



Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра інформатики і програмної інженерії

Лабораторна робота №1
з дисципліни
Технології паралельних обчислень

Виконав:

студент групи ПІ-13:
Бабашев О. Д.

Перевірив:

ст.викл.
Дифучин А. Ю.

Київ 2024

Завдання до комп'ютерного практикуму 1 «Розробка потоків та дослідження пріоритету запуску потоків»

1. Реалізуйте програму імітації руху більярдних кульок, в якій рух кожної кульки відтворюється в окремому потоці (див. презентацію «Створення та запуск потоків в java» та приклад). Спостерігайте роботу програми при збільшенні кількості кульок. Поясніть результати спостереження. Опишіть переваги потокової архітектури програм. 10 балів.
2. Модифікуйте програму так, щоб при потраплянні в «лузу» кульки зникали, а відповідний потік завершував свою роботу. Кількість кульок, яка потрапила в «лузу», має динамічно відображатись у текстовому полі інтерфейсу програми. 10 балів.
3. Виконайте дослідження параметру `priority` потоку. Для цього модифікуйте програму «Більярдна кулька» так, щоб кульки червоного кольору створювались з вищим пріоритетом потоку, в якому вони виконують рух, ніж кульки синього кольору. Спостерігайте рух червоних та синіх кульок при збільшенні загальної кількості кульок. Проведіть такий експеримент. Створіть багато кульок синього кольору (з низьким пріоритетом) і одну червоного кольору, які починають рух в одному й тому ж самому місці більярдного стола, в одному й тому ж самому напрямку та з однаковою швидкістю. Спостерігайте рух кульки з більшим пріоритетом. Повторіть експеримент кілька разів, значно збільшуючи кожного разу кількість кульок синього кольору. Зробіть висновки про вплив пріоритету потоку на його роботу в залежності від загальної кількості потоків. 20 балів.
4. Побудуйте ілюстрацію методу `join()` класу `Thread` через взаємодію потоків, що відтворюють рух більярдних кульок різного кольору. Поясніть результат, який спостерігається. 10 балів.
5. Створіть два потоки, один з яких виводить на консоль символ '-', а інший – символ '|'. Запустіть потоки в основній програмі так, щоб вони виводили свої символи в рядок. Виведіть на консоль 100 таких рядків. Поясніть виведений результат. 10 балів. Використовуючи найпростіші методи управління потоками, добийтесь почергового виведення на консоль символів. 15 балів.
6. Створіть клас `Counter` з методами `increment()` та `decrement()`, які збільшують та зменшують значення лічильника відповідно. Створіть два потоки, один з яких збільшує 100000 разів значення лічильника, а інший – зменшує 100000 разів значення лічильника. Запустіть потоки на одночасне виконання. Спостерігайте останнє значення лічильника. Поясніть результат. 10 балів. Використовуючи синхронізований доступ, добийтесь правильної роботи лічильника при одночасній роботі з ним двох і більше потоків. Опрацюйте використання таких способів синхронізації: синхронізований метод, синхронізований блок, блокування об'єкта. Порівняйте способи синхронізації. 15 балів.

Завдання 1

Лістинг коду:

Ball.java

```
package Task1;

import java.awt.*;
import java.awt.geom.Ellipse2D;
import java.util.Random;

//
// By Cay S. Horstman
//

class Ball {
    private Component canvas;
    private static final int XSIZE = 20;
    private static final int YSIZE = 20;
    private int x = 0;
    private int y = 0;
    private int dx = 2;
    private int dy = 2;

    public Ball(Component c){
        this.canvas = c;

        if(Math.random() < 0.5){
            x = new Random().nextInt(this.canvas.getWidth());
            y = 0;
        }else{
            x = 0;
            y = new Random().nextInt(this.canvas.getHeight());
        }
    }
}
```

```
        y = new Random().nextInt(this.canvas.getHeight());
    }
}
```

```
public void draw (Graphics2D g2){
    g2.setColor(Color.darkGray);
    g2.fill(new Ellipse2D.Double(x,y,XSIZE,YSIZE));

}
```

```
public void move(){
    x+=dx;
    y+=dy;
    if(x<0){
        x = 0;
        dx = -dx;
    }
    if(x+XSIZE>=this.canvas.getWidth()){
        x = this.canvas.getWidth()-XSIZE;
        dx = -dx;
    }
    if(y<0){
        y=0;
        dy = -dy;
    }
    if(y+YSIZE>=this.canvas.getHeight()){
        y = this.canvas.getHeight()-YSIZE;
        dy = -dy;
    }
    this.canvas.repaint();
}
```

```
    }  
}
```

BallCanvas.java

```
package Task1;  
  
import javax.swing.*.*;  
import java.awt.*.*;  
import java.util.ArrayList;  
  
public class BallCanvas extends JPanel{  
    private final ArrayList<Ball> balls = new ArrayList<>();  
  
    public void add(Ball b){  
        this.balls.add(b);  
    }  
  
    @Override  
    public void paintComponent(Graphics g){  
        super.paintComponent(g);  
        Graphics2D g2 = (Graphics2D)g;  
        for(int i=0; i<balls.size();i++){  
            Ball b = balls.get(i);  
            b.draw(g2);  
        }  
    }  
}
```

Bounce.java

```
package Task1;  
  
import javax.swing.*.*;  
  
public class Bounce {  
    public static void main(String[] args) {
```

```

        BounceFrame frame = new BounceFrame();
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        frame.setVisible(true);
        System.out.println("Thread name = " +
            Thread.currentThread().getName());
    }
}

```

BounceFrame.java

```

package Task1;

import javax.swing.*.*;
import java.awt.*.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class BounceFrame extends JFrame {
    private BallCanvas canvas;
    public static final int WIDTH = 450;
    public static final int HEIGHT = 350;
    public static final int BALL_COUNT = 200;

    public BounceFrame() {
        this.setSize(WIDTH, HEIGHT);
        this.setTitle("Bounce programm");
        this.canvas = new BallCanvas();
        System.out.println("In Frame Thread name = "
            + Thread.currentThread().getName());
        Container content = this.getContentPane();
        content.add(this.canvas, BorderLayout.CENTER);
        JPanel buttonPanel = new JPanel();
    }
}

```

```

buttonPanel.setBackground(Color.lightGray);
JButton buttonStart = new JButton("Start");
JButton buttonStop = new JButton("Stop");
buttonStart.addActionListener(new ActionListener() {

    @Override
    public void actionPerformed(ActionEvent e) {
        for (int i = 1; i <= BALL_COUNT; i++) {
            Ball b = new Ball(canvas);
            canvas.add(b);

            BallThread thread = new BallThread(b);
            thread.start();
            System.out.println("Thread name = " +
                               thread.getName());
        }
    }
});
buttonStop.addActionListener(new ActionListener() {

    @Override
    public void actionPerformed(ActionEvent e) {
        System.exit(0);
    }
});
buttonPanel.add(buttonStart);
buttonPanel.add(buttonStop);
content.add(buttonPanel, BorderLayout.SOUTH);
}
}

```

BallThread.java

```
package Task1;
```

```
public class BallThread extends Thread {
```

```
    private final Ball b;
```

```
    public BallThread(Ball ball){
```

```
        b = ball;
```

```
    }
```

```
    @Override
```

```
    public void run(){
```

```
        try{
```

```
            for(int i=1; i<200000; i++){
```

```
                b.move();
```

```
                System.out.println("Thread name = "
```

```
                    + Thread.currentThread().getName());
```

```
                Thread.sleep(5);
```

```
            }
```

```
        } catch (InterruptedException ex) { System.out.println("Interrupted!!!"); }}
```

Результати виконання коду:

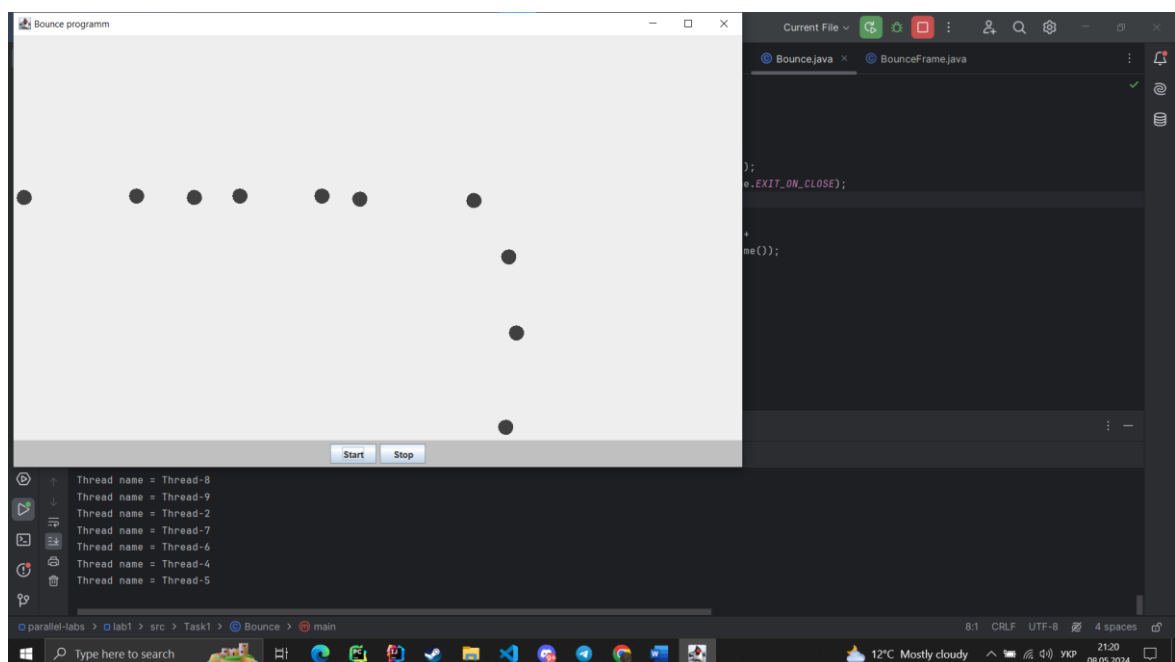


Рисунок 1.1 – Виконання з 10 кульками.

Кульки переміщуються плавно, програма працює швидко.

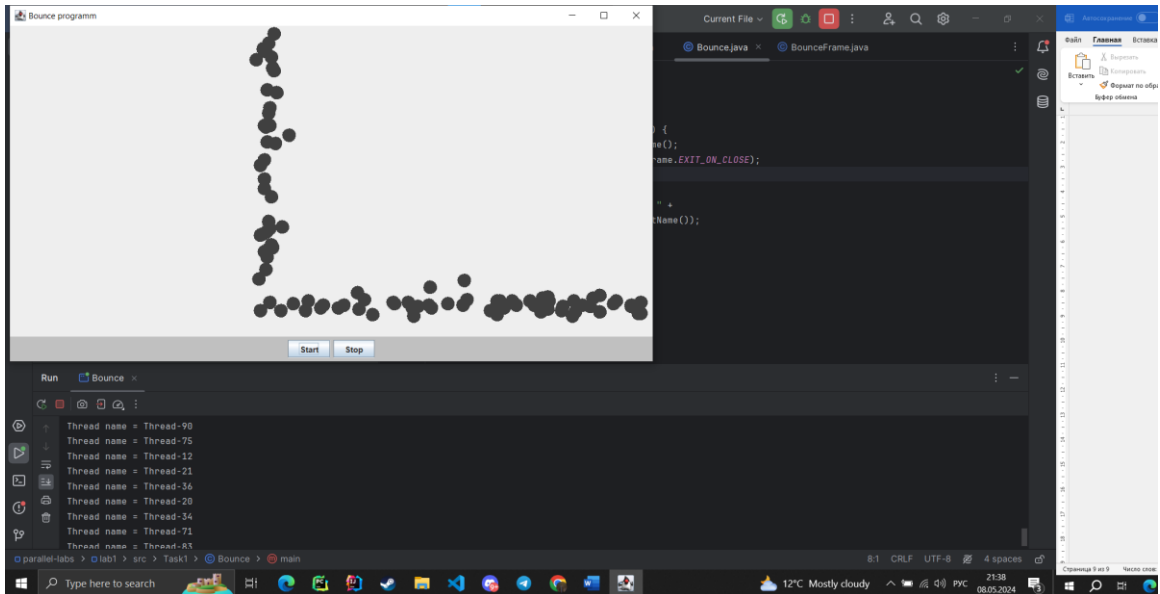


Рисунок 1.2 – Виконання з 100 кульками.

Аналогічна якість роботи програми.

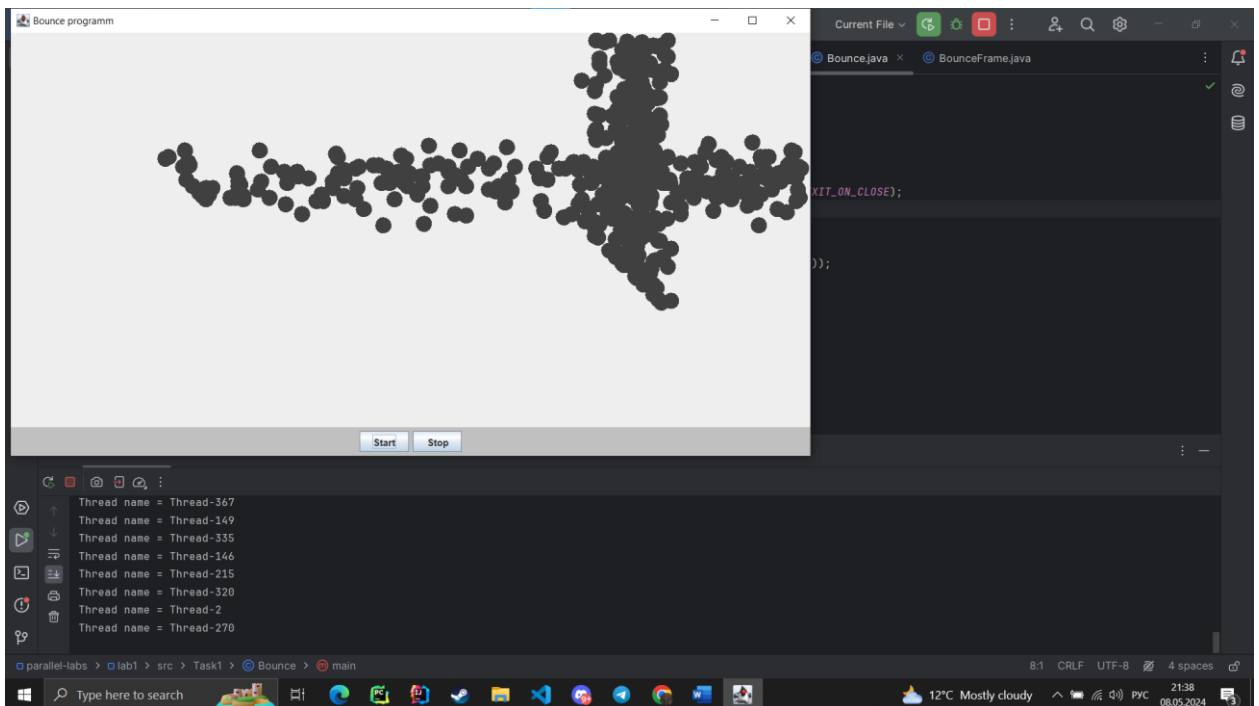


Рисунок 1.3 – Виконання з 500 кульками.

При виконанні програми спостерігаються деякі затримки.

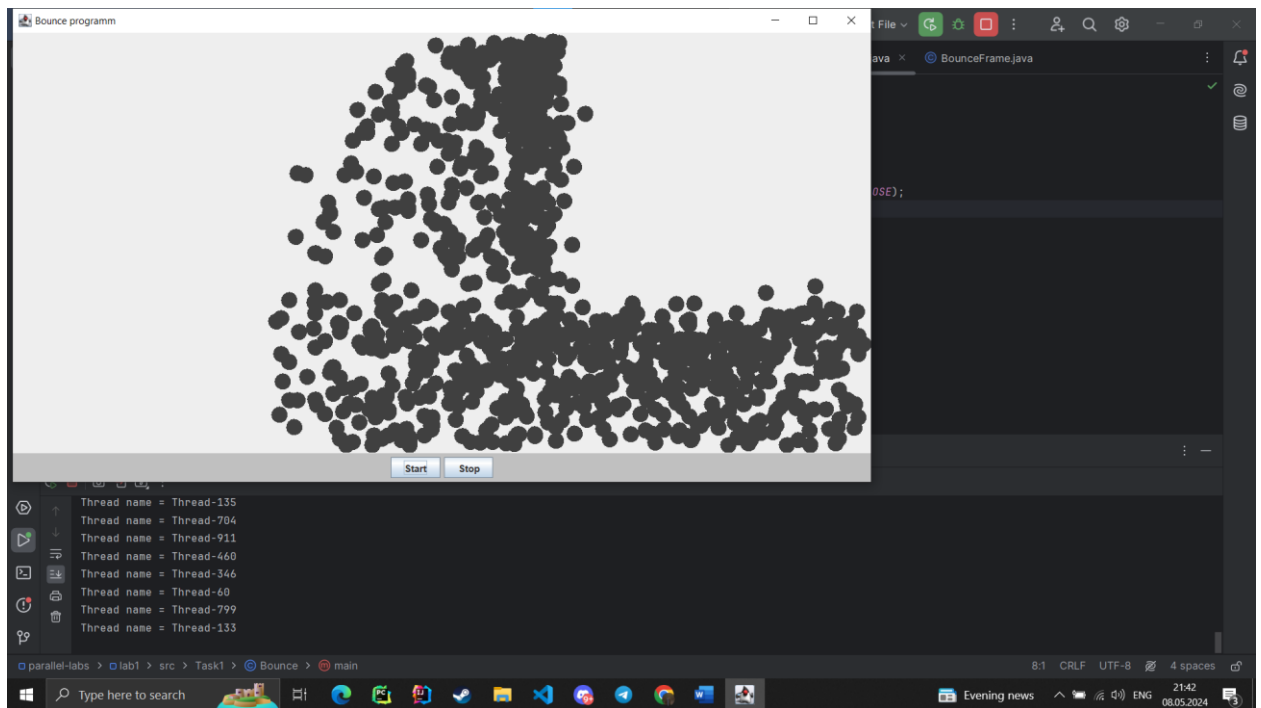


Рисунок 1.4 – Виконання з 1000 кульками.

При виконанні коду з 1000 кульок програма працює повільно, спостерігається перенавантаження системи, кульки переміщуються не плавно.

За результатами експерименту можна помітити, що із збільшенням кількості кульок зростає число потоків, які обробляють ці кульки. Проте, відповідно до спостережень, зі збільшенням потоків погіршується продуктивність програми. Це може бути викликане конфліктами при доступі до ресурсів, таких як обчислювальна потужність чи пам'ять, або ж синхронізацією між потоками, що призводить до затримок у виконанні коду.

Завдання 2

Лістинг коду:

Ball.java

```
package Task2;

import java.awt.*;
import java.awt.geom.Ellipse2D;
import java.util.Random;

public class Ball {
```

```
private final BallCanvas canvas;
private final Sink sink;
private static final int XSIZE = 20;
private static final int YSIZE = 20;
private int x;
private int y;
private int dx = 2;
private int dy = 2;
```

```
public Ball(BallCanvas c, Sink sink){
    this.canvas = c;
    this.sink = sink;
    if(Math.random()<0.5){
        x = new Random().nextInt(this.canvas.getWidth());
        y = 0;
    }else{
        x = 0;
        y = new Random().nextInt(this.canvas.getHeight());
    }
}
```

```
public void draw (Graphics2D g2){
    g2.setColor(Color.darkGray);
    g2.fill(new Ellipse2D.Double(x,y,XSIZE,YSIZE));
}
```

```
public void move(){
    x += dx;
```

```
y += dy;
```

```
if(x<0){
```

```
    x = 0;
```

```
    dx = -dx;
```

```
}
```

```
if(x+XSIZE>=this.canvas.getWidth()){
```

```
    x = this.canvas.getWidth()-XSIZE;
```

```
    dx = -dx;
```

```
}
```

```
if(y<0){
```

```
    y = 0;
```

```
    dy = -dy;
```

```
}
```

```
if(y+YSIZE>=this.canvas.getHeight()){
```

```
    y = this.canvas.getHeight()-YSIZE;
```

```
    dy = -dy;
```

```
}
```

```
if (sink.checkCollision(x, y)) {
```

```
    canvas.remove(this);
```

```
    return;
```

```
}
```

```
    this.canvas.repaint();
```

```
}
```

```
}
```

```
BallCanvas.java
```

```
package Task2;
```

```
import javax.swing.*;
import java.awt.*;
import java.util.ArrayList;
```

```
public class BallCanvas extends JPanel{
    private final ArrayList<Ball> balls = new ArrayList<>();
    private Sink sink;
    private BounceFrame bounceFrame;

    public void add(Ball b){
        this.balls.add(b);
    }

    public void addSink(Sink sink) {
        this.sink = sink;
    }

    public void setBounceFrame(BounceFrame bounceFrame) {
        this.bounceFrame = bounceFrame;
    }

    @Override
    public void paintComponent(Graphics g){
        super.paintComponent(g);
        Graphics2D g2 = (Graphics2D)g;
        for(int i=0; i<balls.size();i++){
            Ball b = balls.get(i);
            if (b != null) {
                b.draw(g2);
            }
        }
    }
}
```

```

    }
    g2.setColor(Color.BLACK);
    if (sink != null) {
        sink.draw(g2);
    }
}

public void updateSinkCounter() {
    int ballsInSink = sink.getBallsInSink();
    bounceFrame.updateSinkLabel(ballsInSink);
}

public synchronized void remove(Ball b) {
    balls.remove(b);
    repaint();
}

public boolean containsBall(Ball b) {
    return balls.contains(b);
}
}

```

BallThread.java

```

package Task2;

public class BallThread extends Thread {
    private final Ball b;
    private final Sink sink;
    private final BallCanvas canvas;

    public BallThread(Ball ball, Sink sink, BallCanvas canvas){
        b = ball;
    }
}

```

```
    this.sink = sink;
    this.canvas = canvas;
}
```

```
@Override
```

```
public void run(){
    try{
        for(int i=1; i<200000; i++){
            b.move();
            if(!canvas.containsBall(b)){
                sink.incrementBallsInSink();
                canvas.updateSinkCounter();
                break;
            }
            System.out.println("Thread name = "
                               + Thread.currentThread().getName());
            Thread.sleep(5);
        }
    } catch (InterruptedException ex){
        System.out.println("Interrupted!!!");
    }
}
```

Bounce.java

```
package Task2;
```

```
import javax.swing.*;
```

```
public class Bounce {
    public static void main(String[] args) {
```

```

        BounceFrame frame = new BounceFrame();
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        frame.setVisible(true);
        System.out.println("Thread name = " +
            Thread.currentThread().getName());
    }
}

```

BounceFrame.java

```

package Task2;

import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class BounceFrame extends JFrame {
    private final BallCanvas canvas;
    private final Sink sink;
    private final JLabel sinkLabel = new JLabel("0");
    public static final int WIDTH = 450;
    public static final int HEIGHT = 350;
    public static final int BALL_COUNT = 100;

    public BounceFrame() {
        this.setSize(WIDTH, HEIGHT);
        this.setTitle("Bounce programm");
        this.canvas = new BallCanvas();

        this.sink = new Sink();
    }
}

```



```
canvas.addSink(sink);
```

```
System.out.println("In Frame Thread name = "  
    + Thread.currentThread().getName());
```

```
Container content = this.getContentPane();  
content.add(this.canvas, BorderLayout.CENTER);
```

```
JPanel buttonPanel = new JPanel();  
buttonPanel.setBackground(Color.lightGray);
```

```
JButton buttonStart = new JButton("Start");  
JButton buttonStop = new JButton("Stop");
```

```
buttonStart.addActionListener(new ActionListener() {
```

```
    @Override
```

```
    public void actionPerformed(ActionEvent e) {
```

```
        for (int i = 1; i <= BALL_COUNT; i++) {
```

```
            Ball b = new Ball(canvas, sink);
```

```
            canvas.add(b);
```

```
            BallThread thread = new BallThread(b, sink, canvas);
```

```
            thread.start();
```

```
            System.out.println("Thread name = " +
```

```
                thread.getName());
```

```
        }
```

```

        }
    });
    buttonStop.addActionListener(e -> System.exit(0));

    // buttonStop.addActionListener(new ActionListener() {
    //     @Override
    //     public void actionPerformed(ActionEvent e) {System.exit(0);
    // }
    //});

    buttonPanel.add(buttonStart);
    buttonPanel.add(buttonStop);
    content.add(buttonPanel, BorderLayout.SOUTH);

    JPanel labelPanel = new JPanel();
    labelPanel.add(sinkLabel);
    labelPanel.setBackground(Color.lightGray);
    content.add(labelPanel, BorderLayout.NORTH);

    canvas.setBounceFrame(this);
}

public void updateSinkLabel(int ballsInSink) {
    sinkLabel.setText(String.valueOf(ballsInSink));
}
}

```

Sink.java

```

package Task2;
import java.awt.*;
import java.awt.geom.Ellipse2D;

```

```

public class Sink {
    private int ballsInSink = 0;
    private static final int XSIZE = 40;
    private static final int YSIZE = 40;
    private final int width = 200;
    private final int height = 85;

    public Sink() {

    }

    public void incrementBallsInSink() {
        ballsInSink++;
    }

    public int getBallsInSink() {

        return ballsInSink;
    }

    public void draw(Graphics2D g2) {
        g2.setColor(Color.BLACK);
        g2.fill(new Ellipse2D.Double(width, height, XSIZE, YSIZE));
        g2.setColor(Color.WHITE);
    }

    public boolean checkCollision(int x, int y) {
        return (x >= width && x <= width + XSIZE &&
            y >= height && y <= height + YSIZE);
    }
}

```

}

Результати виконання коду:

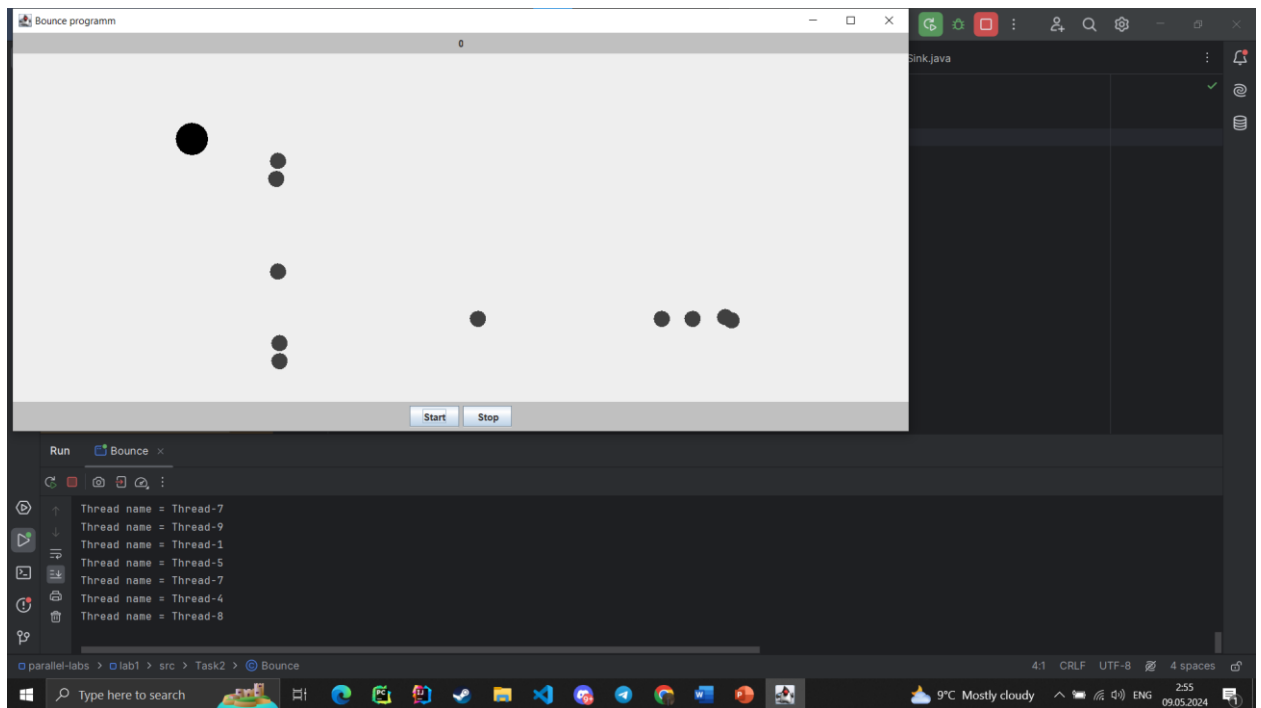


Рисунок 2.1 –Початок виконання.

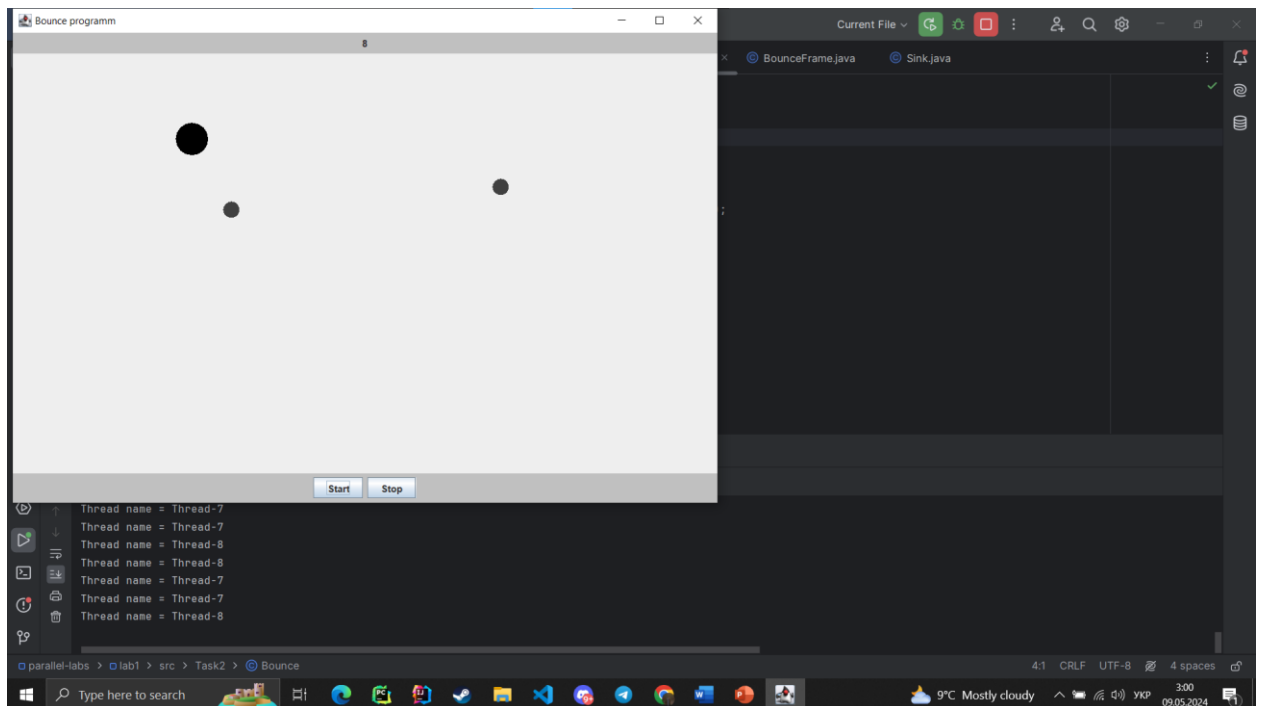


Рисунок 2.2 –Процес виконання.

У консолі лишилось і працюють тільки 2 потоки, всі інші після потрапляння в лунку завершили свою роботу. Можемо зазначити у консолі що працює 2 потоки при 2 лишившихся кульках.

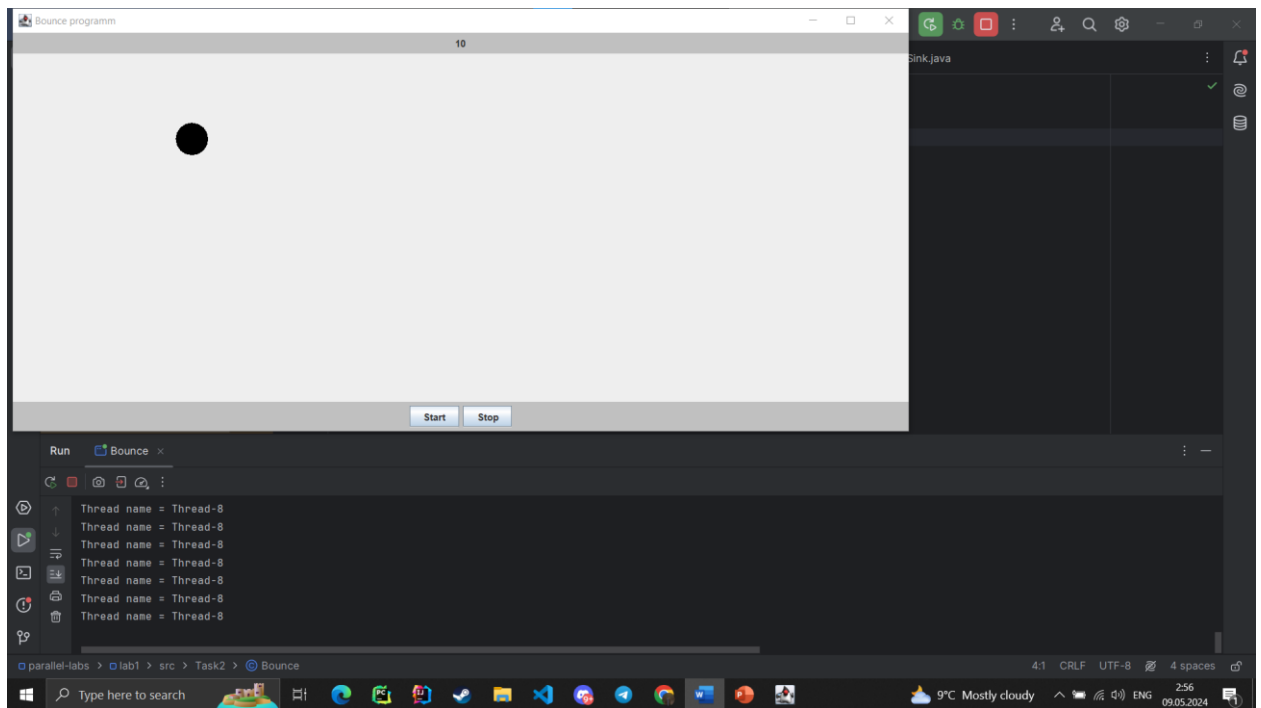


Рисунок 2.3 –Результат виконання.

Всі потоки завершили свою роботу. У лунці всі 10 кульок і всі 10 потоків звершили свою роботу. Результат тестування показує, що програма працює правильно.

Завдання 3

Лістинг коду:

Ball.java

```
package Task3;
import java.awt.*;
import java.awt.geom.Ellipse2D;
```

```
class Ball {
    private Component canvas;
    private static final int XSIZE = 20;
    private static final int YSIZE = 20;
    private int x = 0;
    private int y= 0;
```

```
private int dx = 2;
```

```
private int dy = 2;
```

```
private final Color color;
```

```
public Ball(Component c, Color color, int x, int y){
```

```
    this.canvas = c;
```

```
    this.color = color;
```

```
    this.x = x;
```

```
    this.y = y;
```

```
}
```

```
public void draw (Graphics2D g2){
```

```
    g2.setColor(color);
```

```
    g2.fill(new Ellipse2D.Double(x,y,XSIZE,YSIZE));
```

```
}
```

```
public void move(){
```

```
    x+=dx;
```

```
    y+=dy;
```

```
    if(x<0){
```

```
        x = 0;
```

```
        dx = -dx;
```

```
}
```

```
    if(x+XSIZE>=this.canvas.getWidth()){
```

```
        x = this.canvas.getWidth()-XSIZE;
```

```
        dx = -dx;
```

```
}
```

```

        if(y<0){
            y=0;
            dy = -dy;
        }
        if(y+YSIZE>=this.canvas.getHeight()){
            y = this.canvas.getHeight()-YSIZE;
            dy = -dy;
        }
        this.canvas.repaint();
    }
}

```

BallThread.java

```

package Task3;

public class BallThread extends Thread {
    private final Ball b;

    public BallThread(Ball ball){
        b = ball;
    }

    @Override
    public void run(){
        try{
            for(int i=1; i<200000; i++){
                b.move();
                System.out.println("Thread name = "
                    + Thread.currentThread().getName());
                Thread.sleep(5);
            }
        } catch (InterruptedException ex){

```

```

        System.out.println("Interrupted!!!");
    }
}
}

```

BallCanvas.java

```
package Task3;
```

```

import javax.swing.*;
import java.awt.*;
import java.util.ArrayList;

```

```

public class BallCanvas extends JPanel{
    private final ArrayList<Ball> balls = new ArrayList<>();

    public void add(Ball b){
        this.balls.add(b);
    }

    @Override
    public void paintComponent(Graphics g){
        super.paintComponent(g);
        Graphics2D g2 = (Graphics2D)g;
        for(int i=0; i<balls.size();i++){
            Ball b = balls.get(i);
            b.draw(g2);
        }
    }
}

```

Bounce.java

```
package Task3;
```



```

import javax.swing.*;

public class Bounce {
    public static void main(String[] args) {
        BounceFrame frame = new BounceFrame();
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        frame.setVisible(true);
        System.out.println("Thread name = " +
            Thread.currentThread().getName());
    }
}

```

BounceFrame.java

```

package Task3;

import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class BounceFrame extends JFrame {
    private final BallCanvas canvas;
    public static final int WIDTH = 450;
    public static final int HEIGHT = 350;
    public static final int BALL_COUNT = 100;
    public BounceFrame() {
        this.setSize(WIDTH, HEIGHT);
        this.setTitle("Bounce program");
        this.canvas = new BallCanvas();
    }
}

```

```

Container content = this.getContentPane();
content.add(this.canvas, BorderLayout.CENTER);

JPanel buttonPanel = new JPanel();
buttonPanel.setBackground(Color.lightGray);

JButton buttonStart = new JButton("Start");
JButton buttonStop = new JButton("Stop");
JButton buttonExperiment = new JButton("Experiment");

buttonStart.addActionListener(new ActionListener() {

    @Override
    public void actionPerformed(ActionEvent e) {
        for (int i = 1; i <= BALL_COUNT; i++) {
            Color color = (i % 2 != 0) ? Color.RED : Color.BLUE;
            Ball b = new Ball(canvas, color);
            canvas.add(b);

            BallThread thread = new BallThread(b);

            if (color.equals(Color.RED)) {
                thread.setPriority(Thread.MAX_PRIORITY);
            } else {
                thread.setPriority(Thread.MIN_PRIORITY);
            }

            thread.start();

```

```

        System.out.println("Thread name = " +
            thread.getName());
    }
}
});

```

```

buttonExperiment.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        for (int i = 1; i <= BALL_COUNT; i++) {
            Color color = (i == 1) ? Color.RED : Color.BLUE;
            Ball b = new Ball(canvas, color, 0, 0);
            canvas.add(b);

            BallThread thread = new BallThread(b);
            if (i == 1) {
                thread.setPriority(Thread.MAX_PRIORITY);
            } else {
                thread.setPriority(Thread.MIN_PRIORITY);
            }
            thread.start();
            System.out.println("Thread name = " +
                thread.getName());
        }
    }
});

```

```

buttonStop.addActionListener(e -> System.exit(0));

```

```

buttonPanel.add(buttonStart);

```

```

        buttonPanel.add(buttonStop);

        buttonPanel.add(buttonExperiment);

        content.add(buttonPanel, BorderLayout.SOUTH);
    }
}

```

Результати виконання коду:

У першій частині дослідження запустимо програму, яка буде генерувати однакову кількість синіх та червоних кульок, при цьому у червоних кульок буде максимальний пріоритет.

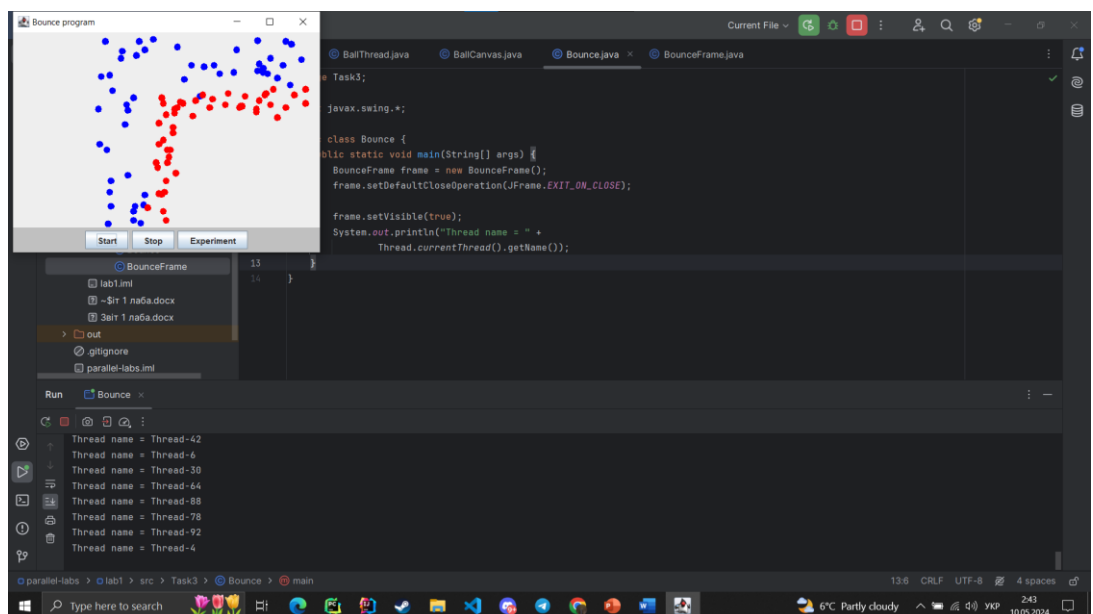


Рисунок 3.1 – Виконання програми зі 100 кульками.

З часом кульки червоного кольору відриваються від кульок з синім, оскільки мають максимальний пріоритет. До того ж червоні рухаються із більш постійною швидкістю, на відміну від синіх, тому з часом вони будуть більш кучно розташовані, ніж сині.

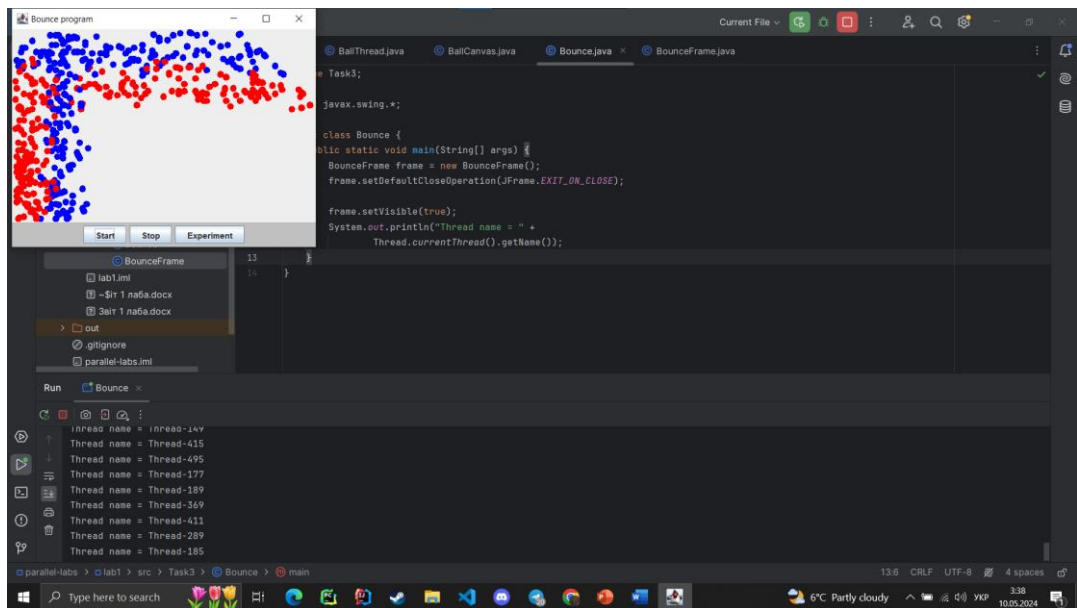


Рисунок 3.2 – Виконання програми зі 500 кульками.

При збільшенні кількості кульок спостерігається пришвидження процесу відриву червоних від синіх. Це пояснюється тим, що червоні мають максимальний пріоритет, більше ресурсів, а звідси більшу швидкість.

У другій частині дослідження проведемо експеримент. Буде генеруватись тільки одна кулька червоного кольору з максимальним пріоритетом, всі інші будуть сині і мати мінімальний пріоритет. Всі кульки будуть генерувати в одній точці, мати однаковий напрямок та однакову швидкість.

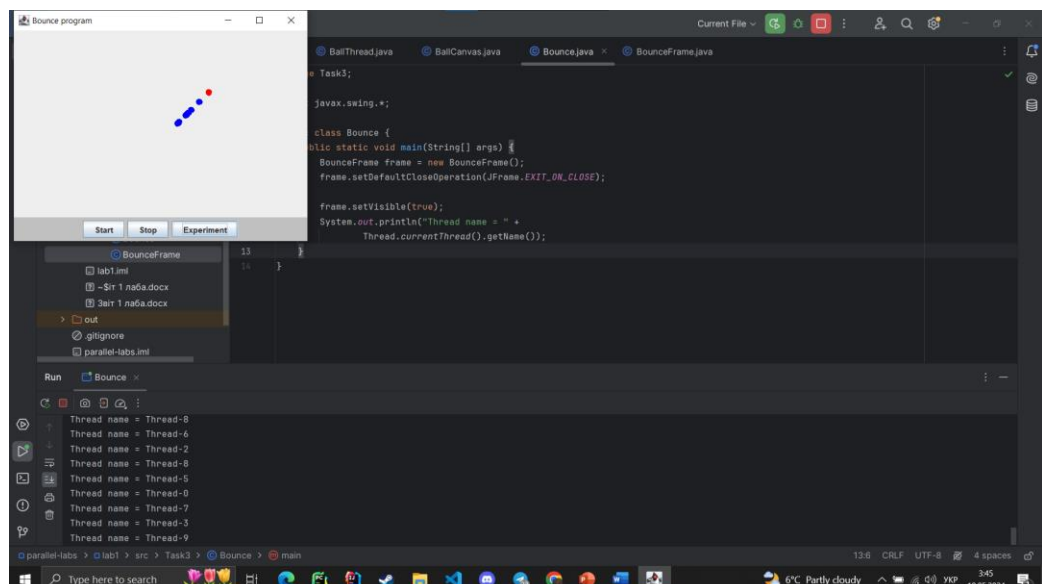


Рисунок 3.3 – Виконання експерименту з 10 кульками.

При запуску 10 кульок, червона кулька із максимальним пріоритетом за довгий проміжок часу виривається вперед, що є цілком логічним, бо вона є швидшою, але під сині кульки з мінімальним пріоритетом видається теж

достатня кількість ресурсів , тому мають велику швидкість і перегнати їх складно.

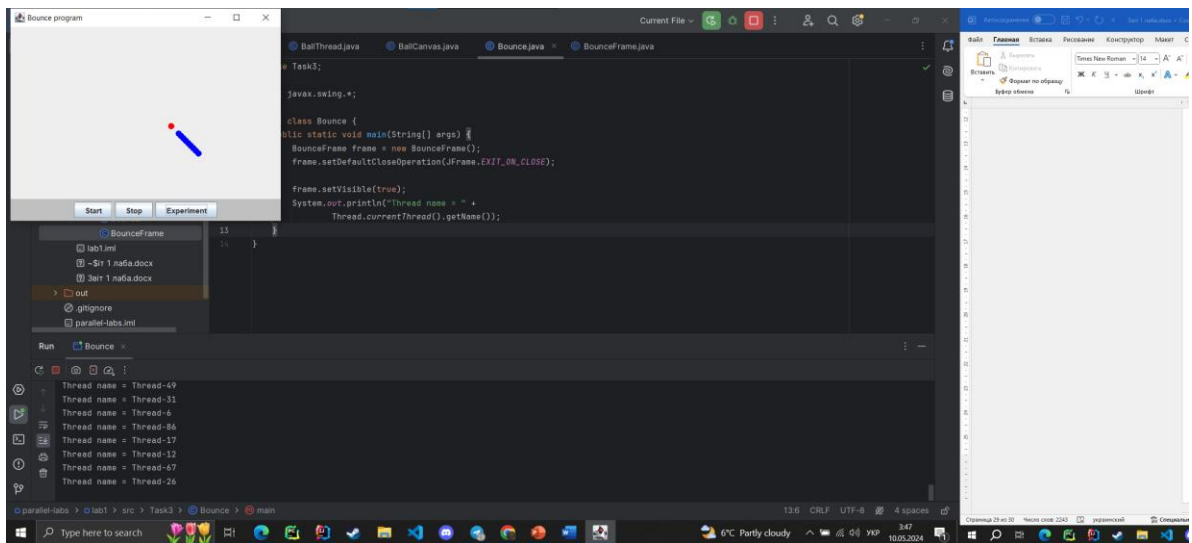


Рисунок 3.4 – Виконання експерименту з 100 кульками.

При проведенні експерименту зі 100 кульками, червона вирвалася вперед набагато швидше ніж при проведенні експерименту із 10 кульками, сині стали повільніші через нестачу ресурсів, бо їх на мінімальному пріоритеті стало більше.

Завдання 4

Лістинг коду:

Ball.java

```
package Task4;

import java.awt.*;
import java.awt.geom.Ellipse2D;
import java.util.Random;
```

```
class Ball {
    private Component canvas;
    private static final int XSIZE = 20;
    private static final int YSIZE = 20;
    private int x = 0;
```

```
private int y = 0;
private int dx = 2;
private int dy = 2;
private final Color color;
```

```
public Ball(Component c, Color color){
    this.canvas = c;
    this.color = color;
    if(Math.random() < 0.5){
        x = new Random().nextInt(this.canvas.getWidth());
        y = 0;
    } else {
        x = 0;
        y = new Random().nextInt(this.canvas.getHeight());
    }
}
```

```
public void draw (Graphics2D g2){
    g2.setColor(color);
    g2.fill(new Ellipse2D.Double(x, y, XSIZE, YSIZE));
}
```

```
public void move(){
    x += dx;
    y += dy;

    if (x < 0){
        x = 0;
        dx = -dx;
    }
}
```

```

        if (x + XSIZE >= this.canvas.getWidth()){
            x = this.canvas.getWidth() - XSIZE;
            dx = -dx;
        }
        if (y < 0){
            y = 0;
            dy = -dy;
        }
        if(y + YSIZE >= this.canvas.getHeight()){
            y = this.canvas.getHeight() - YSIZE;
            dy = -dy;
        }
        this.canvas.repaint();
    }
}

```

BallCanvas.java

```
package Task4;
```

```
import javax.swing.*;
```

```
import java.awt.*;
```

```
import java.util.ArrayList;
```

```
public class BallCanvas extends JPanel{
```

```
    private final ArrayList<Ball> balls = new ArrayList<>();
```

```
    public void add(Ball b){
```

```
        this.balls.add(b);
```

```
    }
```

```
    @Override
```

```
    public void paintComponent(Graphics g){
```



```

        super.paintComponent(g);
        Graphics2D g2 = (Graphics2D)g;
        for(int i=0; i<balls.size();i++){
            Ball b = balls.get(i);
            b.draw(g2);
        }
    }
}

```

BallThread.java

```

package Task4;

public class BallThread extends Thread {
    private final Ball b;

    public BallThread(Ball ball){
        b = ball;
    }

    @Override
    public void run(){
        try{
            for(int i=1; i<200; i++){
                b.move();
                System.out.println("Thread name = "
                    + Thread.currentThread().getName());
                Thread.sleep(5);
            }
        } catch (InterruptedException ex){
            System.out.println("Interrupted!!!");
        }
    }
}

```

```
}
```

Bounce.java

```
package Task4;
```

```
import javax.swing.*;
```

```
public class Bounce {
```

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

```
        BounceFrame frame = new BounceFrame();
```

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

```
        frame.setVisible(true);
```

```
        System.out.println("Thread name = " +
```

```
            Thread.currentThread().getName());
```

```
    }
```

```
}
```

BounceFrame.java

```
package Task4;
```

```
import javax.swing.*;
```

```
import java.awt.*;
```

```
import java.awt.event.ActionEvent;
```

```
import java.awt.event.ActionListener;
```

```
public class BounceFrame extends JFrame {
```

```
    private final BallCanvas canvas;
```

```
    public static final int WIDTH = 450;
```

```
    public static final int HEIGHT = 350;
```

```
    public static final int BALL_COUNT = 10;
```

```

public BounceFrame() {
    this.setSize(WIDTH, HEIGHT);
    this.setTitle("Bounce programm");
    this.canvas = new BallCanvas();

    Container content = this.getContentPane();
    content.add(this.canvas, BorderLayout.CENTER);

    JPanel buttonPanel = new JPanel();
    buttonPanel.setBackground(Color.lightGray);

    JButton buttonStart = new JButton("Start");
    JButton buttonStop = new JButton("Stop");

    buttonStart.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            new Thread(() -> {

                Ball[] balls = new Ball[BALL_COUNT];
                BallThread[] ballThreads = new BallThread[BALL_COUNT];

                for (int i = 0; i < BALL_COUNT; i++) {
                    int red = (int) (Math.random() * 256);
                    int green = (int) (Math.random() * 256);
                    int blue = (int) (Math.random() * 256);
                    Color color = new Color(red, green, blue);

                    Ball b = new Ball(canvas, color);
                    canvas.add(b);
                }
            })
        }
    });
}

```

```

        balls[i] = b;
    }

    for (int i = 0; i < BALL_COUNT; i++) {
        if (i > 0) {
            try {
                ballThreads[i - 1].join(); // Зачекати завершення
попереднього потоку
            } catch (InterruptedException ex) {
                System.err.println(ex.getMessage());
            }
        }
        ballThreads[i] = new BallThread(balls[i]);
        ballThreads[i].start();
    }

    }).start();
}

});

buttonStop.addActionListener(e -> System.exit(0));

buttonPanel.add(buttonStart);
buttonPanel.add(buttonStop);
content.add(buttonPanel, BorderLayout.SOUTH);

}

}

```

Результати виконання коду:

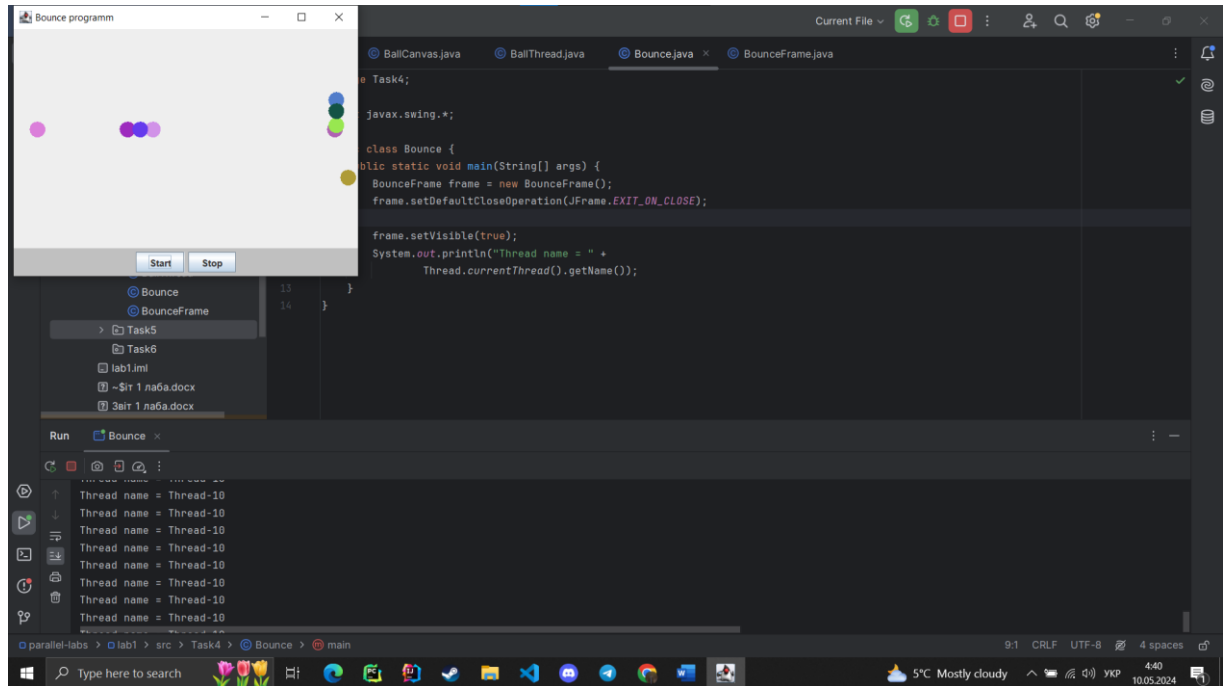


Рисунок 4.1 – Виконання програми.

Спостерігається почергове виконання потоків, кожна кулька чекає на завершення виконання одного потоку і переходить до виконання іншого, при цьому минула кулька перестає рухатись, а наступна починає.

Завдання 5

Лістинг коду:

Sync.java

```
package Task5;
```

```
public class Sync {  
    private volatile boolean permission;  
    private volatile boolean stop;  
    private int symbols;  
    private int lines;
```

```
    public Sync() {  
        permission = true;  
        stop = false;
```

```

    symbols = 0;
    lines = 0;
}

public boolean isStop() {
    return stop;
}

public synchronized void printSymbol(char symbol, boolean control){
    while (this.permission != control && !isStop()) {
        try {
            wait();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

    if(isStop()) {
        notifyAll();
        return;
    }

    System.out.print(symbol);
    permission = !permission;
    symbols++;

    if (symbols == 100) {
        symbols = 0;
        System.out.println();
        lines++;
    }
}

```

```

    }

    if (lines == 100) {
        stop = true;
    }

    notifyAll();
}
}

Main.java
package Task5;

import java.util.Scanner;

public class Main {
    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);
        System.out.println("Оберіть метод виводу: 1 - простий, 2 - синхр");
        int method = scanner.nextInt();

        switch (method) {
            case 1:
                new Thread(new PrintCharThread('-')).start();
                new Thread(new PrintCharThread('|')).start();
                break;
            case 2:
                Sync permission = new Sync();
                new Thread(new PrintCharThreadSync( '-',permission,true)).start();
                new Thread(new PrintCharThreadSync( '|',permission,false)).start();

```

```

        break;
    default:
        System.out.println("Невірний вибір. Завершення програми.");
    }
}
}

```

PrintCharThread.java

```
package Task5;
```

```
class PrintCharThread implements Runnable {
```

```
    private final char symbol;
```

```
    public PrintCharThread(char symbol) {
```

```
        this.symbol = symbol;
```

```
    }
```

```
    @Override
```

```
    public void run() {
```

```
        for (int i = 0; i < 100; i++) {
```

```
            for (int j = 0; j < 100; j++) {
```

```
                System.out.print(symbol);
```

```
            }
```

```
            System.out.println();
```

```
        }
```

```
    }
```

```
}
```

PrintCharThreadSync.java

```
package Task5;
```

```
public class PrintCharThreadSync implements Runnable {
```


private final char symbol;

private final Sync permission;

```
private final boolean control;
```

```
public PrintCharThreadSync(char symbol, Sync permission, boolean control) {
    this.symbol = symbol;
    this.permission = permission;
    this.control = control;
}
```

@Override

```
public void run() {
    while (!permission.isStop()) {
        permission.printSymbol(symbol,control);
    }
}
```

Результати виконання коду:

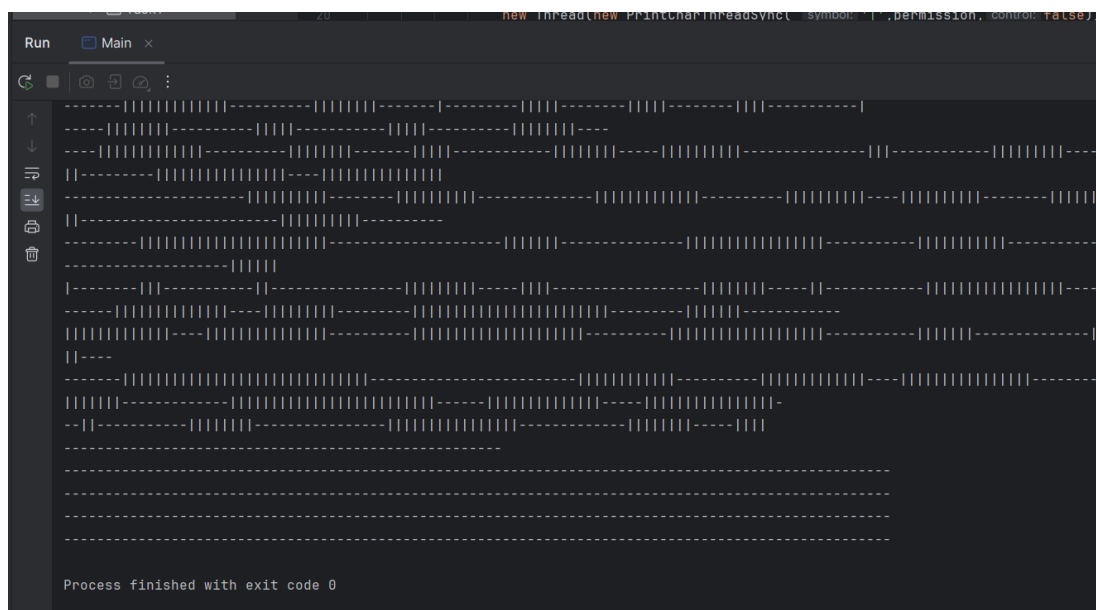


Рисунок 5.1 – Виконання простого виведення символів двома потоками.

Спостерігаємо мішане довільне виведення.

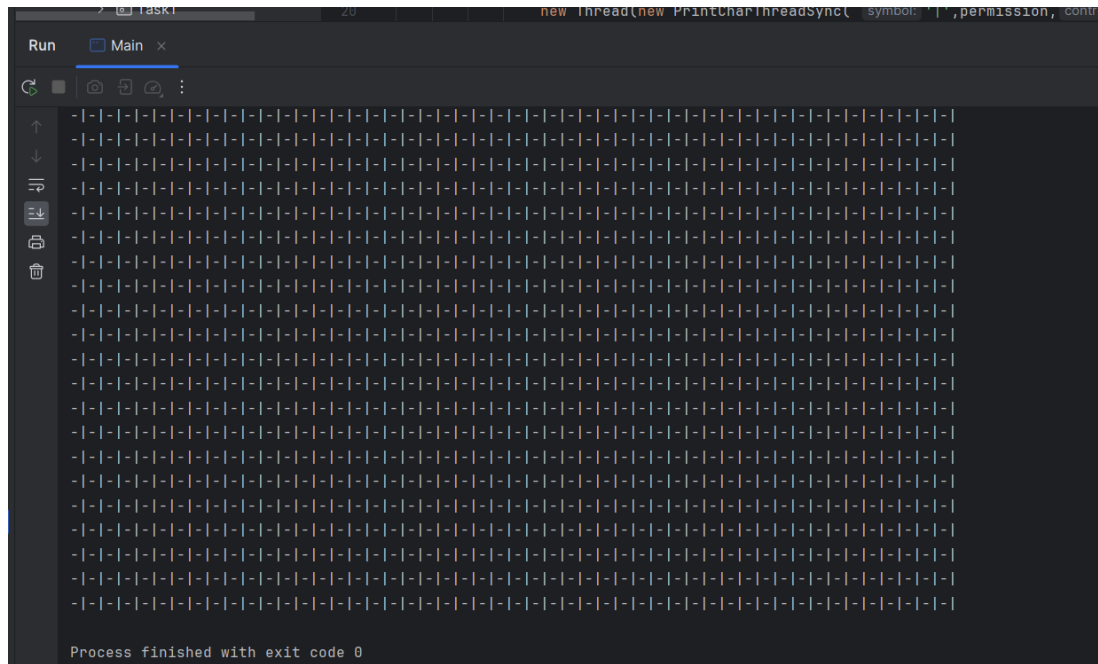


Рисунок 5.2 – Виконання синхронізованого виведення двома потоками.

Спостерігаємо почергове виведення.

Завдання 6

Лістинг коду:

Counter.java

package Task6;

```
public class Counter extends CounterBase {
```

```
    private int value = 0 ;
```

```
    @Override
```

```
    public void increment() {
```

```
        value++;
```

```
    }
```

```
    @Override
```

```
    public void decrement() {
```

```
        value--;  
    }
```

```
    @Override  
    public int getValue() {  
        return value;  
    }  
}
```

CounterBase.java

```
package Task6;
```

```
public abstract class CounterBase {  
    public abstract void increment();  
    public abstract void decrement();  
    public abstract int getValue();  
}
```

DecThread.java

```
package Task6;
```

```
class DecThread extends Thread {  
    private final CounterBase value;  
  
    public DecThread(CounterBase value) {  
        this.value = value;  
    }  
}
```

```
    @Override  
    public void run() {  
        for (int i = 0; i < 100000; i++) {  
            value.decrement();  
        }  
    }  
}
```

```
    }  
    }  
}
```

IncThread.java

```
package Task6;
```

```
class IncThread extends Thread {  
    private final CounterBase value;
```

```
    public IncThread(CounterBase value) {  
        this.value = value;  
    }
```

```
    @Override
```

```
    public void run() {  
        for (int i = 0; i < 1000000; i++) {  
            value.increment();  
        }  
    }  
}
```

LockCounter.java

```
package Task6;
```

```
import java.util.concurrent.locks.Lock;
```

```
import java.util.concurrent.locks.ReentrantLock;
```

```
class LockCounter extends CounterBase {  
    private int value = 0;  
    private final Lock lock = new ReentrantLock();  
    @Override  
    public void increment() {
```

```

        lock.lock();
    try {
        value++;
    } finally {
        lock.unlock();
    }
}

@Override
public void decrement() {
    lock.lock();
    try {
        value--;
    } finally {
        lock.unlock();
    }
}

@Override
public int getValue() {
    lock.lock();
    try {
        return value;
    } finally {
        lock.unlock();
    }
}
}

```

Main.java

```
package Task6;
```

```

public class Main {
    public static void main(String[] args) {

        Counter counter = new Counter();
        runThreads(counter);

        LockCounter lockCounter = new LockCounter();
        runThreads(lockCounter);

        SyncBlockCounter syncBlockCounter = new SyncBlockCounter();
        runThreads(syncBlockCounter);

        SyncMethodCounter syncMethodCounter = new SyncMethodCounter();
        runThreads(syncMethodCounter);
    }

    private static void runThreads(CounterBase value) {
        IncThread incrementThread = new IncThread(value);
        DecThread decrementThread = new DecThread(value);

        incrementThread.start();
        decrementThread.start();

        try {
            incrementThread.join();
            decrementThread.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}

```

```
        System.out.println(value.getClass().getSimpleName() + " = " +  
value.getValue());  
    }
```

```
}
```

SyncBlockCounter.java

```
package Task6;
```

```
class SyncBlockCounter extends CounterBase {
```

```
    private int value = 0;
```

```
    @Override
```

```
    public void increment() {
```

```
        synchronized(this) {
```

```
            value++;
```

```
        }
```

```
    }
```

```
    @Override
```

```
    public void decrement() {
```

```
        synchronized(this) {
```

```
            value--;
```

```
        }
```

```
    }
```

```
    @Override
```

```
    public int getValue() {
```

```
        synchronized(this) {
```

```
            return value;
```

```
        }
```

```
    }
```

```
}
```

SyncMethodCounter.java

```
package Task6;
```

```
class SyncBlockCounter extends CounterBase {
```

```
    private int value = 0;
```

```
    @Override
```

```
    public void increment() {
```

```
        synchronized(this) {
```

```
            value++;
```

```
        }
```

```
    }
```

```
    @Override
```

```
    public void decrement() {
```

```
        synchronized(this) {
```

```
            value--;
```

```
        }
```

```
    }
```

```
    @Override
```

```
    public int getValue() {
```

```
        synchronized(this) {
```

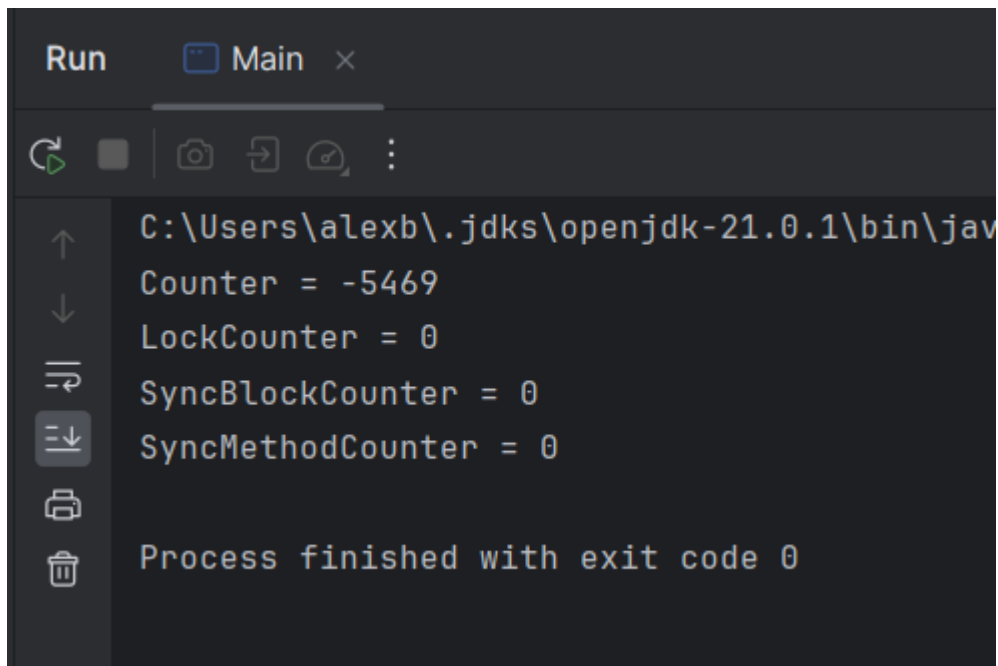
```
            return value;
```

```
        }
```

```
    }
```

```
}
```

Результати виконання коду:



```
Run Main x
C:\Users\alexb\.jdk\openjdk-21.0.1\bin\java
Counter = -5469
LockCounter = 0
SyncBlockCounter = 0
SyncMethodCounter = 0
Process finished with exit code 0
```

Рисунок 6.1 – Робота всіх лічильників.

При паралельній роботі обох потоків, при звичайному каунті виходить не 0, тому що потоки не синхронізовано і один з них спрацьовую поки другий ще не завершив роботу. Через це маєм не 0. Розроблено 3 різні способи синхронізації, всі працюють належим чином.

Синхронізація методу та блоку відрізняється в області застосування. Синхронізація методу охоплює всю його область, тобто весь код методу виконується атомарно одним потоком. У той час, як синхронізація блоку дозволяє синхронізувати лише певний фрагмент коду, що надає більшу гнучкість у керуванні спільними ресурсами між потоками. Блокування об'єкту є більш універсальним механізмом, який дозволяє контролювати доступ до критичних секцій коду, і надає більше можливостей у синхронізації взаємодії між потоками.

Висновок

У цьому практичному завданні було досліджено та відображено реальні випробування з управління потоками на прикладі програми, що моделює рух більярдних куль. Ми вивчали вплив кількості куль на продуктивність програми, досліджуючи її роботу в різних умовах, ми вивчили та застосували метод `join()` для керування потоками, створили потоки та встановили їх пріоритети. Також ми дослідили різні способи синхронізації, щоб уникнути конфліктів за ресурсами. На прикладі завдань з лічильником та виведенням символів ми розглянули, як забезпечити безпеку доступу до спільних ресурсів у багатопоточному середовищі.