

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ  
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»  
ФАКУЛЬТЕТ ІНФОРМАТИКИ ТА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

Кафедра інформатики та програмної інженерії

**Звіт**

З лабораторної роботи № 1 з дисципліни  
«Технології паралельних обчислень»

Тема: «Розробка потоків та дослідження пріоритету запуску потоків»

**Виконав(ла)**

*ІП-14 Бабіч Денис*

(шифр, прізвище, ім'я, по батькові)

**Перевірів**

*Дифучина О. Ю.*

(шифр, прізвище, ім'я, по батькові)

Київ 2024

## ОСНОВНА ЧАСТИНА

**Мета роботи:** Розробка потоків та дослідження пріоритету запуску потоків.

- 1.1. Реалізуйте програму імітації руху більярдних кульок, в якій рух кожної кульки відтворюється в окремому потоці. Спостерігайте роботу програми при збільшенні кількості кульок. Поясніть результати спостереження. Опишіть переваги потокової архітектури програм.

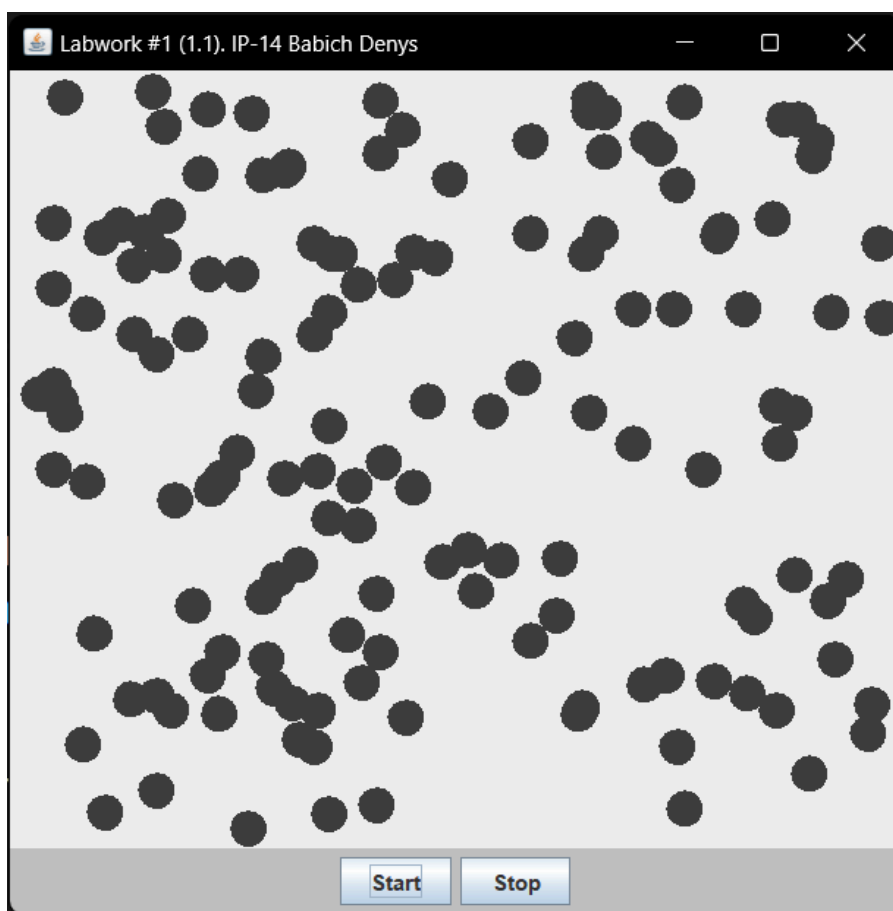


Рисунок 1.1 – Вікно програми

Під час збільшення кількості кульок відбувається сповільнення руху всіх кульок, що свідчить про боротьбу за обчислювальні ресурси.

Потокова архітектура має ряд ключових переваг: ефективне використання обчислювальних ресурсів, потенційний виграш у продуктивності, тощо.

**Лістинг класу BallThread**

```
package task1;

import java.lang.InterruptedException;

public final class BallThread extends Thread
{
    private final int ITERATIONS_COUNT = 10000;
    private final int THREAD_SLEEP_DURATION_MS = 5;

    private Ball ball;

    public BallThread(Ball ball)
    {
        this.ball = ball;
    }

    @Override
    public void run()
    {
        try
        {
            for(int i = 1; i < this.ITERATIONS_COUNT; ++i)
            {
                ball.move();

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

                Thread.sleep(this.THREAD_SLEEP_DURATION_MS);
            }
        }
        catch (InterruptedException e)
        {
            e.printStackTrace();
        }
    }
}
```

```

        }
    }
    catch(InterruptedException ex) { }
}
}

```

### **Лістинг класу MainFrame**

```

package task1;

import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JButton;

import java.awt.Color;
import java.awt.Container;
import java.awt.BorderLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public final class MainFrame extends JFrame
{
    private final int FRAME_WIDTH = 500;
    private final int FRAME_HEIGHT = 500;

    private BallCanvas ballCanvas;

    private JPanel panelButtons;

    private JButton buttonStart;
    private JButton buttonStop;

```

```

private Container container;

public MainFrame()
{
    this.setSize(this.FRAME_WIDTH, this.FRAME_HEIGHT);
    this.ballCanvas = new BallCanvas();

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

    this.container = this.getContentPane();
    this.container.add(this.ballCanvas, BorderLayout.CENTER);

    this.panelButtons = new JPanel();
    this.panelButtons.setBackground(Color.lightGray);

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

    this.buttonStart.addActionListener(new ActionListener()
    {
        @Override
        public void actionPerformed(ActionEvent e)
        {
            Ball ball = new Ball(MainFrame.this.ballCanvas);
            MainFrame.this.ballCanvas.add(ball);

            BallThread thread = new BallThread(ball);

```

```
        thread.start();

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

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

this.panelButtons.add(this.buttonStart);
this.panelButtons.add(this.buttonStop);

this.container.add(this.panelButtons, BorderLayout.SOUTH);
}
}
```

- 1.2. 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.

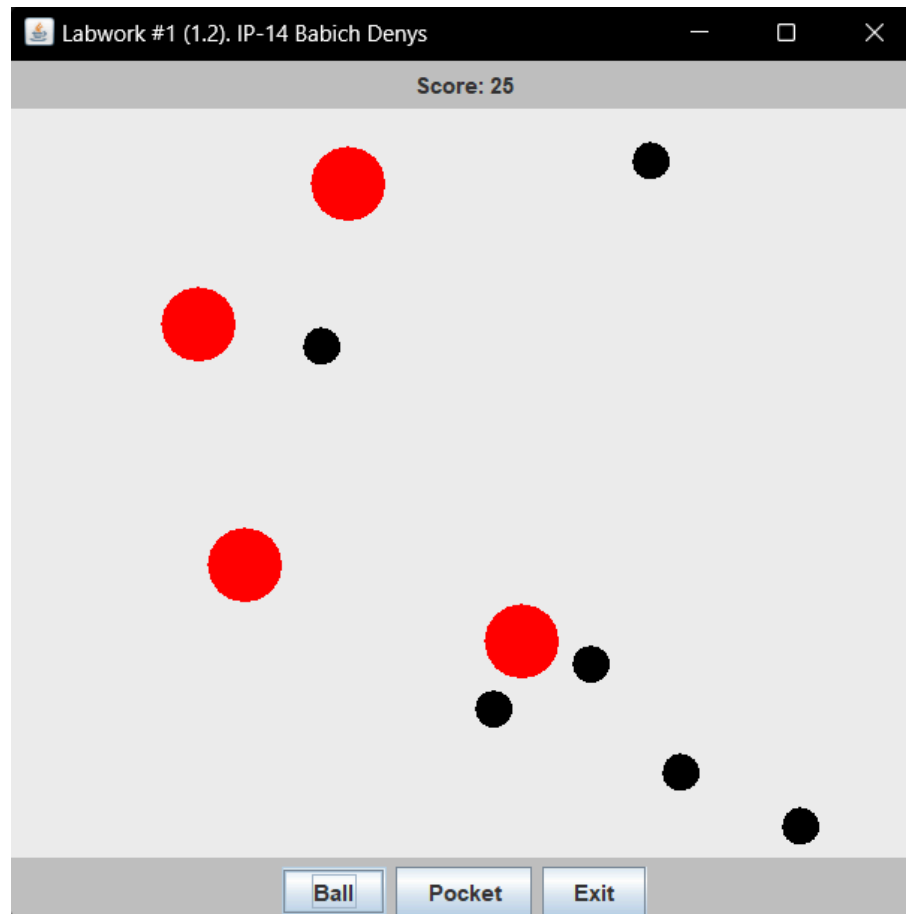


Рисунок 1.2 – Вікно програми

### Лістинг класу Ball

```
package task2;  
  
import java.awt.Color;  
import java.awt.Component;  
import java.awt.Graphics2D;  
import java.awt.geom.Ellipse2D;
```

```
import java.util.Random;

public final class Ball
{
    private final int SIZE_X = 20;
    private final int SIZE_Y = 20;

    private Component component;

    private int x = 0;
    private int y = 0;
    private int dx = 2;
    private int dy = 2;

    public Ball(Component component)
    {
        this.component = component;

        Random random = new Random();

        this.x = random.nextInt(this.component.getWidth());
        this.y = random.nextInt(this.component.getHeight());
    }

    public int getX()
    {
        return this.x;
    }
}
```



```
public int getY()  
{  
    return this.y;  
}
```

```
public void draw(Graphics2D g2)  
{  
    g2.setColor(Color.BLACK);  
    g2.fill(new Ellipse2D.Double(this.x, this.y, this.SIZE_X, this.SIZE_Y));  
}
```

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

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

```
    if(x + this.SIZE_X >= this.component.getWidth())  
    {  
        this.dx = -this.dx;  
        this.x = this.component.getWidth() - this.SIZE_X;  
    }
```

```

    if(this.y < 0)
    {
        this.y = 0;
        this.dy = -this.dy;
    }

    if(this.y + this.SIZE_Y >= this.component.getHeight())
    {
        this.dy = -this.dy;
        this.y = this.component.getHeight() - this.SIZE_Y;
    }

    this.component.repaint();
}

public boolean isInPocket(int pocketPositionX, int pocketPositionY)
{
    double distance = Math.sqrt(Math.pow(this.x - pocketPositionX, 2) +
Math.pow(this.y - pocketPositionY, 2));
    return distance < Pocket.RADIUS;
}
}

```

### **Лістинг класу BallThread**

```

package task2;

import java.lang.InterruptedException;

public final class BallThread extends Thread
{

```

```
private final int ITERATIONS_COUNT = 10000;
private final int THREAD_SLEEP_DURATION_MS = 5;
```

```
private Ball ball;
private Canvas canvas;
```

```
public BallThread(Ball ball, Canvas canvas)
{
    this.ball = ball;
    this.canvas = canvas;
}
```

```
@Override
```

```
public void run()
```

```
{
    try
    {
        for(int i = 1; i < this.ITERATIONS_COUNT; ++i)
        {
            ball.move();

            if (this.canvas.hasCollided(ball))
            {
                this.interrupt();

                System.out.println("Thread name = " +
Thread.currentThread().getName() + ". Status: " +
Thread.currentThread().getState());
                break;
            }
        }
    }
}
```

```

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

        Thread.sleep(this.THREAD_SLEEP_DURATION_MS);
    }
}
catch(InterruptedException ex) { }
}
}

```

### **Лістинг класу Canvas**

```

package task2;

import java.awt.Graphics;
import java.awt.Graphics2D;

import javax.swing.JPanel;

import java.util.ArrayList;

public final class Canvas extends JPanel
{
    private ArrayList<Ball> balls;
    private ArrayList<Pocket> pockets;

    public Canvas()
    {
        this.balls = new ArrayList<Ball>();
        this.pockets = new ArrayList<Pocket>();
    }
}

```

```
}
```

```
public void add(Ball ball)
```

```
{
```

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

```
}
```

```
public void add(Pocket pocket)
```

```
{
```

```
    this.pockets.add(pocket);
```

```
}
```

```
@Override
```

```
public void paintComponent(Graphics graphics)
```

```
{
```

```
    super.paintComponent(graphics);
```

```
    Graphics2D g2 = (Graphics2D) graphics;
```

```
    for(int i = 0; i < this.balls.size(); ++i)
```

```
    {
```

```
        this.balls.get(i).draw(g2);
```

```
    }
```

```
    for(int i = 0; i < this.pockets.size(); ++i)
```

```
    {
```

```
        this.pockets.get(i).draw(g2);
```

```
    }
```

```
}
```

```

public boolean hasCollided(Ball ball)
{
    for (Pocket pocket : this.pockets)
    {
        if (ball.isInPocket(pocket.getX(), pocket.getY()))
        {
            this.balls.remove(ball);

            MainFrame.updateScore();

            return true;
        }
    }

    return false;
}

```

### **Лістинг класу Pocket**

```

package task2;

import java.awt.Color;
import java.awt.Component;
import java.awt.Graphics2D;
import java.awt.geom.Ellipse2D;
import java.util.Random;

public final class Pocket {
    private final int SIZE_X = 40;
    private final int SIZE_Y = 40;

```

```
public static final int RADIUS = 20;

private Component component;

private int x;
private int y;

public Pocket(Component component) {
    this.component = component;

    Random random = new Random();

    x = random.nextInt(this.component.getWidth() - this.SIZE_X);
    y = random.nextInt(this.component.getHeight() - this.SIZE_Y);
}

public int getX() {
    return this.x;
}

public int getY() {
    return this.y;
}

public void draw(Graphics2D g2) {
    g2.setColor(Color.RED);
    g2.fill(new Ellipse2D.Double(x, y, SIZE_X, SIZE_Y));
}
}
```

- 1.3. Виконайте дослідження параметру `priority` потоку. Для цього модифікуйте програму «Більярдна куляка» 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.

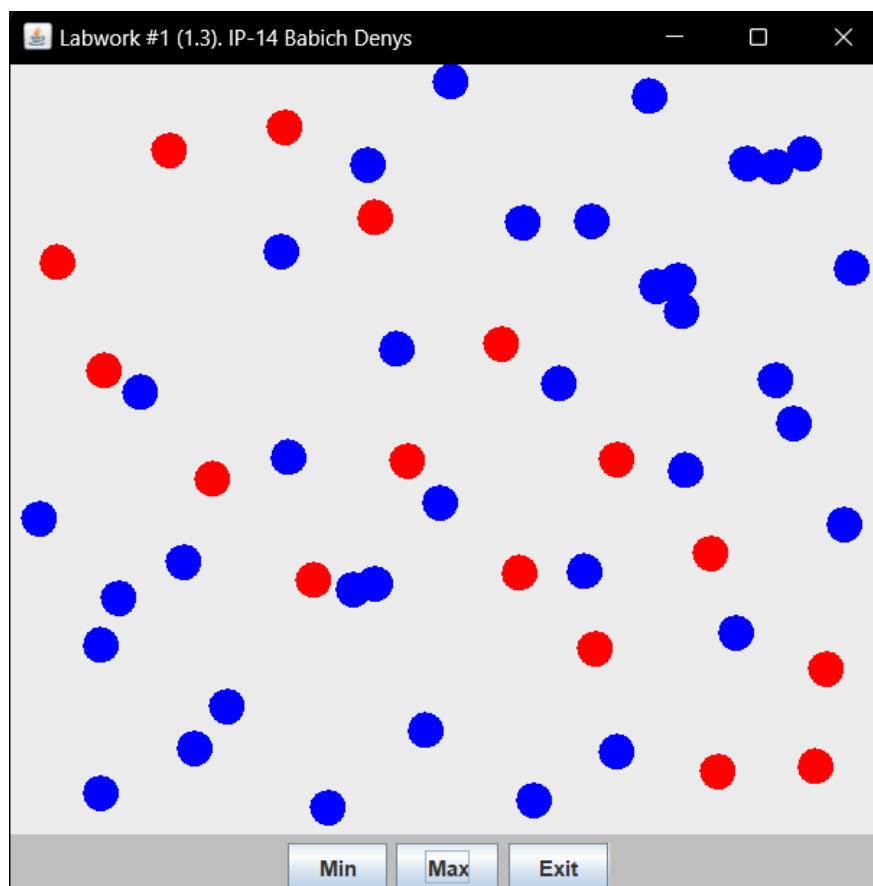


Рисунок 1.3 – Вікно програми



Пріоритет потоку визначає, яким чином операційна система розподіляє процесорний час між потоками. Вищий пріоритет означає, що потік має більше шансів отримати процесорний час. У експерименті з більярдними кульками, кульки з вищим пріоритетом (червоні) мають більше шансів рухатися швидше, ніж кульки з нижчим пріоритетом (сині). Але коли кількість потоків збільшується, різниця в швидкості може стати менш помітною.

### **Лістинг класу MainFrame**

```
package task3;

import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JButton;

import java.awt.Color;
import java.awt.Container;
import java.awt.BorderLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public final class MainFrame extends JFrame
{
    private final int FRAME_WIDTH = 500;
    private final int FRAME_HEIGHT = 500;

    private Canvas canvas;

    private JPanel panelButtons;

    private JButton buttonExit;
```

```

private JButton buttonSpawn;

private Container container;

public MainFrame()
{
    this.canvas = new Canvas();

    this.setSize(this.FRAME_WIDTH, this.FRAME_HEIGHT);

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

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

    this.panelButtons = new JPanel();
    this.panelButtons.setBackground(Color.lightGray);

    this.buttonExit = new JButton("Exit");
    this.buttonSpawn = new JButton("Spawn");

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

```

```

this.buttonSpawn.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        final int MIN_BALLS_COUNT = 500;

        for (int i = 0; i < MIN_BALLS_COUNT; ++i) {
            Ball ball = new Ball(MainFrame.this.canvas, Color.BLUE, 0, 0);
            MainFrame.this.canvas.add(ball);

            BallThread thread = new BallThread(ball);
            thread.setPriority(Thread.MIN_PRIORITY);
            thread.start();
        }

        Ball ball = new Ball(MainFrame.this.canvas, Color.RED, 0, 0);
        MainFrame.this.canvas.add(ball);

        BallThread thread = new BallThread(ball);
        thread.setPriority(Thread.MAX_PRIORITY);
        thread.start();
    }
});

this.panelButtons.add(this.buttonExit);
this.panelButtons.add(this.buttonSpawn);

this.container.add(this.panelButtons, BorderLayout.SOUTH);
}
}

```

- 1.4. Побудуйте ілюстрацію методу `join()` класу `Thread` через взаємодію потоків, що відтворюють рух більярдних кульок різного кольору. Поясніть результат, який спостерігається.

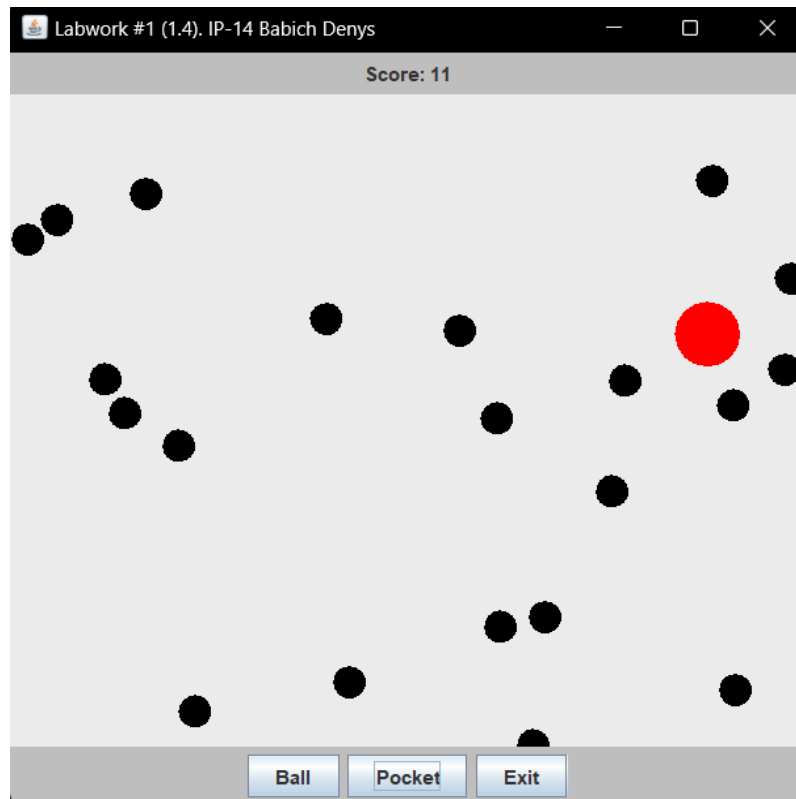


Рисунок 1.4 – Вікно програми

### Лістинг класу `BallThread`

```
package task4;

import java.lang.InterruptedException;

public final class BallThread extends Thread {
    private final int ITERATIONS_COUNT = 10000;
    private final int THREAD_SLEEP_DURATION_MS = 5;
    private Ball ball;
    private Canvas canvas;
    private BallThread previousThread;
```

```

public BallThread(Ball ball, Canvas canvas, BallThread previousThread) {
    this.ball = ball;
    this.canvas = canvas;
    this.previousThread = previousThread;
}

@Override
public void run() {
    try {
        if (previousThread != null) {
            previousThread.join();
        }

        for(int i = 1; i < this.ITERATIONS_COUNT; ++i) {
            ball.move();

            if (this.canvas.hasCollided(ball)) {
                this.interrupt();

                System.out.println("Thread name = " +
Thread.currentThread().getName() + ". Status: " +
Thread.currentThread().getState());
                break;
            }

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

            Thread.sleep(this.THREAD_SLEEP_DURATION_MS);
        }
    }
}

```

```

    }
    catch(InterruptedException ex) { }
}
}

```

### **Лістинг класу MainFrame**

```

package task4;

import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JButton;

import java.awt.Color;
import java.awt.Container;
import java.awt.BorderLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public final class MainFrame extends JFrame
{
    private final int FRAME_WIDTH = 500;
    private final int FRAME_HEIGHT = 500;

    private static int score = 0;

    private static JPanel panelScore;

    private static JLabel labelScore;

```

```
private Canvas canvas;
private Container container;

private JPanel panelButtons;

private JButton buttonStop;
private JButton buttonStart;
private JButton buttonSpawn;

private BallThread currentThread = null;

public MainFrame()
{
    this.canvas = new Canvas();

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

    this.setSize(this.FRAME_WIDTH, this.FRAME_HEIGHT);

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

    this.panelButtons = new JPanel();
    this.panelButtons.setBackground(Color.lightGray);

    MainFrame.panelScore = new JPanel();
    MainFrame.panelScore.setBackground(Color.lightGray);
```

```

MainFrame.labelScore = new JLabel();

this.buttonStop = new JButton("Exit");
this.buttonStart = new JButton("Ball");
this.buttonSpawn = new JButton("Pocket");

this.buttonStart.addActionListener(new ActionListener()
{
    @Override
    public void actionPerformed(ActionEvent e)
    {
        Ball ball = new Ball(MainFrame.this.canvas);
        MainFrame.this.canvas.add(ball);

        BallThread thread = new BallThread(ball, MainFrame.this.canvas,
MainFrame.this.currentThread);

        MainFrame.this.currentThread = thread;
        thread.start();

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

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

```



```

    }
});

this.buttonSpawn.addActionListener(new ActionListener()
{
    @Override
    public void actionPerformed(ActionEvent e)
    {
        Pocket pocket = new Pocket(MainFrame.this.canvas);
        MainFrame.this.canvas.add(pocket);
    }
});

this.panelButtons.add(this.buttonStart);
this.panelButtons.add(this.buttonSpawn);
this.panelButtons.add(this.buttonStop);

MainFrame.panelScore.add(MainFrame.labelScore);

this.container.add(this.panelButtons, BorderLayout.SOUTH);
this.container.add(MainFrame.panelScore, BorderLayout.NORTH);
}

public static synchronized void updateScore()
{
    MainFrame.labelScore.setText("Score: " + (++MainFrame.score));
}
}

```



```

{
    this.character = character;
}

@Override
public void run()
{
    for (int i = 0; i < this.LINES_COUNT; ++i)
    {
        for (int j = 0; j < this.CHARACTERS_COUNT; ++j)
        {
            System.out.print(this.character);

        }

        System.out.println();
    }
}
}

```

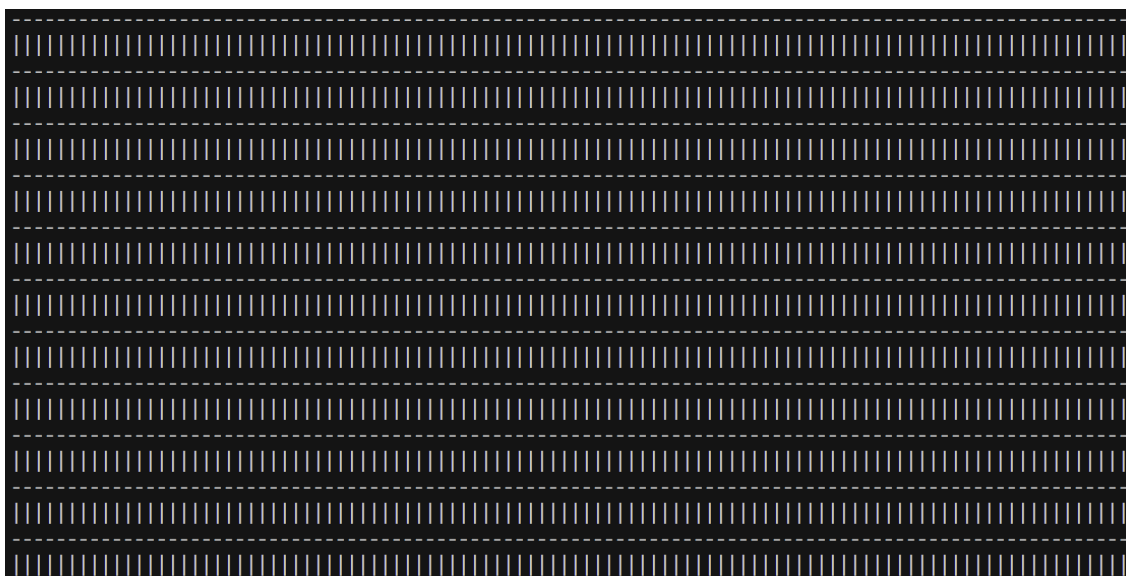


Рисунок 1.6 – Результат виконання у випадку синхронізованого запуску

**Лістинг класу TaskThread**

```
package task5.task5_2;

final class TaskThread extends Thread
{
    private static volatile int linesCount = 0;

    private final int LINES_COUNT = 100;
    private final int CHARACTERS_COUNT = 100;

    private Object lock;
    private char character;

    public TaskThread(char character, Object lock)
    {
        this.lock = lock;
        this.character = character;
    }

    @Override
    public void run()
    {
        synchronized (this.lock)
        {
            try {
                while (TaskThread.linesCount <= LINES_COUNT) {
                    for (int i = 0; i < CHARACTERS_COUNT; ++i) {
                        System.out.print(this.character);
                    }
                }
            }
        }
    }
}
```

```
        ++TaskThread.linesCount;

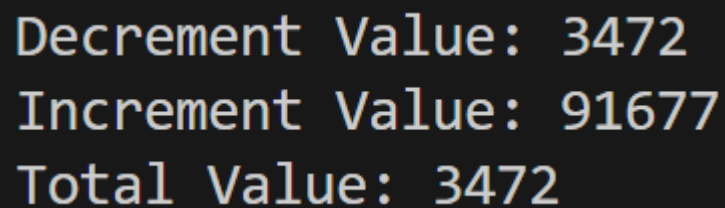
        System.out.println();

        this.lock.notify();

        this.lock.wait();
    }

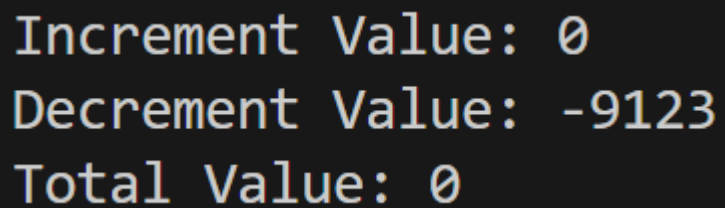
    this.lock.notify();
}
catch (InterruptedException e) {
    e.printStackTrace();
}
}
}
}
```

- 1.6. 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. 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.



```
Decrement Value: 3472
Increment Value: 91677
Total Value: 3472
```

Рисунок 1.7 – Результат виконання без використання жодного способу синхронізації



```
Increment Value: 0
Decrement Value: -9123
Total Value: 0
```

Рисунок 1.8 – Приклад виконання з використанням синхронізації

**Лістинг класу Counter (без синхронізацій)**

```
package task6.task6_1;

public final class Counter
{
    private volatile int value = 0;

    public int getValue()
    {
        return this.value;
    }

    public void increment()
    {
        ++this.value;
    }

    public void decrement()
    {
        --this.value;
    }
}
```

**Лістинг класу Counter (синхронізовані методи)**

```
package task6.task6_2;

public final class Counter
{
    private volatile int value = 0;
```

```

public int getValue()
{
    return this.value;
}

public synchronized void increment()
{
    ++this.value;
}

public synchronized void decrement()
{
    --this.value;
}
}

```

### **Лістинг класу Counter (синхронізований блок)**

```

package task6.task6_3;

public final class Counter
{
    private volatile int value = 0;

    public int getValue()
    {
        return this.value;
    }

    public void increment()

```



```

{
    synchronized(this)
    {
        ++this.value;
    }
}

public void decrement()
{
    synchronized(this)
    {
        --this.value;
    }
}
}

```

### **Лістинг класу Counter (блокування об'єкта)**

```

package task6.task6_4;

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

public final class Counter
{
    private volatile int value = 0;

    private final Lock lock = new ReentrantLock();

    public int getValue()
    {

```

```
        return this.value;
    }

    public void increment() {
        this.lock.lock();

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

    public void decrement()
    {
        this.lock.lock();

        try {
            --this.value;
        }
        finally
        {
            lock.unlock();
        }
    }
}
```

## ВИСНОВКИ

В ході виконання лабораторної роботи було розглянуто такі ключові поняття, як створення та запуск потоків, пріоритети потоків, синхронізація та взаємодія потоків. Під час виконання практичних завдань були визначені переваги потокової архітектури, де кожен потік може виконувати свою роботу паралельно з іншими. Також на практиці було засвоєно, як можна динамічно керувати потоками в реальному часі. Додатково, було досліджено, як пріоритет потоку впливає на його виконання, особливо при великій кількості потоків. Крім того, було засвоєно застосування методу `join()` класу `Thread` для координації взаємодії між потоками. На практиці було проаналізовано взаємодію та конкурувати потоків за ресурси, а також важливість синхронізації при роботі з загальними ресурсами у багатопоточному середовищі.