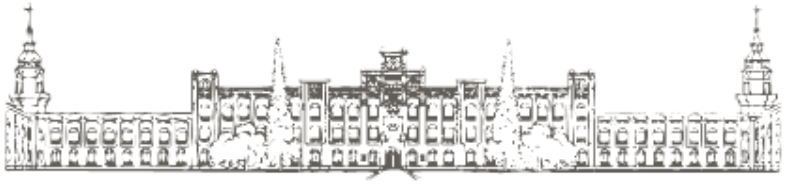


МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
КАФЕДРА ІНФОРМАТИКИ ТА ПРОГРАМНОЇ ІНЖЕНЕРІЇ

Звіт

до лабораторного практикуму 1

«Розробка потоків та дослідження пріоритету запуску потоків»

з дисципліни

«Технології паралельних обчислень»

Виконав:
студент групи ІП-01
Пашковський Євгеній

Київ – 2023

Завдання

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.
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.
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.
4. Побудуйте ілюстрацію методу join() класу Thread через взаємодію потоків, що відтворють рух більярдних кульок різного кольору. Поясніть результат, який спостерігається.
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. Використовуючи найпростіші методи управління потоками, добийтесь почергового виведення на консоль символів.

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.

Хід виконання

Виконання коду відбувалося на пристрої з такими характеристиками:

- Процесор Intel Core i7-8550U. Базова частота 1.8-2.0ГГц, але фактично може досягати 4ГГц у пікових навантаженнях. Містить 4 фізичних ядра та 8 логічних процесорів.
- 16Гб оперативної пам’яті з частотою 2400МГц
- ОС Windows 10.

Програма імітації руху більярдних кульок, в якій рух кожної кульки відтворюється в окремому потоці:

Ball:

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

class Ball {
    private static final int X_SIZE = 20;
    private static final int Y_SIZE = 20;
    private final Component canvas;
    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());
        }
    }

    public static void f() {
        int a = 0;
    }

    public void draw(Graphics2D g2) {
        g2.setColor(Color.BLUE);
        g2.fill(new Ellipse2D.Double(x, y, X_SIZE, Y_SIZE));
    }

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

```
        y += dy;
        if (x < 0) {
            x = 0;
            dx = -dx;
        }
        if (x + X_SIZE >= this.canvas.getWidth()) {
            x = this.canvas.getWidth() - X_SIZE;
            dx = -dx;
        }
        if (y < 0) {
            y = 0;
            dy = -dy;
        }
        if (y + Y_SIZE >= this.canvas.getHeight()) {
            y = this.canvas.getHeight() - Y_SIZE;
            dy = -dy;
        }

        this.canvas.repaint();
    }

    public int getXCentered() {
        return x + X_SIZE;
    }

    public int getYCentered() {
        return y + Y_SIZE;
    }
}
```

BallCanvas:

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

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

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

    public void removeBall(Ball ball) {
        this.balls.remove(ball);
        this.repaint();
    }

    @Override
    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        Graphics2D g2 = (Graphics2D) g;

        for (int i = 0; i < balls.size(); i++) {
            Ball ball = this.balls.get(i);
            ball.draw(g2);
        }
```

```
    }
}
```

BallThread:

```
import java.util.Arrays;

public class BallThread extends Thread {
    private final Ball ball;

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

    @Override
    public void run() {
        try {
            while (true) {
                ball.move();
                System.out.println("Thread name = " +
Thread.currentThread().getName());
                Thread.sleep(5);
            }
        } catch (InterruptedException ignored) {

        }
    }
}
```

Bounce:

```
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:

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

public class BounceFrame extends JFrame {
    public static final int WIDTH = 500;
    public static final int HEIGHT = 500;
    private static final int HOLE_SIZE = 50;
    private final BallCanvas canvas;

    public BounceFrame() {
        this.setSize(WIDTH, HEIGHT);
        this.setTitle("Bounce program");
```

```
        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 button100Start = new JButton("Start 100");
        JButton buttonStop = new JButton("Stop");

        JTextField textField = new JTextField("0", 2);

        buttonStart.addActionListener(e -> {

            Ball ball = new Ball(canvas);
            canvas.addBall(ball);

            BallThread thread = new BallThread(ball);
            thread.start();
            System.out.println("Thread name = " +
thread.getName());
        });
        button100Start.addActionListener(e -> {
            for (int i = 0; i < 100; i++) {
                Ball ball = new Ball(canvas);
                canvas.addBall(ball);

                BallThread thread = new BallThread(ball);
                thread.start();
                System.out.println("Thread name = " +
thread.getName());
            }
        });
        buttonStop.addActionListener(e -> System.exit(0));

        buttonPanel.add(buttonStart);
        buttonPanel.add(button100Start);
        buttonPanel.add(buttonStop);
        buttonPanel.add(textField);

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

Зникнення під час попадання в «лузу»:

Hole:

```
import java.awt.*;

public class Hole {
    private final int size;
    private final Component canvas;
```

```
    private final HolePosition position;

    public Hole(Component canvas, HolePosition position, int size)
{
        this.canvas = canvas;
        this.position = position;
        this.size = size;
    }

    public boolean isPointInside(int x, int y) {
        Position2D position2D = this.getPosition2D();
        return x > position2D.x && y > position2D.y && x <
position2D.x + size && y < position2D.y + size;
    }

    public void draw(Graphics2D g2) {
        g2.setColor(Color.BLACK);
        Position2D position2D = this.getPosition2D();
        g2.fill(new Rectangle(position2D.x, position2D.y, size,
size));
    }

    public Position2D getPosition2D() {
        return switch (position) {
            case LEFT_TOP -> new Position2D(0, 0);
            case RIGHT_TOP -> new Position2D(canvas.getWidth() -
size, 0);
            case LEFT_BOTTOM -> new Position2D(0,
canvas.getHeight() - size);
            case RIGHT_BOTTOM -> new Position2D(canvas.getWidth() -
size, canvas.getHeight() - size);
        };
    }
}
```

HolePosition:

```
public enum HolePosition {
    LEFT_TOP,
    RIGHT_TOP,
    LEFT_BOTTOM,
    RIGHT_BOTTOM
}
```

BounceFrame:

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

public class BounceFrame extends JFrame {
    public static final int WIDTH = 500;
    public static final int HEIGHT = 500;
    private static final int HOLE_SIZE = 50;
    private final BallCanvas canvas;

    private int ballsInAHole = 0;
```

```
    public BounceFrame() {
        this.setSize(WIDTH, HEIGHT);
        this.setTitle("Bounce program");

        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");

        JTextField textField = new JTextField("0", 2);


        Hole leftTopHole = new Hole(canvas, HolePosition.LEFT_TOP,
HOLE_SIZE);
        Hole rightTopHole = new Hole(canvas,
HolePosition.RIGHT_TOP, HOLE_SIZE);
        Hole leftBottomHole = new Hole(canvas,
HolePosition.LEFT_BOTTOM, HOLE_SIZE);
        Hole rightBottomHole = new Hole(canvas,
HolePosition.RIGHT_BOTTOM, HOLE_SIZE);

        Hole[] holes = {leftTopHole, rightTopHole, leftBottomHole,
rightBottomHole};
        canvas.addHoles(holes);

        buttonStart.addActionListener(e -> {
            Ball ball = new Ball(canvas);
            canvas.addBall(ball);

            BallThread thread = new BallThread(ball, () -> {
                ballsInAHole++;
                textField.setText(String.valueOf(ballsInAHole));
                canvas.removeBall(ball);
            }, holes);
            thread.start();
            System.out.println("Thread name = " +
thread.getName());
        });
        buttonStop.addActionListener(e -> System.exit(0));

        buttonPanel.add(buttonStart);
        buttonPanel.add(buttonStop);
        buttonPanel.add(textField);

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

TextFieldSetter:

```
public interface TextFieldSetter {
    void updateBallsInAHole();
}
```

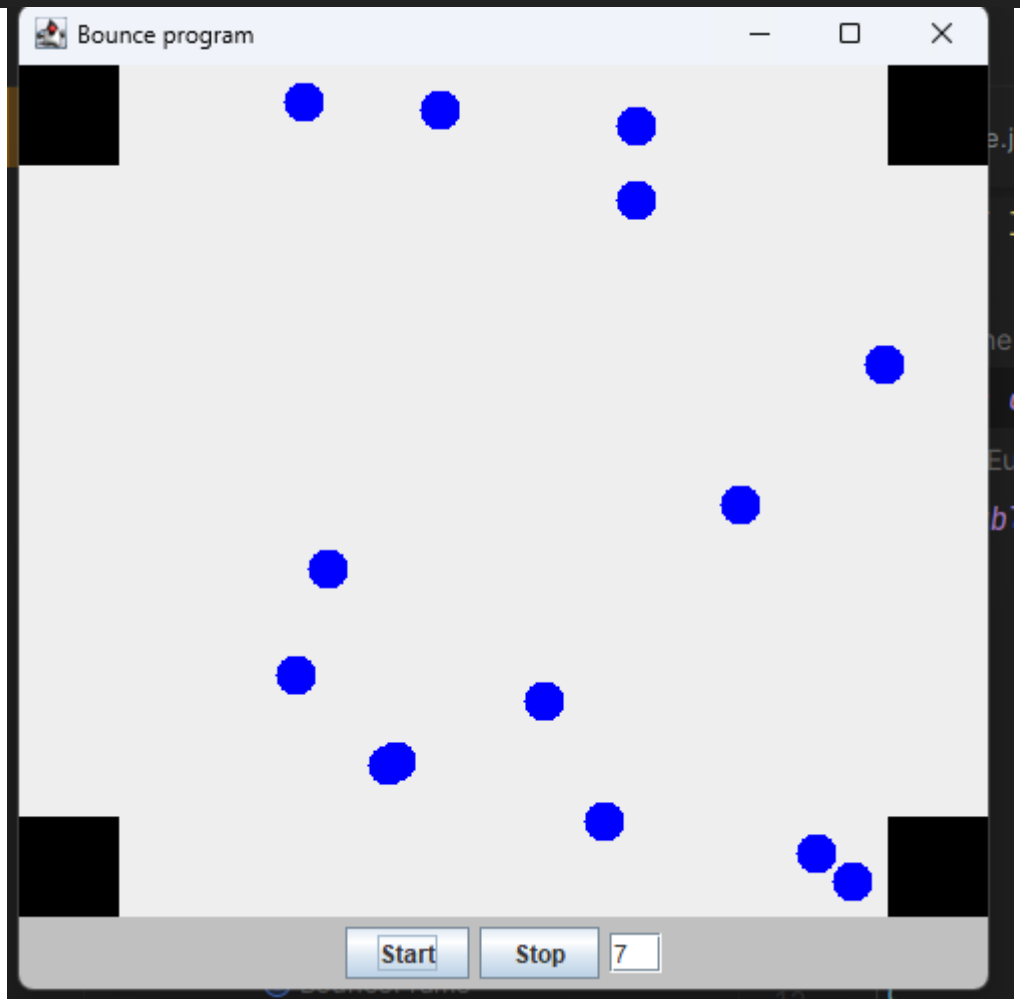
BallThread:

```
import java.util.Arrays;

public class BallThread extends Thread {
    private final Ball ball;
    private final TextFieldSetter ballsInAHoleUpdater;
    private final Hole[] holes;

    public BallThread(Ball ball, TextFieldSetter
ballsInAHoleUpdater, Hole[] holes) {
        this.ball = ball;
        this.ballsInAHoleUpdater = ballsInAHoleUpdater;
        this.holes = holes;
    }

    @Override
    public void run() {
        try {
            while (true) {
                ball.move();
                if (Arrays.stream(holes).anyMatch((hole ->
hole.isPointInside(ball.getXCentered(), ball.getYCentered())))) {
                    this.ballsInAHoleUpdater.updateBallsInAHole();
                    break;
                }
                System.out.println("Thread name = " +
Thread.currentThread().getName());
                Thread.sleep(5);
            }
        } catch (InterruptedException ignored) {

        }
    }
}
```

BallCanvas:

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

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

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

    public void addHole(Hole hole) {
        this.holes.add(hole);
    }
```

```
    public void addHoles(Hole[] holes) {
        Collections.addAll(this.holes, holes);
    }

    public void removeBall(Ball ball) {
        this.balls.remove(ball);
        this.repaint();
    }

    @Override
    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        Graphics2D g2 = (Graphics2D) g;

        for (int i = 0; i < holes.size(); i++) {
            Hole hole = this.holes.get(i);
            hole.draw(g2);
        }

        for (int i = 0; i < balls.size(); i++) {
            Ball ball = this.balls.get(i);
            ball.draw(g2);
        }
    }
}
```

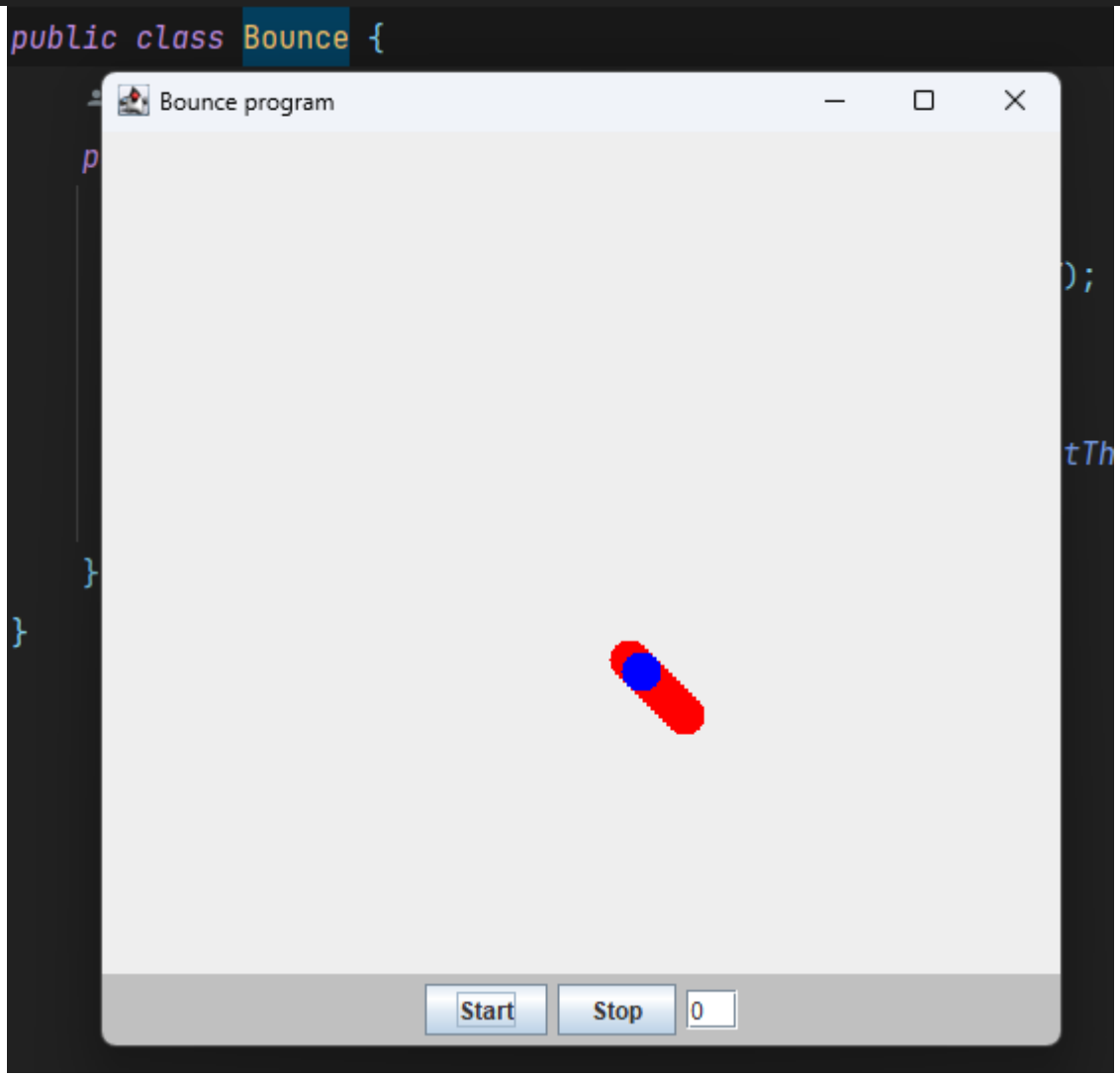


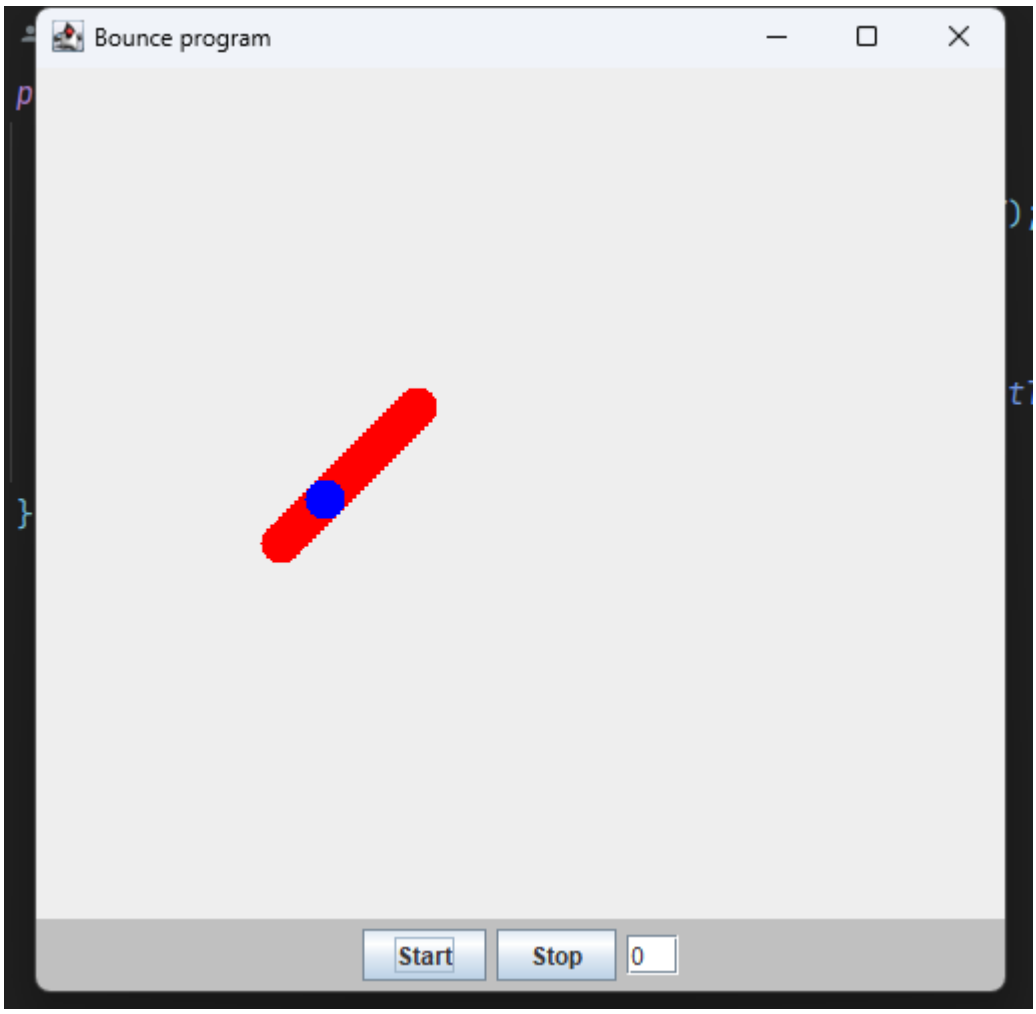
Дослідження пріоритету потоку:

BounceFrame:

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

public class BounceFrame extends JFrame {
    public static final int WIDTH = 500;
    public static final int HEIGHT = 500;
    private static final int HOLE_SIZE = 50;
    private final BallCanvas canvas;

    private int ballsInAHole = 0;


    public BounceFrame() {
        this.setSize(WIDTH, HEIGHT);
        this.setTitle("Bounce program");

        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");

        JTextField textField = new JTextField("0", 2);

        Hole[] holes = {};

        buttonStart.addActionListener(e -> {
            for (int i = 0; i < 1000; i++) {
                Ball ball = new Ball(canvas, BallPriority.HIGH);
                canvas.addBall(ball);

                BallThread thread = new BallThread(ball, () -> {
                    ballsInAHole++;

textField.setText(String.valueOf(ballsInAHole));
                    canvas.removeBall(ball);
                }, holes);
                thread.setPriority(5);
                thread.start();
            }

            Ball ball = new Ball(canvas, BallPriority.STANDARD);
            canvas.addBall(ball);

            BallThread thread = new BallThread(ball, () -> {
                ballsInAHole++;
                textField.setText(String.valueOf(ballsInAHole));
                canvas.removeBall(ball);
```

```
            }, holes);
            thread.setPriority(10);
            thread.start();
            System.out.println("Thread name = " +
thread.getName());
        });
        buttonStop.addActionListener(e -> System.exit(0));

        buttonPanel.add(buttonStart);
        buttonPanel.add(buttonStop);
        buttonPanel.add(textField);

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





Синхронізація потоків, що виводять різні символи для роботи почергово:

Main:

```
public class Main {
    public static void main(String[] args) {
        CharPrinter charPrinter = new CharPrinter();
        CharThread verticalLineThread = new CharThread('|',
charPrinter);
        CharThread dashThread = new CharThread('-', charPrinter);

        verticalLineThread.start();
        dashThread.start();
    }
}
```

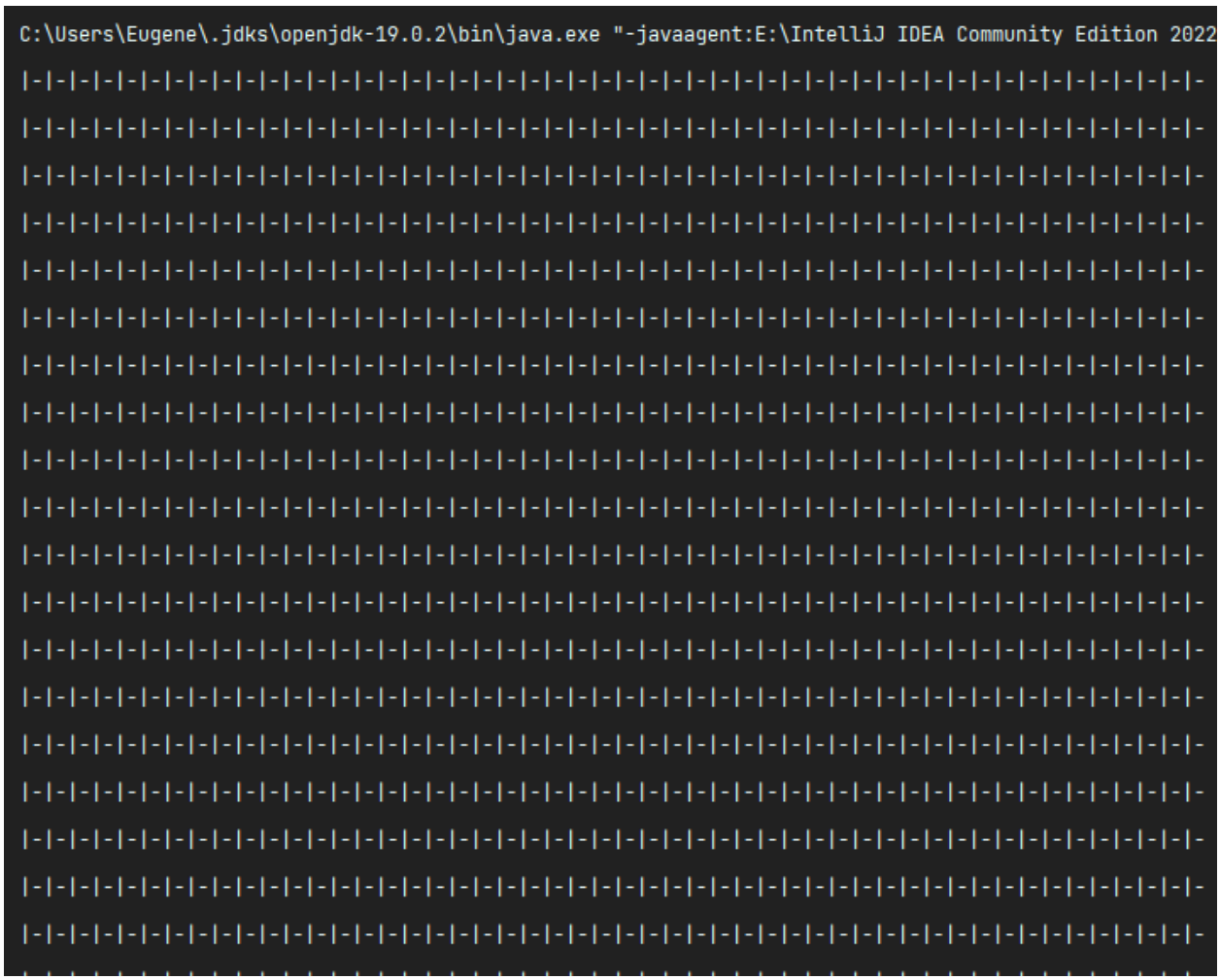
CharThread:

```
public class CharThread extends Thread {

    private final char threadChar;
    private final CharPrinter charPrinter;

    public CharThread(char threadChar, CharPrinter charPrinter) {
        this.threadChar = threadChar;
```

```
        this.charPrinter = charPrinter;
    }

    @Override
    public void run() {
//        for (int i = 0; i < 100000; i++) {
//            this.charPrinter.print(threadChar);
//        }
        synchronized (this.charPrinter) {
            for (int i = 0; i < 100000; i++) {
                while (this.charPrinter.getLastCharPrinted() ==
threadChar) {
                    try {
                        this.charPrinter.wait();
                    } catch (InterruptedException ignored) {

                    }
                }
                this.charPrinter.print(threadChar);
                this.charPrinter.notify();
            }
        }
    }
}
```

CharPrinter:

```
public class CharPrinter {
    private char lastCharPrinted;
    private int printedChars = 0;
    public void print(char c) {
        System.out.print(c);
        lastCharPrinted = c;
        printedChars++;
        if (printedChars == 100) {
            System.out.print('\n');
            printedChars = 0;
        }
    }

    public char getLastCharPrinted() {
        return lastCharPrinted;
    }
}
```



Лічильник:

Main:

```
public class Main {
    public static void main(String[] args) {
        Counter counter = new Counter();

        IncrementThread incrementThread = new
IncrementThread(counter);
        DecrementThread decrementThread = new
DecrementThread(counter);

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

        try {
            incrementThread.join();
            decrementThread.join();
            System.out.println(counter.getC());
        } catch (InterruptedException ignored) {

        }
    }
}
```

IncrementThread:

```java
public class IncrementThread extends Thread {
    private final Counter counter;

    public IncrementThread(Counter counter) {
        this.counter = counter;
    }

    @Override
    public void run() {
        for (int i = 0; i < 100_000_000; i++) {
            this.counter.increment();
        }
    }
}
```

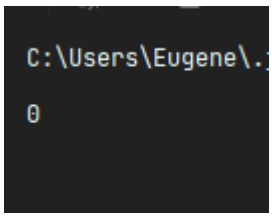
DecrementThread:

```java
public class DecrementThread extends Thread {
    private final Counter counter;

    public DecrementThread(Counter counter) {
        this.counter = counter;
    }

    @Override
    public void run() {
        for (int i = 0; i < 100_000_000; i++) {
            this.counter.decrement();
        }
    }
}
```

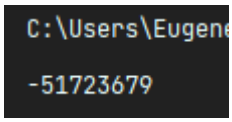
Counter:

```java
public class Counter {

    private int c = 0;

    public void increment() {
        synchronized(this) {
            c++;
        }
    }

    public synchronized void decrement() {
        c--;
    }

    public int getC() {
        return c;
    }
}
```

Синхронізоване виконання:



Несинхронізоване виконання:



Висновки

Виконуючи роботу комп’ютерного практикуму, було реалізовано алгоритми для дослідження потоків, їх керуючих та інформаційних методів, пріоритету, синхронізації.