

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ ІНФОРМАТИКИ ТА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

Кафедра інформатики та програмної інженерії

**Звіт**

З комп'ютерного практикуму № 1 з дисципліни  
«Технології паралельних обчислень»

Тема: «Розробка потоків та дослідження пріоритету запуску потоків»

**Виконав(ла)**

*ІП-15 Мешков Андрій*

\_\_\_\_\_  
(шифр, прізвище, ім'я, по батькові)

**Перевірів**

*Дифучина О. Ю.*

\_\_\_\_\_  
(шифр, прізвище, ім'я, по батькові)

Київ 2024

## 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. 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: 8 синхронізований метод, синхронізований блок, блокування об'єкта. Порівняйте способи синхронізації. 15 балів.

## ХІД РОБОТИ

### Завдання 1.

При збільшенні кількості кульок, можна спостерігати наступне:

1. **Підвищене навантаження на процесор:** Кожен потік вимагає ресурсів процесора для виконання своїх завдань. Збільшення кількості потоків призводить до більшого навантаження на процесор.

2. **Час переключення контексту:** Операційна система витрачає час на переключення між потоками, що може знизити загальну продуктивність при великій кількості потоків.

3. **Пам'ять:** Кожен потік потребує пам'яті для свого стека. При великій кількості потоків це може призвести до нестачі пам'яті.

4. **Конкуренція за ресурси:** Потоки можуть конкурувати за доступ до спільних ресурсів, що може викликати блокування або зупинку програми.

Переваги потокової архітектури програм

1. **Паралельне виконання:** Потоки дозволяють програмі виконувати декілька завдань одночасно, що підвищує ефективність використання ресурсів багатоядерних процесорів.

2. **Відгук системи:** Потоки дозволяють підтримувати високу відгук системи, оскільки один потік може обробляти події користувача, поки інші виконують тривалі обчислення.

3. **Складні обчислення:** Паралельні потоки можуть використовуватися для прискорення складних обчислень шляхом розподілу задачі на підзадачі, які виконуються одночасно.

4. **Модульність:** Розділення програм на потоки може зробити код більш модульним і легшим для підтримки.

5. **Асинхронність:** Потоки дозволяють виконувати асинхронні операції, наприклад, обробку вводу/виводу, не блокуючи основний потік.

Лістинг коду:

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());
    }
}
```

Ball.java

```
package task1;

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

public 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());
        }
    }
}
```

```

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 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);
        }
    }
}

```

```

    }
}

```

## 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 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) {

                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);
    }
}

```

## BounceThread.java

```

package task1;

public class BallThread extends Thread {
    private Ball b;

    public BallThread(Ball ball){
        b = ball;
    }
    @Override
    public void run(){
        try{
            for(int i=1; i<10000; i++){
                b.move();
                System.out.println("Thread name = "
                    + Thread.currentThread().getName());
                Thread.sleep(5);
            }
        } catch (InterruptedException ex){
        }
    }
}

```



Результат:

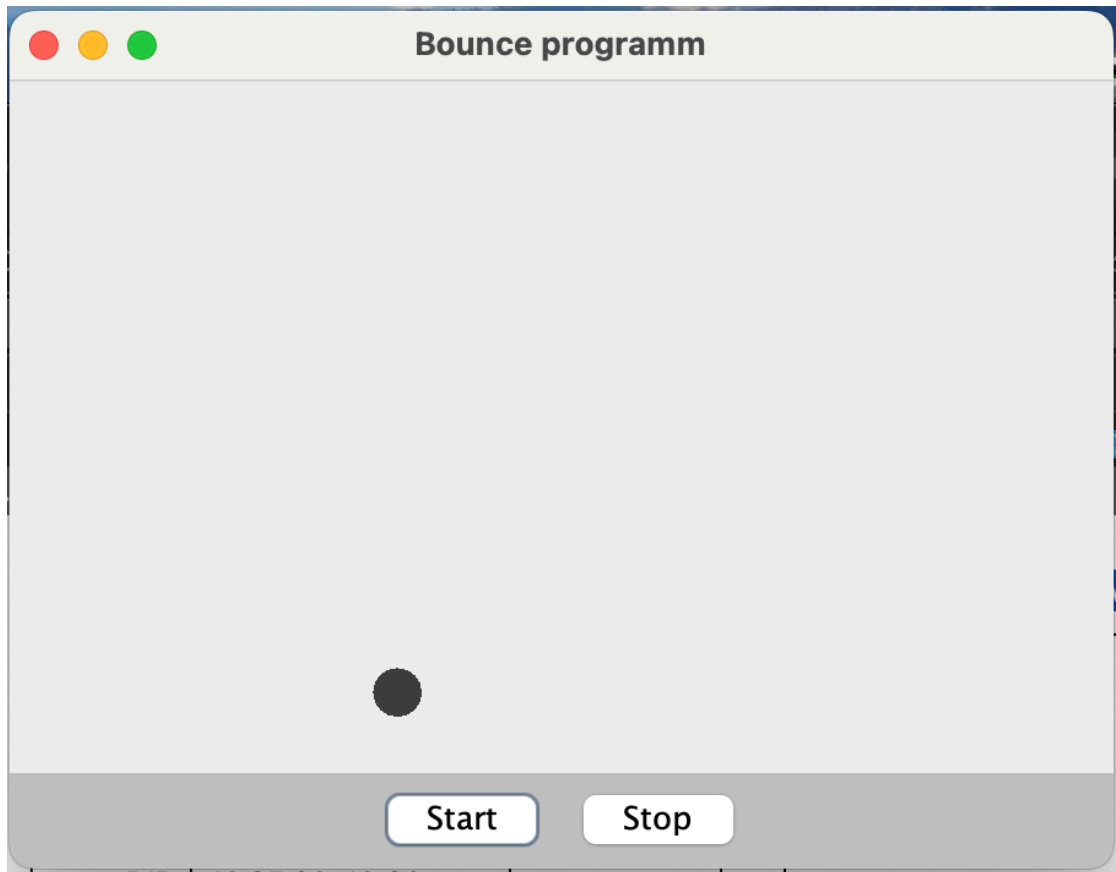


Рисунок 1 – Результат запуску програми 1

## Завдання 2.

Модифікуємо програму, додамо лузи.

Лістинг коду:

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());
    }
}
```

Ball.java

```
package task2;

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

public 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());
        }
    }
}
```

```

    }
}

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();
}

public boolean isInHole(Hole hole) {
    return Math.abs(hole.getX() - x) < XSIZE / 2 && Math.abs(hole.getY() - y) < YSIZE /
2;
}
}

```

## BallCanvas.java

```

package task2;

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

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

```

```

public int downtroddenBallsCount = 0;

public BallCanvas(ArrayList<Hole> holes, JLabel label) {
    this.holes = holes;
    this.label = label;
}

public void add(Ball b) {
    this.balls.add(b);
    BallThread thread = new BallThread(b, holes, this);
    thread.start();
    System.out.println("Thread name = " +
        thread.getName());
}

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

public synchronized void removeBall(Ball b) {
    downtroddenBallsCount += 1;
    label.setText(String.valueOf(downtroddenBallsCount));
    balls.remove(b);
}
}

```

## BounceFrame.java

```

package task2;

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

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

```

```

public BounceFrame() {

    this.setSize(WIDTH, HEIGHT);
    this.setTitle("Bounce programm");

    JLabel downtroddenBallsCountLabel = new JLabel();
    downtroddenBallsCountLabel.setText(String.valueOf(0));

    ArrayList<Hole> holes = new ArrayList<>();
    int maxX = BounceFrame.WIDTH - Hole.XSIZE;
    int maxY = BounceFrame.HEIGHT - Hole.YSIZE - 67;
    holes.add(new Hole(this, 0, 0));
    holes.add(new Hole(this, 0, maxY));
    holes.add(new Hole(this, maxX, 0));
    holes.add(new Hole(this, maxX, maxY));
    holes.add(new Hole(this, maxX / 2, 0));
    holes.add(new Hole(this, maxX / 2, maxY));
    this.canvas = new BallCanvas(holes, downtroddenBallsCountLabel);

    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) {
            Ball b = new Ball(canvas);
            canvas.add(b);
        }
    });

    buttonStop.addActionListener(new ActionListener() {

        @Override
        public void actionPerformed(ActionEvent e) {

            System.exit(0);
        }
    });

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

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

```

```
}
```

## BounceThread.java

```
package task2;

import java.util.ArrayList;

public class BallThread extends Thread {
    private Ball b;
    private ArrayList<Hole> holes;
    private BallCanvas canvas;

    public BallThread(Ball ball, ArrayList<Hole> holes, BallCanvas canvas){
        b = ball;
        this.holes = holes;
        this.canvas = canvas;
    }

    @Override
    public void run(){
        try{
            while(!isBallInHoles()) {
                b.move();
                System.out.println("Thread name = "
                    + Thread.currentThread().getName());
                Thread.sleep(5);
            }
        } catch (InterruptedException ex){
        }
        canvas.removeBall(b);
        canvas.updateUI();
    }

    private boolean isBallInHoles() {
        for(int i=0; i<holes.size();i++) {
            Hole hole = holes.get(i);
            if (b.isInHole(hole)) {
                return true;
            }
        }
        return false;
    }
}
```

## Hole.java

```
package task2;

import java.awt.*;
```

```
import java.awt.geom.Ellipse2D;

public class Hole {
    public static final int XSIZE = 20;
    public static final int YSIZE = 20;

    private Component canvas;
    private int x;
    private int y;

    public Hole(Component c, int x, int y) {
        this.canvas = c;
        this.x = x;
        this.y = y;
    }

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

    public int getX() {
        return x;
    }

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

Результат:

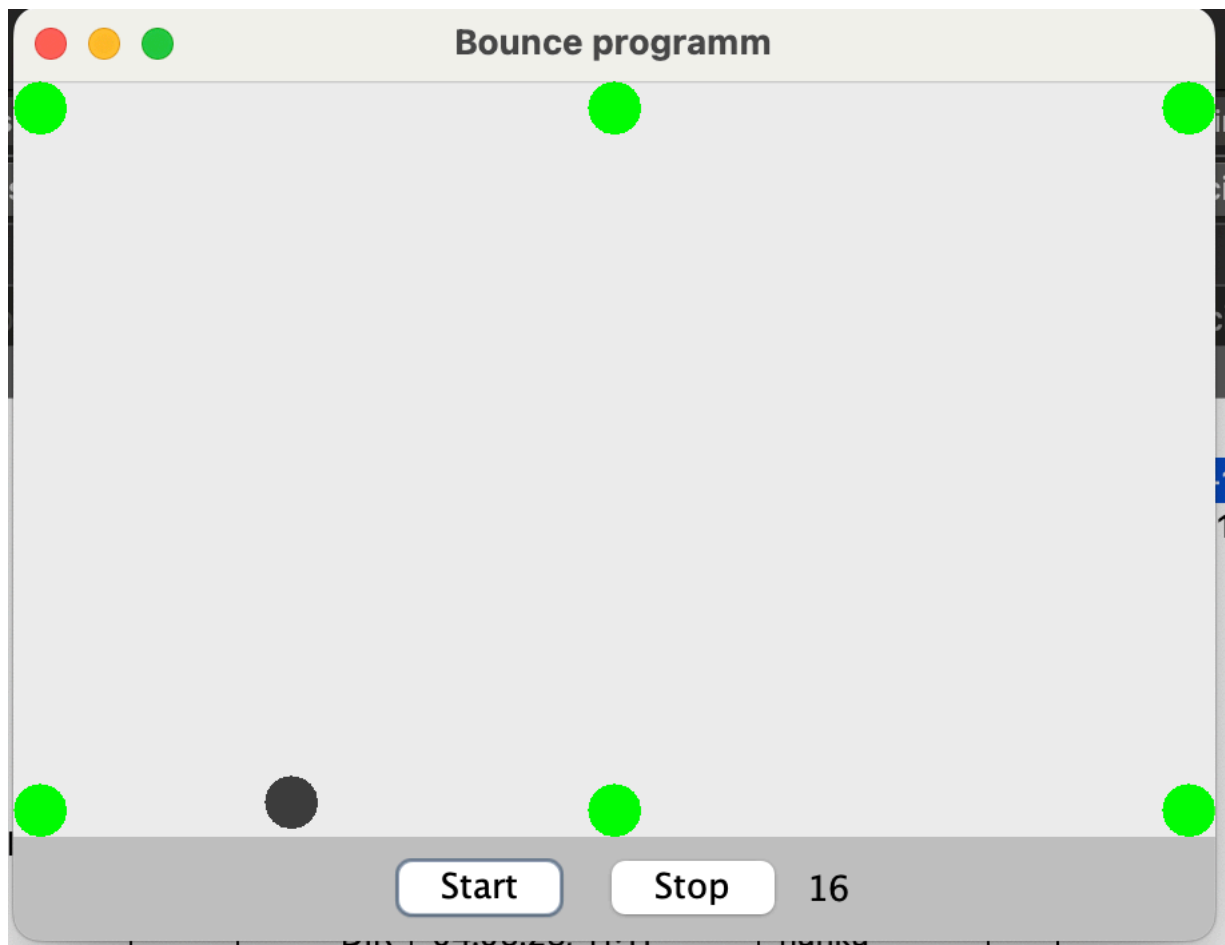


Рисунок 2 – Результат запуску програми 2



### Завдання 3.

Модифікуємо програму.

Було досліджено вплив пріоритету потоку на швидкість польоту куль. Було виявлено, що розподіл ресурсів на операційній системі MacOS є рівномірним, а отже кулька з більшим пріоритетом не летить швидше.

Лістинг коду:

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);
    }
}
```

Ball.java

```
package task3;
import java.awt.*;
import java.awt.geom.Ellipse2D;

public 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 Boolean isBlue;

    public Ball(Component c, Boolean isBlue){
        this.canvas = c;
        this.isBlue = isBlue;
        x = 0;
        y = 30;
    }
}
```

```

public void draw (Graphics2D g2){
    if (isBlue) {
        g2.setColor(Color.blue);
    } else {
        g2.setColor(Color.red);
    }
    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 task3;

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

public class BallCanvas extends JPanel {
    private 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);
        }
    }
}

```

## 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 BallCanvas canvas;
    public static final int WIDTH = 450;
    public static final int HEIGHT = 350;
    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 button1 = new JButton("1:2");
        JButton button2 = new JButton("1:500");
        JButton button3 = new JButton("1:1000");
        JButton buttonStop = new JButton("Stop");
        button1.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {

                for(int i=0; i<2; i++) {
                    Ball b = new Ball(canvas, true);
                    canvas.add(b);
                    BallThread thread = new BallThread(b);
                    thread.setPriority(Thread.MIN_PRIORITY);
                    thread.start();
                }
                Ball b = new Ball(canvas, false);
                canvas.add(b);
                BallThread thread = new BallThread(b);
                thread.setPriority(Thread.MAX_PRIORITY);
                thread.start();
            }
        });
    }
}

```

```

button2.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {

        for(int i=0; i<500; i++) {
            Ball b = new Ball(canvas, true);
            canvas.add(b);
            BallThread thread = new BallThread(b);
            thread.setPriority(Thread.MAX_PRIORITY);
            thread.start();
        }
        Ball b = new Ball(canvas, false);
        canvas.add(b);
        BallThread thread = new BallThread(b);
        thread.setPriority(Thread.MIN_PRIORITY);
        thread.start();
    }
});
button3.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {

        for(int i=0; i<1000; i++) {
            Ball b = new Ball(canvas, true);
            canvas.add(b);
            BallThread thread = new BallThread(b);
            thread.setPriority(Thread.MIN_PRIORITY);
            thread.start();
        }
        Ball b = new Ball(canvas, false);
        canvas.add(b);
        BallThread thread = new BallThread(b);
        thread.setPriority(Thread.MAX_PRIORITY);
        thread.start();
    }
});
buttonStop.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {

        System.exit(0);
    }
});

buttonPanel.add(button1);
buttonPanel.add(button2);
buttonPanel.add(button3);
buttonPanel.add(buttonStop);

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

```

## BounceThread.java

```
package task3;

public class BallThread extends Thread {
    private Ball b;

    public BallThread(Ball ball){
        b = ball;
    }
    @Override
    public void run(){
        try{
            for(int i=1; i<10000; i++){
                b.move();
                Thread.sleep(5);
            }
        } catch (InterruptedException ex){
        }
    }
}
```

Результат:

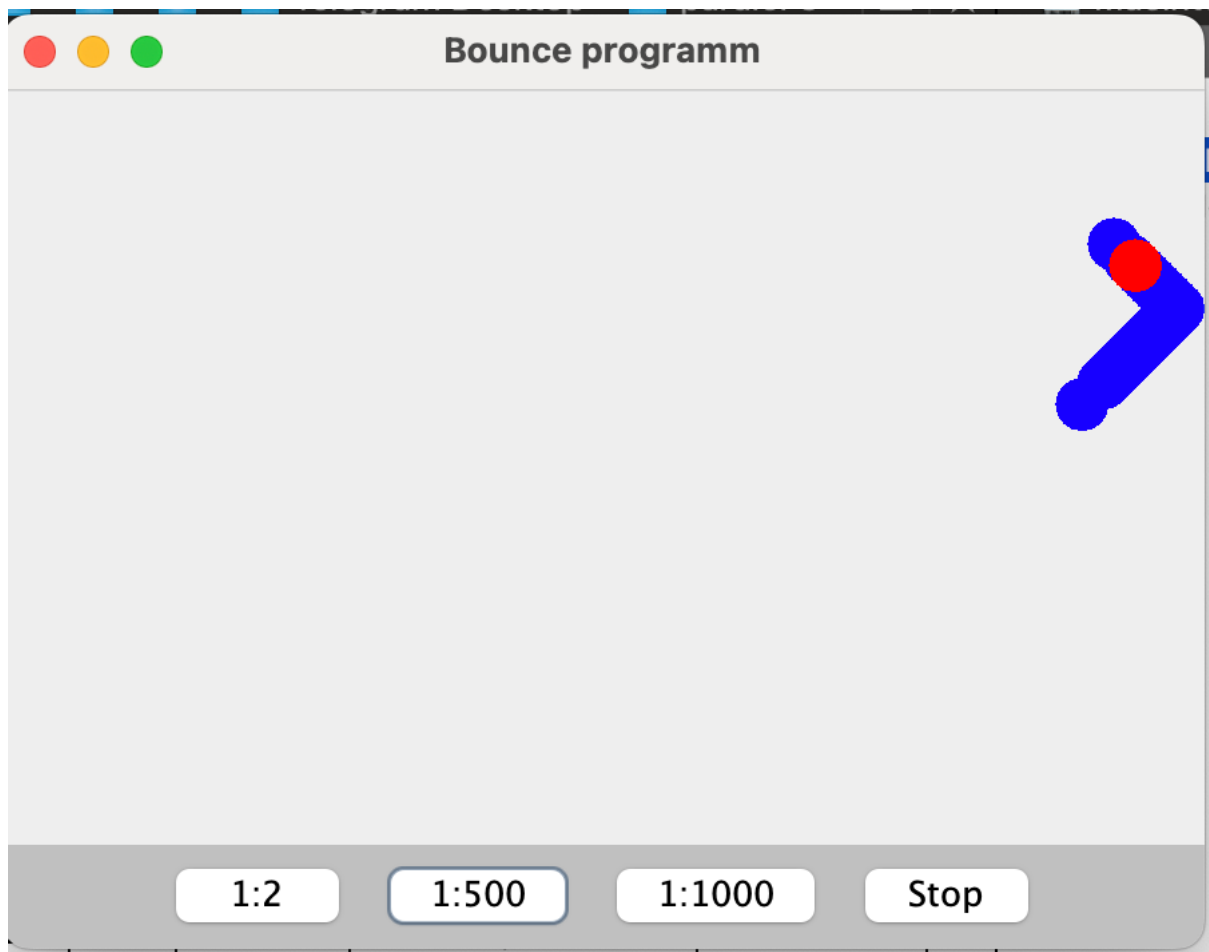


Рисунок 3 – Результат запуску програми 3

Завдання 4.

Модифікуємо програму.

Було розроблено програму, яка за рахунок методу `join()` змушує одну кульку чекати завершення руху іншої.

Лістинг коду:

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());
    }
}
```

Ball.java

```
package task4;

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

public 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 boolean isBlue;

    public Ball(Component c, boolean isBlue){
        this.canvas = c;
        this.isBlue = isBlue;
        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){
    if (isBlue) {
        g2.setColor(Color.blue);
    } else {
        g2.setColor(Color.red);
    }
    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();
}

public boolean isInHole(Hole hole) {
    return Math.abs(hole.getX() - x) < XSIZE / 2 && Math.abs(hole.getY() - y) < YSIZE /
2;
}
}

```

## BallCanvas.java

```

package task4;

import javax.swing.*.*;

```



```

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

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

    public int downtroddenBallsCount = 0;

    public BallCanvas(ArrayList<Hole> holes, JLabel label) {
        this.holes = holes;
        this.label = label;
    }

    public void test() {
        Ball blueBall = new Ball(this, true);
        Ball redBall = new Ball(this, false);
        this.balls.add(blueBall);
        this.balls.add(redBall);
        BallThread thread1 = new BallThread(blueBall, holes, this);
        BallThread thread2 = new BallThread(redBall, holes, this, thread1);
        thread1.start();
        thread2.start();
    }

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

    public synchronized void removeBall(Ball b) {
        downtroddenBallsCount += 1;
        label.setText(String.valueOf(downtroddenBallsCount));
        balls.remove(b);
    }
}

```

BounceFrame.java

```
package task4;
```

```

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

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

        this.setSize(WIDTH, HEIGHT);
        this.setTitle("Bounce programm");

        JLabel downtroddenBallsCountLabel = new JLabel();
        downtroddenBallsCountLabel.setText(String.valueOf(0));

        ArrayList<Hole> holes = new ArrayList<>();
        int maxX = BounceFrame.WIDTH - Hole.XSIZE;
        int maxY = BounceFrame.HEIGHT - Hole.YSIZE - 67;
        holes.add(new Hole(this, 0, 0));
        holes.add(new Hole(this, 0, maxY));
        holes.add(new Hole(this, maxX, 0));
        holes.add(new Hole(this, maxX, maxY));
        holes.add(new Hole(this, maxX / 2, 0));
        holes.add(new Hole(this, maxX / 2, maxY));
        this.canvas = new BallCanvas(holes, downtroddenBallsCountLabel);

        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("Test");
        JButton buttonStop = new JButton("Stop");

        buttonStart.addActionListener(new ActionListener() {

            @Override
            public void actionPerformed(ActionEvent e) {
                canvas.test();
            }
        });

        buttonStop.addActionListener(new ActionListener() {

            @Override
            public void actionPerformed(ActionEvent e) {

```

```

        System.exit(0);
    }
});

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

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

```

## BounceThread.java

```

package task4;

import java.util.ArrayList;

import static java.util.Objects.isNull;

public class BallThread extends Thread {
    private Ball b;
    private ArrayList<Hole> holes;
    private BallCanvas canvas;
    private Thread threadForWait;

    public BallThread(Ball ball, ArrayList<Hole> holes, BallCanvas canvas){
        b = ball;
        this.holes = holes;
        this.canvas = canvas;
        this.threadForWait = null;
    }

    public BallThread(Ball ball, ArrayList<Hole> holes, BallCanvas canvas, Thread
threadForWait){
        b = ball;
        this.holes = holes;
        this.canvas = canvas;
        this.threadForWait = threadForWait;
    }

    @Override
    public void run(){
        try{
            if (!isNull(threadForWait)) {
                threadForWait.join();
            }
            while(!isBallInHoles()) {
                b.move();
                System.out.println("Thread name = "
                    + Thread.currentThread().getName());
            }
        }
    }
}

```

```

        Thread.sleep(5);
    }
} catch (InterruptedException ex) {
}
canvas.removeBall(b);
canvas.updateUI();
}

private boolean isBallInHoles() {
    for (int i=0; i<holes.size(); i++) {
        Hole hole = holes.get(i);
        if (b.isInHole(hole)) {
            return true;
        }
    }
    return false;
}
}

```

## Hole.java

```

package task4;

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

public class Hole {
    public static final int XSIZE = 20;
    public static final int YSIZE = 20;

    private Component canvas;
    private int x;
    private int y;

    public Hole(Component c, int x, int y) {
        this.canvas = c;
        this.x = x;
        this.y = y;
    }

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

    public int getX() {
        return x;
    }

    public int getY() {
        return y;
    }
}

```

```
}  
}
```

Результат:

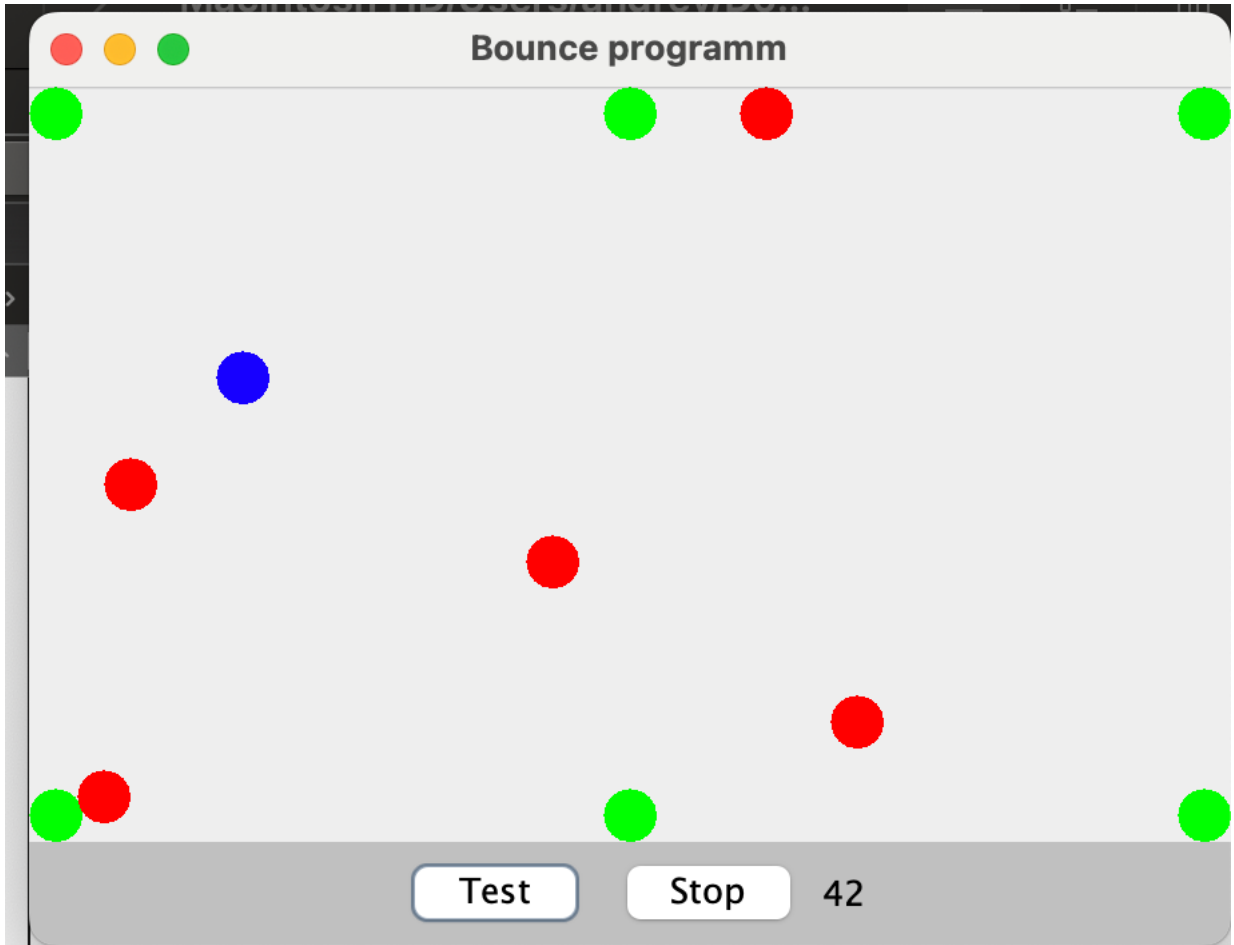


Рисунок 4 – Результат запуску програми 4

## Завдання 5.

Було реалізовано програму, що містить 2 потоки, що виводять на екран різні символи. Було модифіковано програму, щоб потоки синхронізувалися і символи виводилися по чергово.

Лістинг коду:

Task.java

```
package task5;

public class Task {
    public static void main(String[] args) {
        for (int i = 0; i < 100; i++) {
            System.out.print(i);
            Outputer outputer = new Outputer();
            OutputThread thread1 = new OutputThread('|', outputer);
            OutputThread thread2 = new OutputThread('-', outputer);
            thread1.start();
            thread2.start();
            try {
                thread1.join();
                thread2.join();
            } catch (InterruptedException e) {
                throw new RuntimeException(e);
            }
            System.out.println();
        }
        System.out.println();
        for (int i = 0; i < 100; i++) {
            System.out.print(i);
            Outputer outputer = new Outputer();
            OutputSyncThread thread1 = new OutputSyncThread('|', outputer);
            OutputSyncThread thread2 = new OutputSyncThread('-', outputer);
            thread1.start();
            thread2.start();
            try {
                thread1.join();
                thread2.join();
            } catch (InterruptedException e) {
                throw new RuntimeException(e);
            }
            System.out.println();
        }
    }
}
```

Outputer.java

```

package task5;

public class Outputer {

    public char lastChar = '-';

    public void setLastChar(char ch) {
        lastChar = ch;
    }

    public void output(char ch) {
        System.out.print(ch);
    }

}

```

### OutputThread.java

```

package task5;

public class OutputThread extends Thread {

    private char symbol;
    private Outputer outputer;

    public OutputThread(char symbol, Outputer outputer) {
        this.symbol = symbol;
        this.outputer = outputer;
    }

    @Override
    public void run() {
        for (int i = 0; i < 100; i++) {
            outputer.output(symbol);
        }
    }

}

```

### OutputSyncThread.java

```

package task5;

public class OutputSyncThread extends Thread {

    private char symbol;
    private Outputer outputer;

    public OutputSyncThread(char symbol, Outputer outputer) {
        this.symbol = symbol;
    }
}

```



```
        this.outputer = outputer;
    }

    @Override
    public void run() {
        for (int i = 0; i < 100; i++) {
            synchronized (outputer) {
                while (outputer.lastChar == symbol) {
                    try {
                        outputer.wait();
                    } catch (InterruptedException e) {
                    }
                }
                outputer.output(symbol);
                outputer.setLastChar(symbol);
                outputer.notifyAll();
            }
        }
    }
}
```

Результат:

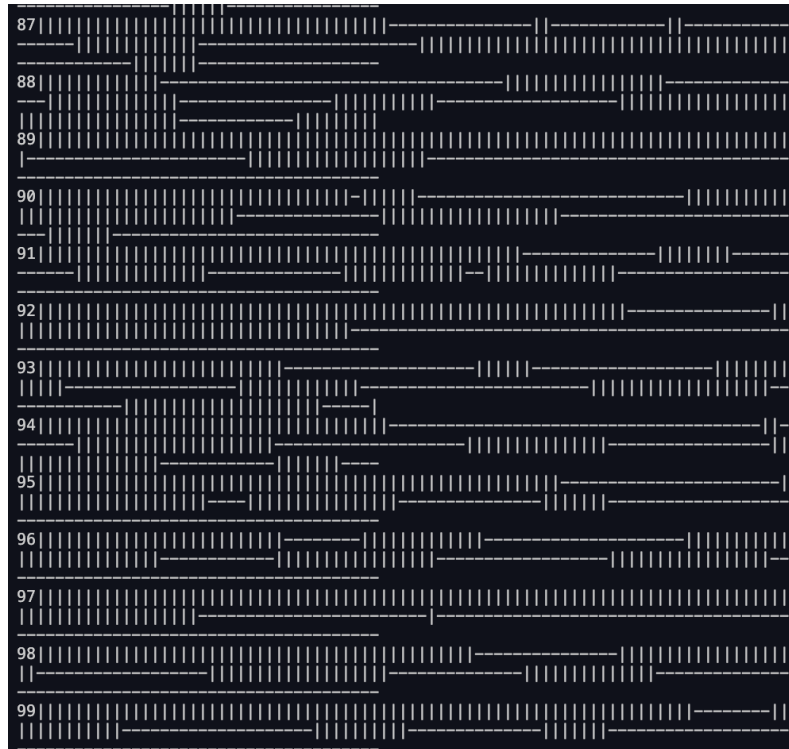


Рисунок 5 – Результат запуску програми 5 без синхронізації

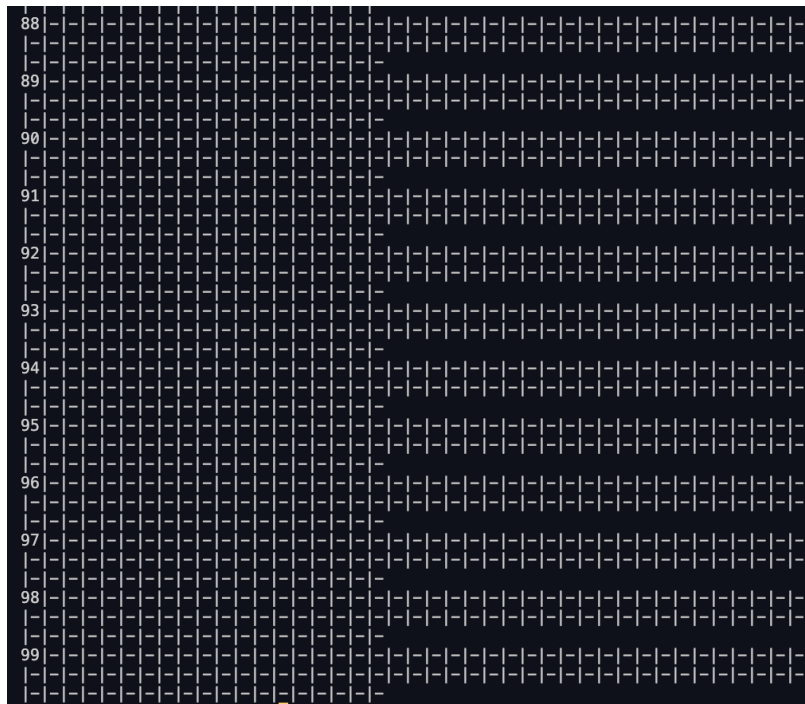


Рисунок 6 – Результат запуску програми 5 з синхронізацією

## Завдання 6.

Було створено програму, що містить 2 потоки, один потік збільшує число, інший зменшує. Було модифіковано програму для синхронізації потоків шляхом синхронізованого методу, синхронізованого блоку і блокованого об'єкту.

**Результат без синхронізації:**

- Значення лічильника буде випадковим і непередбачуваним.
- Це відбувається через те, що операції інкременту та декременту не атомарні. Інструкції можуть переплітатися, що призводить до неправильних результатів.

**Порівняння способів синхронізації**

Спосіб синхронізації	Переваги	Недоліки
Синхронізований метод	Простота реалізації, коректність результатів	Блокує весь метод, зниження продуктивності
Синхронізований блок	Ефективніше використання блокування, захист критичних секцій коду	Більш складна реалізація
Блокування об'єкта	Гнучкість, додаткові можливості (tryLock, Condition)	Складна реалізація, потребує ручного управління

Лістинг коду:

Task.java

```
package task6;

public class Task {
    public static void main(String[] args) {
        Counter counter = new Counter();
        CounterSyncBlock counter = new CounterSyncBlock();
        CounterSyncMethod counter = new CounterSyncMethod();
        CounterWithLock counter = new CounterWithLock();

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

        incrementThread.start();
        decrementThread.start();
    }
}
```

```

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

        System.out.println("Final counter value without synchronization: " +
            counter.getCount());
        System.out.println("Final counter value with synchronized method: " +
            counter.getCount());
        System.out.println("Final counter value with synchronized block: " +
            counter.getCount());
        System.out.println("Final counter value with blocked object: " +
            counter.getCount());
    }
}

```

### Counter.java

```

package task6;

public class Counter {
    private int count = 0;

    public void increment() {
        count++;
    }

    public void decrement() {
        count--;
    }

    public int getCount() {
        return count;
    }
}

```

### IncrementThread.java

```

package task6;

public class IncrementThread extends Thread {
    private Counter counter;
    public IncrementThread(Counter counter) {
        this.counter = counter;
    }

    private CounterSyncBlock counter;
    public IncrementThread(CounterSyncBlock counter) {

```

```

        this.counter = counter;
    }

    private CounterSyncMethod counter;
    public IncrementThread(CounterSyncMethod counter) {
        this.counter = counter;
    }

    private CounterWithLock counter;
    public IncrementThread(CounterWithLock counter) {
        this.counter = counter;
    }

    @Override
    public void run() {
        for (int i = 0; i < 100000; i++) {
            counter.increment();
        }
    }
}

```

### DecrementThread.java

```

package task6;

public class DecrementThread extends Thread {
    private Counter counter;
    public DecrementThread(Counter counter) {
        this.counter = counter;
    }

    private CounterSyncBlock counter;
    public DecrementThread(CounterSyncBlock counter) {
        this.counter = counter;
    }

    private CounterSyncMethod counter;
    public DecrementThread(CounterSyncMethod counter) {
        this.counter = counter;
    }

    private CounterWithLock counter;
    public DecrementThread(CounterWithLock counter) {
        this.counter = counter;
    }

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

```

```
}
```

### CounterSyncBlock.java

```
package task6;

public class CounterSyncBlock {
    private int count = 0;
    private final Object lock = new Object();

    public void increment() {
        synchronized (lock) {
            count++;
        }
    }

    public void decrement() {
        synchronized (lock) {
            count--;
        }
    }

    public int getCount() {
        return count;
    }
}
```

### CounterSyncMethod.java

```
package task6;

public class CounterSyncMethod {
    private int count = 0;

    public synchronized void increment() {
        count++;
    }

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

    public int getCount() {
        return count;
    }
}
```

## CounterWithLock.java

```
package task6;

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

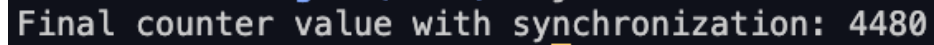
public class CounterWithLock {
    private int count = 0;
    private final Lock lock = new ReentrantLock();

    public void increment() {
        lock.lock();
        try {
            count++;
        } finally {
            lock.unlock();
        }
    }

    public void decrement() {
        lock.lock();
        try {
            count--;
        } finally {
            lock.unlock();
        }
    }

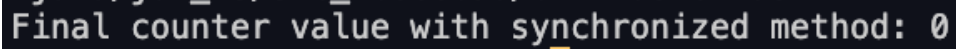
    public int getCount() {
        return count;
    }
}
```

Результат:



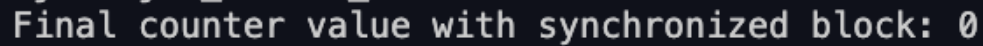
```
Final counter value with synchronization: 4480
```

Рисунок 7 – Результат запуску програми 6 без синхронізації



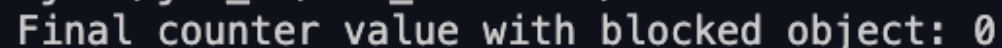
```
Final counter value with synchronized method: 0
```

Рисунок 8 – Результат запуску програми 6 з синхронізованим методом



```
Final counter value with synchronized block: 0
```

Рисунок 9 – Результат запуску програми 6 з синхронізованим блоком



```
Final counter value with blocked object: 0
```

Рисунок 10 – Результат запуску програми 6 з блокуванням об'єкту



## ВИСНОВКИ

В результаті роботи над комп'ютерним практикумом було виконано 6 завдань на мові програмування Java.

Завдання 1: Було реалізовано програму, що імітує рух більярдних кульок. Рух кожної кульки здійснюється у окремому потоці

Завдання 2: Було модифіковано завдання 1 шляхом додавання луз. При потраплянні шара у лузу потік завершує свою роботу.

Завдання 3: Було досліджено вплив пріоритету потоку на швидкість польоту куль. Було виявлено, що розподіл ресурсів на операційній системі MacOS є рівномірним, а отже кулька з більшим пріоритетом не летить швидше.

Завдання 4: Було розроблено програму, яка за рахунок методу `join()` змушує одну кульку чекати завершення руху іншої.

Завдання 5: Було реалізовано програму, що містить 2 потоки, що виводять на екран різні символи. Було модифіковано програму, щоб потоки синхронізувалися і символи виводилися по чергово.

Завдання 6: Було створено програму, що містить 2 потоки, один потік збільшує число, інший зменшує. Було модифіковано програму для синхронізації потоків шляхом синхронізованого методу, синхронізованого блоку і блокованого об'єкту.