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

Кафедра інформатики та програмної інженерії

Звіт

Комп‘ютерного практикуму № 1 з дисципліни
«Технології паралельних та розподілених обчислень»

«Розробка потоків та дослідження пріоритету запуску потоків»

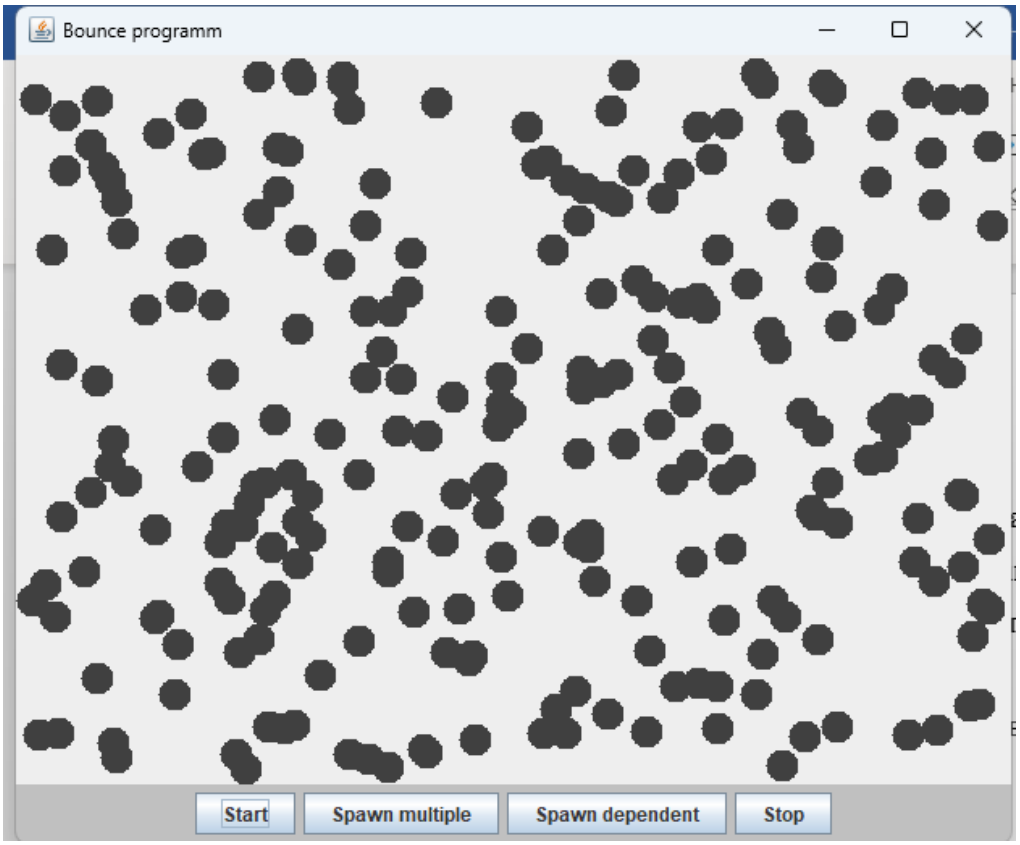
Виконав(ла) *ІП-01 Корнієнко В.С.* _____
(шифр, прізвище, ім'я, по батькові)

Перевірив(ла) *Стеценко І. В.* _____
(прізвище, ім'я, по батькові)

Київ 2023

Хід роботи

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.**



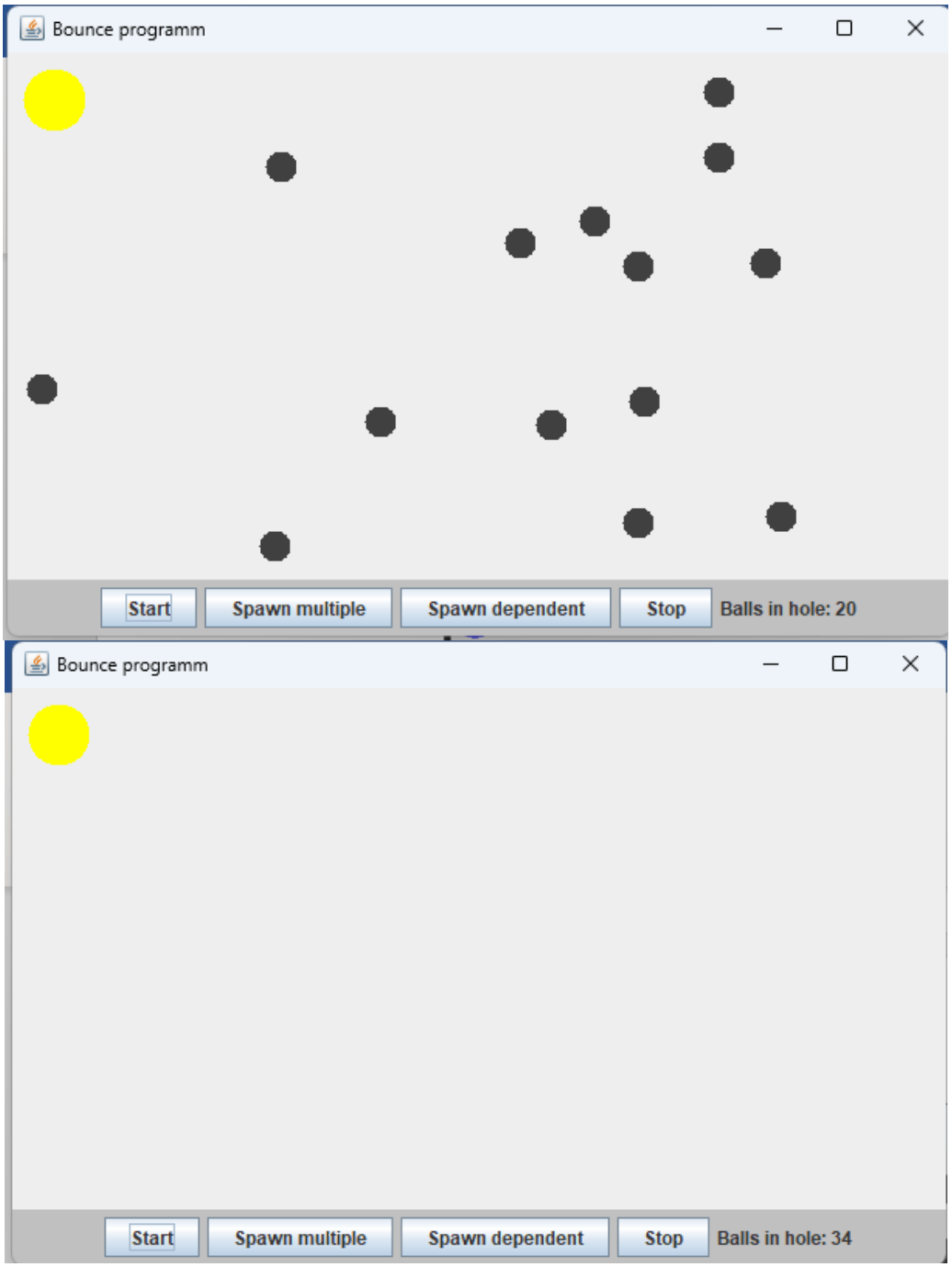
Під час спостереження було помічено, що при збільшенні кількості кульок їх рух стає менш плавним. Кожній кульці потрібно більше часу для розрахунку її наступного положення. Це пояснюється обмеженою кількістю потоків, яку може обробити процесор. Незважаючи на те, що є багато кульок, процесор має обробити їх усі разом, розподіляючи обчислювальні ресурси між ними. З більшою кількістю потоків на кожен з них припадає менше ресурсів, що впливає на продуктивність та призводить до затримок у роботі.

Переваги потокової архітектури:

1. **Паралельне виконання:** Завдяки потокам можна виконувати кілька завдань одночасно, що поліпшує продуктивність.

2. **Ефективне використання ресурсів:** Потокова архітектура дозволяє розподіляти завдання між багатоядерними процесорами, що забезпечує більш швидке виконання.
3. **Відмовостійкість та реагування на події:** Потокова архітектура дозволяє програмі реагувати на події в реальному часі і виконувати роботу одночасно з іншими завданнями. (Наприклад у випадку нашої програми, багатопотоковість дозволяє нам зчитувати користувацькі дії та рухати кульки одночасно)

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.**

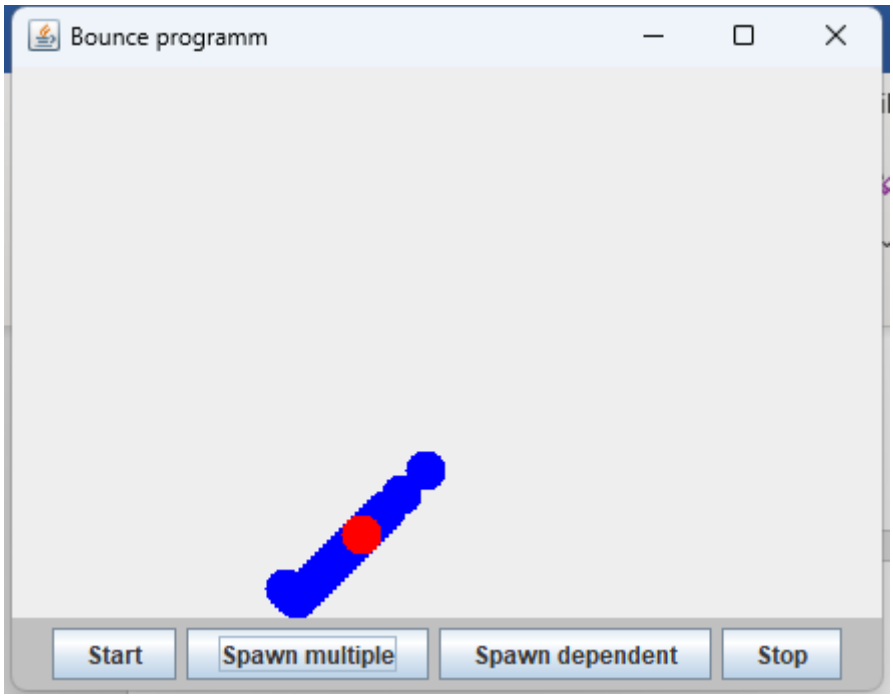


Бачимо що при потраплянні кульок в жовту лунку вони зникають, а значення поля «Balls in hole» збільшується

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.**

Для цього завдання була створена кнопка з назвою «Spawn multiple».

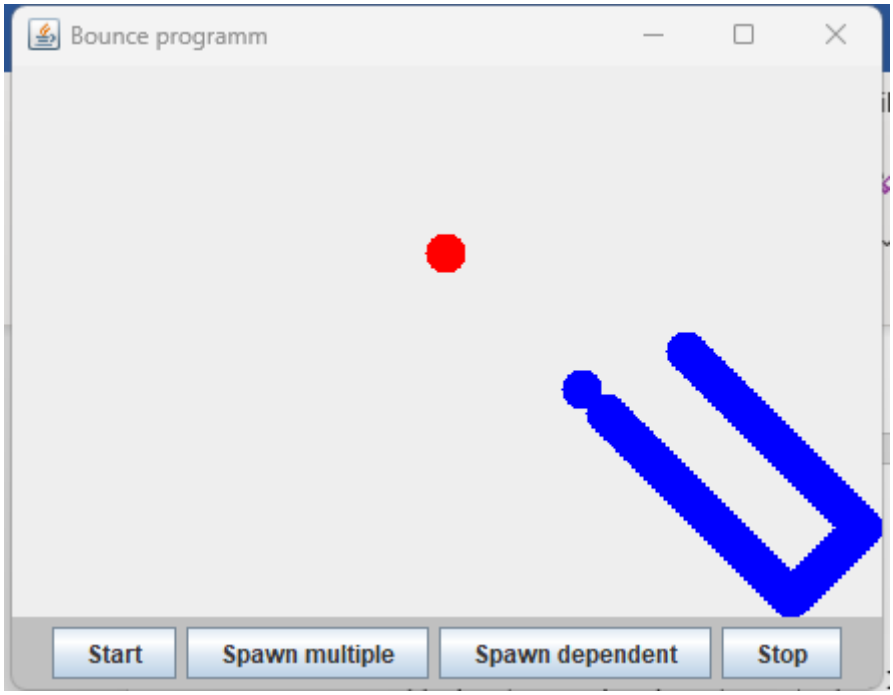
Задамо значення кількості кульок 3 низьким пріоритетом 200. Можемо спостерігати наступний результат:



За умови, коли кількість синіх кульок невелика (200), червона кулька майже завжди рухається по центру. Це можна пояснити тим, що наявний обчислювальний ресурс вистачає для всіх кульок і немає необхідності

надавати пріоритетність в їх розрахунках. Тобто пріоритети потоків майже не враховуються.

Тепер запустимо програму з більшим значенням кількості кульок з низьким пріоритетом. Нехай це буде 5000. Маємо наступний результат:

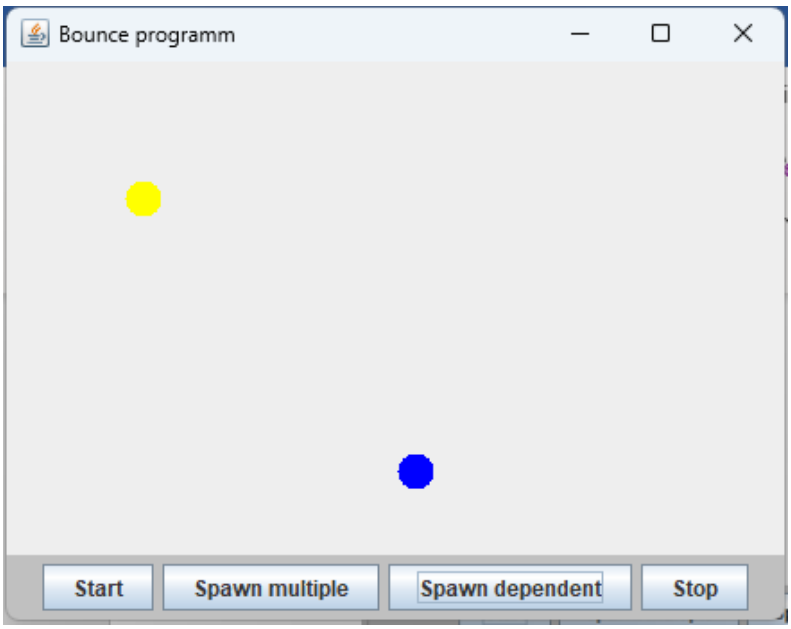


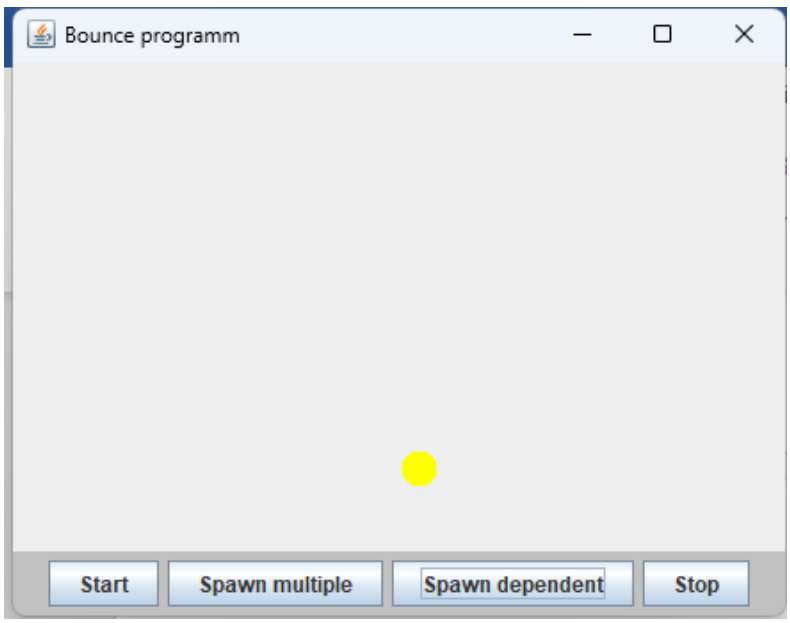
При збільшенні кількості синіх кульок (5000), спостерігається інше явище. Червона кулька випереджає значно випереджає усі інші, оскільки обчислювального ресурсу не вистачає для всіх. Операційна система, (зокрема Windows 11) повинна більш обережно розподіляти ресурси. Експеримент демонструє, що принцип справедливості майже не працює на даній системі.

В даному випадку можна зробити висновок, що в умовах обмеженого обчислювального ресурсу в Windows 11 потокам з вищим пріоритетом виділяється значно більше ресурсів.

4. Побудуйте ілюстрацію для методу join() класу Thread з використанням руху більярдних кульок різного кольору. Поясніть результат, який спостерігається. **10 балів.**

Щоб наочно показати роботу методу join() класу Thread створимо кнопку «Spawn dependent».
При запуску програми і натисненні кнопки можемо побачити наступний результат:

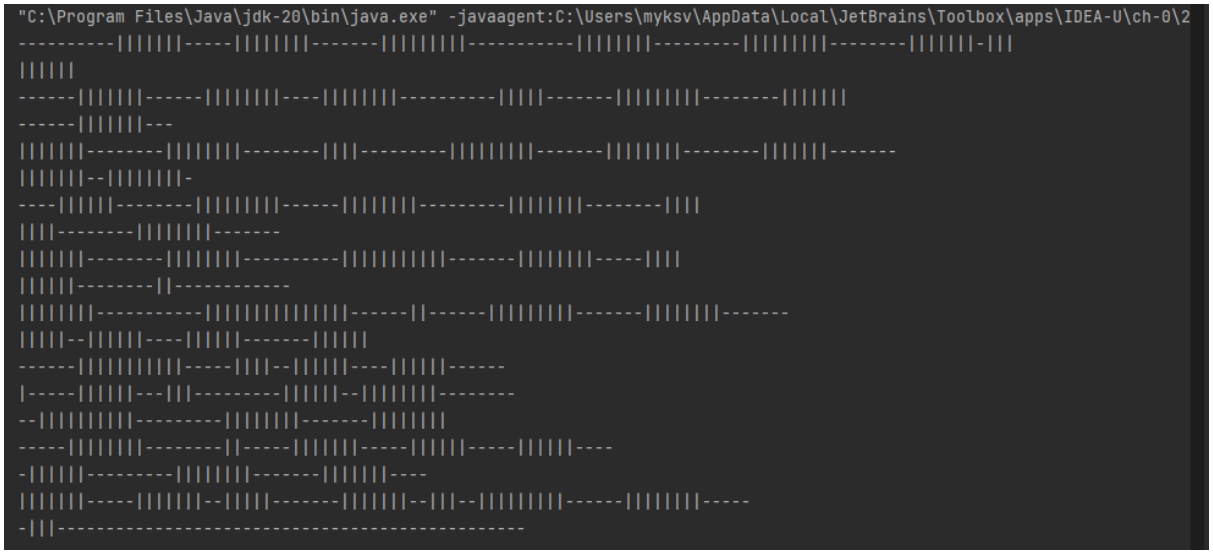




На початку, жовта кулька стоїть на місці, тоді як синя кулька рухається. Однак, коли потік, що відповідає за синю кульку, викликає операцію join() для жовтої кульки, остання починає рухатись.

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.**

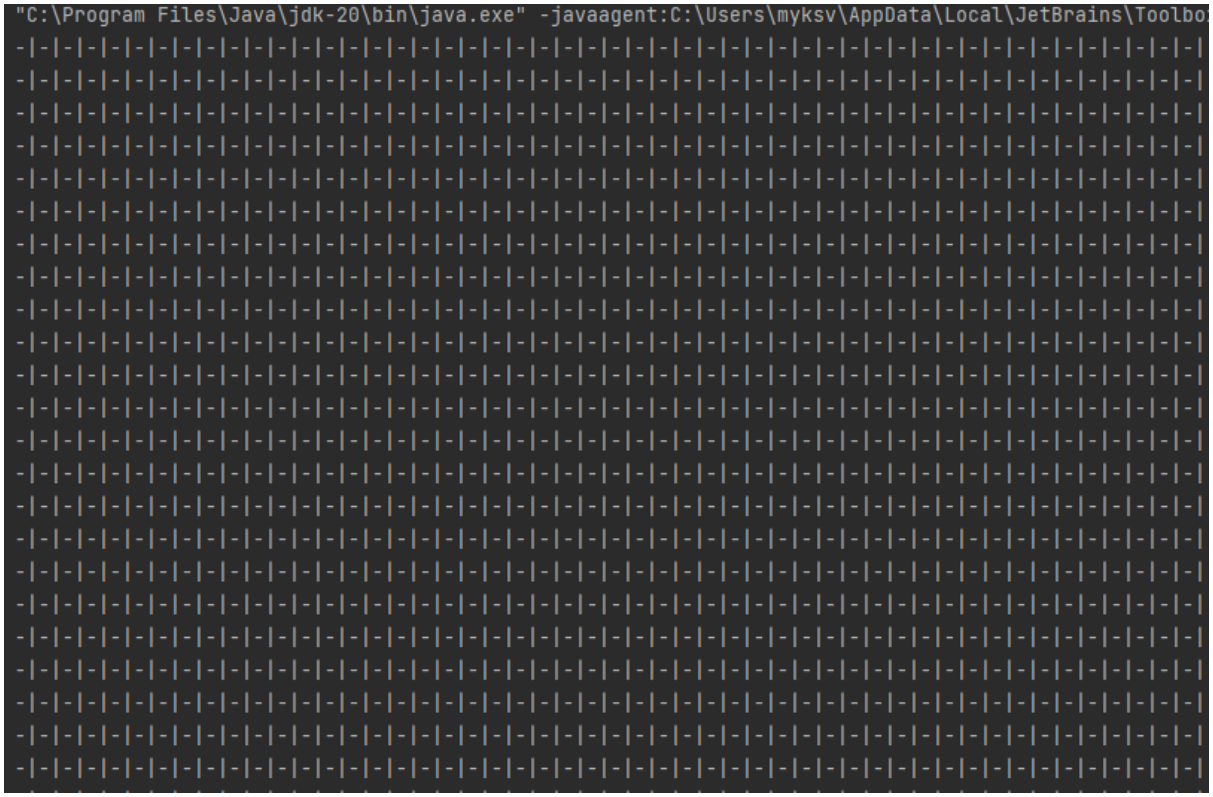
Створимо несинхронізований варіант програми. При запуску її на виконання бачимо наступні результати:



Символи '-' та '|' не будуть виводитися послідовно через паралельний характер виконання потоків. У програмі ми створюємо два потоки, один для виведення символу '-', інший - для виведення символу '|'. Коли обидва потоки запускаються паралельно, вони можуть взаємодіяти зі спільним ресурсом, яким є консоль, і конкретний порядок виводу символів може варіюватись залежно від швидкості виконання кожного потоку та планування операційної системи.

Наприклад, перший потік може встигнути вивести декілька символів '-', після чого другий потік може отримати доступ до консолі і вивести символ '|', і так далі. Цей процес взаємодії між потоками може призводити до змінюваного порядку виведення символів.

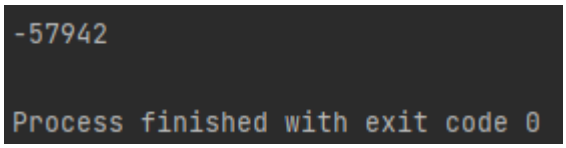
Тепер створимо синхронізований варіант програми та також запустимо її. Бачимо наступні результати:



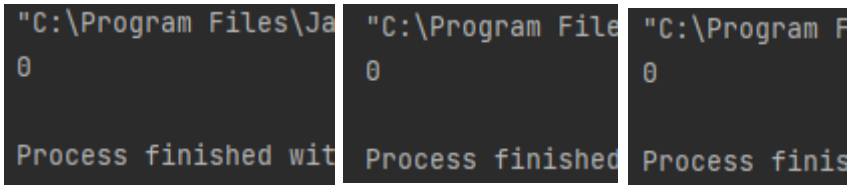
Бачимо що символи виводяться послідовно. За допомогою механізму wait-notify, доступ до консолі було синхронізовано, що дозволяє символам виводитися послідовно. Тепер перший потік зачекає, доки другий потік виведе символ, і другий потік виводить символ, після чого сигналізує першому потоку про успішне виконання операції.

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.**

Спочатку запустимо програму Counter без жодної синхронізації. Маємо наступний результат:



Бачимо результат, що не дорівнює 0. Опишемо причини цього далі, а зараз запустимо програму з синхронізованим методом, синхронізованим блоком та локером. Маємо такі результати:



Всі 3 синхронізації допомагають нам отримати правильний результат.

Як можна побачити, без синхронізації результат не стає нулем. Цю проблему називають помилкою узгодженості пам'яті (Memory consistency error). Оскільки операція increment або decrement не є атомарною, вона складається з трьох етапів: читання значення, зміни значення та запису. Тому потоки можуть одночасно прочитати значення, перший потік змінює актуальне значення, але не встигає його записати. Другий потік, прочитавши старе значення, змінює його, і в цей час перший потік записує оновлене значення. Але другий потік просто перезаписує його, тому виникає помилка.

Розглянемо використані методи синхронізації потоків:

1. Синхронізований метод здійснюється за допомогою ключового слова "synchronized" у сигнатурі методу. Коли метод викликається, монітор захоплює об'єкт, до якого він належить, що призводить до того, що інший потік починає очікувати, доки цей об'єкт не буде звільнений.

2. Синхронізований блок здійснює синхронізацію даних за допомогою структури synchronized(object){}. Тепер монітор захоплює не поточний об’єкт, а об’єкт, який передається в дужки «synchronized».

3. Блокування об'єкта використовує для синхронізації Локери, які є класами, що реалізують інтерфейс Lock. Локери надають додаткові можливості, такі як перевірка стану поточного об'єкта (чи він заблокований), спроба блокування, повторне блокування певну кількість разів та отримання відповідної інформації.

Висновок.

У цій лабораторній роботі були досліджені основні операції з управління потоками. Ми провели серію експериментів, щоб вивчити вплив кількості потоків на роботу програми і як пріоритетність впливає на розподіл обчислювальних ресурсів. Ми також розглянули метод синхронізації і пояснили їх відмінності.

Лістинг коду програми:

Завдання 1 – 4

Ball.java

```
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 Color color;

    public int getX() {
        return x;
    }

    public int getY() {
        return y;
    }

    public int getXsize(){
        return XSIZE;
    }

    public int getYsize(){
        return YSIZE;
    }

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

        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 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(this.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

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

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

    public void setHole(Hole hole) {
        this.hole = hole;
    }

    public void add(Ball b){
```

```
        this.balls.add(b);
    }
    public void remove(Ball b){
        this.balls.remove(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);
        }
        hole.draw(g2);
    }
}
```

BallThread.java

```
public class BallThread extends Thread {
    private Ball ball;
    private BallCanvas canvas;
    private BounceFrame bounceFrame;
    private int framesCount = 10_000;

    public BallThread(Ball ball, BallCanvas canvas,
BounceFrame bounceFrame) {
        this.ball = ball;
        this.canvas = canvas;
        this.bounceFrame = bounceFrame;
    }

    public BallThread(Ball ball, BallCanvas canvas,
BounceFrame bounceFrame, int framesCount) {
        this.ball = ball;
        this.canvas = canvas;
        this.bounceFrame = bounceFrame;
        this.framesCount = framesCount;
    }


    @Override
    public void run() {
        try {
            for (int i = 1; i < framesCount; i++) {
                ball.move();
                System.out.println("Thread name = "
                        + Thread.currentThread().getName());
                Thread.sleep(5);
```

```
                if (bounceFrame.getHole().BallInHole(ball)) {
                    canvas.remove(ball);
                    canvas.repaint();
                    bounceFrame.incBallsInHole();
                    return;
                }
            }
        } catch (InterruptedException ex) {
        }
    }
}
```

Bounce.java

```
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

```
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());

    }
}
```

DependentBallThread.java

```
public class DependentBallThread extends Thread {
    private Ball ball;
    private BallCanvas canvas;
    private BounceFrame bounceFrame;
    private BallThread ballThread;
```

```
    private int framesCount = 10_000;

    public DependentBallThread(Ball ball, BallCanvas canvas,
BounceFrame bounceFrame, BallThread ballThread) {
        this.ball = ball;
        this.canvas = canvas;
        this.bounceFrame = bounceFrame;
        this.ballThread = ballThread;
    }

    public DependentBallThread(Ball ball, BallCanvas canvas,
BounceFrame bounceFrame, BallThread ballThread, int
framesCount) {
        this.ball = ball;
        this.canvas = canvas;
        this.bounceFrame = bounceFrame;
        this.ballThread = ballThread;
        this.framesCount = framesCount;
    }



    @Override
    public void run() {
        try {
            ballThread.join();
            for (int i = 1; i < framesCount; i++) {
                ball.move();
                System.out.println("Thread name = "
                        + Thread.currentThread().getName());
                Thread.sleep(5);

                if (bounceFrame.getHole().BallInHole(ball)) {
                    canvas.remove(ball);
                    canvas.repaint();
                    bounceFrame.incBallsInHole();
                    return;
                }
            }
        } catch (InterruptedException ex) {
        }
    }
}
```

Hole.java

```
import java.awt.*;

public class Hole {
    private int x;
```

```
    private int y;
    private int radius;

    public Hole(int x, int y, int radius) {
        this.x = x;
        this.y = y;
        this.radius = radius;
    }

    public boolean BallInHole(Ball ball){
        int ballX = ball.getX();
        int ballY = ball.getY();
        int ballXSize = ball.getXsize();
        int ballYSize = ball.getYsize();
        int ballCenterX = ballX + ballXSize/2;
        int ballCenterY = ballY + ballYSize/2;
        int holeCenterX = x + radius;
        int holeCenterY = y + radius;
        int distance = (int) Math.sqrt(Math.pow(ballCenterX -
holeCenterX, 2) + Math.pow(ballCenterY - holeCenterY, 2));

        return distance <= radius;
    }

    public void draw(Graphics2D g2){
        g2.setColor(Color.YELLOW);
        g2.fillOval(x, y, radius*2, radius*2);
    }
}
```

Завдання 5

Main.java

```
import jdk.jshell.spi.ExecutionControl;

import java.util.List;

// Press Shift twice to open the Search Everywhere dialog and
type `show whitespaces`,
// then press Enter. You can now see whitespace characters in
your code.
public class Main {
    public static void main(String[] args) {
//        NonSyncSymbPrinter nonSyncSymbPrinter = new
NonSyncSymbPrinter('-', 500);
//        NonSyncSymbPrinter nonSyncSymbPrinter1 = new
NonSyncSymbPrinter('|', 500);
//        nonSyncSymbPrinter.start();
//        nonSyncSymbPrinter1.start();
//
//        try {
//            nonSyncSymbPrinter.join();
//            nonSyncSymbPrinter1.join();
//        } catch (InterruptedException e) {
//            throw new RuntimeException(e);
//        }
//
//        System.out.println();
        char[] symbols = {'-', '|'};
        Synchronizer synchronizer = new Synchronizer(symbols);
        Thread[] threads = new Thread[symbols.length];
        for (int i = 0; i < symbols.length; i++) {
            threads[i] = new Thread(new
SymbolPrinter(symbols[i], synchronizer, 5000));
        }
        for (Thread value : threads) {
            value.start();
        }

        try {
            for (Thread thread : threads) {
                thread.join();
            }
        } catch (InterruptedException e) {
            throw new RuntimeException(e);
        }

    }
}
```

NonSyncSymbPrinter.java

```
public class NonSyncSymbPrinter extends Thread {
    private final char symbol;
    private final int elementsCount;

    public NonSyncSymbPrinter(char symbol, int elementsCount)
{
        this.symbol = symbol;
        this.elementsCount = elementsCount;
    }

    @Override
    public void run() {
        for (int i = 0; i < elementsCount; i++) {
            System.out.print(symbol);
            if (i % 50 == 0 && i != 0) {
                System.out.println();
            }
        }
    }
}
```

SymbPrinter.java

```
public class SymbolPrinter implements Runnable {
    private final char symbol;
    private final Synchronizer synchronizer;
    private final int iterationCount;

    public char getSymbol() {
        return symbol;
    }

    public SymbolPrinter(char symbol, Synchronizer
synchronizer, int iterationCount) {
        this.symbol = symbol;
        this.synchronizer = synchronizer;
        this.iterationCount = iterationCount;
    }

    @Override
    public void run() {
        for (int i = 0; i < iterationCount; i++) {
            synchronized (synchronizer) {
                while (synchronizer.getActiveSymbol() !=
symbol) {
                    try {
                        synchronizer.wait();
                    } catch (InterruptedException e) {
```

```
                        throw new RuntimeException(e);
                    }
                }

                System.out.print(symbol);
                synchronizer.updateActiveIndex();
            }
        }
    }
}
```

Synchronizer.java

```
public class Synchronizer {
    private final char[] symbols;
    private int activeIndex;
    private int counter;

    public Synchronizer(char[] symbols) {
        this.symbols = symbols;
    }

    public char getActiveSymbol() {
        return symbols[activeIndex];
    }

    public void updateActiveIndex() {
        activeIndex++;
        activeIndex %= symbols.length;
        counter++;

        if (counter % 100 == 0 && counter != 0) {
            System.out.println();
        }

        notifyAll();
    }
}
```

Завдання 6

Counter.java

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

public class Counter {
    private int value;
    private final ReentrantLock lock;

    public Counter(int value) {
        this.value = value;
        lock = new ReentrantLock();
    }

    public int getValue() {
        return value;
    }

    public void incrementNonSync() {
        value++;
    }

    public void decrementNonSync() {
        value--;
    }
    public synchronized void incrementSyncMethod() {
        value++;
    }

    public synchronized void decrementSyncMethod() {
        value--;
    }

    public void incrementSyncBlock() {
        synchronized (this) {
            value++;
        }
    }

    public void decrementSyncBlock() {
        synchronized (this) {
            value--;
        }
    }

    public void incrementLock(){
        lock.lock();
        value++;
        lock.unlock();
```

```
    }

    public void decrementLock(){
        lock.lock();
        value--;
        lock.unlock();
    }
}
```

DecrementThread.java

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

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

    @Override
    public void run() {
        for (int i = 0; i < times; i++) {
//            counter.decrementNonSync(); // non-synchronized
decrement
            counter.decrementSyncMethod(); // synchronized
method decrement
//            counter.decrementSyncBlock(); // synchronized
block decrement
//            counter.decrementLock(); // lock decrement
        }

    }
}
```

IncrementThread.java

```
public class IncrementThread extends Thread{
    private final Counter counter;
    private final int times;
    public IncrementThread(Counter counter, int times) {
        this.counter = counter;
        this.times = times;
    }

    @Override
    public void run() {
        for (int i = 0; i < times; i++) {
//            counter.incrementNonSync(); // non-synchronized
```

```
increment
            counter.incrementSyncMethod(); // synchronized
method increment
//            counter.incrementSyncBlock(); // synchronized
block increment
//            counter.incrementLock(); // lock increment
        }
    }
}
```

Main.java

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

public class Main {
    public static void main(String[] args) {
        Counter counter = new Counter(0);
        HashSet<Thread> threads = new HashSet<>();

        threads.add(new IncrementThread(counter, 1_000_000));
//          threads.add(new IncrementThread(counter,
1_000_000));
        threads.add(new DecrementThread(counter, 1_000_000));

        for (Thread thread : threads) {
            thread.start();
        }
        try {
            for (Thread thread : threads) {
                thread.join();
            }
        } catch (InterruptedException e) {
            throw new RuntimeException(e);
        }

        System.out.println(counter.getValue());
    }
}
```