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 Галько М.В. _____
(шифр, прізвище, ім'я, по батькові)

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

Київ 2022

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.**

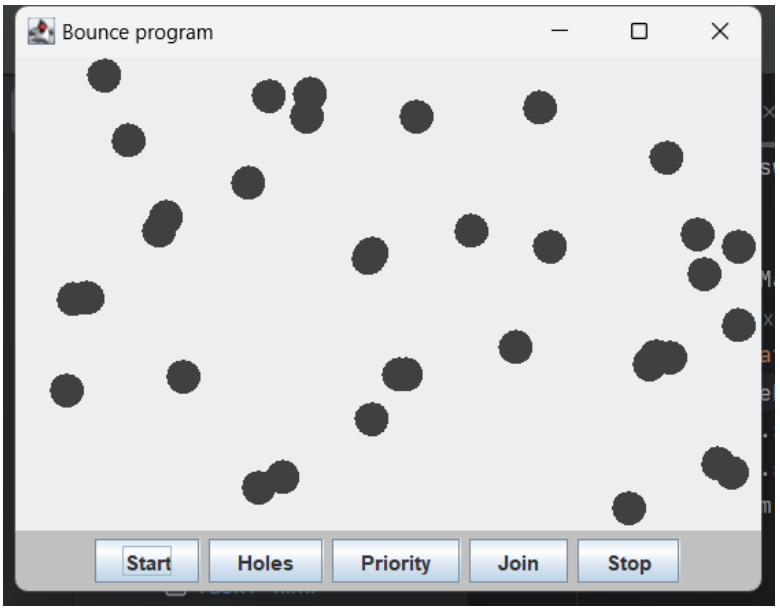

Рис. 1 – Виконання пункту 1 при невеликій кількості кульок

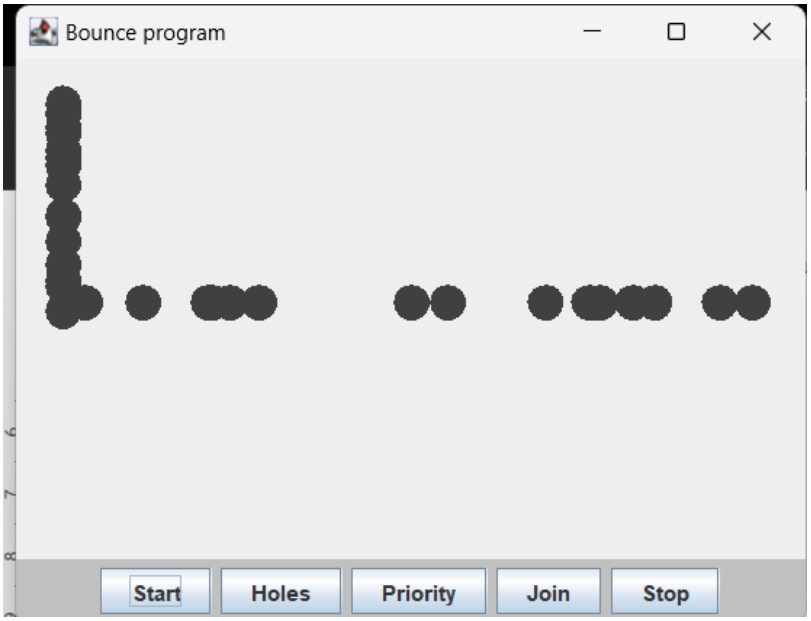

Рис. 2 – Виконання п.1 при великій кількості кульок

У даному завданні реалізовано програму імітації руху більярдних кульок, де кожна кулька має власний потік. Під час розробки програми, було протестовано роботу з різною кількістю кульок (рис. 1).

Спостерігаючи роботу програми, при збільшенні кількості кульок та досягненні певної межі, програма починає тормозити, а деякі кульки застрягають в певних місцях і не рухаються (більше 300 кульок). Це пов'язано з обмеженнями ресурсів системи та обчислювальної потужності.

При збільшенні кількості кульок (рис. 2), програмі потрібно більше ресурсів для обчислення руху кожної кульки в окремому потоці. Це може призвести до затримок у виконанні обчислень та сповільнення роботи програми. Крім того, при великій кількості кульок, збільшується ймовірність виникнення конфліктів та гонки за ресурсами між потоками, що може призвести до неправильної роботи програми.

У нашому випадку, при досягненні певної межі, деякі кульки можуть застрягати в певних місцях і не рухатися. Це може бути пов'язано з колізіями та взаємодією між кульками. При великій кількості кульок, збільшується ймовірність зіткнень та перекриття шляхів руху, що може призвести до неправильного руху або застрягання.

Використання окремих потоків для кожної кульки дозволяє досягти паралельного виконання їх руху. Потокова архітектура дозволяє реалізувати багатозадачність, що дозволяє програмі виконувати одночасно декілька операцій на різних потоках, забезпечуючи більш гнучку та швидку обробку даних.

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.**

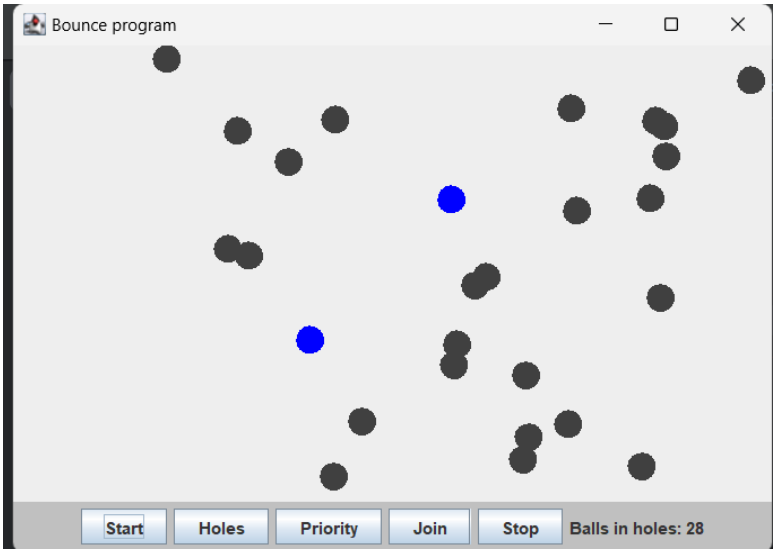

Рис. 3 – Модифікація ПЗ із лунками

Для реалізації цих змін, я включила перевірку, чи кульки дотинаються до синіх статичних кіл – лузи. Якщо так, кулька вважається потрапленою в лузу. У цьому випадку, відповідний потік, який відповідає за рух даної кульки, завершує свою роботу. Крім того, збільшується лічильник кульок, які потрапили в лузу, і ця кількість динамічно оновлюється 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. **20 балів.**

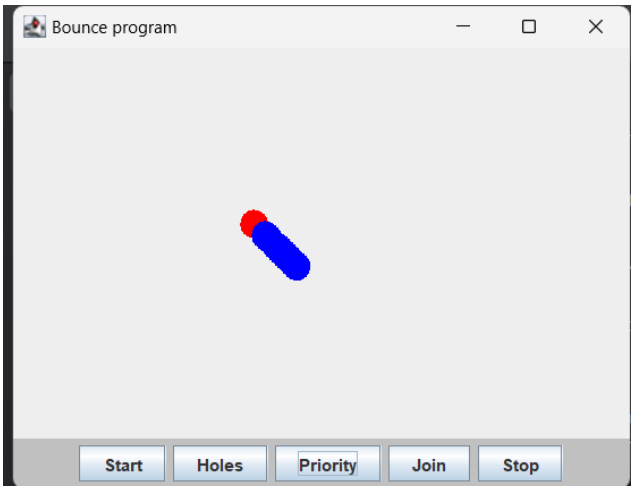

Рис. 4 – priority потік із 100 синіми кульками

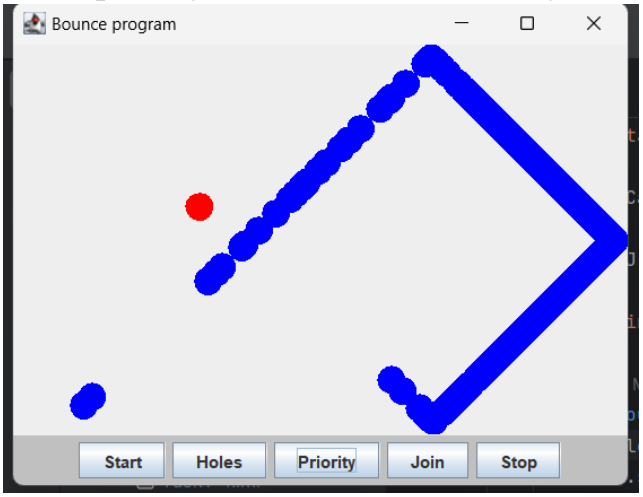

Рис. 5 – priority потік із 1000 синіми кульками

При невеликій кількості синіх кульок (рис. 4), червона кулька рухається на початку черги (створюється першою). Також із часом, черга стає менш щільною, але не значно. Це можна пояснити тим, що при

невеликій кількості кульок обчислювальних ресурсів вистачає на всі кульки і пріоритети стають менш суттєвими.

Коли є лише декілька кульок, кожна з них має достатньо обчислювальних ресурсів для виконання своїх обчислень і зміни своїх координат. В такій ситуації не виникає сильної конкуренції за обчислювальні ресурси, і кожна кулька може виконувати свої обчислення без затримок.

Однак, при збільшенні кількості кульок (рис. 5), зростає конкуренція за обчислювальні ресурси. Кульки стають залежними від доступності ресурсів, що може призводити до затримок у виконанні їхніх обчислень та змін координат. Пріоритети в такій ситуації стають більш важливими, оскільки деякі кульки можуть отримувати більше обчислювального часу та ресурсів, ніж інші.

Отже, при невеликій кількості кульок немає сильної потреби у встановленні пріоритетів, оскільки обчислювальні ресурси розподіляються рівномірно. Проте, при збільшенні кількості кульок пріоритети стають важливішими для ефективного використання обчислювальних ресурсів та запобігання затримок у роботі програми.

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

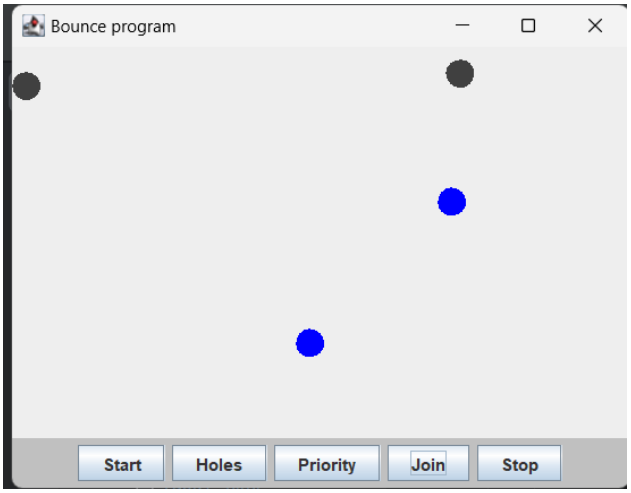

Рис. 6 – Ілюстрація методу join

Спочатку ліва чорна кулька з рис. 6 стоїть нерухомо, а права рухається. Коли рухома кулька закінчує свій життєвий цикл (у нашому випадку потрапляє у синю лузу), виконується операція join(), і кулька зліва починає свій рух.

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.**

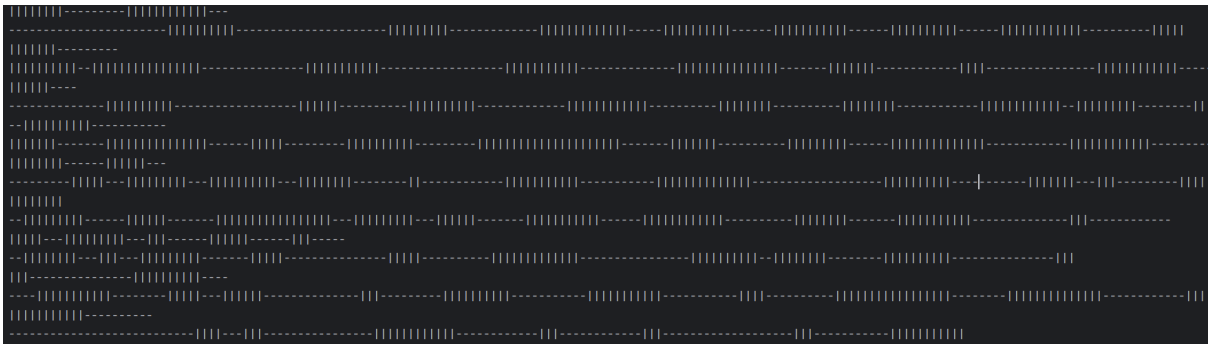

Рис. 7 – пункт 5 ПЗ без впровадження додаткових методів

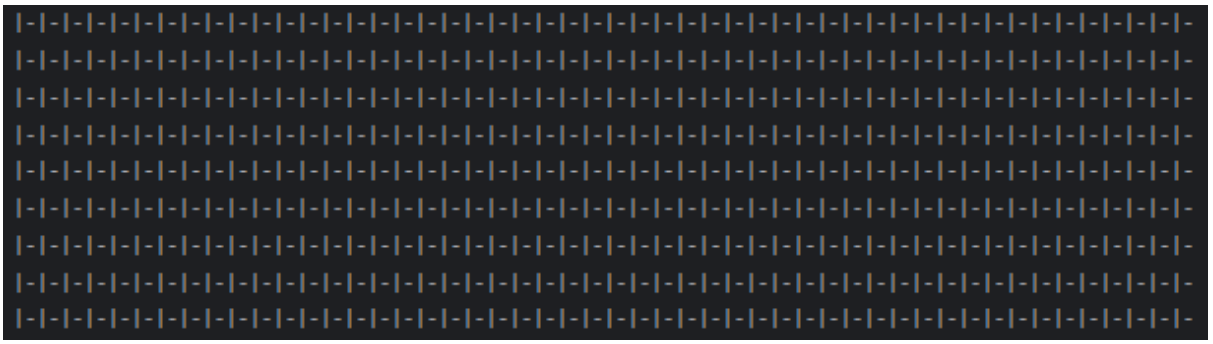

Рис. 8 – пункт 5 із методами wait, notifyAll

З рис. 7 видно, що символи не чергуються в рядку і різна кількість символів може з'являтись на консолі. Це пояснюється тим, що операція виводу на консоль є унарною і синхронізованою для всієї системи, тому одночасно може виводитись лише один символ. В такому випадку не встановлюється жодного порядку, що регулював би, який саме потік має виводити символ, що призводить до хаотичного виводу.

Тому, при запуску двох потоків, які виводять символи '-', '|' в рядок, вивід на консоль стає хаотичним, оскільки відсутній механізм встановлення послідовності між потоками. Кожен потік намагається вивести свій символ на консоль без регуляції порядку, що призводить до непередбачуваного розташування символів у рядку.

За допомогою методів wait() та notifyAll() було досягнуто почергового виведення символів на консоль (рис. 8). Коли потік 1 виводить символ '-', він викликає метод wait() і переходить у режим очікування. Потім, коли основна програма викликає метод notifyAll(), усі потоки пробуджуються і змагаються за доступ до виводу. Таким чином, потік 2

має можливість вивести символ '|', а потім перейти у режим очікування за допомогою методу wait(). Цей процес повторюється сто разів, що забезпечує почергове виведення символів '-' та '|' у рядку.

Цей підхід дозволяє уникнути хаотичного розташування символів та забезпечує регулювання порядку виводу між потоками, що покращує зрозумілість та передбачуваність.

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.**

Значення лічильників без синхронізації: -27950, 15751, 25406;

Значення через використання синхронізованого метода, синхронізованого блоку та блокування об’єкта: 0.

Без використання методів синхронізації виконання програми не виходить 0 через проблему гонки (race condition). Гонка виникає тоді, коли два або більше потоки намагаються змінювати спільний ресурс одночасно і непередбачувано, що може призвести до некоректних результатів.

Коли два або більше потоки одночасно намагаються змінити значення лічильника, виникають проблеми консистентності даних. Наприклад, якщо два потоки одночасно збільшують лічильник, може статися так, що обидва потоки прочитують початкове значення лічильника, виконують свою операцію збільшення, а потім записують результат у пам'ять. Це може призвести до того, що обидва потоки перезапишуть одне одного, тобто одне збільшення може бути втрачене.

Порівнюючи способи синхронізації - синхронізований метод, синхронізований блок і блокування об'єкта - можна відзначити наступні особливості:

Синхронізований метод:

1. Застосовується до цілого методу.
2. Забезпечує автоматичний захист доступу до критичної секції коду.
3. Може мати великий розмір, оскільки весь метод є блоком синхронізації.
4. Є зручним і простим у використанні, але може мати обмежену гнучкість у визначенні областей синхронізації.

Синхронізований блок:

1. Застосовується до певної частини коду, використовуючи блокування об'єкта.
2. Дозволяє точніше визначити місце, яке потребує синхронізації, і, таким чином, може мати менший розмір.

3. Забезпечує автоматичний захист доступу до критичної секції коду.
4. Надає більшу гнучкість у визначенні областей синхронізації, оскільки можна використовувати будь-який об'єкт як м'ютекс для блокування.

Блокування об'єкта:

1. Надає максимальну гнучкість у визначенні областей синхронізації.
2. Використовується для блокування об'єкта, який визначає область синхронізації.
3. Дозволяє використовувати будь-який об'єкт як блокований.
4. Має більшу гнучкість, але вимагає вручну виконувати блокування і розблокування об'єкта.

Висновок

В ході цієї лабораторної роботи було вивчено основні операції з управління потоками. Була проведена серія експериментів, щоб дослідити вплив кількості потоків на роботу програми і розподіл обчислювальних ресурсів. Також було розглянуто методи синхронізації та пояснено їх різницю.

Під час експериментів було встановлено, що збільшення кількості потоків може призвести до збільшення загального часу виконання програми через змагальність за обчислювальні ресурси. Також виявлено, що пріоритетність потоків може впливати на розподіл обчислювального ресурсу, але в ситуаціях з невеликою кількістю потоків пріоритети майже не мають суттєвого значення.

Також були вивчені та застосовані методи синхронізації: синхронізований метод, синхронізований блок і блокування об'єкта. Кожен з цих методів має свої переваги і обмеження, але всі вони служать для забезпечення правильної роботи лічильника при одночасній роботі з ним двох або більше потоків.

Код для 1-4 завдань:

Main

```
import javax.swing.*;

public class Main {
   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.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.Random;

public class BounceFrame extends JFrame {
   public static final int WIDTH = 450;
   public static final int HEIGHT = 350;
   private Canvas canvas;
   private JLabel label;
   private int ballsInHoles = 0;

   public BounceFrame() {
       int lowPriorityBallsCount = 1000;
       this.setSize(WIDTH, HEIGHT);
       this.setTitle("Bounce program");
       this.canvas = new Canvas(this);
       Container content = this.getContentPane();
       content.add(this.canvas, BorderLayout.CENTER);
       JPanel buttonPanel = new JPanel();
       buttonPanel.setBackground(Color.lightGray);
       System.out.println("In Frame Thread name = "
               + Thread.currentThread().getName());

       JButton buttonStart = new JButton("Start");
       JButton buttonHolesSwitch = new JButton("Holes");
       JButton buttonColorBalls = new JButton("Priority");
       JButton buttonJoin = new JButton("Join");
       JButton buttonStop = new JButton("Stop");

       label = new JLabel();
       setLabelBallsInHoles();

       Hole h1 = new Hole(canvas, 300, 100, 20);
       Hole h2 = new Hole(canvas, 200, 200, 20);
       canvas.add(h1);
       canvas.add(h2);

       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());
           }
       });
       buttonHolesSwitch.addActionListener(new ActionListener() {
           @Override
           public void actionPerformed(ActionEvent e) {
               canvas.switchHolesEnabled();
               canvas.repaint();
           }
       });
       buttonColorBalls.addActionListener(new ActionListener() {
           @Override
```

```
            public void actionPerformed(ActionEvent e) {
                int x = new Random().nextInt(canvas.getWidth());
                int y = new Random().nextInt(canvas.getWidth());

                Ball b = new Ball(canvas, x, y, Color.red);
                canvas.add(b);
                BallThread thread = new BallThread(b);
                thread.setPriority(Thread.MAX_PRIORITY);
                thread.start();

                for (int i = 0; i < lowPriorityBallsCount; i++) {
                    b = new Ball(canvas, x, y, Color.blue);
                    canvas.add(b);
                    thread = new BallThread(b);
                    thread.setPriority(Thread.MIN_PRIORITY);
                    thread.start();
                }
            }
        });

        buttonJoin.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                Ball b1 = new Ball(canvas);
                Ball b2 = new Ball(canvas);
                canvas.add(b1);
                canvas.add(b2);
                BallThreadJoin joinThread = new BallThreadJoin (new BallThread(b1), new
BallThread(b2));
                joinThread.start();
            }
        });
        buttonStop.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                System.exit(0);
            }
        });

        buttonPanel.add(buttonStart);
        buttonPanel.add(buttonHolesSwitch);
        buttonPanel.add(buttonColorBalls);
        buttonPanel.add(buttonJoin);
        buttonPanel.add(buttonStop);
        buttonPanel.add(label);
        content.add(buttonPanel, BorderLayout.SOUTH);
    }

    public void setLabelBallsInHoles() {
        label.setText("Balls in holes: " + canvas.getBallsInHoles());
    }
}
```

Canvas

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

public class Canvas extends JPanel {
    private BounceFrame frame;
    private ArrayList<Ball> balls = new ArrayList<>();
    private ArrayList<Hole> holes = new ArrayList<>();
    private int ballsInHoles = 0;
    public Canvas(BounceFrame frame) {
        this.frame = frame;
    }

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

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

```
    @Override
    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        Graphics2D g2 = (Graphics2D) g;
            for (Hole hole : holes) {
                if (hole.isEnabled()) {
                    hole.draw(g2);
                }
            }
        for (Ball ball : balls) {
            ball.draw(g2);
        }
    }

    public List<Hole> getHoles() {
        return holes;
    }

    public int getBallsInHoles() {
        return ballsInHoles;
    }

    public void switchHolesEnabled() {
        for (Hole hole : holes) {
            hole.switchEnabled();
        }
    }

    public synchronized void remove(Ball b) {
        balls.remove(b);
        ballsInHoles++;
        frame.setLabelBallsInHoles();
    }
}
```

Ball

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

public class Ball {
    private Canvas 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 = Color.darkGray;

    public Ball(Canvas 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 Ball(Canvas c, int x, int y, Color color) {
        this.canvas = c;
        this.x = x;
        this.y = y;
        this.color = color;
    }

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


    public void draw(Graphics2D g2) {
```

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

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

    public int getX() { return x; }
    public int getY() { return y; }
    public int getRadius() { return XSIZE / 2;}
    public Canvas getCanvas() { return canvas;}
    public Color getColor() {return color;}
}
```

BallThread

```
import java.util.List;

public class BallThread extends Thread {
    private Ball b;
    private List<Hole> holes;

    public BallThread(Ball ball) {
        b = ball;
        holes = b.getCanvas().getHoles();
    }

    @Override
    public void run() {
        try {
            for (int i = 1; i < 10000; i++) {
                b.move();
                for (Hole hole : holes) {
                    if (hole.checkBall(b)) {
                        b.getCanvas().remove(b);
                        b.getCanvas().repaint();
                        return;
                    }
                }
                System.out.println("Thread name = "
                        + Thread.currentThread().getName());
                Thread.sleep(5);
            }
        } catch (InterruptedException ex) {
        }
    }
}
```

BallThreadJoin

```
public class BallThreadJoin extends Thread {

    private final BallThread th1;
    private final BallThread th2;

    public BallThreadJoin(BallThread th1, BallThread th2) {
```

```
        this.th1 = th1;
        this.th2 = th2;
    }

    public void run () {
        th1.start();
        try {
            th1.join();
        } catch (InterruptedException ex) {
            throw new RuntimeException(ex);
        };
        th2.start();
    }
}
```

Hole

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

public class Hole {
    private int x;
    private int y;
    private Component canvas;
    private int size;
    private boolean isHoleEnabled = true;

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

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

    public boolean checkBall(Ball b) {
        if (!isHoleEnabled) {
            return false;
        }
        int ballCenterX = b.getX() + b.getRadius();
        int ballCenterY = b.getY() + b.getRadius();
        int holeCenterX = x + size / 2;
        int holeCenterY = y + size / 2;
        int distance = (int) Math.sqrt(Math.pow(ballCenterX - holeCenterX, 2) +
Math.pow(ballCenterY - holeCenterY, 2));
        return distance <= size / 2 + b.getRadius();
    }
    public void switchEnabled() {isHoleEnabled = !isHoleEnabled;}
    public boolean isEnabled() {return isHoleEnabled;}
}
```

Код для завдання 5

Main

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

        printer.addSymbolThread(new SymbolThread('|'));
        printer.addSymbolThread(new SymbolThread('-'));

        Sync permission = new Sync();
        printer.addSyncSymbolThread(new SyncSymbolThread('|', permission, true));
        printer.addSyncSymbolThread(new SyncSymbolThread('-', permission, false));

        //printer.printThreads();
        printer.printSyncThreads();
    }
}
```

Printer

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

public class Printer {
    List<SymbolThread> threads;
    List<SyncSymbolThread> syncThreads;

    public Printer() {
        threads = new ArrayList<>();
        syncThreads = new ArrayList<>();
    }

    public void addSymbolThread(SymbolThread symbolThread) {
        threads.add(symbolThread);
    }

    public void addSyncSymbolThread(SyncSymbolThread symbolThread) {
        syncThreads.add(symbolThread);
    }

    public void printThreads() {
        for (SymbolThread thread : threads) {
            thread.start();
        }
    }

    public void printSyncThreads() {
        for (SyncSymbolThread thread : syncThreads) {
            thread.start();
        }
    }
}
```

SymbolThread

```
public class SymbolThread extends Thread {
    char symbol;
    public SymbolThread(char symbol) {
        this.symbol = symbol;
    }

    @Override
    public void run() {
        for (int j = 0; j < 1000; j++) {
            for (int i = 0; i < 100; i++) {
                System.out.print(symbol);
            }
            System.out.println();
        }
    }
}
```

```
public class SyncSymbolThread extends Thread {
```

```
    private final char symbol;
    private final Sync sync;
    private final boolean controlValue;

    public SyncSymbolThread(char s, Sync sync, boolean controlValue) {
        this.symbol = s;
        this.sync = sync;
        this.controlValue = controlValue;
    }

    @Override
    public void run() {
        while (true) {
            sync.waitAndChange(controlValue, symbol);
            if (sync.isFinished()) {
                return;
            }
        }
    }
}
```

Sync

```
public class Sync {
    private boolean permission = true;
    private boolean finished = false;
    private int num = 0;

    public synchronized boolean isFinished() {
        return finished;
    }

    public synchronized void waitAndChange(boolean controlValue, char symbol) {
        while (this.permission != controlValue) {
            try {
                wait();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        System.out.print(symbol);
        this.permission = !permission;
        num++;
        if (num % 100 == 0) System.out.println();
        if (num == 1000) finished = true;
        notifyAll();
    }
}
```

Код для завдання 6:

Main

```java
import java.util.List;

public class Main {
   public static void main(String[] args) throws InterruptedException {
       int iterations = 100000;
       Counter counter = new Counter();
       List<Thread> threads = List.of(
               new Thread(new Runnable() {
                   @Override
                   public void run() {
                       for (int i = 0; i < iterations; i++) {
                           counter.increment();
//                             counter.incrementSync();
//                             counter.incrementSyncBlock();
//                             counter.incrementLock();
                       }
                   }
               }),
               new Thread(new Runnable() {
                   @Override
                   public void run() {
                       for (int i = 0; i < iterations; i++) {
                           counter.decrement();
//                             counter.decrementSync();
//                             counter.decrementSyncBlock();
//                             counter.decrementLock();
                       }
                   }
               })
       );

       for (Thread thread : threads) thread.start();
       for (Thread thread : threads) thread.join();
       System.out.print(counter.getCount());
   }
}
```

Counter

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

public class Counter {
   static int count = 0;
   private ReentrantLock locker = new ReentrantLock();
   public Counter() {
   }

   public void increment() {
       count++;
   }

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

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

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

   public void decrement() {
       count--;
   }

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

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

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

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