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

Звіт
з дисципліни «Технології розподілених обчислень»
Комп’ютерний практикум №1
«Розробка потоків та дослідження пріоритету запуску потоків»

Виконав:
Студент III курсу
гр. ІП-33
Соколов О. В.

Прийняв:
Дифучин А. Ю.
“14” Лютого 2026 р.

Київ – 2026

Комп’ютерний практикум №1

Тема: Розробка потоків та дослідження пріоритету запуску потоків.

Мета: Здобути практичні навички багатопоточної розробки мовою Java. Дослідити подію гонки потоків на прикладі лічильника за допомогою різних методів синхронізації і їх відмінність від непотокобезпечних операцій.

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. 15 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. 25 балів.

4. Побудуйте ілюстрацію методу join() класу Thread через взаємодію потоків, що відтворюють рух більярдних кульок різного кольору. Поясніть результат, який спостерігається. 25 балів.

5. 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.

Виконання

Завдання 1

За основу проєкту взято приклад із презентації «Створення та запуск потоків у Java». Реалізовано програму імітації руху більярдних кульок, у якій рух кожної кульки виконується в окремому потоці. Клас BallThread розширює Thread; у методі run() у циклі викликається ball.move() та Thread.sleep() для заданого інтервалу. При натисканні «Add Ball» або «Add 10» створюються нова кулька та новий потік, який одразу запускається. Канва відображає кількість кульок, кількість потоків кульок, FPS, навантаження CPU процесу JVM, кількість потоків у станах RUNNABLE та WAITING, а також графік залежності FPS і CPU від часу (у правому верхньому куті). UI та ця інформативність не змінюють логіку потоків і структуру класів.

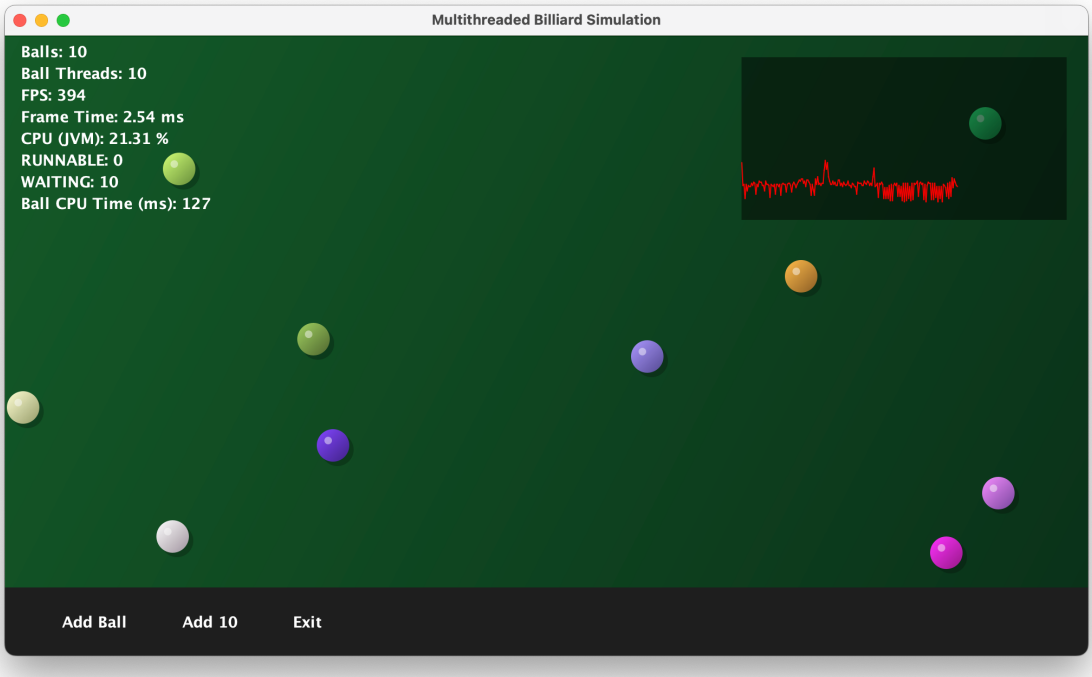

Рисунок 1.1 – Інтерфейс програми із невеликою кількістю кульок

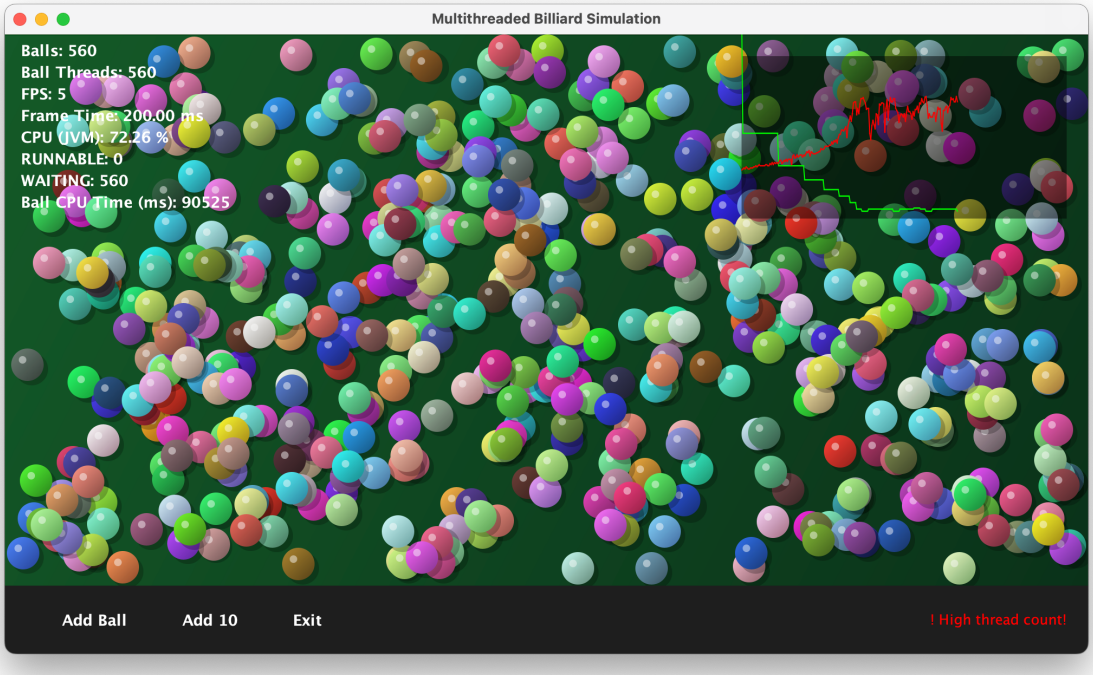

Рисунок 1.2 – Перевірка впливу кількості кульок на продуктивність

При збільшенні кількості кульок спостерігається наступне: зростає кількість одночасно активних потоків; навантаження на CPU (червона лінія на графіку) збільшується; FPS (зелена лінія) падає, тобто програма починає працювати повільніше. При великій кількості кульок (десятки та більше) інтерфейс може стати менш плавним.

Продуктивність програми зменшується при більшій кількості кульок тому, що кожна кулька – окремий потік, який періодично переміщує кульку та викликає перемальовування полотна. Більше кульок – більше потоків, більше контекстних перемикань і навантаження на CPU, а також частіші виклики repaint(). Перемикання між потоками та оновленням GUI обмежує швидкодію, тому FPS знижується.

Окремий потік на кожну кульку дає змогу відтворювати рух паралельно: кульки рухаються одночасно, а не по черзі. Код руху однієї кульки залишається простим (цикл переміщень із затримкою), без явної черги завдань.

Програма коректно реалізує імітацію руху більярдних кульок у окремих потоках. Спостереження підтверджують, що збільшення кількості кульок веде до зростання навантаження на CPU та зниження FPS через збільшення числа потоків і частоту оновлень інтерфейсу.

Завдання 2

Програму модифіковано так, що на полі з’явилися шість луз: чотири в кутах та дві посередині довгих сторін стола. Лузи відмальовуються як темні кола. Після кожного кроку руху кульки в BallThread.run() викликається перевірка canvas.isBallInPocket(ball): якщо центр кульки потрапляє в межах радіусу лузи, кулька вважається потрапившою. У такому разі викликається canvas.pocketBall(ball), який у потокобезпечному режимі видаляє кульку зі списку на полі, збільшує лічильник потрапивших у лузу та оновлює текст у інтерфейсі; потік виходить із циклу (break) і після завершення run() викликає canvas.unregisterBallThread(this), тобто відповідний потік коректно завершує роботу. Кількість кульок, що потрапили в лузу, динамічно відображається.

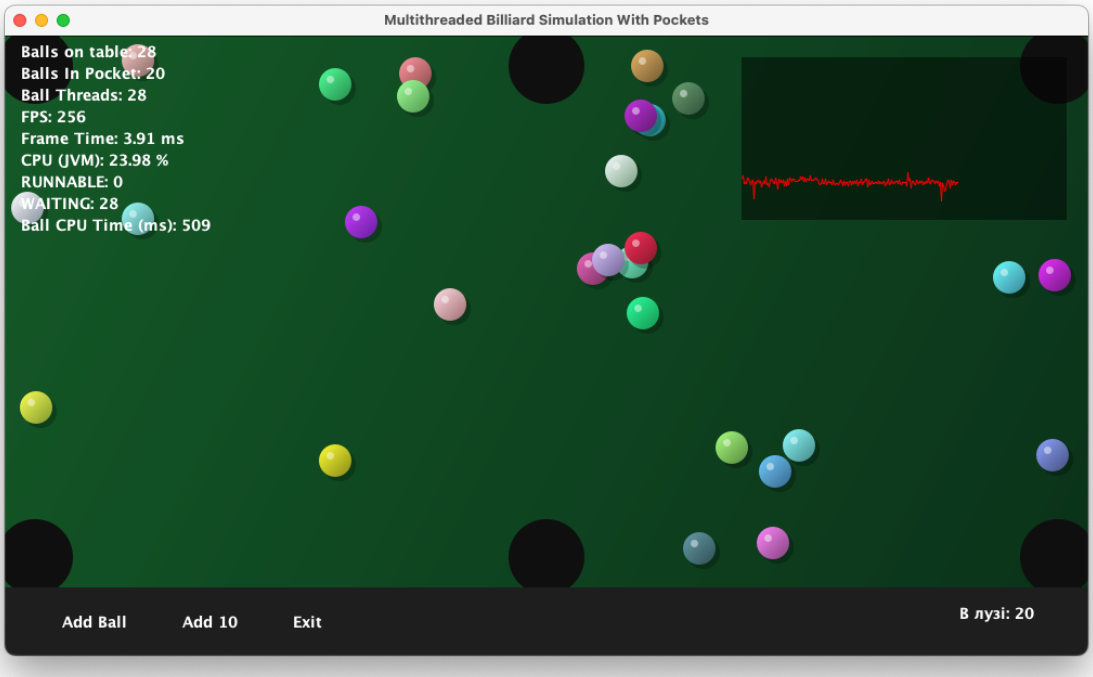

Рисунок 2.1 – Інтерфейс програми із лузами

Завдання 3

Програму модифіковано так, щоб червоні кульки створювалися з вищим пріоритетом потоку (Thread.MAX_PRIORITY), а сині – з нижчим (Thread.MIN_PRIORITY). Введено тип кульки Ball.BallType (RED, BLUE), кожен тип має свій пріоритет потоку; при створенні BallThread йому передається пріоритет і викликається setPriority(priority) перед start(). У інтерфейсі додано кнопки: «Add Red», «Add Blue», «Add 10 Blue», а також режим експерименту «Experiment: 1 Red + N Blue» з полем N: створюється одна червона та N синіх кульок з одним і тим же початковим місцем (наприклад, лівий край по центру висоти), однаковим напрямком (наприклад, dx=2, dy=0) та однаковою швидкістю; усі потоки запускаються після створення кульок, щоб експеримент був коректним.

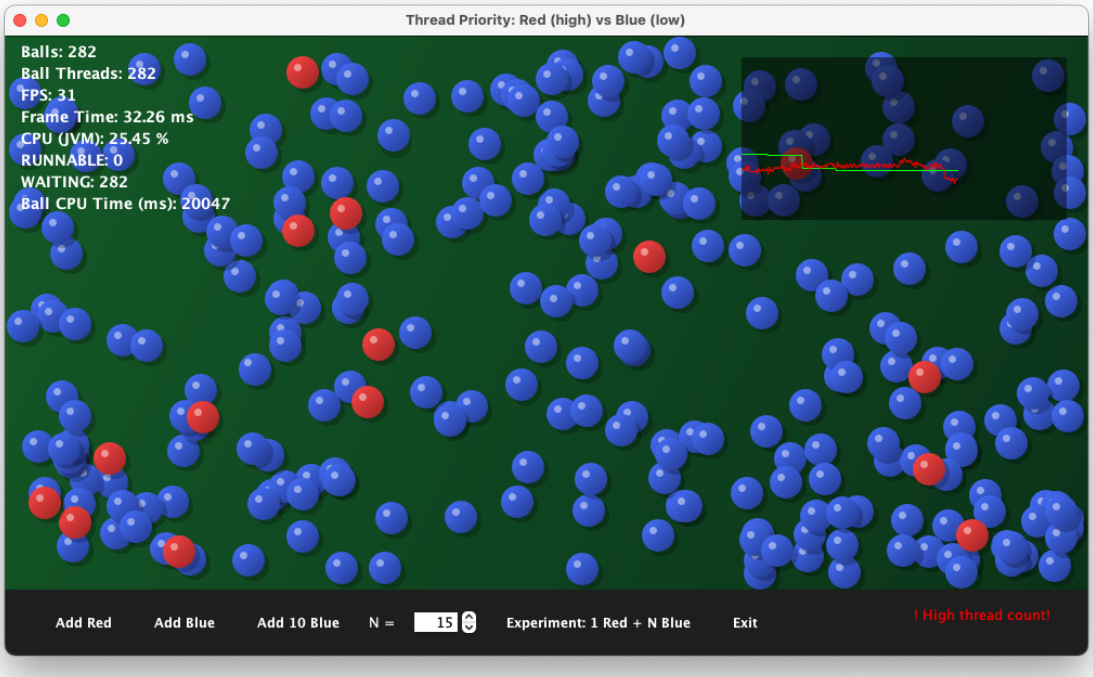

Рисунок 3.1 – Змішана кількість червоних та синіх кульок

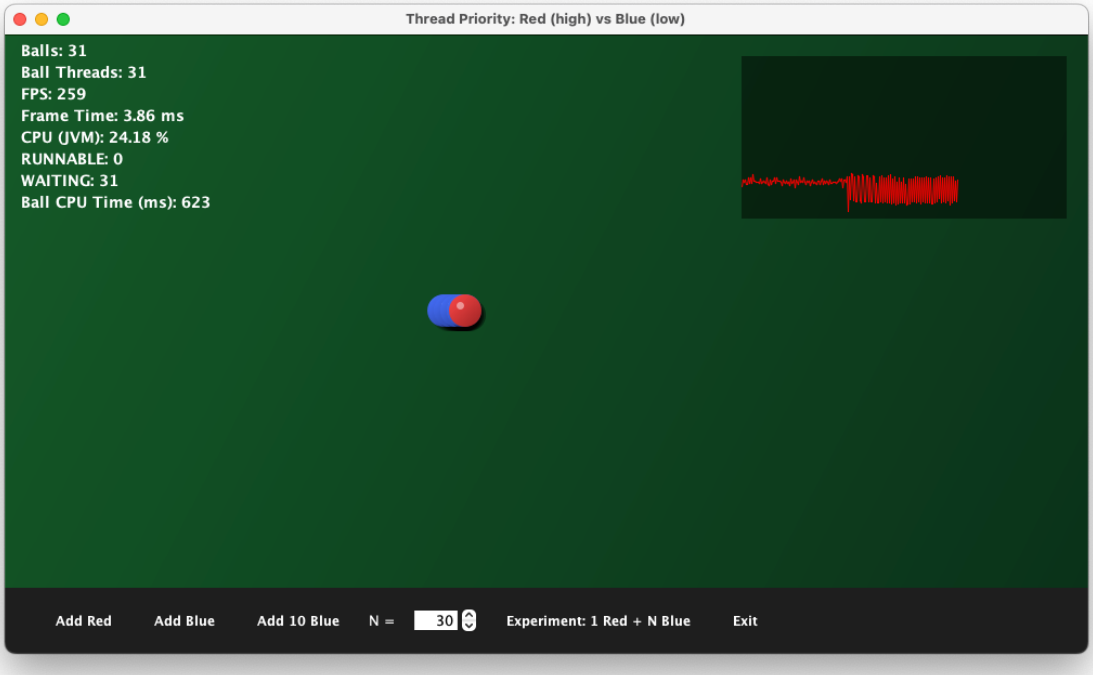

Рисунок 3.2 – Одна червона кулька та невелика кількість синіх

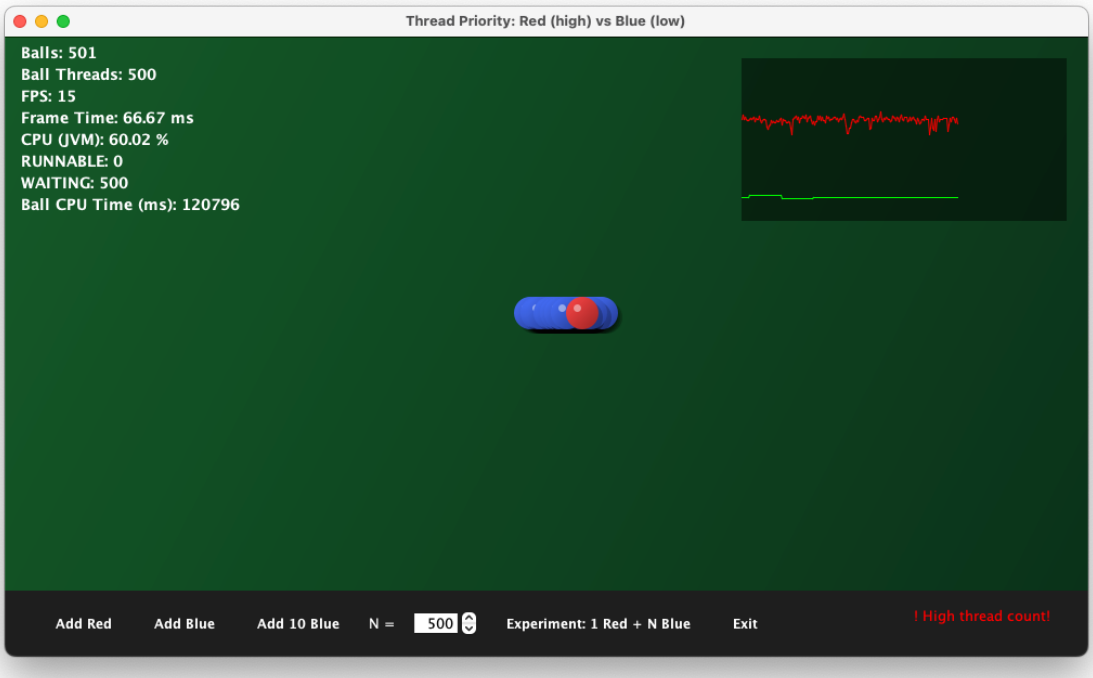

Рисунок 3.3 – Одна червона кулька та велика кількість синіх

При змішаній кількості червоних і синіх кульок червоні часто рухаються плавніше або «випереджають» сині при однаковій логіці руху, оскільки планувальник ОС частіше надає час потокам з вищим пріоритетом. У режимі експерименту (1 червона + багато синіх, однаковий старт): при невеликому N (наприклад, 10–30) червона кулька зазвичай випереджає сині; при великому N (100, 200, 500) сині кульки можуть «розтягуватися» вперед, а червона залишатися в центрі – це нормально: сумарно 200 синіх потоків отримують набагато більше часу CPU, ніж один червоний, тому позиції синіх оновлюються частіше і вони випереджають.

Завдання 4

Було реалізовано демонстрацію методу join() класу Thread через потоки, що відтворюють рух більярдних кульок різного кольору (червона, синя, зелена). Кожна кулька рухається в окремому потоці; потік виконує фіксовану кількість кроків, після чого завершується. Кнопка експерименту запускає допоміжний потік, який:

1. створює червону кульку та її потік і запускає його;
2. викликає redThread.join() — блокується до завершення потоку червоної кульки;
3. після повернення з join() створює синю кульку та її потік і запускає;
4. викликає blueThread.join();
5. аналогічно запускає потік зеленої кульки та викликає greenThread.join().

Рисунок 4.1 – Інтерфейс програми із кнопкою запуску експерименту join()

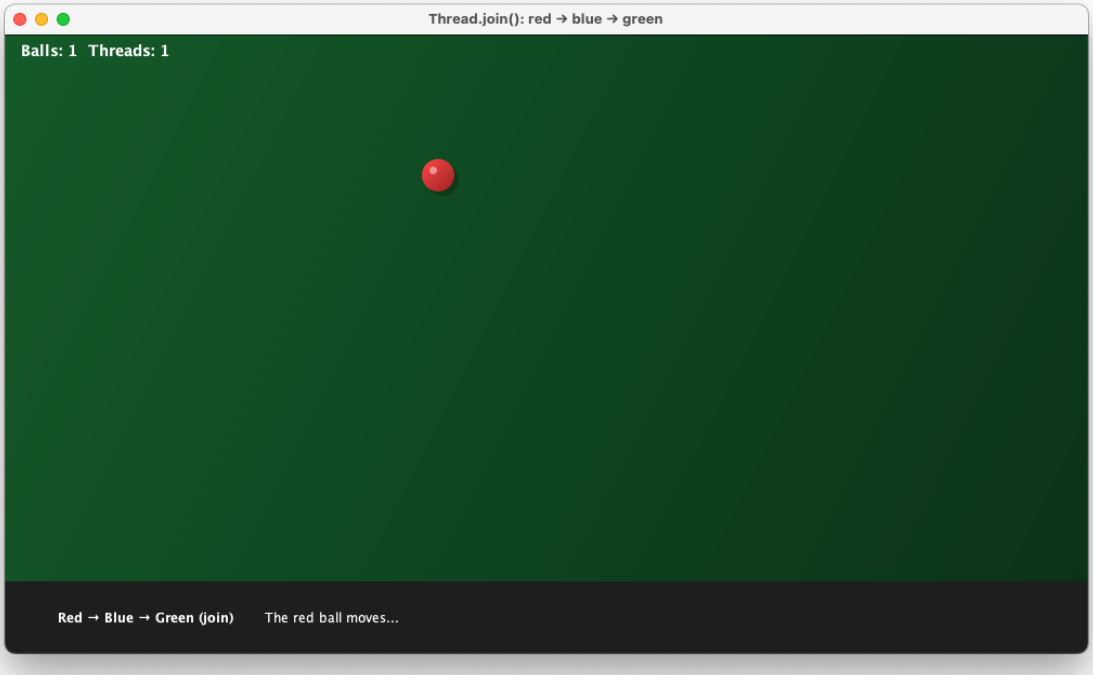

Рисунок 4.2 – Рух червоної кульки

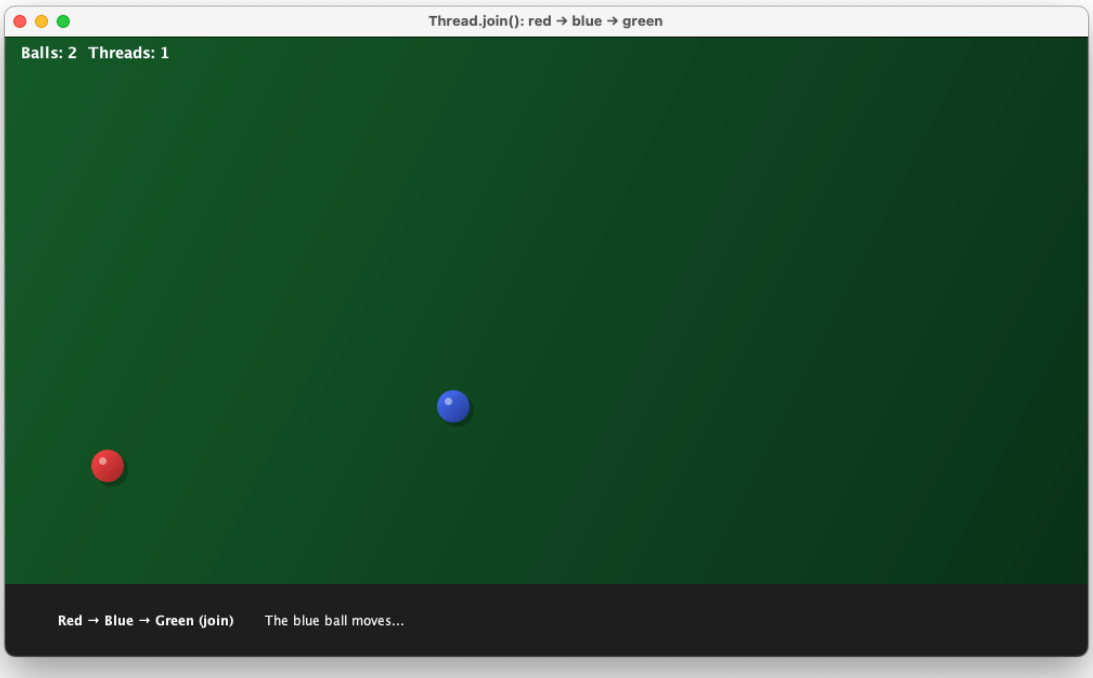

Рисунок 4.3 – Червона кулька завершила рух, синя створилась та почала рух

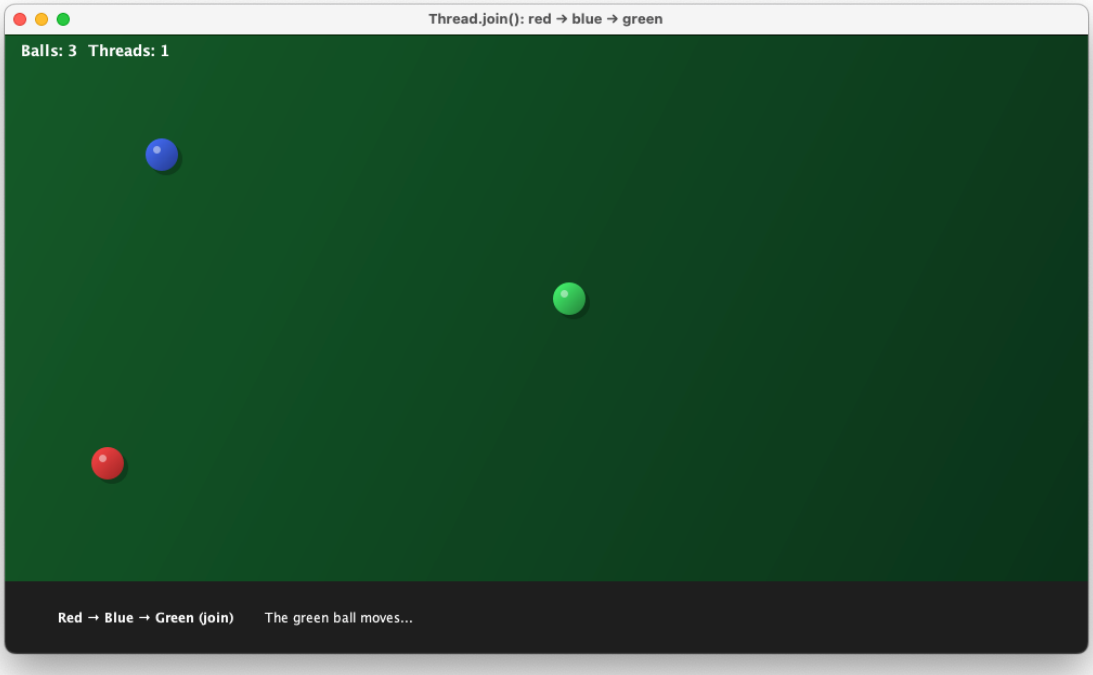

Рисунок 4.4 – Синя кулька завершила рух, зелена створилась та почала рух

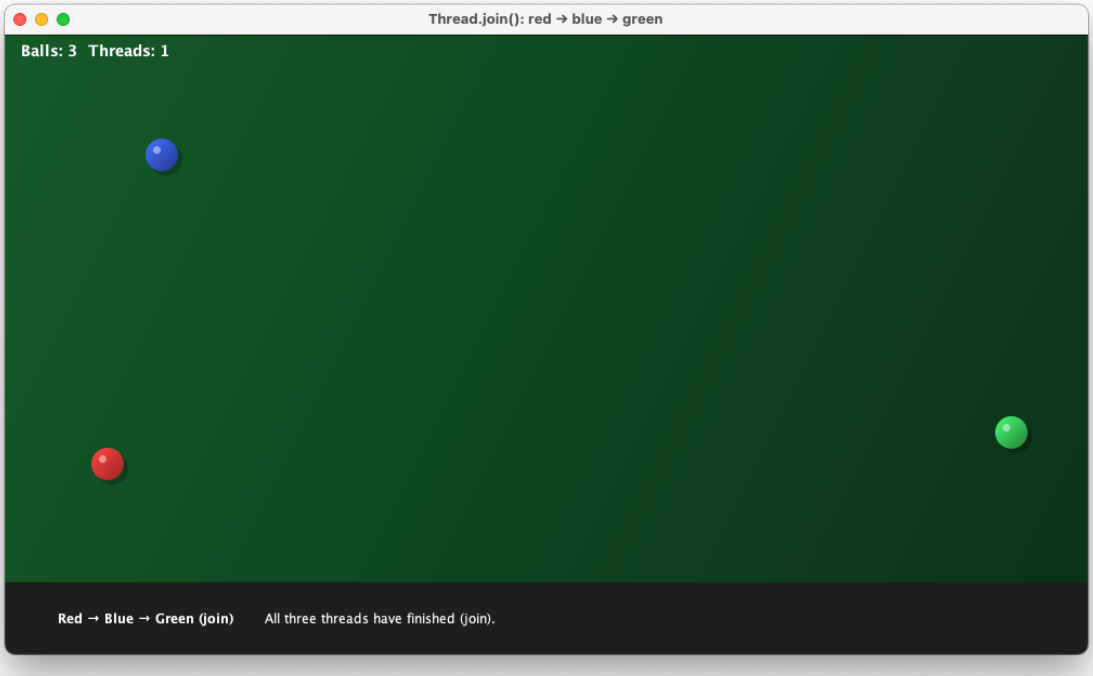

Рисунок 4.5 – Зелена кулька завершила рух, кінець експерименту

Спочатку на полі рухається лише червона кулька. Після того як її потік завершується (вичерпано кількість кроків), з’являється синя кулька і рухається сама. Після завершення потоку синьої з’являється зелена. Тобто кульки рухаються послідовно, а не одночасно: наступна починає рух лише після завершення попередньої. Це безпосередньо демонструє сенс join(): поточний потік (тут – допоміжний) чекає завершення викликаного потоку, перш ніж продовжити виконання. Без join() усі три потоки були б запущені одразу і кульки рухались би паралельно.

Метод join() дозволяє одному потоку дочекатися завершення іншого. У експерименті це реалізовано як послідовний запуск потоків кульок: червона –> синя –> зелена; спостережувана послідовність руху підтверджує, що наступний потік стартує лише після повернення з join().

Завдання 5

Було створено клас Counter з методами increment() та decrement(), які збільшують і зменшують значення лічильника. Клас CounterDemo запускає два потоки: один викликає increment() 100000 разів, інший – decrement() 100000 разів; потоки стартують одночасно, основний потік чекає їх завершення через join() і виводить остаточне значення. Очікуваний коректний результат: 0.

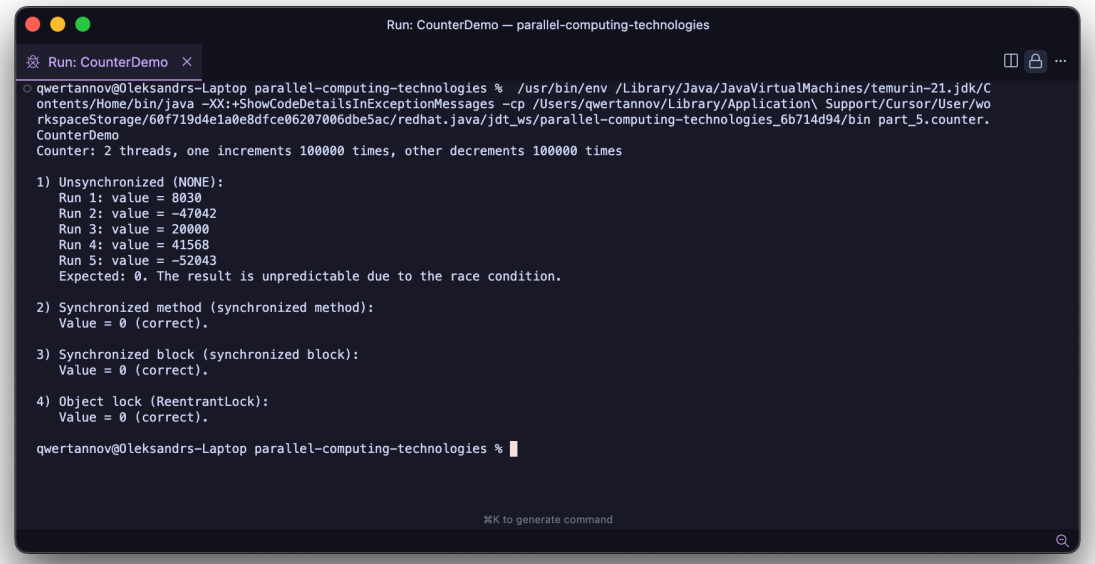

Рисунок 5.1 – Запуск експерименту із лічильником

При режимі Counter.SyncType.NONE операції value++ та value-- не є атомарними (читання – зміна – запис можуть переплітатися між потоками). Спостерігаються різні остаточні значення при повторних запусках. Це гонка потоків (race condition): кілька потоків одночасно змінюють спільний стан без координації, наслідок залежить від порядку виконання і є непередбачуваним.

Опис способів синхронізації:

1. Синхронізований метод. Методи increment() та decrement() викликають допоміжні incrementSyncMethod() та decrementSyncMethod(), позначені як synchronized. Монітор – об’єкт his (сам лічильник). Одночасно виконуватиметься лише один із цих методів для даного об’єкта, тому результат завжди 0.

2. Синхронізований блок. Усередині increment() / decrement() використовується блок synchronized (this) { value++; } або synchronized (this) { value--; }. Монітор також this. Ефект той самий, що й у синхронізованого методу, але критична ділянка обмежена одним виразом; решта коду (наприклад, перемикання за типом) може виконуватися поза блоком.

3. Блокування об’єкта. Використовується ReentrantLock: перед зміною значення викликається lock.lock(), після – lock.unlock() у блоці finally. Явне блокування дає гнучкість (наприклад, умовні блокування, спроба захоплення з таймаутом); потік, що володіє замком, може повторно його захопити (reentrant). Результат так само коректний – 0.

Усі три способи забезпечують взаємне виключення і правильний результат. Синхронізований метод – найпростіший у написанні, але блокує весь метод. Синхронізований блок дозволяє мінімізувати критичну ділянку і зменшити час утримання монітора. ReentrantLock дає більше можливостей (умовні черги, fair lock, tryLock) і може бути ефективнішим у складних сценаріях; необхідно не забувати викликати unlock() у finally.

Висновок

У межах комп’ютерного практикуму №1 отримано практичні навички багатопоточної розробки мовою Java та досліджено основні механізми роботи з потоками. Реалізовано п’ять частин завдань.

У частині 1 побудовано програму імітації руху більярдних кульок, де кожна кулька виконується в окремому потоці; перевірено вплив кількості потоків на навантаження CPU та FPS і сформульовано переваги потокової архітектури. У частині 2 додано лузи, завершення потоку при потраплянні кульки в лузу та динамічне відображення лічильника в інтерфейсі. У частині 3 досліджено пріоритет потоків: червоні та сині кульки створюються з різними пріоритетами; спостереження показали, що пріоритет дає лише ймовірнісну перевагу і при великій кількості потоків його вплив зменшується. У частині 4 проілюстровано метод join(): послідовний запуск потоків кульок (червона –> синя –> зелена) наочно демонструє очікування завершення одного потоку перед стартом наступного. У частині 5 створено клас Counter і продемонстровано гонку потоків при одночасному збільшенні та зменшенні лічильника двома потоками; реалізовано три способи синхронізації (синхронізований метод, синхронізований блок, блокування об’єкта ReentrantLock) і порівняно їх.

Підсумовуючи, виконані завдання охоплюють створення та запуск потоків, завершення потоку за подією, використання пріоритетів, координацію потоків за допомогою join() та забезпечення потокобезпеки спільного стану за допомогою синхронізації. Отримані навички є основою для подальшої розробки багатопоточних та розподілених програм.

Код програми

Посилання на GitHub репозиторій: https://github.com/sokolovgit/parallel-computing-technologies

Задача 1

```
===== Ball.java =====
package part_1.bounce;

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

public class Ball {

    private BallCanvas canvas;

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

    private int size;
    private Color baseColor;

    public Ball(BallCanvas c) {
        this.canvas = c;

        updateSize();

        Random rand = new Random();

        int w = Math.max(1, canvas.getWidth() - size);
        int h = Math.max(1, canvas.getHeight() - size);
        switch (rand.nextInt(4)) {
            case 0 -> {
                x = rand.nextInt(w);
                y = 0;
            }
            case 1 -> {
                x = rand.nextInt(w);
                y = h;
            }
            case 2 -> {
                x = 0;
                y = rand.nextInt(h);
            }
            default -> {
                x = w;
                y = rand.nextInt(h);
            }
        }

        baseColor = new Color(
                rand.nextInt(200) + 30,
                rand.nextInt(200) + 30,
                rand.nextInt(200) + 30);
```

```
    }

    private void updateSize() {
        int min = Math.min(canvas.getWidth(), canvas.getHeight());
        size = Math.max(Config.BALL_MIN_SIZE, min / 30);
    }

    public void draw(Graphics2D g2) {

        updateSize();

        g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
                RenderingHints.VALUE_ANTIALIAS_ON);

        g2.setColor(new Color(0, 0, 0, 60));
        g2.fillOval(x + 4, y + 4, size, size);

        GradientPaint gradient = new GradientPaint(
                x, y, baseColor.brighter(),
                x + size, y + size, baseColor.darker());

        g2.setPaint(gradient);
        g2.fill(new Ellipse2D.Double(x, y, size, size));

        g2.setColor(new Color(255, 255, 255, 120));
        g2.fillOval(x + size / 4, y + size / 4, size / 4, size / 4);
    }

    public void move() {

        updateSize();

        x += dx;
        y += dy;

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

        if (x + size >= canvas.getWidth()) {
            x = canvas.getWidth() - size;
            dx = -dx;
        }

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

        if (y + size >= canvas.getHeight()) {
            y = canvas.getHeight() - size;
            dy = -dy;
        }

        canvas.repaint();
    }
}
```

===== BallCanvas.java =====

```
package part_1.bounce;

import javax.swing.*;
import java.awt.*;
import java.lang.Thread.State;
import java.lang.management.*;
import java.util.*;
import java.util.List;

public class BallCanvas extends JPanel {

   private List<Ball> balls = new ArrayList<>();
   private List<BallThread> ballThreads = new ArrayList<>();

   private int sleepTime = Config.DEFAULT_SLEEP_MS;

   private long lastTime = System.nanoTime();
   private int frames = 0;
   private int fps = 0;

   private ThreadMXBean threadBean = ManagementFactory.getThreadMXBean();

   private com.sun.management.OperatingSystemMXBean osBean =
(com.sun.management.OperatingSystemMXBean) ManagementFactory
         .getOperatingSystemMXBean();

   private double processCpuLoad = 0;
   private int runnableThreads = 0;
   private int waitingThreads = 0;
   private long totalBallCpuTime = 0;

   private LinkedList<Integer> fpsHistory = new LinkedList<>();
   private LinkedList<Double> cpuHistory = new LinkedList<>();
   private final int GRAPH_POINTS = 200;

   public BallCanvas() {
      setDoubleBuffered(true);
      threadBean.setThreadCpuTimeEnabled(true);
   }

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

   public void registerBallThread(BallThread t) {
      ballThreads.add(t);
   }

   public void unregisterBallThread(BallThread t) {
      ballThreads.remove(t);
   }

   public int getSleepTime() {
      return sleepTime;
   }

   public void setSleepTime(int sleepTime) {
      this.sleepTime = sleepTime;
   }
```

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

   g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
         RenderingHints.VALUE_ANTIALIAS_ON);

   drawBackground(g2);

   for (Ball b : balls) {
      b.draw(g2);
   }

   updateFPS();
   updateSystemStats();
   updateThreadStats();
   updateCpuTime();
   updateGraph();

   drawOverlay(g2);
   drawGraph(g2);
}

private void drawBackground(Graphics2D g2) {
   GradientPaint table = new GradientPaint(
         0, 0, new Color(20, 90, 40),
         getWidth(), getHeight(), new Color(10, 50, 25));
   g2.setPaint(table);
   g2.fillRect(0, 0, getWidth(), getHeight());
}

private void updateFPS() {
   frames++;
   long current = System.nanoTime();

   if (current - lastTime >= 1_000_000_000) {
      fps = frames;
      frames = 0;
      lastTime = current;
   }
}

private void updateSystemStats() {
   processCpuLoad = osBean.getProcessCpuLoad() * 100.0;
}

private void updateThreadStats() {

   runnableThreads = 0;
   waitingThreads = 0;

   for (BallThread t : ballThreads) {
      State state = t.getState();

      switch (state) {
         case RUNNABLE -> runnableThreads++;
         case WAITING, TIMED_WAITING -> waitingThreads++;
         case BLOCKED, NEW, TERMINATED -> {
         }
```

```
        }
    }
}

private void updateCpuTime() {
    totalBallCpuTime = 0;

    for (BallThread t : ballThreads) {
        long id = t.threadId();
        totalBallCpuTime += threadBean.getThreadCpuTime(id);
    }
}

private void updateGraph() {
    fpsHistory.add(fps);
    cpuHistory.add(processCpuLoad);

    if (fpsHistory.size() > GRAPH_POINTS)
        fpsHistory.removeFirst();
    if (cpuHistory.size() > GRAPH_POINTS)
        cpuHistory.removeFirst();
}

private void drawOverlay(Graphics2D g2) {

    double frameTime = fps > 0 ? 1000.0 / fps : 0;

    g2.setColor(Color.WHITE);
    g2.setFont(new Font("Consolas", Font.BOLD, 14));

    int y = 20;

    g2.drawString("Balls: " + balls.size(), 15, y);
    y += 20;
    g2.drawString("Ball Threads: " + ballThreads.size(), 15, y);
    y += 20;
    g2.drawString("FPS: " + fps, 15, y);
    y += 20;
    g2.drawString(String.format("Frame Time: %.2f ms", frameTime), 15, y);
    y += 20;
    g2.drawString(String.format("CPU (JVM): %.2f %%", processCpuLoad), 15, y);
    y += 20;
    g2.drawString("RUNNABLE: " + runnableThreads, 15, y);
    y += 20;
    g2.drawString("WAITING: " + waitingThreads, 15, y);
    y += 20;
    g2.drawString("Ball CPU Time (ms): " + totalBallCpuTime / 1_000_000, 15, y);
}

private void drawGraph(Graphics2D g2) {

    int width = 300;
    int height = 150;

    int x = getWidth() - width - 20;
    int y = 20;

    g2.setColor(new Color(0, 0, 0, 120));
    g2.fillRect(x, y, width, height);
```

```
        g2.setColor(Color.GREEN);
        drawLineGraph(g2, fpsHistory, x, y, width, height, 100);

        g2.setColor(Color.RED);
        drawLineGraphDouble(g2, cpuHistory, x, y, width, height, 100);
    }

    private void drawLineGraph(Graphics2D g2, List<Integer> data,
            int x, int y, int width, int height, int maxValue) {

        if (data.size() < 2)
            return;

        int step = width / GRAPH_POINTS;
        int prevX = x;
        int prevY = y + height - (data.get(0) * height / maxValue);

        for (int i = 1; i < data.size(); i++) {
            int currentX = x + i * step;
            int currentY = y + height - (data.get(i) * height / maxValue);

            g2.drawLine(prevX, prevY, currentX, currentY);

            prevX = currentX;
            prevY = currentY;
        }
    }

    private void drawLineGraphDouble(Graphics2D g2, List<Double> data,
            int x, int y, int width, int height, int maxValue) {

        if (data.size() < 2)
            return;

        int step = width / GRAPH_POINTS;
        int prevX = x;
        int prevY = y + height - ((int) (data.get(0) * height / maxValue));

        for (int i = 1; i < data.size(); i++) {
            int currentX = x + i * step;
            int currentY = y + height - ((int) (data.get(i) * height / maxValue));

            g2.drawLine(prevX, prevY, currentX, currentY);

            prevX = currentX;
            prevY = currentY;
        }
    }
}

===== BallThread.java =====
package part_1.bounce;

public class BallThread extends Thread {

    private Ball ball;
    private BallCanvas canvas;

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

```
        this.canvas = canvas;
    }

    @Override
    public void run() {
        canvas.registerBallThread(this);
        try {
            for (int i = 0; i < Config.BALL_MOVES_COUNT; i++) {

                ball.move();

                Thread.sleep(canvas.getSleepTime());

            }
        } catch (InterruptedException er) {
            er.printStackTrace();
        }
        canvas.unregisterBallThread(this);
    }
}
```

===== Bounce.java =====

```
package part_1.bounce;

import javax.swing.*;

public class Bounce {

    public static void main(String[] args) {
        BounceFrame frame = new BounceFrame();
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);

        System.out.println("Main Thread: " + Thread.currentThread().getName());
    }

}
```

===== BounceFrame.java =====

```
package part_1.bounce;

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

public class BounceFrame extends JFrame {

    private BallCanvas canvas;
    private JLabel warningLabel;

    public BounceFrame() {

        try {
            UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
        } catch (Exception ignored) {
        }

        setSize(Config.FRAME_WIDTH, Config.FRAME_HEIGHT);
        setTitle("Multithreaded Billiard Simulation");
        setLocationRelativeTo(null);
```

```
        canvas = new BallCanvas();
        add(canvas, BorderLayout.CENTER);

        JPanel controlPanel = new JPanel(new BorderLayout());
        controlPanel.setBackground(new Color(30, 30, 30));
        controlPanel.setBorder(BorderFactory.createEmptyBorder(10, 20, 10, 20));

        JPanel leftPanel = new JPanel(new FlowLayout(FlowLayout.LEFT, 15, 5));
        leftPanel.setOpaque(false);

        JButton addOne = createButton("Add Ball");
        JButton addTen = createButton("Add 10");
        JButton exit = createButton("Exit");

        addOne.addActionListener(e -> addBall());
        addTen.addActionListener(e -> {
            for (int i = 0; i < 10; i++)
                addBall();
        });
        exit.addActionListener(e -> System.exit(0));

        leftPanel.add(addOne);
        leftPanel.add(addTen);
        leftPanel.add(exit);

        warningLabel = new JLabel("");
        warningLabel.setForeground(Color.RED);

        controlPanel.add(leftPanel, BorderLayout.WEST);
        controlPanel.add(warningLabel, BorderLayout.EAST);

        add(controlPanel, BorderLayout.SOUTH);
    }

    private JButton createButton(String text) {
        JButton button = new JButton(text);
        button.setFocusPainted(false);
        button.setFont(new Font("Segoe UI", Font.BOLD, 14));
        button.setForeground(Color.WHITE);
        button.setBackground(new Color(60, 120, 200));
        button.setCursor(new Cursor(Cursor.HAND_CURSOR));
        button.setBorder(BorderFactory.createEmptyBorder(8, 18, 8, 18));

        button.addMouseListener(new java.awt.event.MouseAdapter() {
            public void mouseEntered(java.awt.event.MouseEvent evt) {
                button.setBackground(new Color(80, 150, 240));
            }

            public void mouseExited(java.awt.event.MouseEvent evt) {
                button.setBackground(new Color(60, 120, 200));
            }
        });

        return button;
    }

    private void addBall() {
        Ball ball = new Ball(canvas);
        canvas.add(ball);
```

```
        BallThread thread = new BallThread(ball, canvas);
        thread.start();

        if (Thread.activeCount() > Config.WARNING_THREAD_COUNT) {
            warningLabel.setText("! High thread count!");
        } else {
            warningLabel.setText("");
        }
    }
}
```

===== Config.java =====

```
package part_1.bounce;

public class Config {

    public static final int FRAME_WIDTH = 1000;
    public static final int FRAME_HEIGHT = 600;
    public static final int BALL_MIN_SIZE = 30;

    public static final int BALL_MOVES_COUNT = 1_000_000;

    public static final int DEFAULT_SLEEP_MS = 5;

    public static final int WARNING_THREAD_COUNT = 100;
}
```

Задача 2

===== Ball.java =====

```
package part_2.bounce;

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

public class Ball {

   private BallCanvas canvas;

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

   private int size;
   private Color baseColor;

   public Ball(BallCanvas c) {
      this.canvas = c;

      updateSize();

      Random rand = new Random();

      int w = Math.max(1, canvas.getWidth() - size);
      int h = Math.max(1, canvas.getHeight() - size);
      switch (rand.nextInt(4)) {
         case 0 -> {
            x = rand.nextInt(w);
            y = 0;
         }
         case 1 -> {
            x = rand.nextInt(w);
            y = h;
         }
         case 2 -> {
            x = 0;
            y = rand.nextInt(h);
         }
         default -> {
            x = w;
            y = rand.nextInt(h);
         }
      }

      baseColor = new Color(
            rand.nextInt(200) + 30,
            rand.nextInt(200) + 30,
            rand.nextInt(200) + 30);
   }

   private void updateSize() {
      int min = Math.min(canvas.getWidth(), canvas.getHeight());
      size = Math.max(Config.BALL_MIN_SIZE, min / 30);
   }
```

```
public void draw(Graphics2D g2) {

    updateSize();

    g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
            RenderingHints.VALUE_ANTIALIAS_ON);

    g2.setColor(new Color(0, 0, 0, 60));
    g2.fillOval(x + 4, y + 4, size, size);

    GradientPaint gradient = new GradientPaint(
            x, y, baseColor.brighter(),
            x + size, y + size, baseColor.darker());

    g2.setPaint(gradient);
    g2.fill(new Ellipse2D.Double(x, y, size, size));

    g2.setColor(new Color(255, 255, 255, 120));
    g2.fillOval(x + size / 4, y + size / 4, size / 4, size / 4);
}

public void move() {

    updateSize();

    x += dx;
    y += dy;

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

    if (x + size >= canvas.getWidth()) {
        x = canvas.getWidth() - size;
        dx = -dx;
    }

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

    if (y + size >= canvas.getHeight()) {
        y = canvas.getHeight() - size;
        dy = -dy;
    }

    canvas.repaint();
}

public int getBallX() {
    return x;
}

public int getBallY() {
    return y;
}

public int getBallSize() {
```

```
        return size;
    }
}

===== BallCanvas.java =====
package part_2.bounce;

import javax.swing.*;
import java.awt.*;
import java.lang.Thread.State;
import java.lang.management.*;
import java.util.*;
import java.util.List;

public class BallCanvas extends JPanel {

    private final List<Ball> balls = new ArrayList<>();
    private final List<BallThread> ballThreads = new ArrayList<>();

    private int pocketedCount = 0;
    private JLabel pocketedLabel;

    private int sleepTime = Config.DEFAULT_SLEEP_MS;

    private long lastTime = System.nanoTime();
    private int frames = 0;
    private int fps = 0;

    private ThreadMXBean threadBean = ManagementFactory.getThreadMXBean();

    private com.sun.management.OperatingSystemMXBean osBean =
(com.sun.management.OperatingSystemMXBean) ManagementFactory
            .getOperatingSystemMXBean();

    private double processCpuLoad = 0;
    private int runnableThreads = 0;
    private int waitingThreads = 0;
    private long totalBallCpuTime = 0;

    private LinkedList<Integer> fpsHistory = new LinkedList<>();
    private LinkedList<Double> cpuHistory = new LinkedList<>();
    private final int GRAPH_POINTS = 200;

    public BallCanvas() {
        setDoubleBuffered(true);
        threadBean.setThreadCpuTimeEnabled(true);
    }

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

    public void setPocketedLabel(JLabel label) {
        this.pocketedLabel = label;
    }

    public int getPocketedCount() {
        return pocketedCount;
```

```
}

private List<Point> getPocketCenters() {
    int w = getWidth();
    int h = getHeight();
    int in = Config.POCKET_INSET;
    return List.of(
            new Point(in, in),
            new Point(w - in, in),
            new Point(in, h - in),
            new Point(w - in, h - in),
            new Point(w / 2, in),
            new Point(w / 2, h - in));
}

public boolean isBallInPocket(Ball ball) {
    int cx = ball.getBallX() + ball.getBallSize() / 2;
    int cy = ball.getBallY() + ball.getBallSize() / 2;
    int r = Config.POCKET_RADIUS;
    for (Point p : getPocketCenters()) {
        if (Math.hypot(cx - p.x, cy - p.y) <= r)
            return true;
    }
    return false;
}

public void pocketBall(Ball ball) {
    synchronized (balls) {
        if (balls.remove(ball)) {
            pocketedCount++;

            if (pocketedLabel != null) {
                int count = pocketedCount;
                SwingUtilities.invokeLater(() -> pocketedLabel.setText("В лузі: " + count));
            }

            SwingUtilities.invokeLater(this::repaint);
        }
    }
}

public void registerBallThread(BallThread t) {
    ballThreads.add(t);
}

public void unregisterBallThread(BallThread t) {
    ballThreads.remove(t);
}

public int getSleepTime() {
    return sleepTime;
}

public void setSleepTime(int sleepTime) {
    this.sleepTime = sleepTime;
}

@Override
protected void paintComponent(Graphics g) {
    super.paintComponent(g);
```

```
    Graphics2D g2 = (Graphics2D) g;

    g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
            RenderingHints.VALUE_ANTIALIAS_ON);

    drawBackground(g2);

    List<Ball> copy;
    synchronized (balls) {
        copy = new ArrayList<>(balls);
    }
    for (Ball b : copy) {
        b.draw(g2);
    }

    updateFPS();
    updateSystemStats();
    updateThreadStats();
    updateCpuTime();
    updateGraph();

    drawOverlay(g2);
    drawGraph(g2);
}

private void drawBackground(Graphics2D g2) {
    int w = getWidth();
    int h = getHeight();
    GradientPaint table = new GradientPaint(
            0, 0, new Color(20, 90, 40),
            w, h, new Color(10, 50, 25));
    g2.setPaint(table);
    g2.fillRect(0, 0, w, h);
    drawPockets(g2);
}

private void drawPockets(Graphics2D g2) {
    int r = Config.POCKET_RADIUS;
    g2.setColor(new Color(15, 15, 15));
    for (Point p : getPocketCenters()) {
        g2.fillOval(p.x - r, p.y - r, 2 * r, 2 * r);
    }
}

private void updateFPS() {
    frames++;
    long current = System.nanoTime();

    if (current - lastTime >= 1_000_000_000) {
        fps = frames;
        frames = 0;
        lastTime = current;
    }
}

private void updateSystemStats() {
    processCpuLoad = osBean.getProcessCpuLoad() * 100.0;
}

private void updateThreadStats() {
```

```
    runnableThreads = 0;
    waitingThreads = 0;

    for (BallThread t : ballThreads) {
        State state = t.getState();

        switch (state) {
            case RUNNABLE -> runnableThreads++;
            case WAITING, TIMED_WAITING -> waitingThreads++;
            case BLOCKED, NEW, TERMINATED -> {
            }
        }
    }
}

private void updateCpuTime() {
    totalBallCpuTime = 0;

    for (BallThread t : ballThreads) {
        long id = t.threadId();
        totalBallCpuTime += threadBean.getThreadCpuTime(id);
    }
}

private void updateGraph() {
    fpsHistory.add(fps);
    cpuHistory.add(processCpuLoad);

    if (fpsHistory.size() > GRAPH_POINTS)
        fpsHistory.removeFirst();
    if (cpuHistory.size() > GRAPH_POINTS)
        cpuHistory.removeFirst();
}

private void drawOverlay(Graphics2D g2) {

    double frameTime = fps > 0 ? 1000.0 / fps : 0;

    g2.setColor(Color.WHITE);
    g2.setFont(new Font("Consolas", Font.BOLD, 14));

    int y = 20;

    g2.drawString("Balls on table: " + balls.size(), 15, y);
    y += 20;
    g2.drawString("Balls In Pocket: " + pocketedCount, 15, y);
    y += 20;
    g2.drawString("Ball Threads: " + ballThreads.size(), 15, y);
    y += 20;
    g2.drawString("FPS: " + fps, 15, y);
    y += 20;
    g2.drawString(String.format("Frame Time: %.2f ms", frameTime), 15, y);
    y += 20;
    g2.drawString(String.format("CPU (JVM): %.2f %%", processCpuLoad), 15, y);
    y += 20;
    g2.drawString("RUNNABLE: " + runnableThreads, 15, y);
    y += 20;
    g2.drawString("WAITING: " + waitingThreads, 15, y);
    y += 20;
```

```
    g2.drawString("Ball CPU Time (ms): " + totalBallCpuTime / 1_000_000, 15, y);
}

private void drawGraph(Graphics2D g2) {

    int width = 300;
    int height = 150;

    int x = getWidth() - width - 20;
    int y = 20;

    g2.setColor(new Color(0, 0, 0, 120));
    g2.fillRect(x, y, width, height);

    g2.setColor(Color.GREEN);
    drawLineGraph(g2, fpsHistory, x, y, width, height, 100);

    g2.setColor(Color.RED);
    drawLineGraphDouble(g2, cpuHistory, x, y, width, height, 100);
}

private void drawLineGraph(Graphics2D g2, List<Integer> data,
        int x, int y, int width, int height, int maxValue) {

    if (data.size() < 2)
        return;

    int step = width / GRAPH_POINTS;
    int prevX = x;
    int prevY = y + height - (data.get(0) * height / maxValue);

    for (int i = 1; i < data.size(); i++) {
        int currentX = x + i * step;
        int currentY = y + height - (data.get(i) * height / maxValue);

        g2.drawLine(prevX, prevY, currentX, currentY);

        prevX = currentX;
        prevY = currentY;
    }
}

private void drawLineGraphDouble(Graphics2D g2, List<Double> data,
        int x, int y, int width, int height, int maxValue) {

    if (data.size() < 2)
        return;

    int step = width / GRAPH_POINTS;
    int prevX = x;
    int prevY = y + height - ((int) (data.get(0) * height / maxValue));

    for (int i = 1; i < data.size(); i++) {
        int currentX = x + i * step;
        int currentY = y + height - ((int) (data.get(i) * height / maxValue));

        g2.drawLine(prevX, prevY, currentX, currentY);

        prevX = currentX;
        prevY = currentY;
```

```
        }
    }
}

===== BallThread.java =====
package part_2.bounce;

public class BallThread extends Thread {

    private Ball ball;
    private BallCanvas canvas;

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

    @Override
    public void run() {
        canvas.registerBallThread(this);
        try {
            for (int i = 0; i < Config.BALL_MOVES_COUNT; i++) {
                ball.move();

                if (canvas.isBallInPocket(ball)) {
                    canvas.pocketBall(ball);
                    break;
                }

                Thread.sleep(canvas.getSleepTime());
            }
        } catch (InterruptedException er) {
            er.printStackTrace();
        }
        canvas.unregisterBallThread(this);
    }
}

===== Bounce.java =====
package part_2.bounce;

import javax.swing.*;

public class Bounce {

    public static void main(String[] args) {
        BounceFrame frame = new BounceFrame();
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);

        System.out.println("Main Thread: " + Thread.currentThread().getName());
    }

}

===== BounceFrame.java =====
package part_2.bounce;

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

```
public class BounceFrame extends JFrame {

    private BallCanvas canvas;
    private JLabel warningLabel;
    private JLabel pocketedLabel;

    public BounceFrame() {

        try {
            UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
        } catch (Exception ignored) {
        }

        setSize(Config.FRAME_WIDTH, Config.FRAME_HEIGHT);
        setTitle("Multithreaded Billiard Simulation With Pockets");
        setLocationRelativeTo(null);

        canvas = new BallCanvas();
        add(canvas, BorderLayout.CENTER);

        pocketedLabel = new JLabel("В лузі: 0");
        pocketedLabel.setFont(new Font("Segoe UI", Font.BOLD, 14));
        pocketedLabel.setForeground(Color.WHITE);
        canvas.setPocketedLabel(pocketedLabel);

        JPanel controlPanel = new JPanel(new BorderLayout());
        controlPanel.setBackground(new Color(30, 30, 30));
        controlPanel.setBorder(BorderFactory.createEmptyBorder(10, 20, 10, 20));

        JPanel leftPanel = new JPanel(new FlowLayout(FlowLayout.LEFT, 15, 5));
        leftPanel.setOpaque(false);

        JButton addOne = createButton("Add Ball");
        JButton addTen = createButton("Add 10");
        JButton exit = createButton("Exit");

        addOne.addActionListener(e -> addBall());
        addTen.addActionListener(e -> {
            for (int i = 0; i < 10; i++)
                addBall();
        });
        exit.addActionListener(e -> System.exit(0));

        leftPanel.add(addOne);
        leftPanel.add(addTen);
        leftPanel.add(exit);

        warningLabel = new JLabel("");
        warningLabel.setForeground(Color.RED);

        JPanel rightPanel = new JPanel(new FlowLayout(FlowLayout.RIGHT, 15, 5));
        rightPanel.setOpaque(false);
        rightPanel.add(pocketedLabel);
        rightPanel.add(warningLabel);

        controlPanel.add(leftPanel, BorderLayout.WEST);
        controlPanel.add(rightPanel, BorderLayout.EAST);

        add(controlPanel, BorderLayout.SOUTH);
```

```
    }

    private JButton createButton(String text) {
        JButton button = new JButton(text);
        button.setFocusPainted(false);
        button.setFont(new Font("Segoe UI", Font.BOLD, 14));
        button.setForeground(Color.WHITE);
        button.setBackground(new Color(60, 120, 200));
        button.setCursor(new Cursor(Cursor.HAND_CURSOR));
        button.setBorder(BorderFactory.createEmptyBorder(8, 18, 8, 18));

        button.addMouseListener(new java.awt.event.MouseAdapter() {
            public void mouseEntered(java.awt.event.MouseEvent evt) {
                button.setBackground(new Color(80, 150, 240));
            }

            public void mouseExited(java.awt.event.MouseEvent evt) {
                button.setBackground(new Color(60, 120, 200));
            }
        });

        return button;
    }

    private void addBall() {
        Ball ball = new Ball(canvas);
        canvas.add(ball);

        BallThread thread = new BallThread(ball, canvas);
        thread.start();

        if (Thread.activeCount() > Config.WARNING_THREAD_COUNT) {
            warningLabel.setText("! High thread count!");
        } else {
            warningLabel.setText("");
        }
    }
}

===== Config.java =====
package part_2.bounce;

public class Config {

    public static final int FRAME_WIDTH = 1000;
    public static final int FRAME_HEIGHT = 600;
    public static final int BALL_MIN_SIZE = 30;

    public static final int BALL_MOVES_COUNT = 1_000_000;

    public static final int DEFAULT_SLEEP_MS = 5;

    public static final int WARNING_THREAD_COUNT = 100;

    public static final int POCKET_INSET = 28;

    public static final int POCKET_RADIUS = 35;
}
```

Задача 3

===== Ball.java =====

```
package part_3.bounce;

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

public class Ball {

    public enum BallType {
        RED(Config.THREAD_PRIORITY_HIGH),
        BLUE(Config.THREAD_PRIORITY_LOW);

        private final int threadPriority;

        BallType(int threadPriority) {
            this.threadPriority = threadPriority;
        }

        public int getThreadPriority() {
            return threadPriority;
        }
    }

    private final BallCanvas canvas;

    private int x;
    private int y;
    private int dx;
    private int dy;

    private int size;
    private final Color baseColor;
    private final BallType type;

    public Ball(BallCanvas c, BallType type) {
        this.canvas = c;
        this.type = type;
        this.dx = 2;
        this.dy = 2;

        updateSize();

        Random rand = new Random();
        int w = Math.max(1, canvas.getWidth() - size);
        int h = Math.max(1, canvas.getHeight() - size);
        switch (rand.nextInt(4)) {
            case 0 -> {
                x = rand.nextInt(w);
                y = 0;
            }
            case 1 -> {
                x = rand.nextInt(w);
                y = h;
            }
            case 2 -> {
                x = 0;
                y = rand.nextInt(h);
```

```
            }
            default -> {
                x = w;
                y = rand.nextInt(h);
            }
        }

        baseColor = type == BallType.RED ? new Color(220, 50, 50) : new Color(50, 80, 200);
    }

    public Ball(BallCanvas c, int startX, int startY, int dx, int dy, BallType type) {
        this.canvas = c;
        this.type = type;
        this.x = startX;
        this.y = startY;
        this.dx = dx;
        this.dy = dy;

        updateSize();
        baseColor = type == BallType.RED ? new Color(220, 50, 50) : new Color(50, 80, 200);
    }

    private void updateSize() {
        int min = Math.min(canvas.getWidth(), canvas.getHeight());
        size = Math.max(Config.BALL_MIN_SIZE, min / 30);
    }

    public BallType getType() {
        return type;
    }

    public void draw(Graphics2D g2) {

        updateSize();

        g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
                RenderingHints.VALUE_ANTIALIAS_ON);

        g2.setColor(new Color(0, 0, 0, 60));
        g2.fillOval(x + 4, y + 4, size, size);

        GradientPaint gradient = new GradientPaint(
                x, y, baseColor.brighter(),
                x + size, y + size, baseColor.darker());

        g2.setPaint(gradient);
        g2.fill(new Ellipse2D.Double(x, y, size, size));

        g2.setColor(new Color(255, 255, 255, 120));
        g2.fillOval(x + size / 4, y + size / 4, size / 4, size / 4);
    }

    public void move() {

        updateSize();

        x += dx;
        y += dy;

        if (x < 0) {
```

```
            x = 0;
            dx = -dx;
        }

        if (x + size >= canvas.getWidth()) {
            x = canvas.getWidth() - size;
            dx = -dx;
        }

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

        if (y + size >= canvas.getHeight()) {
            y = canvas.getHeight() - size;
            dy = -dy;
        }

        canvas.repaint();
    }

    public int getBallX() {
        return x;
    }

    public int getBallY() {
        return y;
    }

    public int getBallSize() {
        return size;
    }
}
```

===== BallCanvas.java =====

```
package part_3.bounce;

import javax.swing.*;
import java.awt.*;
import java.lang.Thread.State;
import java.lang.management.*;
import java.util.*;
import java.util.List;

public class BallCanvas extends JPanel {

    private final List<Ball> balls = new ArrayList<>();
    private final List<BallThread> ballThreads = new ArrayList<>();

    private int sleepTime = Config.DEFAULT_SLEEP_MS;

    private long lastTime = System.nanoTime();
    private int frames = 0;
    private int fps = 0;

    private ThreadMXBean threadBean = ManagementFactory.getThreadMXBean();

    private com.sun.management.OperatingSystemMXBean osBean =
(com.sun.management.OperatingSystemMXBean) ManagementFactory
```

```
        .getOperatingSystemMXBean();

private double processCpuLoad = 0;
private int runnableThreads = 0;
private int waitingThreads = 0;
private long totalBallCpuTime = 0;

private LinkedList<Integer> fpsHistory = new LinkedList<>();
private LinkedList<Double> cpuHistory = new LinkedList<>();
private final int GRAPH_POINTS = 200;

public BallCanvas() {
    setDoubleBuffered(true);
    threadBean.setThreadCpuTimeEnabled(true);
}

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

public void registerBallThread(BallThread t) {
    ballThreads.add(t);
}

public void unregisterBallThread(BallThread t) {
    ballThreads.remove(t);
}

public int getSleepTime() {
    return sleepTime;
}

public void setSleepTime(int sleepTime) {
    this.sleepTime = sleepTime;
}

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

    g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
            RenderingHints.VALUE_ANTIALIAS_ON);

    drawBackground(g2);

    List<Ball> copy;
    synchronized (balls) {
        copy = new ArrayList<>(balls);
    }

    for (Ball b : copy) {
        if (b.getType() == Ball.BallType.BLUE)
            b.draw(g2);
    }
    for (Ball b : copy) {
        if (b.getType() == Ball.BallType.RED)
            b.draw(g2);
```

```
    }

    updateFPS();
    updateSystemStats();
    updateThreadStats();
    updateCpuTime();
    updateGraph();

    drawOverlay(g2);
    drawGraph(g2);
}

private void drawBackground(Graphics2D g2) {
    int w = getWidth();
    int h = getHeight();
    GradientPaint table = new GradientPaint(
            0, 0, new Color(20, 90, 40),
            w, h, new Color(10, 50, 25));
    g2.setPaint(table);
    g2.fillRect(0, 0, w, h);
}

private void updateFPS() {
    frames++;
    long current = System.nanoTime();

    if (current - lastTime >= 1_000_000_000) {
        fps = frames;
        frames = 0;
        lastTime = current;
    }
}

private void updateSystemStats() {
    processCpuLoad = osBean.getProcessCpuLoad() * 100.0;
}

private void updateThreadStats() {

    runnableThreads = 0;
    waitingThreads = 0;

    for (BallThread t : ballThreads) {
        State state = t.getState();

        switch (state) {
            case RUNNABLE -> runnableThreads++;
            case WAITING, TIMED_WAITING -> waitingThreads++;
            case BLOCKED, NEW, TERMINATED -> {
            }
        }
    }
}

private void updateCpuTime() {
    totalBallCpuTime = 0;

    for (BallThread t : ballThreads) {
        long id = t.threadId();
        totalBallCpuTime += threadBean.getThreadCpuTime(id);
```

```
    }
}

private void updateGraph() {
    fpsHistory.add(fps);
    cpuHistory.add(processCpuLoad);

    if (fpsHistory.size() > GRAPH_POINTS)
        fpsHistory.removeFirst();
    if (cpuHistory.size() > GRAPH_POINTS)
        cpuHistory.removeFirst();
}

private void drawOverlay(Graphics2D g2) {

    double frameTime = fps > 0 ? 1000.0 / fps : 0;

    g2.setColor(Color.WHITE);
    g2.setFont(new Font("Consolas", Font.BOLD, 14));

    int y = 20;

    g2.drawString("Balls: " + balls.size(), 15, y);
    y += 20;
    g2.drawString("Ball Threads: " + ballThreads.size(), 15, y);
    y += 20;
    g2.drawString("FPS: " + fps, 15, y);
    y += 20;
    g2.drawString(String.format("Frame Time: %.2f ms", frameTime), 15, y);
    y += 20;
    g2.drawString(String.format("CPU (JVM): %.2f %%", processCpuLoad), 15, y);
    y += 20;
    g2.drawString("RUNNABLE: " + runnableThreads, 15, y);
    y += 20;
    g2.drawString("WAITING: " + waitingThreads, 15, y);
    y += 20;
    g2.drawString("Ball CPU Time (ms): " + totalBallCpuTime / 1_000_000, 15, y);
}

private void drawGraph(Graphics2D g2) {

    int width = 300;
    int height = 150;

    int x = getWidth() - width - 20;
    int y = 20;

    g2.setColor(new Color(0, 0, 0, 120));
    g2.fillRect(x, y, width, height);

    g2.setColor(Color.GREEN);
    drawLineGraph(g2, fpsHistory, x, y, width, height, 100);

    g2.setColor(Color.RED);
    drawLineGraphDouble(g2, cpuHistory, x, y, width, height, 100);
}

private void drawLineGraph(Graphics2D g2, List<Integer> data,
        int x, int y, int width, int height, int maxValue) {
```

```
        if (data.size() < 2)
            return;

        int step = width / GRAPH_POINTS;
        int prevX = x;
        int prevY = y + height - (data.get(0) * height / maxValue);

        for (int i = 1; i < data.size(); i++) {
            int currentX = x + i * step;
            int currentY = y + height - (data.get(i) * height / maxValue);

            g2.drawLine(prevX, prevY, currentX, currentY);

            prevX = currentX;
            prevY = currentY;
        }
    }

    private void drawLineGraphDouble(Graphics2D g2, List<Double> data,
            int x, int y, int width, int height, int maxValue) {

        if (data.size() < 2)
            return;

        int step = width / GRAPH_POINTS;
        int prevX = x;
        int prevY = y + height - ((int) (data.get(0) * height / maxValue));

        for (int i = 1; i < data.size(); i++) {
            int currentX = x + i * step;
            int currentY = y + height - ((int) (data.get(i) * height / maxValue));

            g2.drawLine(prevX, prevY, currentX, currentY);

            prevX = currentX;
            prevY = currentY;
        }
    }
}
```

===== BallThread.java =====

```
package part_3.bounce;

public class BallThread extends Thread {

    private final Ball ball;
    private final BallCanvas canvas;

    public BallThread(Ball ball, BallCanvas canvas, int priority) {
        this.ball = ball;
        this.canvas = canvas;
        setPriority(priority);
    }

    @Override
    public void run() {
        canvas.registerBallThread(this);
        try {
            for (int i = 0; i < Config.BALL_MOVES_COUNT; i++) {
                ball.move();
```

```
            Thread.sleep(canvas.getSleepTime());
          }
       } catch (InterruptedException er) {
          er.printStackTrace();
       }
       canvas.unregisterBallThread(this);
    }
}

===== Bounce.java =====
package part_3.bounce;

import javax.swing.*;

public class Bounce {

   public static void main(String[] args) {
      BounceFrame frame = new BounceFrame();
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      frame.setVisible(true);

      System.out.println("Main Thread: " + Thread.currentThread().getName());
   }

}

===== BounceFrame.java =====
package part_3.bounce;

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

public class BounceFrame extends JFrame {

   private BallCanvas canvas;
   private JLabel warningLabel;
   private JSpinner experimentBlueCount;

   public BounceFrame() {

      try {
         UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
      } catch (Exception ignored) {
      }

      setSize(Config.FRAME_WIDTH, Config.FRAME_HEIGHT);
      setTitle("Thread Priority: Red (high) vs Blue (low)");
      setLocationRelativeTo(null);

      canvas = new BallCanvas();
      add(canvas, BorderLayout.CENTER);

      JPanel controlPanel = new JPanel(new BorderLayout());
      controlPanel.setBackground(new Color(30, 30, 30));
      controlPanel.setBorder(BorderFactory.createEmptyBorder(10, 20, 10, 20));

      JPanel leftPanel = new JPanel(new FlowLayout(FlowLayout.LEFT, 15, 5));
      leftPanel.setOpaque(false);

      JButton addRed = createButton("Add Red", new Color(180, 60, 60));
```

```
        JButton addBlue = createButton("Add Blue", new Color(60, 90, 180));
        JButton add10Blue = createButton("Add 10 Blue", new Color(60, 90, 180));
        JButton experimentBtn = createButton("Experiment: 1 Red + N Blue", new Color(100, 100,
50));
        JButton exit = createButton("Exit", new Color(80, 80, 80));

        addRed.addActionListener(e -> addBall(Ball.BallType.RED));
        addBlue.addActionListener(e -> addBall(Ball.BallType.BLUE));
        add10Blue.addActionListener(e -> {
            for (int i = 0; i < 10; i++)
                addBall(Ball.BallType.BLUE);
        });
        experimentBtn.addActionListener(e -> runExperiment());
        exit.addActionListener(e -> System.exit(0));

        experimentBlueCount = new JSpinner(new SpinnerNumberModel(15, 5, 500, 5));
        experimentBlueCount.setMaximumSize(new Dimension(70, 30));
        JLabel expLabel = new JLabel("N =");
        expLabel.setForeground(Color.WHITE);

        leftPanel.add(addRed);
        leftPanel.add(addBlue);
        leftPanel.add(add10Blue);
        leftPanel.add(expLabel);
        leftPanel.add(experimentBlueCount);

        leftPanel.add(experimentBtn);
        leftPanel.add(exit);

        warningLabel = new JLabel("");
        warningLabel.setForeground(Color.RED);

        JPanel rightPanel = new JPanel(new FlowLayout(FlowLayout.RIGHT, 15, 5));
        rightPanel.setOpaque(false);
        rightPanel.add(warningLabel);

        controlPanel.add(leftPanel, BorderLayout.WEST);
        controlPanel.add(rightPanel, BorderLayout.EAST);

        add(controlPanel, BorderLayout.SOUTH);
    }

    private JButton createButton(String text, Color bg) {
        JButton button = new JButton(text);
        button.setFocusPainted(false);
        button.setFont(new Font("Segoe UI", Font.BOLD, 12));
        button.setForeground(Color.WHITE);
        button.setBackground(bg);
        button.setCursor(new Cursor(Cursor.HAND_CURSOR));
        button.setBorder(BorderFactory.createEmptyBorder(8, 12, 8, 12));

        button.addMouseListener(new java.awt.event.MouseAdapter() {
            public void mouseEntered(java.awt.event.MouseEvent evt) {
                button.setBackground(brighter(bg));
            }

            public void mouseExited(java.awt.event.MouseEvent evt) {
                button.setBackground(bg);
            }
        });
```

```
    return button;
}

private static Color brighter(Color c) {
    return new Color(
            Math.min(255, c.getRed() + 40),
            Math.min(255, c.getGreen() + 40),
            Math.min(255, c.getBlue() + 40));
}

private void addBall(Ball.BallType type) {
    Ball ball = new Ball(canvas, type);
    canvas.add(ball);
    startBallThread(ball);
    updateWarning();
}

private void runExperiment() {
    int n = 20;
    try {
        Object v = experimentBlueCount.getValue();
        if (v instanceof Number)
            n = ((Number) v).intValue();
    } catch (Exception ignored) {
    }
    n = Math.max(1, Math.min(500, n));

    int w = canvas.getWidth();
    int h = canvas.getHeight();
    if (w <= 0)
        w = Config.FRAME_WIDTH;
    if (h <= 0)
        h = Config.FRAME_HEIGHT - 80;
    int size = Math.max(Config.BALL_MIN_SIZE, Math.min(w, h) / 30);
    int startX = 80;
    int startY = Math.max(0, h / 2 - size / 2);
    int dx = 2;
    int dy = 0;

    Ball redBall = new Ball(canvas, startX, startY, dx, dy, Ball.BallType.RED);
    canvas.add(redBall);

    java.util.List<Ball> blueBalls = new java.util.ArrayList<>();
    for (int i = 0; i < n; i++) {
        Ball blueBall = new Ball(canvas, startX, startY, dx, dy, Ball.BallType.BLUE);
        canvas.add(blueBall);
        blueBalls.add(blueBall);
    }

    startBallThread(redBall);
    for (Ball b : blueBalls)
        startBallThread(b);

    updateWarning();
}

private void startBallThread(Ball ball) {
    BallThread thread = new BallThread(ball, canvas, ball.getType().getThreadPriority());
    thread.start();
```

```
    }

    private void updateWarning() {
        if (Thread.activeCount() > Config.WARNING_THREAD_COUNT) {
            warningLabel.setText("! High thread count!");
        } else {
            warningLabel.setText("");
        }
    }
}
```

===== Config.java =====

```
package part_3.bounce;

public class Config {

    public static final int FRAME_WIDTH = 1000;
    public static final int FRAME_HEIGHT = 600;
    public static final int BALL_MIN_SIZE = 30;

    public static final int BALL_MOVES_COUNT = 1_000_000;

    public static final int DEFAULT_SLEEP_MS = 5;

    public static final int WARNING_THREAD_COUNT = 100;

    public static final int THREAD_PRIORITY_HIGH = Thread.MAX_PRIORITY;
    public static final int THREAD_PRIORITY_LOW = Thread.MIN_PRIORITY;
}
```

Задача 4

===== Ball.java =====

```
package part_4.bounce;

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

public class Ball {

   public enum BallType {
      RED,
      BLUE,
      GREEN
   }

   private final BallCanvas canvas;
   private int x, y, dx = 2, dy = 2;
   private int size;
   private final Color baseColor;
   private final BallType type;

   public Ball(BallCanvas c, BallType type) {
      this.canvas = c;
      this.type = type;
      updateSize();
      Random rand = new Random();
      int w = Math.max(1, canvas.getWidth() - size);
      int h = Math.max(1, canvas.getHeight() - size);
      switch (rand.nextInt(4)) {
         case 0 -> { x = rand.nextInt(w); y = 0; }
         case 1 -> { x = rand.nextInt(w); y = h; }
         case 2 -> { x = 0; y = rand.nextInt(h); }
         default -> { x = w; y = rand.nextInt(h); }
      }
      baseColor = switch (type) {
         case RED -> new Color(220, 50, 50);
         case BLUE -> new Color(50, 80, 200);
         case GREEN -> new Color(50, 180, 80);
      };
   }

   private void updateSize() {
      int min = Math.min(canvas.getWidth(), canvas.getHeight());
      size = Math.max(Config.BALL_MIN_SIZE, min / 30);
   }

   public BallType getType() { return type; }

   public void draw(Graphics2D g2) {
      updateSize();
      g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
RenderingHints.VALUE_ANTIALIAS_ON);
      g2.setColor(new Color(0, 0, 0, 60));
      g2.fillOval(x + 4, y + 4, size, size);
      GradientPaint gradient = new GradientPaint(x, y, baseColor.brighter(), x + size, y + size,
baseColor.darker());
      g2.setPaint(gradient);
      g2.fill(new Ellipse2D.Double(x, y, size, size));
      g2.setColor(new Color(255, 255, 255, 120));
```

```
        g2.fillOval(x + size / 4, y + size / 4, size / 4, size / 4);
    }

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

===== BallCanvas.java =====
package part_4.bounce;

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

public class BallCanvas extends JPanel {

    private final List<Ball> balls = new ArrayList<>();
    private final List<BallThread> ballThreads = new ArrayList<>();
    private int sleepTime = Config.DEFAULT_SLEEP_MS;

    public BallCanvas() {
        setDoubleBuffered(true);
    }

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

    public void registerBallThread(BallThread t) {
        ballThreads.add(t);
    }

    public void unregisterBallThread(BallThread t) {
        ballThreads.remove(t);
    }

    public int getSleepTime() { return sleepTime; }

    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        Graphics2D g2 = (Graphics2D) g;
        g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
RenderingHints.VALUE_ANTIALIAS_ON);

        int w = getWidth();
        int h = getHeight();
        g2.setPaint(new GradientPaint(0, 0, new Color(20, 90, 40), w, h, new Color(10, 50, 25)));
        g2.fillRect(0, 0, w, h);
```

```
        List<Ball> copy;
        synchronized (balls) {
            copy = new ArrayList<>(balls);
        }
        for (Ball b : copy)
            b.draw(g2);

        g2.setColor(Color.WHITE);
        g2.setFont(new Font("Consolas", Font.BOLD, 14));
        g2.drawString("Balls: " + balls.size() + "  Threads: " + ballThreads.size(), 15, 20);
    }
}
```

===== BallThread.java =====

```
package part_4.bounce;

public class BallThread extends Thread {

    private final Ball ball;
    private final BallCanvas canvas;
    private final int maxMoves;

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

    @Override
    public void run() {
        canvas.registerBallThread(this);
        try {
            for (int i = 0; i < maxMoves; i++) {
                ball.move();
                Thread.sleep(canvas.getSleepTime());
            }
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
        }
        canvas.unregisterBallThread(this);
    }
}
```

===== Bounce.java =====

```
package part_4.bounce;

import javax.swing.*;

public class Bounce {

    public static void main(String[] args) {
        BounceFrame frame = new BounceFrame();
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

===== BounceFrame.java =====

```
package part_4.bounce;
```

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

public class BounceFrame extends JFrame {

   private final BallCanvas canvas;
   private JLabel statusLabel;

   public BounceFrame() {
      try {
         UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
      } catch (Exception ignored) {
      }

      setSize(Config.FRAME_WIDTH, Config.FRAME_HEIGHT);
      setTitle("Thread.join(): red → blue → green");
      setLocationRelativeTo(null);

      canvas = new BallCanvas();
      add(canvas, BorderLayout.CENTER);

      JPanel controlPanel = new JPanel(new FlowLayout(FlowLayout.LEFT, 15, 8));
      controlPanel.setBackground(new Color(30, 30, 30));
      controlPanel.setBorder(BorderFactory.createEmptyBorder(10, 20, 10, 20));

      JButton joinDemo = createButton("Red → Blue → Green (join)", new Color(80, 120, 60));
      joinDemo.addActionListener(e -> runJoinDemo());

      statusLabel = new JLabel(
            "Click the button: first the red ball moves, then the blue ball moves, then the green ball
moves.");
      statusLabel.setForeground(Color.WHITE);
      statusLabel.setFont(statusLabel.getFont().deriveFont(12f));

      controlPanel.add(joinDemo);
      controlPanel.add(statusLabel);

      add(controlPanel, BorderLayout.SOUTH);
   }

   private JButton createButton(String text, Color bg) {
      JButton b = new JButton(text);
      b.setFocusPainted(false);
      b.setFont(new Font("Segoe UI", Font.BOLD, 12));
      b.setForeground(Color.WHITE);
      b.setBackground(bg);
      b.setCursor(new Cursor(Cursor.HAND_CURSOR));
      b.setBorder(BorderFactory.createEmptyBorder(8, 14, 8, 14));
      b.addMouseListener(new java.awt.event.MouseAdapter() {
         public void mouseEntered(java.awt.event.MouseEvent evt) {
            b.setBackground(brighter(bg));
         }

         public void mouseExited(java.awt.event.MouseEvent evt) {
            b.setBackground(bg);
         }
      });
      return b;
   }
```

```
    private static Color brighter(Color c) {
        return new Color(
                Math.min(255, c.getRed() + 35),
                Math.min(255, c.getGreen() + 35),
                Math.min(255, c.getBlue() + 35));
    }

    private void runJoinDemo() {
        statusLabel.setText("The red ball moves...");
        Thread worker = new Thread(() -> {
            try {
                Ball red = new Ball(canvas, Ball.BallType.RED);
                canvas.add(red);
                BallThread redThread = new BallThread(red, canvas,
Config.BALL_MOVES_JOIN_DEMO);
                redThread.start();
                redThread.join();
                SwingUtilities.invokeLater(() -> statusLabel.setText("The blue ball moves..."));

                Ball blue = new Ball(canvas, Ball.BallType.BLUE);
                canvas.add(blue);
                BallThread blueThread = new BallThread(blue, canvas,
Config.BALL_MOVES_JOIN_DEMO);
                blueThread.start();
                blueThread.join();
                SwingUtilities.invokeLater(() -> statusLabel.setText("The green ball moves..."));

                Ball green = new Ball(canvas, Ball.BallType.GREEN);
                canvas.add(green);
                BallThread greenThread = new BallThread(green, canvas,
Config.BALL_MOVES_JOIN_DEMO);
                greenThread.start();
                greenThread.join();

                SwingUtilities.invokeLater(() -> statusLabel.setText("All three threads have finished
(join)."));
            } catch (InterruptedException ex) {
                Thread.currentThread().interrupt();
                SwingUtilities.invokeLater(() -> statusLabel.setText("Interrupted."));
            }
        });
        worker.start();
    }
}

===== Config.java =====
package part_4.bounce;

public class Config {

    public static final int FRAME_WIDTH = 1000;
    public static final int FRAME_HEIGHT = 600;
    public static final int BALL_MIN_SIZE = 30;

    public static final int DEFAULT_SLEEP_MS = 5;

    public static final int BALL_MOVES_JOIN_DEMO = 1000;
}
```

Задача 5

===== Counter.java =====

```
package part_5.counter;

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

public class Counter {

    public enum SyncType {
        NONE,
        SYNC_METHOD,
        SYNC_BLOCK,
        LOCK
    }

    private int value;
    private final SyncType syncType;
    private final Lock lock = new ReentrantLock();

    public Counter(SyncType syncType) {
        this.syncType = syncType;
    }

    public void increment() {
        switch (syncType) {
            case NONE -> value++;
            case SYNC_METHOD -> incrementSyncMethod();
            case SYNC_BLOCK -> {
                synchronized (this) {
                    value++;
                }
            }
            case LOCK -> {
                lock.lock();
                try {
                    value++;
                } finally {
                    lock.unlock();
                }
            }
        }
    }

    public void decrement() {
        switch (syncType) {
            case NONE -> value--;
            case SYNC_METHOD -> decrementSyncMethod();
            case SYNC_BLOCK -> {
                synchronized (this) {
                    value--;
                }
            }
            case LOCK -> {
                lock.lock();
                try {
                    value--;
                } finally {
                    lock.unlock();
```

```
            }
        }
    }
}

    private synchronized void incrementSyncMethod() {
        value++;
    }

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

    public int getValue() {
        return value;
    }
}

===== CounterDemo.java =====
package part_5.counter;

public class CounterDemo {

    private static final int ITERATIONS = 100_000;

    public static void main(String[] args) throws InterruptedException {
        System.out.println("Counter: 2 threads, one increments 100000 times, other decrements 100000
times\n");

        runUnsafe();
        runSyncMethod();
        runSyncBlock();
        runLock();
    }

    private static void runUnsafe() throws InterruptedException {
        System.out.println("1) Unsynchronized (NONE):");
        for (int run = 1; run <= 5; run++) {
            Counter c = new Counter(Counter.SyncType.NONE);
            Thread inc = new Thread(() -> {
                for (int i = 0; i < ITERATIONS; i++)
                    c.increment();
            });
            Thread dec = new Thread(() -> {
                for (int i = 0; i < ITERATIONS; i++)
                    c.decrement();
            });
            inc.start();
            dec.start();
            inc.join();
            dec.join();
            System.out.println("   Run " + run + ": value = " + c.getValue());
        }
        System.out.println("   Expected: 0. The result is unpredictable due to the race condition.\n");
    }

    private static void runSyncMethod() throws InterruptedException {
        System.out.println("2) Synchronized method (synchronized method):");
        Counter c = new Counter(Counter.SyncType.SYNC_METHOD);
        Thread inc = new Thread(() -> {
```

```
            for (int i = 0; i < ITERATIONS; i++)
                c.increment();
        });
        Thread dec = new Thread(() -> {
            for (int i = 0; i < ITERATIONS; i++)
                c.decrement();
        });
        inc.start();
        dec.start();
        inc.join();
        dec.join();
        System.out.println("   Value = " + c.getValue() + " (correct).\n");
    }

    private static void runSyncBlock() throws InterruptedException {
        System.out.println("3) Synchronized block (synchronized block):");
        Counter c = new Counter(Counter.SyncType.SYNC_BLOCK);
        Thread inc = new Thread(() -> {
            for (int i = 0; i < ITERATIONS; i++)
                c.increment();
        });
        Thread dec = new Thread(() -> {
            for (int i = 0; i < ITERATIONS; i++)
                c.decrement();
        });
        inc.start();
        dec.start();
        inc.join();
        dec.join();
        System.out.println("   Value = " + c.getValue() + " (correct).\n");
    }

    private static void runLock() throws InterruptedException {
        System.out.println("4) Object lock (ReentrantLock):");
        Counter c = new Counter(Counter.SyncType.LOCK);
        Thread inc = new Thread(() -> {
            for (int i = 0; i < ITERATIONS; i++)
                c.increment();
        });
        Thread dec = new Thread(() -> {
            for (int i = 0; i < ITERATIONS; i++)
                c.decrement();
        });
        inc.start();
        dec.start();
        inc.join();
        dec.join();
        System.out.println("   Value = " + c.getValue() + " (correct).\n");
    }
}
```