Міністерство освіти і науки України
Національний технічний університет України «КПІ» імені Ігоря Сікорського
Кафедра обчислювальної техніки ФІОТ

ЗВІТ
з лабораторної роботи №1
з навчальної дисципліни «Технології паралельних обчислень»

Тема:
Розробка потоків та дослідження пріоритету запуску потоків

Виконав
Гордуз О.С., ІТ-03

Перевірив
Дифучина О.Ю.

Дата _____

Оцінка _____

Київ 2023

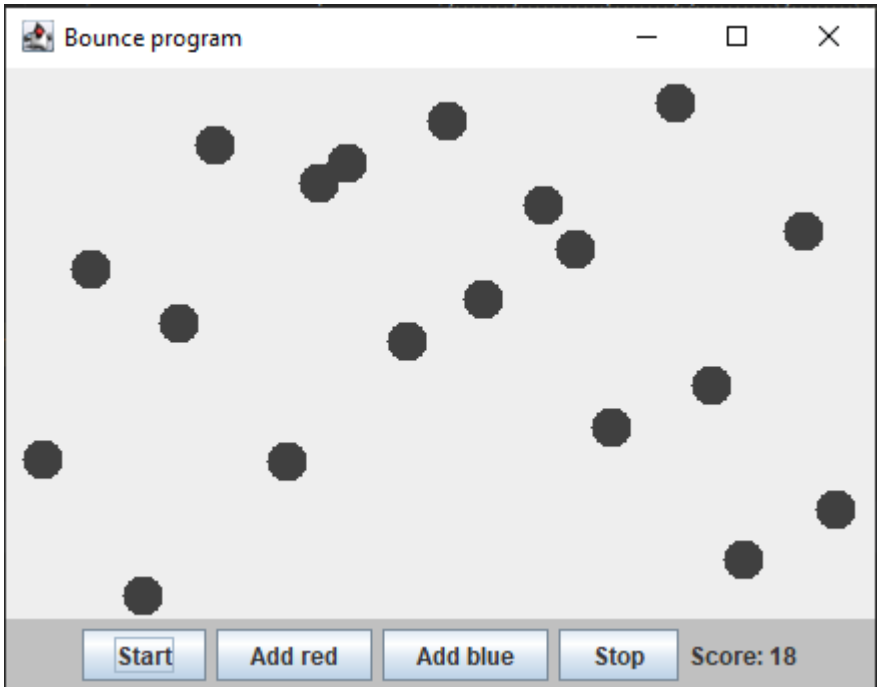
Завдання

1. Реалізуйте програму імітації руху більярдних кульок, в якій рух кожної кульки відтворюється в окремому потоці (див. презентацію «Створення та запуск потоків в java» та приклад). Спостерігайте роботу програми при збільшенні кількості кульок. Поясніть результати спостереження. Опишіть переваги потокової архітектури програм. **10 балів.**
2. Модифікуйте програму так, щоб при потраплянні в «лузу» кульки зникали, а відповідний потік завершував свою роботу. Кількість кульок, яка потрапила в «лузу», має динамічно відображатись у текстовому полі інтерфейсу програми. **10 балів.**
3. Виконайте дослідження параметру priority потоку. Для цього модифікуйте програму «Більярдна кулька» так, щоб кульки червоного кольору створювались з вищим пріоритетом потоку, в якому вони виконують рух, ніж кульки синього кольору. Спостерігайте рух червоних та синіх кульок при збільшенні загальної кількості кульок. Проведіть такий експеримент. Створіть багато кульок синього кольору (з низьким пріоритетом) і одну червоного кольору, які починають рух в одному й тому ж самому місці більярдного стола, в одному й тому ж самому напрямку та з однаковою швидкістю. Спостерігайте рух кульки з більшим пріоритетом. Повторіть експеримент кілька разів, значно збільшуючи кожного разу кількість кульок синього кольору. Зробіть висновки про вплив пріоритету потоку на його роботу в залежності від загальної кількості потоків. **20 балів.**
4. Побудуйте ілюстрацію для методу join() класу Thread з використанням руху більярдних кульок різного кольору. Поясніть результат, який спостерігається. **10 балів.**
5. Створіть два потоки, один з яких виводить на консоль символ '-', а інший – символ '|'. Запустіть потоки в основній програмі так, щоб вони виводили свої символи в рядок. Виведіть на консоль 100 таких рядків. Поясніть виведений результат. **10 балів.** Використовуючи найпростіші методи управління потоками, добийтесь почергового виведення на консоль символів. **15 балів.**
6. Створіть клас Counter з методами increment() та decrement(), які збільшують та зменшують значення лічильника відповідно. Створіть два потоки, один з яких збільшує 100000 разів значення лічильника, а інший – зменшує 100000 разів значення лічильника. Запустіть потоки на одночасне виконання. Спостерігайте останнє значення лічильника. Поясніть результат. **10 балів.** Використовуючи синхронізований доступ, добийтесь правильної роботи лічильника при одночасній роботі з ним двох і більше потоків. Опрацюйте використання таких способів синхронізації:

синхронізований метод, синхронізований блок, блокування об'єкта. Порівняйте способи синхронізації. **15 балів.**

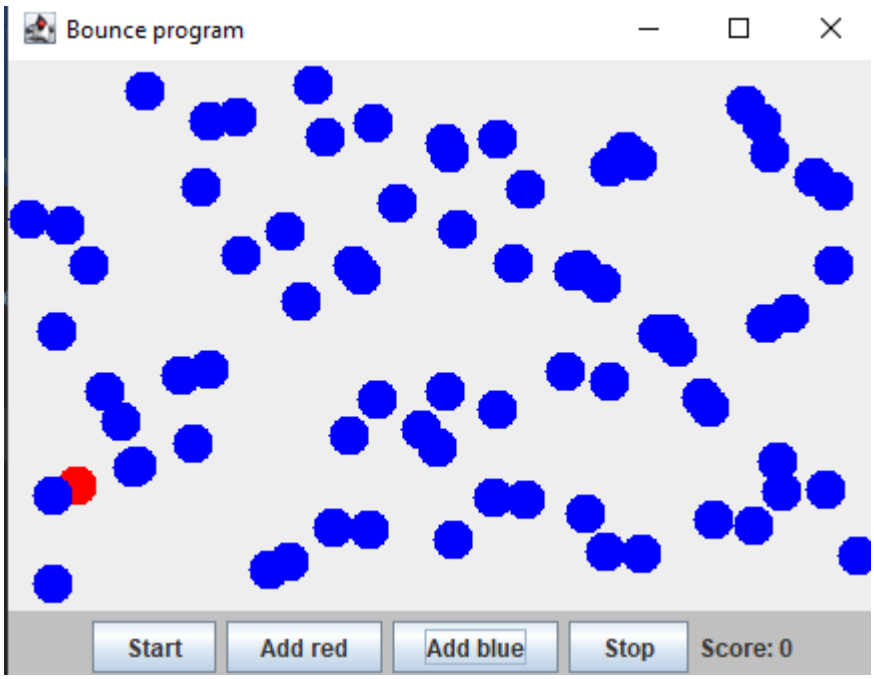
2. Додамо лічильник і 6 лунок (3 зверху і 3 знизу)

```
if ((x < 5 && (y < ySize / 2 || abs(y - (h / 2)) < ySize / 2 || h - y < ySize / 2))||
   (abs(x - w / 2) < 5 && (y < ySize / 2 || h - y < ySize / 2)) ||
   (w - x < xSize / 2 && (y < ySize / 2 || abs(y - (h / 2)) < ySize / 2 || h - y < ySize / 2))) {
    b.delete();
    if (s != null) {
        s.updateScore();
    }
    return;
}
```



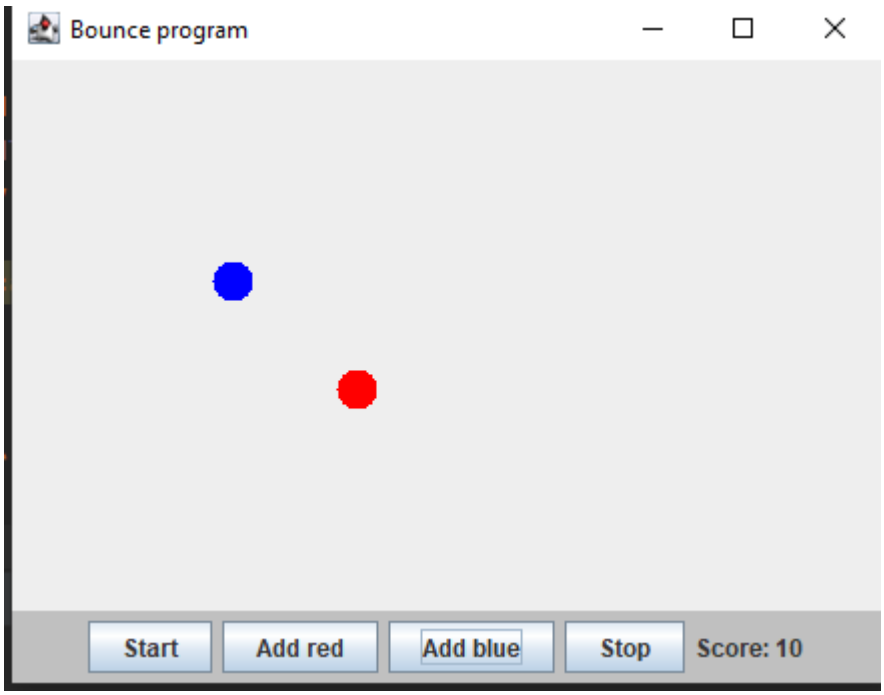
3. Кульки рухаються однаково без залежності від кількості кульок

```
                Ball b = new Ball(canvas);
                b.setColor("red");
                canvas.add(b);

                BallThread thread = new BallThread(b, s);
                thread.setPriority(Thread.MAX_PRIORITY);
                thread.start();
                lastRedThread = thread;
                System.out.println("Thread name = " +
                        thread.getName());
            }
        });
        buttonBlue.addActionListener(new ActionListener() {

            @Override
            public void actionPerformed(ActionEvent e) {

                Ball b = new Ball(canvas);
                b.setColor("blue");
                canvas.add(b);

                BallThread thread = new BallThread(b, s);//, lastRedThread);
                thread.setPriority(Thread.MIN_PRIORITY);
                thread.start();
                System.out.println("Thread name = " +
                        thread.getName());
```



4. Модифікуємо клас BallThread, щоб ми могли передавати в нього потік, на який будемо чекати. Реалізуємо поведінку, при якому сині кульки будуть чекати завершення роботи останньої випущеної червоної кульки

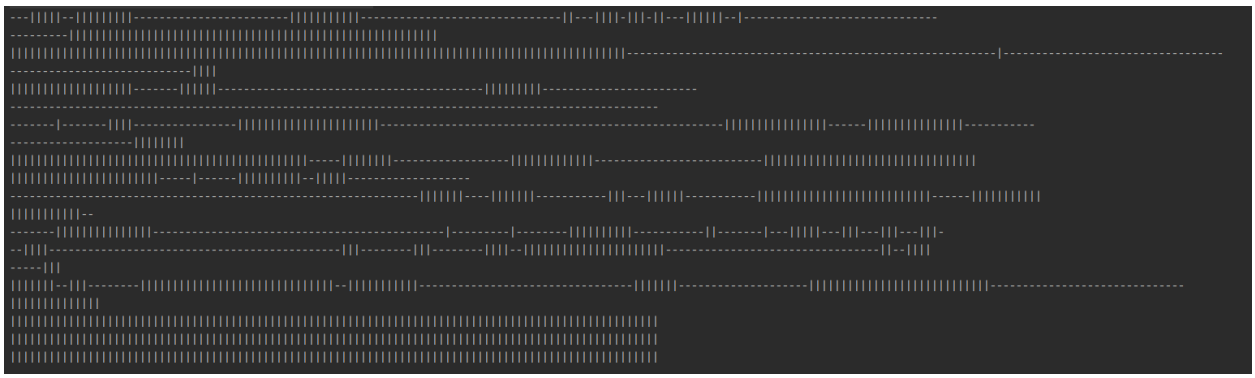
```
public BallThread(Ball ball, Scoreboard s, Thread redThread){
    b = ball;
    this.s = s;
    this.redThread = redThread;
}

@Override
public void run(){
    if (redThread != null)
        try {
            redThread.join();
        } catch (Exception e) {
```



5. Напишемо класс WriterThread, що й буде писати в консоль символи

```
@Override
public void run() {
    for (int i = 0; i < 10; i++) {
        for (int j = 0; j < 100; j++) {
            System.out.print(c);
        }
        System.out.println();
    }
}
```

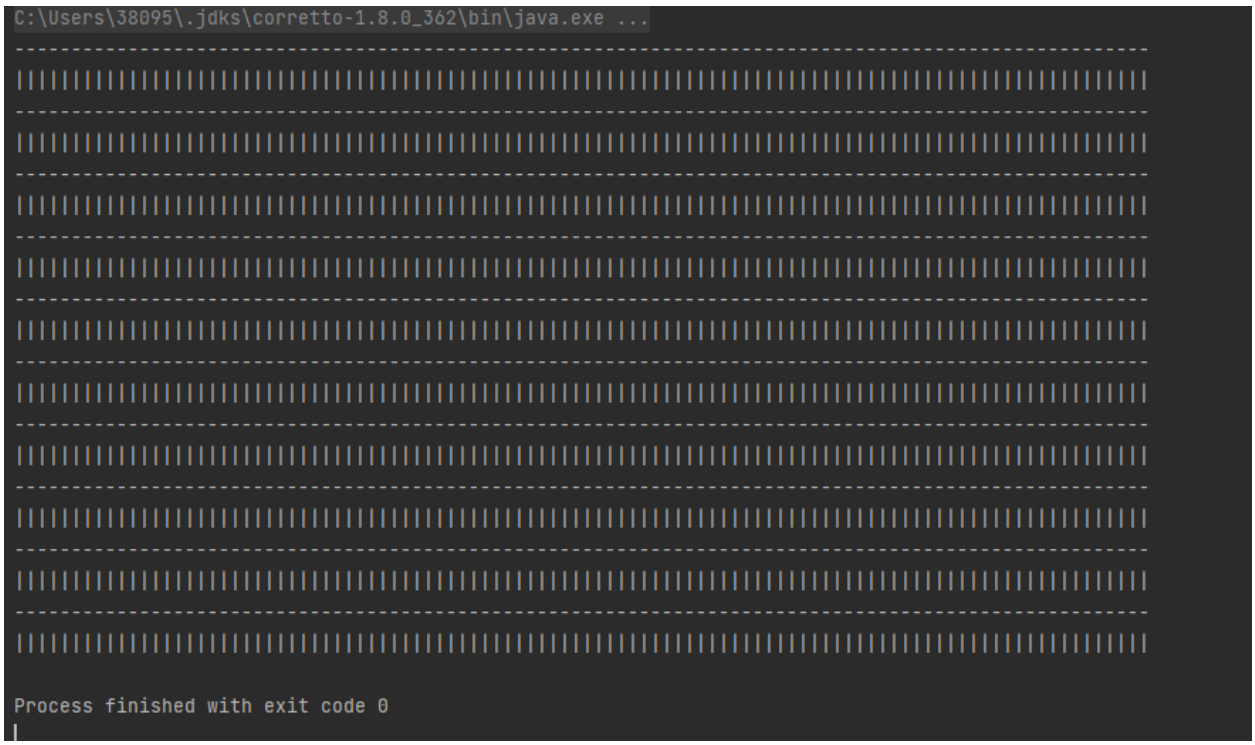
```
public class Main {
    public static void main(String[] args) {
        Thread t1 = new WriterThread( c: '-');
        Thread t2 = new WriterThread( c: '|');
        t1.start();
        t2.start();

    }
}
```



Модифікуємо код так, щоб рядки виводились по черзі

```
@Override
public void run() {
    for (int i = 0; i < 10; i++) {
        for (int j = 0; j < 100; j++) {
            System.out.print(c);
        }
        System.out.println();
        t.interrupt();
        try{
            if (i < 9)
                Thread.sleep( millis: 100000);
        } catch (Exception e) {
            System.out.println("Oops, some error");
        }
    }
}
```

```
public class Main {
    public static void main(String[] args) {
        WriterThread t1 = new WriterThread( c: '-');
        WriterThread t2 = new WriterThread( c: '|');
        t1.setT(t2);
        t2.setT(t1);
        t1.start();
        try {
            Thread.sleep( millis: 1000);
        } catch (Exception e) {
            System.out.println("Oops, some error");
        }
        t2.start();
    }
}
```



6.

```java
public class Main {
    public static void main(String[] args) {
        Counter c = new Counter();
        Thread t1 = new Thread(new RunManyTimes( n: 1000000, c::decrement));
        Thread t2 = new Thread(new RunManyTimes( n: 1000000, c::increment));
        t1.start();
        t2.start();
        try {
            t1.join();
            t2.join();
            System.out.println(c.get());
        } catch (Exception e) {
            System.out.println("Oops, something went wrong");
        }

    }
}
```

```java
public class RunManyTimes implements Runnable{
    private int n;
    private Runnable r;
    public RunManyTimes(int n, Runnable r) {
        this.n = n;
        this.r = r;
    }

    @Override
    public void run() {
        for (int i = 0; i < n; i++) {
            r.run();
        }
    }
}
```

```
-12472

Process finished with exit code 0
```

Спробуємо модифікувати програму так, щоб вийшов 0

- синхронізований метод

```java
public class Counter {
    private int i;

    public int get() {
        return i;
    }

    public synchronized void increment() {
        i = i + 1;
    }

    public synchronized void decrement() {
        i = i - 1;
    }

}
```

- синхронізований блок

```
public static void main(String[] args) {
    Counter c = new Counter();
    Thread t1 = new Thread(new RunManyTimes( n: 1000000, () -> {
        synchronized (c) {c.decrement();}
    }));
    Thread t2 = new Thread(new RunManyTimes( n: 1000000, () -> {
        synchronized (c) {c.increment();}
    }));
    t1.start();
    t2.start();
    try {
        t1.join();
        t2.join();
        System.out.println(c.get());
    } catch (Exception e) {
        System.out.println("Oops, something went wrong");
    }
```

- блокування об’єкта

```
public static void main(String[] args) {
    Counter c = new Counter();
    Lock l = new ReentrantLock();
    Thread t1 = new Thread(new RunManyTimes( n: 1000000, () -> {
        l.lock();
        c.decrement();
        l.unlock();
    }));
    Thread t2 = new Thread(new RunManyTimes( n: 1000000, () -> {
        l.lock();
        c.increment();
        l.unlock();
    }));
    t1.start();
    t2.start();
    try {
        t1.join();
        t2.join();
        System.out.println(c.get());
    } catch (Exception e) {
        System.out.println("Oops, something went wrong");
    }
```

За своєю суттю всі три методи роблять одне й те саме, проте перший читається набагато краще.