Звіт

Автор: Пумня О., КІТ118Б

Дата: 15.02.2020

Лабораторна робота №13

ПАРАЛЕЛЬНЕ ВИКОНАННЯ. БАГАТОПОТОЧНІСТЬ

Мета. Ознайомлення з моделлю потоків Java. Організація паралельного виконання декількох частин програми.

Вимоги:

1. Використовуючи програми рішень попередніх задач, продемонструвати можливість паралельної обробки елементів контейнера: створити не менше трьох додаткових потоків, на яких викликати відповідні методи обробки контейнера.
2. Забезпечити можливість встановлення користувачем максимального часу виконання (таймаута) при закінченні якого обробка повинна припинятися незалежно від того знайдений кінцевий результат чи ні.
3. Для паралельної обробки використовувати алгоритми, що не змінюють початкову колекцію.
4. Кількість елементів контейнера повинна бути досить велика, складність алгоритмів обробки колекції повинна бути зіставна, а час виконання приблизно однаковий, наприклад:
    - пошук мінімуму або максимуму;
    - обчислення середнього значення або суми;
    - підрахунок елементів, що задовольняють деякій умові;
    - відбір за заданим критерієм;
    - власний варіант, що відповідає обраній прикладної області.

ОПИС ПРОГРАМИ

Опис змінних

`GenericList<Integer> numbers – список чисел`

`Thread thr1 – потік, що шукає кількість парних і непарних чисел масиву`

`Thread thr2 – потік, що рахує середнє значення масиву чисел`

`Thread thr3 – потік, що шукає мінімальне та максимальне значення масиву чисел`

Ієрархія та структура класів

`Pumnya13 – точка входу в програму`

`FirstThread – реалізує пошук кількості парних і непарних чисел масиву`

`SecondThread – реалізує розрахунок середнього значення масиву чисел`

ThirdThread – реалізує пошук мінімального та максимального значення масиву чисел

ТЕКСТ ПРОГРАМИ

Текст файлу Pumnya13.java

```
package labs.pumnya13;
import labs.pumnya09.GenericList;
import java.util.Random;

public class Pumnya13 {
    private Pumnya13(){
    }

    public static void main(String[] args) {
        GenericList<Integer> numbers = new GenericList<>();

        for (int i = 0; i < 100000; i++) {
            numbers.pushBack(new Random().nextInt(100000));
        }
        Thread thr1 = new FirstThread(numbers, 0.2);
        Thread thr2 = new SecondThread(numbers, 0.2);
        Thread thr3 = new ThirdThread(numbers, 0.2);

        thr1.start();
        thr2.start();
        thr3.start();
    }
}
```

Текст файлу FirshThread.java

```
package labs.pumnya13;
import labs.pumnya09.GenericList;

public class FirstThread extends Thread {
    /** Принимает список в виде массива. */
    private int[] listToArr;
    /**
     * Значение таймера.
     * Поток работает, пока это значение больше 0-я.
     */
    private double timeOut;
    /** Простой конструктор. */
    public FirstThread() {
    }
    /**
     * Конструктор с параметрами.
     * Устанавливает список значений, а также таймер.
     * @param list данные для обработки
     * @param timeOut устанавливает значение таймера
     */
    public FirstThread(GenericList<Integer> list, double timeOut) {
        this.timeOut = timeOut;
        Object[] temp = list.toArray();
        this.listToArr = new int[temp.length];
        for(int i = 0; i < temp.length; i++) {
            this.listToArr[i] = (int) temp[i];
        }
    }
    /**
     * Переопределение метода запуска потока.
     * Поиск кол-ва четных и нечетных значений в массиве чисел.
     */
    @Override
    public void run() {
        long startTime = System.nanoTime();
        int even = 0, odd = 0;
        for (int i = 0; i < listToArr.length; i++) {
```

```
            if (listToArr[i] % 2 == 0) {
                even++;
            } else {
                odd++;
            }
            long timeTotal = System.nanoTime() - startTime;
            try {
                double convert = timeTotal*10e-9;
                if(convert > timeOut) {
                    throw new Exception();
                }
            } catch (Exception e) {
                this.interrupt();
                System.out.println("FirstThread был прерван!");
                return;
            }
        }
        System.out.println("Четные: " + even + " | Нечетные: " + odd);
    }
}
```

Текст файлу SecondThread.java

```
package labs.pumnya13;
import labs.pumnya09.GenericList;

public class SecondThread extends Thread {
    /** Принимает список в виде массива. */
    private int[] listToArr;
    /**
     * Значение таймера.
     * Поток работает, пока это значение больше 0-я.
     */
    private double timeOut;
    /** Простой конструктор. */
    public SecondThread() {
    }
    /**
     * Конструктор с параметрами.
     * Устанавливает список значений, а также таймер.
     * @param list данные для обработки
     * @param timeOut устанавливает значение таймера
     */
    public SecondThread(GenericList<Integer> list, double timeOut) {
        this.timeOut = timeOut;
        Object[] temp = list.toArray();
        this.listToArr = new int[temp.length];
        for(int i = 0; i < temp.length; i++) {
            this.listToArr[i] = (int) temp[i];
        }
    }
    /**
     * Переопределение метода запуска потока.
     * Определение среднего значения массива целых чисел.
     */
    @Override
    public void run() {
        long startTime = System.nanoTime();
        float avg = 0;
        for (int i = 0; i < listToArr.length; i++) {
            avg += listToArr[i];
            long timeTotal = System.nanoTime() - startTime;
            try {
                double convert = timeTotal*10e-9;
                if(convert > timeOut) {
                    throw new Exception();
                }
            } catch (Exception e) {
                this.interrupt();
```

```
                System.out.println("SecondThread был прерван!");
                return;
            }
        }
        avg /= listToArr.length;
        System.out.println("Среднее значение: " + avg);
    }
}
```

Текст файлу ThirdThread.java

```
package labs.pumnya13;
import labs.pumnya09.GenericList;

public class ThirdThread extends Thread {
    /** Принимает список в виде массива. */
    private int[] listToArr;
    /**
     * Значение таймера.
     * Поток работает, пока это значение больше 0-я.
     */
    private double timeOut;
    /** Простой конструктор. */
    public ThirdThread() {
    }
    /**
     * Конструктор с параметрами.
     * Устанавливает список значений, а также таймер.
     * @param list данные для обработки
     * @param timeOut устанавливает значение таймера
     */
    public ThirdThread(GenericList<Integer> list, double timeOut) {
        this.timeOut = timeOut;
        Object[] temp = list.toArray();
        this.listToArr = new int[temp.length];
        for(int i = 0; i < temp.length; i++) {
            this.listToArr[i] = (int) temp[i];
        }
    }
    /**
     * Переопределение метода запуска потока.
     * Поиск минимального и максимального значения.
     */
    @Override
    public void run() {
        long startTime = System.nanoTime();
        int min = Integer.MAX_VALUE;
        int max = 0;
        for (int i = 0; i < listToArr.length; i++) {
            if (min > listToArr[i]) {
                min = listToArr[i];
            } else if (max < listToArr[i]) {
                max = listToArr[i];
            }
            long timeTotal = System.nanoTime() - startTime;
            try {
                double convert = timeTotal*10e-9;
                if(convert > timeOut) {
                    throw new Exception();
                }
            } catch (Exception e) {
                this.interrupt();
                System.out.println("ThirdThread был прерван!");
                return;
            }
        }
        System.out.println("Min: " + min + " | Max: " + max);
    }
}
```

ВАРІАНТИ ВИКОРИСТАННЯ

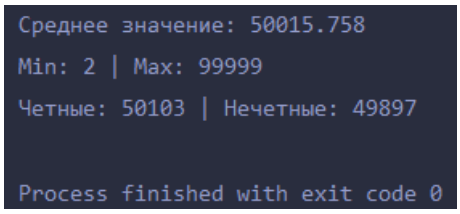

Рисунок 1 – Результат роботи програми

ВИСНОВКИ

При виконанні лабораторної роботи набуто практичних навичок створення власних класів, що реалізують потоки. Створено програму, яка паралельно знаходить середнє, мінімальне та максимальне значення масиву чисел, а також знаходить кількість парних і непарних чисел.