# ЛАБОРАТОРНАЯ РАБОТА №6

# МНОГОПОТОЧНОСТЬ В JAVA

Работу выполнил: Мовенко Константин, ИС/б-21-2-о

Цель работы

Изучить основные приемы работы с потоками в Java и получить практические навыки написания многопоточных Java-приложений.

Задачи

1. В главном классе реализовать статический метод сортировки массива (Таблица 1) и вывода результата сортировки на экран с помощью вызова *System.out.println(…)*, формат вывода: “имя\_текущего\_потока: [отсортированный\_массив]”. Сигнатура функции сортировки *printSorted(int[] array)*;
2. В методе main реализовать цикл генерации 3 массивов случайных чисел длины N и для каждого сгенерированного массива вызывать метод сортировки в новом потоке. Установить для потоков приоритеты в соответствии с вариантом и после этого запустить потоки в произвольном порядке. Запустить программу несколько раз и пронаблюдать за соблюдением приоритетов;
3. В методе сортировки заменить вывод всего массива на поэлементный вывод с помощью вызова *System.out.print(…)*, причем массив должен выводиться по 100 элементов в строке и с двух сторон отделяться строками, состоящими из символов «\*». Перед строками содержимого массива выводить имя текущего потока. Проверить корректность вывода при запуске этой программы;
4. Реализовать синхронизацию потоков в момент записи значений в поток вывода. Запустить исправленную программу и проверить корректность вывода;

Вариант задания

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Метод сортировки | Направление сортировки | Приоритеты потоков | Размер массива N |
| 1 | Метод пузырька | по убыванию | 1-8 2-5 3-1 | 10000 |

Таблица 1 – Вариант задания

Ход работы

Изначально программа была написана без синхронизации с помощью критических секций. Были запущены три потока обработки массивов, выводящих массивы одной операцией println(). Никаких ошибок не возникло (Рисунок 1). Синхронизация не потребовалась.

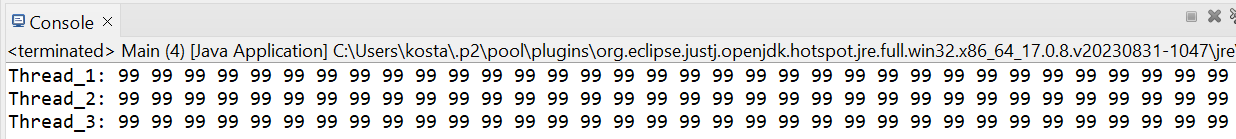


Рисунок 1 – Вывод массивов одной строкой

Затем были запущены потоки, выводящие массивы поэлементно, в несколько операций. Из-за того, что работа потоков была несинхронизированной, строки выводимых массивов перемешались друг с другом (Рисунок 2).

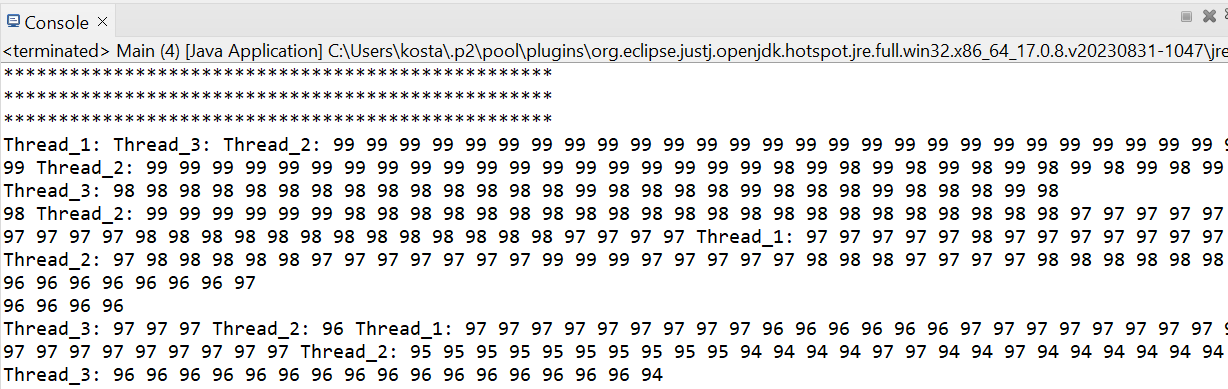


Рисунок 2 – Несинхронизированный вывод массивов поэлементно

Программа была дописана – методы, вызываемые внутри потоков, получили модификатор synchronized, тем самым заключены в неделимые критические секции. При повторном запуске потоков вывод отсортированных массивов был упорядочен (Рисунок 3).

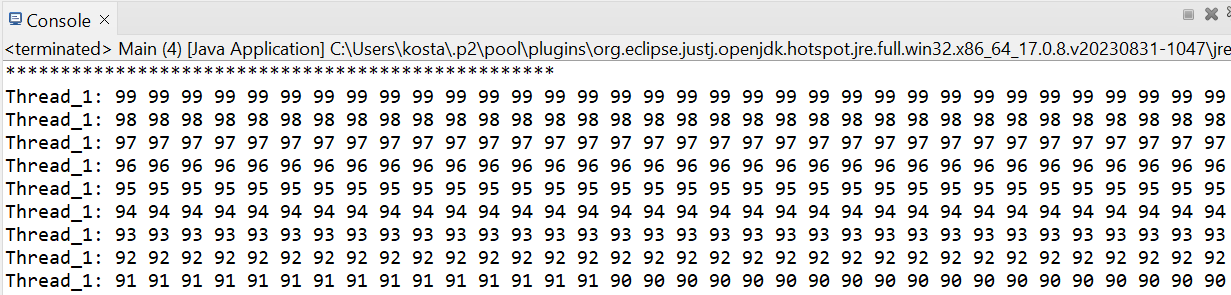


Рисунок 3 – Синхронизированный вывод массивов поэлементно

Текст программы

Листинг 1 – Main.java

**import** java.util.Random;

**import** **static** java.lang.Math.*abs*;

**public** **class** Main

{

**static** **final** **int** ***N*** = 10000; // размер массивов

**static** **final** **boolean** ***PRINT\_SEQUENTLY*** = **true**; // способ вывода (true - поэлементно)

// Сортировка массива методом пузырька

**public** **static** **synchronized** **void** bubbleSort(**int**[] arr)

{

**for** (**int** i = 0; i < arr.length - 1; i++)

{

**for** (**int** j = 0; j < arr.length - i - 1; j++)

{

**if** (arr[j] < arr[j + 1])

{

**int** temp = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = temp;

}

}

}

}

// Сортировка и вывод массива одним вызовом println()

**public** **static** **synchronized** **void** printSorted(**int**[] arr)

{

*bubbleSort*(arr);

String line = Thread.*currentThread*().getName() + ": ";

**for** (**int** i = 0; i < ***N***; i++)

{

line += arr[i] + " ";

}

System.***out***.println(line);

}

// Сортировка и вывод массива поэлементно

**public** **static** **void** printSortedSequently(**int**[] arr)

{

*bubbleSort*(arr);

System.***out***.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

**for**(**int** i = 0; i < ***N***/100; i++)

{

System.***out***.print(Thread.*currentThread*().getName() + ": ");

**for** (**int** j = 0; j < 100; j++)

{

System.***out***.print(arr[i\*100 + j] + " ");

}

System.***out***.println();

}

System.***out***.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

}

// Главный метод - выполнение программы

**public** **static** **void** main(String[] args) **throws** InterruptedException

{

// Создание массивов из N целых чисел

**int**[] arr\_1 = **new** **int**[***N***], arr\_2 = **new** **int**[***N***],

arr\_3 = **new** **int**[***N***];

// Заполнение массивов случайными числами [0; 99]

Random rand = **new** Random();

**for** (**int** i = 0; i < ***N***; i++)

{

arr\_1[i] = *abs*(rand.nextInt() % 100);

arr\_2[i] = *abs*(rand.nextInt() % 100);

arr\_3[i] = *abs*(rand.nextInt() % 100);

}

// Определение потоков обработки массивов

Thread thread\_1, thread\_2, thread\_3;

**if** (!***PRINT\_SEQUENTLY***) // вывод одним вызовом

{

thread\_1 = **new** Thread(() -> *printSorted*(arr\_1));

thread\_2 = **new** Thread(() -> *printSorted*(arr\_2));

thread\_3 = **new** Thread(() -> *printSorted*(arr\_3));

}

**else** // вывод поэлементно

{

thread\_1 = **new** Thread(() -> *printSortedSequently*(arr\_1));

thread\_2 = **new** Thread(() -> *printSortedSequently*(arr\_2));

thread\_3 = **new** Thread(() -> *printSortedSequently*(arr\_3));

}

// Имена потоков

thread\_1.setName("Thread\_1");

thread\_2.setName("Thread\_2");

thread\_3.setName("Thread\_3");

// Приоритеты потоков

thread\_1.setPriority(8);

thread\_2.setPriority(5);

thread\_3.setPriority(1);

// Запуск потоков

thread\_1.start();

thread\_2.start();

thread\_3.start();

}

}

Вывод

В ходе работы были получены навыки создания многопоточных приложений с помощью средств языка Java. Были изучены принципы разбиения процесса на потоки, их свойства и способы обработки, понятие волатильных данных, возможные ошибки взаимодействия потоков и способ их синхронизации через критические секции (synchronized).