**Лабораторная работа 1**

**08.02.2024**

**Разработка многопоточных приложений на языке Java**

**Источники**

1. Хорстманн, Кей С. Java. Библиотека профессионала, том 1. Основы. 11-е изд.: Пер. с англ. СПб.: ООО "Диалектика", 2019.
2. Блинов, И. Н., Романчик, В. С. Java from EPAM : учеб.-метод. пособие / И. Н. Блинов, В. С. Романчик. —2-е издание. Минск: Четыре четверти, 2021. —560 с.
3. <https://math.hws.edu/javanotes/>
4. Paul Deitel, Harvey Deitel - Java How to Program, Early Objects-Pearson Education (2015)

**Правила оформления отчета**

* Берем файл с условием.
* Переименовываем – добавляем свою фамилию к имени файла.
* Вставляем в «***Отчет***» результаты выполнения.

***Задание 1.***

Изучить (вспомнить) способы создания и запуска потока в Java.

[2, 369-370]

[1,12.1, 662-667]

***Задание 2.***

***2.1.***

Скомпилировать и запустить программу:

* class ThreadTest1

[3]

<https://math.hws.edu/javanotes/c12/s1.html>

Чтобы помочь вам понять, как несколько потоков выполняются параллельно, мы рассмотрим пример программы ThreadTest1.java. Эта программа создает несколько потоков. Каждый поток выполняет одну и ту же задачу. Задача — посчитать количество простых чисел меньше 10 000 000. (Конкретная выполняемая задача не важна для наших целей, поскольку она занимает нетривиальное количество времени. Это демонстрационная программа; в реальной программе было бы глупо иметь несколько потоков, которые сделайте то же самое, и метод, используемый для подсчета простых чисел, очень неэффективен.)

Главный поток не ожидает завершения дочерних потоков.

***Знать:***

* как замерять время выполнения вычислений

long startTime = System.currentTimeMillis();

int count = countPrimes(2,MAX);

long elapsedTime = System.currentTimeMillis() - startTime;

System.out.println("Thread " + id + " counted " +

count + " primes in " + (elapsedTime/1000.0) + " seconds.");

***2.2.***

Жизненный цикл потока [2, 371-372].

***2.3.***

[3]

<https://math.hws.edu/javanotes/c12/s1.html>

Добавить, чтобы главный поток ожидал завершения дочерних потоков.

***Знать:***

* Определение состояния потока.
* Ожидание завершения потока.

***2.4.***

1. Последовательная программа.

Реализовать решение задачи «Посчитать количество простых чисел меньше 10 000 000» в одном потоке.

2. Многопоточная (параллельная) программа

class ThreadTest1 – разделить работу между потоками.

3. Измерить время работы Последовательной и Многопоточной программ.

4. Провести эксперименты с многопоточной реализацией с разным количеством потоков.

5. Результата экспериментов сохранить.

***Отчет:***

***Задание 1***

package org.example;  
  
*/\*\*  
 \* A program that starts several threads, each of which performs the  
 \* same computation. The user specifies the number of threads. The  
 \* point is to see that the threads finish in an indeterminate order.  
 \*/*public class ThreadTest1 {  
  
 private final static int *MAX* = 1\_000\_000\_000;  
  
 public static void main(String[] args) {  
 long startTime = System.*currentTimeMillis*();  
 int count = *countPrimes*(2,*MAX*);  
 long elapsedTime = System.*currentTimeMillis*() - startTime;  
 System.*out*.println("Code counted " +  
 count + " primes in " + (elapsedTime/1000.0) + " seconds.");  
 }  
  
  
 */\*\*  
 \* Compute and return the number of prime numbers in the range  
 \* min to max, inclusive.  
 \*/* private static int countPrimes(int min, int max) {  
 int count = 0;  
 for (int i = min; i <= max; i++)  
 if (*isPrime*(i))  
 count++;  
 return count;  
 }  
  
  
 */\*\*  
 \* Test whether x is a prime number.  
 \* x is assumed to be greater than 1.  
 \*/* private static boolean isPrime(int x) {  
 assert x > 1;  
 int top = (int)Math.*sqrt*(x);  
 for (int i = 2; i <= top; i++)  
 if ( x % i == 0 )  
 return false;  
 return true;  
 }  
  
  
}

**Задание 2**

package org.example;  
  
*/\*\*  
 \* This program counts the number of prime integers between 3000001 and 6000000.  
 \* The work is divided among one to six threads. The number of threads is  
 \* chosen by the user.  
 \*/*public class ThreadTest2 {  
  
 */\*\*  
 \* The starting point for the range of integers that are tested for primality.  
 \* The range is from (start+1) to (2\*start). Note the value of start is chosen  
 \* to be divisible by 2, 3, 4, 5, and 6 to make it easy to divide up the range  
 \* among the threads.  
 \*/* private static final int *START* = 1;  
 private static final int *END* = 1\_000\_000\_000;  
  
 */\*\*  
 \* The total number of primes found. Each thread counts the number of primes in  
 \* a different range of integers. After it finishes counting, it adds its count  
 \* to the total.  
 \*/* private static int *total*;  
  
 */\*\*  
 \* Adds x to total. This method is synchronized so that it can be safely used by  
 \* several different threads.  
 \*/* synchronized private static void addToTotal(int x) {  
 *total* = *total* + x;  
 System.*out*.println(*total* + " primes found so far.");  
 }  
  
 */\*\*  
 \* A Thread belonging to this class will count primes in a specified range  
 \* of integers. The range is from min to max, inclusive, where min and max  
 \* are given as parameters to the constructor. After counting, the thread  
 \* outputs a message about the number of primes that it has found, and it  
 \* adds its count to the overall total by calling the addToTotal(int) method.  
 \*/* private static class CountPrimesThread extends Thread {  
 int count = 0;  
 int min, max;  
 public CountPrimesThread(int min, int max) {  
 this.min = min;  
 this.max = max;  
 }  
 public void run() {  
 count = *countPrimes*(min,max);  
 System.*out*.println("There are " + count +  
 " primes between " + min + " and " + max);  
 *addToTotal*(count);  
 }  
 }  
  
 */\*\*  
 \* Counts the primes in the range from (START+1) to (2\*START), using a specified number  
 \* of threads. The total elapsed time is printed.  
 \** ***@param*** *numberOfThreads  
 \*/* private static void countPrimesWithThreads(int numberOfThreads) {  
 int increment = (*END* - *START*)/numberOfThreads;  
 System.*out*.println("\nCounting primes between " + (*START* + 1) + " and "  
 + (*END*) + " using " + numberOfThreads + " threads...\n");  
 long startTime = System.*currentTimeMillis*();  
 CountPrimesThread[] worker = new CountPrimesThread[numberOfThreads];  
 for (int i = 0; i < numberOfThreads; i++)  
 worker[i] = new CountPrimesThread( *START*+i\*increment+1, *START*+(i+1)\*increment );  
 *total* = 0;  
 for (int i = 0; i < numberOfThreads; i++)  
 worker[i].start();  
 for (int i = 0; i < numberOfThreads; i++) {  
 while (worker[i].isAlive()) {  
 try {  
 worker[i].join();  
 }  
 catch (InterruptedException e) {  
 }  
 }  
 }  
 long elapsedTime = System.*currentTimeMillis*() - startTime;  
 System.*out*.println("\nThe number of primes is " + *total* + ".");  
 System.*out*.println("\nTotal elapsed time: " + (elapsedTime/1000.0) + " seconds.\n");  
 }  
  
 */\*\*  
 \* Gets the number of threads from the user and counts primes using that many threads.  
 \*/* public static void main(String[] args) {  
 int processors = Runtime.*getRuntime*().availableProcessors();  
 if (processors == 1)  
 System.*out*.println("Your computer has only 1 available processor.\n");  
 else  
 System.*out*.println("Your computer has " + processors + " available processors.\n");  
 int numberOfThreads = 0;  
 while (numberOfThreads < 1 || numberOfThreads > 6) {  
 System.*out*.print("How many threads do you want to use (from 1 to 6) ? ");  
 numberOfThreads = TextIO.*getlnInt*();  
 if (numberOfThreads < 1 || numberOfThreads > 6)  
 System.*out*.println("Please enter 1, 2, 3, 4, 5, or 6 !");  
 }  
 *countPrimesWithThreads*(numberOfThreads);  
 }  
  
 */\*\*  
 \* Count the primes between min and max, inclusive.  
 \*/* private static int countPrimes(int min, int max) {  
 int count = 0;  
 for (int i = min; i <= max; i++)  
 if (*isPrime*(i))  
 count++;  
 return count;  
 }  
  
 */\*\*  
 \* Test whether x is a prime number.  
 \* x is assumed to be greater than 1.  
 \*/* private static boolean isPrime(int x) {  
 int top = (int)Math.*sqrt*(x);  
 for (int i = 2; i <= top; i++)  
 if ( x % i == 0 )  
 return false;  
 return true;  
 }  
  
}

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Размерность задачи | Время выполнения последовательной программы | Параллельная программа - 2 потока | | | | Параллельная программа - 4 потока | | |
| Время выполнения | Ускорение | | Эффективность | Время выполнения | Ускорение | Эффективность |
| 1 000 000 | 0,247 | 0,159 | | 1,553459 | 0,7767295 | 0,108 | 2,287037037 | 0,571759259 |
| 10 000 000 | 6,288 | 4,154 | | 1,513721 | 0,7568608 | 2,374 | 2,648694187 | 0,662173547 |
| 100 000 000 | 163,846 | 110,372 | | 1,484488 | 0,7422444 | 66,012 | 2,482063867 | 0,620515967 |
| 1 000 000 000 | 2022,573 | 1094,281 | | 1,848312 | 0,9241561 | 681.098 | 2,969577065 | 0,742394266 |

***2.5.***

Многопоточная программа 2.

Creating and Executing Threads with the Executor Framework [4, 23.3].

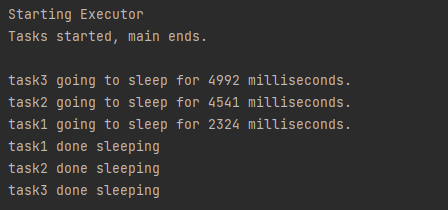
***Отчет:***

***TaskExecutor class***

package org.example;  
  
  
import java.util.concurrent.ExecutorService;  
import java.util.concurrent.Executors;  
  
public class TaskExecutor  
 {  
 public static void main(String[] args)  
 {  
 // create and name each runnable  
 PrintTask task1 = new PrintTask("task1");  
 PrintTask task2 = new PrintTask("task2");  
 PrintTask task3 = new PrintTask("task3");  
  
 System.*out*.println("Starting Executor");  
  
 // create ExecutorService to manage threads  
 ExecutorService executorService = Executors.*newCachedThreadPool*();  
  
 // start the three PrintTasks  
 executorService.execute(task1); // start task1  
 executorService.execute(task2); // start task2  
 executorService.execute(task3); // start task3  
  
 executorService.shutdown();  
  
  
 System.*out*.printf("Tasks started, main ends.%n%n");  
 }  
 } // end class TaskExecutor

**PrintTask class**

package org.example;  
  
import java.security.SecureRandom;  
  
public class PrintTask implements Runnable  
{  
 private static final SecureRandom *generator* = new SecureRandom();  
 private final int sleepTime ; // random sleep time for thread  
 private final String taskName;  
  
 // constructor  
 public PrintTask(String taskName)  
 {  
 this.taskName = taskName;  
 // pick random sleep time between 0 and 5 seconds  
 sleepTime = *generator*.nextInt(5000); // milliseconds  
 }  
 // method run contains the code that a thread will execute  
 public void run()  
 {  
 try // put thread to sleep for sleepTime amount of time  
 {  
 System.*out*.printf("%s going to sleep for %d milliseconds.%n",  
 taskName, sleepTime);  
  
 Thread.*sleep*(sleepTime); // put thread to sleep  
 }  
 catch (InterruptedException exception)  
 {  
 exception.printStackTrace();  
 Thread.*currentThread*().interrupt(); // re-interrupt the thread  
 }  
  
 // print task name  
 System.*out*.printf("%s done sleeping%n", taskName);  
 }  
} // end class PrintTask

**Результат выполнения программы**

***Задание 3.***

Задача: Calculate() выполняет трудоемкие вычисления

Цель: Улучшить отзывчивость программы

Средство: Разделить по потокам пользовательский интерфейс (консольное приложение) и трудоемкие вычисления

После запуска вычислений в отдельном потоке предоставить пользователю выбор: выполнить все вычисления или преждевременно завершить вычисления. Протестировать оба варианта.

***Отчет:***

package org.example;  
  
import java.util.Scanner;  
import java.util.concurrent.atomic.AtomicInteger;  
  
public class ThreadTest2 {  
  
 private static final int *START* = 1;  
 private static final int *END* = 1\_000\_000\_000;  
  
 private static AtomicInteger *total* = new AtomicInteger(0);  
 private static volatile boolean *stopFlag* = false;  
  
 private static class CountPrimesThread extends Thread {  
 private int count = 0;  
 private int min, max;  
  
 public CountPrimesThread(int min, int max) {  
 this.min = min;  
 this.max = max;  
 }  
  
 public void run() {  
 count = *countPrimes*(min, max);  
 *total*.addAndGet(count);  
 System.*out*.println("There are " + count + " primes between " + min + " and " + max);  
 }  
 }  
  
 private static void calculate(int numberOfThreads) {  
 int increment = (*END* - *START*) / numberOfThreads;  
 System.*out*.println("Counting primes between " + (*START* + 1) + " and " + (*END*) + " using " + numberOfThreads + " threads...");  
 long startTime = System.*currentTimeMillis*();  
 CountPrimesThread[] worker = new CountPrimesThread[numberOfThreads];  
 for (int i = 0; i < numberOfThreads; i++)  
 worker[i] = new CountPrimesThread(*START* + i \* increment + 1, *START* + (i + 1) \* increment);  
 *total*.set(0);  
 for (int i = 0; i < numberOfThreads; i++)  
 worker[i].start();  
  
 // Ожидаем нажатия клавиши Enter для прерывания программы  
 System.*out*.println("Press Enter to stop the calculation...");  
 Thread stopThread = new Thread(() -> {  
 Scanner scanner = new Scanner(System.*in*);  
 scanner.nextLine();  
 *stopFlag* = true;  
 });  
 stopThread.start();  
  
 for (int i = 0; i < numberOfThreads; i++) {  
 try {  
 worker[i].join();  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
  
 long elapsedTime = System.*currentTimeMillis*() - startTime;  
 System.*out*.println("The number of primes found is " + *total*.get() + ".");  
 System.*out*.println("Total elapsed time: " + (elapsedTime / 1000.0) + " seconds.");  
 }  
  
 public static void main(String[] args) {  
 int processors = Runtime.*getRuntime*().availableProcessors();  
 if (processors == 1)  
 System.*out*.println("Your computer has only 1 available processor.");  
 else  
 System.*out*.println("Your computer has " + processors + " available processors.");  
  
 int numberOfThreads = 0;  
 while (numberOfThreads < 1 || numberOfThreads > 6) {  
 System.*out*.print("How many threads do you want to use (from 1 to 6)? ");  
 numberOfThreads = new Scanner(System.*in*).nextInt();  
 if (numberOfThreads < 1 || numberOfThreads > 6)  
 System.*out*.println("Please enter a number from 1 to 6!");  
 }  
  
 *calculate*(numberOfThreads);  
 }  
  
 private static int countPrimes(int min, int max) {  
 int count = 0;  
 for (int i = min; i <= max; i++) {  
 if (*stopFlag*) {  
 break;  
 }  
 if (*isPrime*(i)) {  
 count++;  
 }  
 }  
 return count;  
 }  
  
 private static boolean isPrime(int x) {  
 if (x < 2) {  
 return false;  
 }  
 for (int i = 2; i <= Math.*sqrt*(x); i++) {  
 if (x % i == 0) {  
 return false;  
 }  
 }  
 return true;  
 }  
}