|  |
| --- |
| https://lh6.googleusercontent.com/QcftzNtI05T0Y6fjdSh1Rr2rt8oqZ1IvnLvbn1jLJ7CCyteVir3k-xBLv4SL1wAgWJsRhmmJSR0UW-RP63_GQenE4vVWv05BRoZTsmIcBccVTnfxwmsnNMvjg599x9SqZd8E3dkd |
| МИНОБРНАУКИ РОССИИ |
| Федеральное государственное бюджетное образовательное учреждение высшего образования  **«МИРЭА - Российский технологический университет»**  **РТУ МИРЭА** |

Институт информационных технологий (ИТ)

Кафедра инструментального и прикладного программного обеспечения (ИиППО)

|  |  |
| --- | --- |
| **ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ №1** | |
| **по дисциплине** | |
| «Разработка клиент-серверных приложений»  на тему  **«Примитивы синхронизации и их использование при создании клиентсерверных приложений »** | |
| Выполнил студент группы ИКБО-24-20 | Ефимцев С.М. |
|  |  |
| Принял ассистент | Зарипов Е.А. |

Практическая работа выполнена «22» сентября 2023 г.

(подпись студента)

Зачтено «\_\_» 2023 г.

(подпись преподавателя)

СОДЕРЖАНИЕ

[Цель работы 2](#__RefHeading___Toc519_3170985454)

[Описание выполнения работы 3](#__RefHeading___Toc521_3170985454)

[Вывод 14](#__RefHeading___Toc523_3170985454)

# **Цель работы**

Цель работы заключается в разработке программ на Java с использованием многопоточности.

# Задание

Задание 1

Дан массив из 10000 элементов. Необходимо написать несколько реализаций некоторой функции F в зависимости от варианта. Функция должна быть реализована следующими способами:

1. Последовательно
2. С использованием многопоточности (Thread, Future, и т. д.)
3. С использованием ForkJoin. После каждой операции с элементом массива (сравнение, сложение) добавить задержку в 1 мс при помощи Thread.sleep(1); Провести сравнительный анализ затрат по времени и памяти при запуске каждого из вариантов реализации. Варианты функций (выбор варианта осуществляется по формуле «Номер в списке группы % 3»)
4. Поиск суммы элементов массива.
5. Поиск максимального элемента в массиве.
6. Поиск минимального элемента в массиве.

Задание 2

Программа запрашивает у пользователя на вход число. Программа имитирует обработку запроса пользователя в виде задержки от 1 до 5 секунд выводит результат: число, возведенное в квадрат. В момент выполнения запроса пользователь имеет возможность отправить новый запрос. Реализовать с использованием Future.

Задание 3

1. Реализовать следующую многопоточную систему. Файл. Имеет следующие характеристики:

* Тип файла (например XML, JSON, XLS)
* Размер файла — целочисленное значение от 10 до 100.
* Генератор файлов -- генерирует файлы с задержкой от 100 до 1000 мс.
* Очередь — получает файлы из генератора. Вместимость очереди — 5 файлов.
* Обработчик файлов — получает файл из очереди. Каждый обработчик имеет параметр — тип файла, который он может обработать. Время обработки файла: «Размер файла\*7мс»

Система должна удовлетворять следующими условиям:

* Должна быть обеспечена потокобезопасность.
* Работа генератора не должна зависеть от работы обработчиков, и наоборот.
* Если нет задач, то потоки не должны быть активны.
* Если нет задач, то потоки не должны блокировать другие потоки.
* Должна быть сохранена целостность данных.

# Описание выполнения работы

**Задание 1**

В соответствии с вариантом было выбрано задание 2 — Поиск максимального элемента в массиве.

В листингах 1 и 2 представлен программный код для последовательной реализации части 1 задания 1.

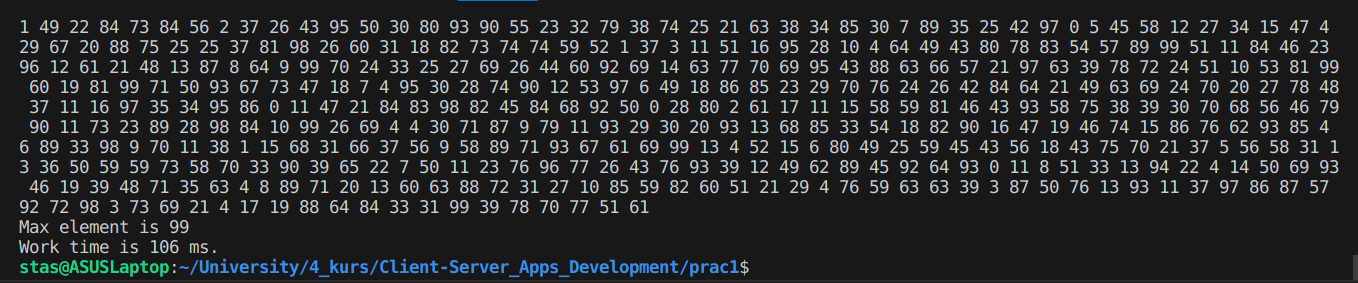
Листинг 1 — Файл task1/task1\_1/Main.java

|  |
| --- |
| package task1.task1\_1;  public class Main {  public static void main(String[] args) {  long startTime = System.currentTimeMillis();  ArrayStuff obj = new ArrayStuff();  obj.printArray();  System.out.println("\nMax element is " + obj.maxElemSearch());  long endTime = System.currentTimeMillis();  System.out.println("Work time is " + (endTime - startTime) + " ms.");  }  } |

Листинг 2 — Файл task1/task1\_1/ArrayStuff.java

|  |
| --- |
| package task1.task1\_1;  import java.util.Random;  public class ArrayStuff extends Thread {  final int SIZE = 10000;  int[] randomIntArray = new int[SIZE];  int upperBound = 100;  long workTime;  ArrayStuff() {  Random random = new Random();  for (int i = 0; i < SIZE; i++) {  this.randomIntArray[i] = random.nextInt(this.upperBound);  }  }  public void printArray() {  for (int i = 0; i < SIZE; i++) {  System.out.print(this.randomIntArray[i] + " ");  }  }  public int maxElemSearch() {  int biggest = 0;  for (int i = 0; i < SIZE; i++) {  if (this.randomIntArray[i] > biggest)  biggest = this.randomIntArray[i];  }  return biggest;  }  } |

Результат работы программы части 1 задания 1 показан на рисунке 1.

Рисунок 1 — Результат работы программы части 1 задания 1

В листингах 3, 4 и 5 представлен программый код части 2 задания 2.

Листинг 3 — Файл task1/task1\_2/Main.java

|  |
| --- |
| package task1.task1\_2;  import java.util.Arrays;  import java.util.Random;  public class Main {  public static void main(String[] args) {  final int SIZE = 10000;  int[] randomIntArray = new int[SIZE];  int upperBound = 100;  final int threadsNumber = 2;  Random random = new Random();  for (int i = 0; i < SIZE; i++) {  randomIntArray[i] = random.nextInt(upperBound);  }  int[] firstCrop = Arrays.copyOfRange(randomIntArray, 0, SIZE / threadsNumber);  int[] secondCrop = Arrays.copyOfRange(randomIntArray, SIZE / threadsNumber, SIZE);  FirstThread t1 = new FirstThread(firstCrop, SIZE / threadsNumber, upperBound);  SecondThread t2 = new SecondThread(secondCrop, SIZE / threadsNumber, upperBound);  t1.start();  t2.start();  try {  t1.join();  t2.join();  } catch (InterruptedException e) {  e.printStackTrace();  }  t1.printArray();  t2.printArray();  System.out.println("\nThe biggest element is " + ((t1.biggest > t2.biggest) ? t1.biggest : t2.biggest));  long firstTime = t1.workTime;  System.out.println("\nTime of the first is " + firstTime + " ms.");  long secondTime = t2.workTime;  System.out.println("\nTime of the second is " + secondTime + " ms.");  System.out.println("\nTotal time s is " + (firstTime + secondTime) + " ms.");  }  } |

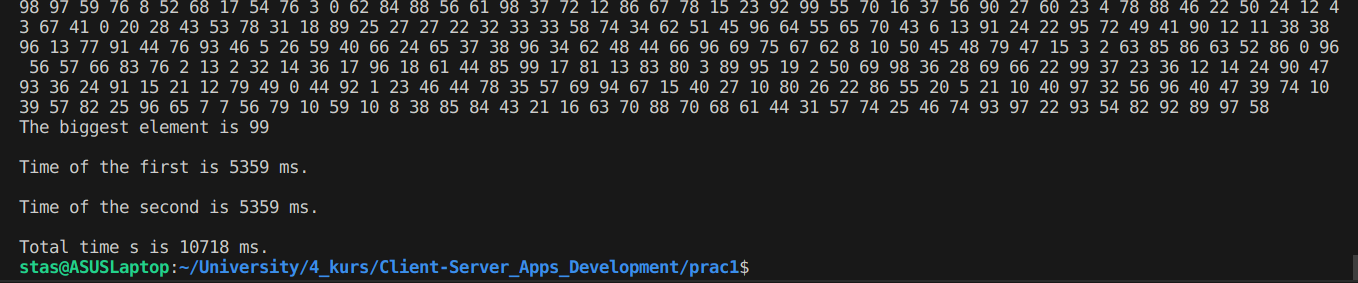
Листинг 4 — Файл task1/task1\_2/FirstThread.java

|  |
| --- |
| package task1.task1\_2;  public class FirstThread extends Thread {  public long workTime;  final int SIZE;  int[] randomIntArray;  int upperBound;  int biggest;  public boolean doStop = false;  public synchronized void doStop() {  this.doStop = true;  }  FirstThread(int[] createdArray, int SIZE, int upperBound) {  this.randomIntArray = createdArray;  this.SIZE = SIZE;  this.upperBound = upperBound;  }  @Override  public void run() {  try {  long startTime = System.currentTimeMillis();  int biggest = 0;  for (int i = 0; i < this.SIZE; i++) {  Thread.sleep(1);  if (this.randomIntArray[i] > biggest)  biggest = randomIntArray[i];  }  System.out.println("The biggest number is " + biggest);  long endTime = System.currentTimeMillis();  this.workTime = endTime - startTime;  this.biggest = biggest;  } catch (InterruptedException e) {  e.printStackTrace();  }  }  public void printArray() {  for (int i = 0; i < SIZE; i++) {  System.out.print(this.randomIntArray[i] + " ");  }  }  } |

Листинг 5 — Файл task1/task1\_2/SecondThread.java

|  |
| --- |
| package task1.task1\_2;  public class SecondThread extends Thread {  long workTime;  final int SIZE;  int[] randomIntArray;  int upperBound;  int biggest;  public boolean doStop = false;  public synchronized void doStop() {  this.doStop = true;  }  SecondThread(int[] createdArray, int SIZE, int upperBound) {  this.randomIntArray = createdArray;  this.SIZE = SIZE;  this.upperBound = upperBound;  }  @Override  public void run() {  try {  long startTime = System.currentTimeMillis();  int biggest = 0;  for (int i = 0; i < this.SIZE; i++) {  Thread.sleep(1);  if (this.randomIntArray[i] > biggest)  biggest = randomIntArray[i];  }  System.out.println("The biggest number is " + biggest);  long endTime = System.currentTimeMillis();  this.workTime = endTime - startTime;  this.biggest = biggest;  } catch (InterruptedException e) {  e.printStackTrace();  }  }  public void printArray() {  for (int i = 0; i < SIZE; i++) {  System.out.print(this.randomIntArray[i] + " ");  }  }  } |

На рисунке 2 показан результат работы программы части 2 задания 2.

Рисунок 2 — Результат работы программы части 2 задания 1

В листингах 6 и 7 представлен программный код части 3 задания 1.

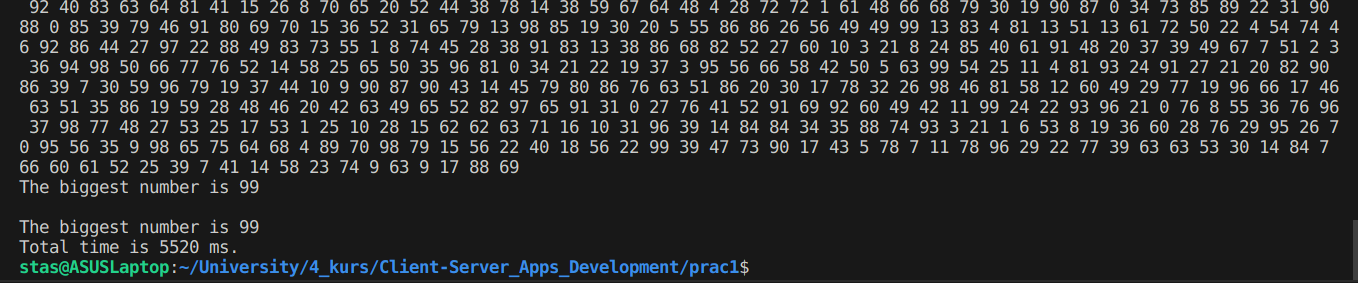
Листинг 6 — Файл task1/task1\_3/Main.java

|  |
| --- |
| package task1.task1\_3;  import java.util.Random;  import java.util.concurrent.ForkJoinPool;  public class Main {  final int SIZE = 10000;  int[] randomIntArray = new int[SIZE];  int upperBound = 100;  public void fillingInArray() {  Random random = new Random();  for (int i = 0; i < SIZE; i++) {  randomIntArray[i] = random.nextInt(upperBound);  }  }  public void printArray() {  for (int i = 0; i < SIZE; i++) {  System.out.print(this.randomIntArray[i] + " ");  }  }  public static void main(String[] args) {  long startTime = System.currentTimeMillis();  MainClass obj = new MainClass();  obj.fillingInArray();  obj.printArray();  ForkJoinPool pool = new ForkJoinPool();  Action action = new Action(obj.randomIntArray, obj.SIZE);  pool.invoke(action);  long endTime = System.currentTimeMillis();  System.out.println("Total time is " + (endTime - startTime) + " ms.");  }  } |

Листинг 6 — Файл task1/task1\_3/Action.java

|  |
| --- |
| package task1.task1\_3;  import java.util.Arrays;  import java.util.concurrent.RecursiveAction;  public class Action extends RecursiveAction {  private int SIZE;  int[] randomIntArray = new int[SIZE];  Action(int[] createdArray, int SIZE) {  this.randomIntArray = createdArray;  this.SIZE = SIZE;  }  @Override  protected void compute() {  if (SIZE <= 5000) {  try {  int biggest = 0;  for (int i = 0; i < this.SIZE; i++) {  Thread.sleep(1);  if (this.randomIntArray[i] > biggest)  biggest = randomIntArray[i];  }  System.out.println("\nThe biggest number is " + biggest);  } catch (InterruptedException e) {  e.printStackTrace();  }  } else {  int[] firstCrop = Arrays.copyOfRange(randomIntArray, 0, SIZE / 2);  int[] secondCrop = Arrays.copyOfRange(randomIntArray, SIZE / 2, SIZE);  Action a1 = new Action(firstCrop, SIZE / 2);  Action a2 = new Action(secondCrop, SIZE / 2);  invokeAll(a1, a2);  }  }  } |

На рисунке 3 показан результат работы программы части 3 задания 2.

Рисунок 3 — Результат работы программы части 3 задания 1

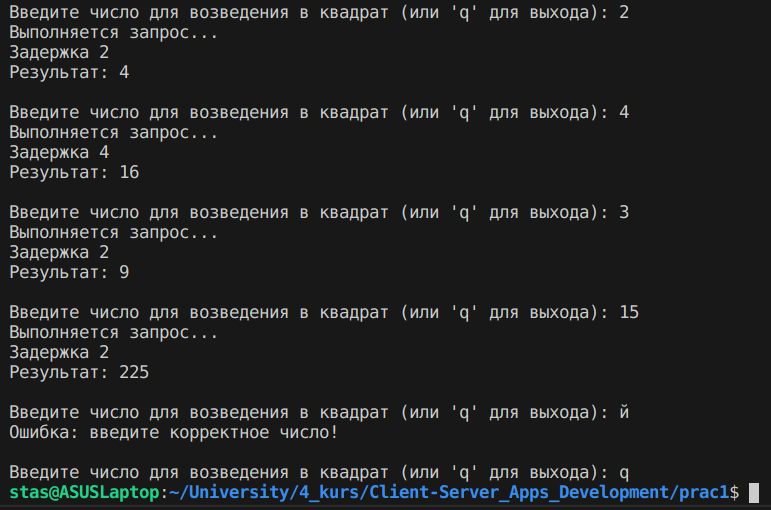
**Задание 2**

В листинге 7 представлен программный код задания 2.

Листинг 7 — Файл task2/Main.java

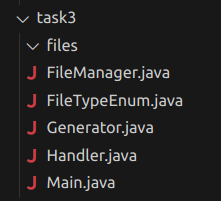
|  |
| --- |
| package task2;  import java.util.Random;  import java.util.Scanner;  import java.util.concurrent.\*;  public class Main {  public static void main(String[] args) {  ExecutorService executorService = Executors.newSingleThreadExecutor();  try {  Scanner sc = new Scanner(System.in);  while (true) {  System.out.print("\nВведите число для возведения в квадрат (или 'q' для выхода): ");  String in = sc.nextLine();  if (in.equals("q")) {  sc.close();  break;  }  try {  int number = Integer.parseInt(in);  Future<Integer> future = executorService.submit(() -> {  Random rand = new Random();  int delay = rand.nextInt(5); // Генерация задержи  System.out.println("Задержка " + delay);  TimeUnit.SECONDS.sleep(delay);  return number \* number;  });  System.out.println("Выполняется запрос...");  while (!future.isDone()) {  }  Object result = future.get();  System.out.println("Результат: " + result);  } catch (NumberFormatException e) {  System.out.println("Ошибка: введите корректное число!");  } catch (InterruptedException | ExecutionException e) {  System.out.println("Ошибка при выполнении запроса: " + e.getMessage());  }  }  } finally {  if (executorService != null) {  executorService.shutdown();  }  }  }  } |

На рисунке 4 представлен результат работы программы задания 2.

Рисунок 4 — Результат работы программы задания 2

**Задание 3**

На рисунке 5 представлена структура файлов задания 3.

Рисунок 5 — Структура файлов задания 3

В листингах 8 — 12 представлен программный код для задания 3.

Листинг 8 — Файл task3/Main.java

|  |
| --- |
| package task3;  import java.util.concurrent.ExecutionException;  public class Main {  public static void main(String[] args) throws InterruptedException, ExecutionException {  new FileManager();  }  } |

Листинг 9 — Файл task3/FileManager.java

|  |
| --- |
| package task3;  import java.io.File;  import java.util.concurrent.ArrayBlockingQueue;  import java.util.concurrent.BlockingQueue;  import java.util.concurrent.ExecutorService;  import java.util.concurrent.Executors;  public class FileManager {  private final String dir = "task3/files/";  FileManager() {  BlockingQueue<File> blockQueue = new ArrayBlockingQueue<>(5);  ExecutorService generatorPool = Executors.newFixedThreadPool(8);  ExecutorService handlerPool = Executors.newFixedThreadPool(8);  int fileCount = 0;  String fileName = "";  while (fileCount < 10) {  fileName = Integer.toString(fileCount);  Generator fileGenerator = new Generator(fileName, dir, blockQueue);  generatorPool.submit(fileGenerator);  Handler fileHandler = new Handler(blockQueue);  handlerPool.submit(fileHandler);  fileCount++;  }  generatorPool.shutdown();  handlerPool.shutdown();  }  } |

Листинг 10 — Файл task3/FileTypeEnum.java

|  |
| --- |
| package task3;  public enum FileTypeEnum {  XML("xml"),  JSON("json"),  XLS("xls");  private final String type;  FileTypeEnum(String type) {  this.type = type;  }  @Override  public String toString() {  return type;  }  } |

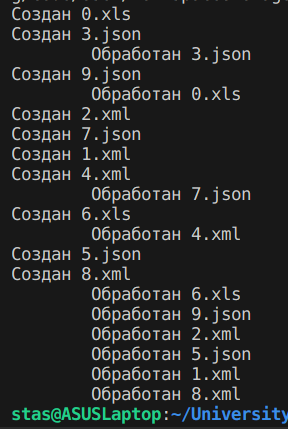
Листинг 11 — Файл task3/Generator.java

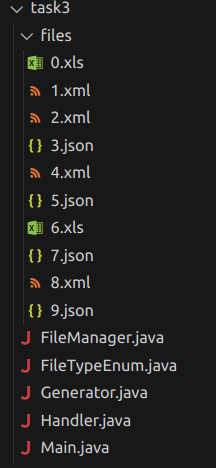
|  |
| --- |
| package task3;  import java.io.File;  import java.io.IOException;  import java.io.RandomAccessFile;  import java.util.concurrent.BlockingQueue;  import java.util.concurrent.TimeUnit;  public class Generator implements Runnable {  private String fileName;  private String dir;  private BlockingQueue<File> blockQueue;  public Generator(String fileName, String dir, BlockingQueue<File> blockQueue) {  this.fileName = fileName;  this.dir = dir;  this.blockQueue = blockQueue;  }  @Override  public void run() {  try {  int delay = (int) (Math.random() \* 900 + 100);  TimeUnit.MILLISECONDS.sleep(delay);  } catch (InterruptedException e) {  e.printStackTrace();  }  File file;  int fileType = (int) (Math.random() \* FileTypeEnum.values().length);  synchronized (Thread.currentThread()) {  file = new File(dir + fileName + "." + FileTypeEnum.values()[fileType]);  }  try {  file.createNewFile();  System.out.println("Создан " + file.getName());  RandomAccessFile raFile = new RandomAccessFile(file, "rw");  raFile.setLength((int) (Math.random() \* 90 + 10));  raFile.close();  blockQueue.put(file);  } catch (IOException | InterruptedException e) {  e.printStackTrace();  }  }  } |

Листинг 12 — Файл task3/Handler.java

|  |
| --- |
| package task3;  import java.io.File;  import java.util.concurrent.BlockingQueue;  public class Handler implements Runnable {  private BlockingQueue<File> fileBlockingQueue;  public Handler(BlockingQueue<File> fileBlockingQueue) {  this.fileBlockingQueue = fileBlockingQueue;  }  @Override  public void run() {  try {  File file;  file = fileBlockingQueue.take();  Thread.sleep(file.length() \* 7);  System.out.println("\tОбработан " + file.getName());  } catch (InterruptedException e) {  e.printStackTrace();  }  }  } |

На рисунках 6 и 7 представлен результат работы программы задания 3.

Рисунок 6 — Результат работы программы задания 3

Рисунок 7 — Созданные в процессе работы программы задания 3 файлы

# Вывод

В процессе выполнения данной практической работы были получены навыки и умения работы работы с многопоточностью в Java, использованы такие инструмменты, как Thread, Future, Fork/Join, ExecutorService и другие.