**Разработка клиент-серверных приложений**

**Зверев А.А. ИКБО-20-21**

**Практическая работа 1**

**Задание 1**

В соответствии с вариантом была решена задача поиска суммы элементов массива. Разработаны реализации на основе последовательного и многопоточного выполнения, а также с применением ForkJoin. В листинге 1.1 представлен код реализации всего задания.

Листинг 1.1 – Код к заданию 1

|  |
| --- |
| package PR1;  import java.util.ArrayList; import java.util.List; import java.util.Random; import java.util.concurrent.\*;  //Zverev public class Ex1 {  //Zverev  public static List<Integer> generateArray10000() { // генерация 10000 рандомных чисел типа Integer в листе  List<Integer> list = new ArrayList<>();  Random random = new Random();  for (int i = 0; i < 10000; i++) {  int randomNumber = random.nextInt();  list.add(randomNumber);  }  return list;  }  //Zverev  public static int findSum(List<Integer> list) throws  InterruptedException {  if (list == null || list.isEmpty()) {  throw new IllegalArgumentException("Список пуст или равен null");  }  int sum = 0;  for (int number : list) {  Thread.sleep(1);  sum += number;  }  return sum;  }  //Zverev  public static int findSumMnogopotok(List<Integer> list) throws  InterruptedException, ExecutionException {  if (list == null || list.isEmpty()) {  throw new IllegalArgumentException("Список пуст или равен null");  }  // Определяем количество доступных процессоров  int numberOfThreads = Runtime.getRuntime().availableProcessors();  System.out.println("Количество доступных процессоров: " + numberOfThreads);  // Создаем пул потоков для выполнения задач  ExecutorService executorService =  Executors.newFixedThreadPool(numberOfThreads);  // Создаем список задач для каждого подсписка  List<Callable<Integer>> tasks = new ArrayList<>();  int batchSize = list.size() / numberOfThreads;   // Разбиваем список на подсписки и создаем задачи для каждого подсписка  for (int i = 0; i < numberOfThreads; i++) {  final int startIndex = i \* batchSize;  final int endIndex = (i == numberOfThreads - 1) ? list.size() : (i + 1) \* batchSize;  tasks.add(() -> findSumInRange(list.subList(startIndex, endIndex)));  }  // Запускаем все задачи и получаем Future объекты для получения результатов  List<Future<Integer>> futures = executorService.invokeAll(tasks);  // Инициализируем переменную для хранения суммы  int sum = 0;  // Обходим результаты каждой задачи и находим сумму  for (Future<Integer> future : futures) {  int partialSum = future.get();  Thread.sleep(1);  sum += partialSum;  }  // Завершаем пул потоков  executorService.shutdown();  // Возвращаем сумму из всех подсписков  return sum;  }  //Zverev  // Функция для поиска суммы в подсписке  private static int findSumInRange(List<Integer> sublist) throws  InterruptedException {  int sum = 0;  for (int number : sublist) {  Thread.sleep(1);  sum += number;  }  return sum;  }  //Zverev  public static int findSumFork(List<Integer> list) {  // Проверяем, что список не пуст и не равен null  if (list == null || list.isEmpty()) {  throw new IllegalArgumentException("Список пуст или равен null");  }  // Создаем пул потоков ForkJoin  ForkJoinPool forkJoinPool = new ForkJoinPool();  // Создаем корневую задачу SumFinderTask для всего списка  SumFinderTask task = new SumFinderTask(list, 0, list.size());  // Выполняем корневую задачу и получаем результат  return forkJoinPool.invoke(task);  }  //Zverev  // Внутренний класс SumFinderTask, расширяющий RecursiveTask для многопоточного выполнения  static class SumFinderTask extends RecursiveTask<Integer> {  private List<Integer> list;  private int start;  private int end;  //Zverev  // Конструктор SumFinderTask для создания задачи для подсписка  SumFinderTask(List<Integer> list, int start, int end) {  this.list = list;  this.start = start;  this.end = end;  }  //Zverev  // Метод compute(), выполняющий вычисления для задачи  @Override  protected Integer compute() {  // Если в подсписке не более 1000 элементов, завершаем разбиение  if (end - start <= 1000) {  try {  return findSumInRange(list.subList(start, end));  } catch (InterruptedException e) {  throw new RuntimeException(e);  }  }  // Найдем середину подсписка  int middle = start + (end - start) / 2;  // Создаем две подзадачи для левой и правой половин подсписка  SumFinderTask leftTask = new SumFinderTask(list, start, middle);  SumFinderTask rightTask = new SumFinderTask(list, middle, end);  // Запускаем подзадачу для левой половины параллельно  leftTask.fork();  // Вычисляем сумму в левой и правой половинах подсписка  int rightResult = rightTask.compute();  int leftResult = leftTask.join();  try {  Thread.sleep(1);  } catch (InterruptedException e) {  throw new RuntimeException(e);  }  // Возвращаем сумму из левой и правой половин  return leftResult + rightResult;  }  //Zverev  public static void main(String[] args) throws InterruptedException,  ExecutionException {  List<Integer> testList = generateArray10000();   long startTime = System.nanoTime();  int result = findSum(testList);  long endTime = System.nanoTime();  long durationInMilliseconds = (endTime - startTime) / 1\_000\_000;  System.out.println("Время выполнения последовательной функции: " +  durationInMilliseconds + " миллисекунд. Результат - "+result);   startTime = System.nanoTime();  result = findSumMnogopotok(testList);  endTime = System.nanoTime();  durationInMilliseconds = (endTime - startTime) / 1\_000\_000;  System.out.println("Время выполнения многопоточной функции: " +  durationInMilliseconds + " миллисекунд. Результат - "+result);   startTime = System.nanoTime();  result = findSumFork(testList);  endTime = System.nanoTime();  durationInMilliseconds = (endTime - startTime) / 1\_000\_000;  System.out.println("Время выполнения форк функции: " +  durationInMilliseconds + " миллисекунд. Результат - "+result);  System.out.println("Автор: " + "Зверев А.А. ИКБО-20-21");  }  }} |

В результате запуска программы получен ответ, представленный на рисунке 1.1.

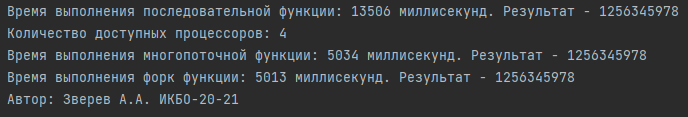


Рисунок 1.1 – Выполнение задания 1

**Задание 2**

В соответствии с заданием была реализована программа возведения пользовательского числа в квадрат. В листинге 1.2 представлен код реализации задания.

Листинг 1.2 – Код к заданию 2

|  |
| --- |
| package PR1;  import java.util.Scanner; import java.util.concurrent.\*;  //Zverev public class Ex2 {  //Zverev  private static CompletableFuture<Integer> calculateSquare(int number) {  // Запуск асинхронной задачи  return CompletableFuture.supplyAsync(() -> {  int delayInSeconds = ThreadLocalRandom.current().nextInt(1, 6);  try {  Thread.sleep(delayInSeconds \* 1000);  } catch (InterruptedException e) {  Thread.currentThread().interrupt();  }  return number \* number;  });  }   //Zverev  public static void main(String[] args) {   // Создаем пул потоков  ExecutorService executorService = Executors.newFixedThreadPool(5);   // Создаем бесконечный цикл для обработки запросов  while (true) {  try {  System.out.println("Введите число (или 'exit' для выхода): ");  Scanner scanner = new Scanner(System.in);  String userInput = scanner.nextLine();  if ("exit".equalsIgnoreCase(userInput)) {  break;  }  int number = Integer.parseInt(userInput);   // Прием выполнения асинхронной функции в другом потоке  calculateSquare(number).thenAcceptAsync(result ->  System.out.println("Результат: " + result), executorService)  .exceptionally(e -> {  System.err.println("Ошибка обработки запроса: " + e.getMessage());  return null;  });   } catch (NumberFormatException e) {  System.err.println("Неверный формат числа. Пожалуйста, введите целое число.");  }  }   executorService.shutdown();  System.out.println("Автор: " + "Зверев А.А. ИКБО-20-21");  } } |

В результате запуска программы получен ответ, представленный на рисунке 1.2.

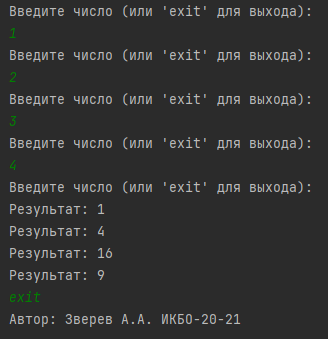


Рисунок 1.2 – Выполнение задания 2

**Задание 3**

В соответствии с заданием была реализована многопоточная программа обработки файлов. В листинге 1.3 представлен код реализации классов файлов, генератора файлов и обработчика файлов.

Листинг 1.3 – Код классов файлов, генератора файлов и обработчика файлов

|  |
| --- |
| package PR1;  import java.util.Random; import java.util.concurrent.BlockingQueue; //Zverev public class File {  private String fileType;  private int fileSize;  public File(String fileType, int fileSize) {  this.fileType = fileType;  this.fileSize = fileSize;  }  public String getFileType() {  return fileType;  }  public int getFileSize() {  return fileSize;  } } //Zverev class FileGenerator implements Runnable {  private BlockingQueue<File> queue;  public FileGenerator(BlockingQueue<File> queue) {  this.queue = queue;  }  @Override  public void run() {  Random random = new Random();  String[] fileTypes = {"XML", "JSON", "XLS"};  while (true) {  try {  Thread.sleep(random.nextInt(901) + 100); // Задержка от 100 до 1000 мс  String randomFileType =  fileTypes[random.nextInt(fileTypes.length)];  int randomFileSize = random.nextInt(91) + 10; // Размер файла от 10 до 100  File file = new File(randomFileType, randomFileSize);  queue.put(file); // Добавляем файл в очередь  } catch (InterruptedException e) {  Thread.currentThread().interrupt();  break;  }  }  } } //Zverev class FileProcessor implements Runnable {  private BlockingQueue<File> queue;  private String allowedFileType;  public FileProcessor(BlockingQueue<File> queue, String allowedFileType) {  this.queue = queue;  this.allowedFileType = allowedFileType;  }  //Zverev  @Override  public void run() {  while (true) {  try {  File file = queue.take(); // Получаем файл из очереди  if (file.getFileType().equals(allowedFileType)) {  long processingTime = file.getFileSize() \* 7; // Время обработки  Thread.sleep(processingTime);  System.out.println("Обработан файл типа " +  file.getFileType() +  " с размером " + file.getFileSize() + ". Время обработки: " +  processingTime + " мс.");  }  } catch (InterruptedException e) {  Thread.currentThread().interrupt();  break;  }  }  } } |

В листинге 1.4 представлен главный класс программы, использующий вышеприведенные классы для реализации заданной функциональности.

Листинг 1.4 – MultithreadedFileProcessing.java

|  |
| --- |
| package PR1;  import java.util.concurrent.BlockingQueue; import java.util.concurrent.LinkedBlockingQueue; public class MultithreadedFileProcessing {  //Zverev  public static void main(String[] args) {  BlockingQueue<File> queue = new LinkedBlockingQueue<>(5);  // Очередь вместимостью 5 файлов  // Создаем генератор файлов и обработчики для разных типов файлов  Thread generatorThread = new Thread(new FileGenerator(queue));  Thread jsonProcessorThread = new Thread(new FileProcessor(queue, "JSON"));  Thread xmlProcessorThread = new Thread(new FileProcessor(queue, "XML"));  Thread xlsProcessorThread = new Thread(new FileProcessor(queue, "XLS"));  // Запускаем потоки  generatorThread.start();  jsonProcessorThread.start();  xmlProcessorThread.start();  xlsProcessorThread.start();  System.out.println("Автор: " + "Зверев А.А. ИКБО-20-21");  } } |

В результате запуска программы получен ответ, представленный на рисунке 1.3.

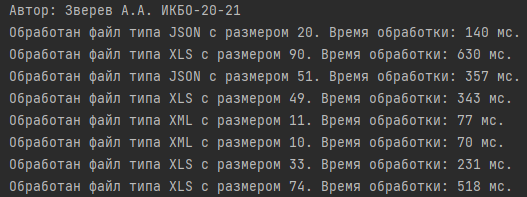


Рисунок 1.3 – Выполнение задания 3

**Практическая работа 2**

**Задание 1**

В соответствии с заданием была реализована программа считывания данных из файла с помощью пакета java.nio. В листинге 2.1 представлен код реализации всего задания.

Листинг 2.1 – ReadTextFile.java

|  |
| --- |
| package PR2;  import java.io.IOException; import java.nio.file.Files; import java.nio.file.Path; import java.nio.file.Paths; import java.util.List; public class ReadTextFile {  //Zverev  public static void main(String[] args) {  String fileName = "sample.txt"; // Имя файла  // Создаем несколько строк текста  String[] lines = {  "Это первая строка текста.",  "Зверев А.А.",  "ИКБО-20-21"  };  // Записываем строки в файл  writeLinesToFile(fileName, lines);  // Читаем содержимое файла и выводим его в стандартный поток вывода  readAndPrintFileContent(fileName);  }  //Zverev  private static void writeLinesToFile(String fileName, String[] lines) {  Path filePath = Paths.get(fileName);  try {  Files.write(filePath, List.of(lines));  System.out.println("Файл успешно создан: " + fileName);  } catch (IOException e) {  System.err.println("Ошибка при записи в файл: " + e.getMessage());  }  }  //Zverev  private static void readAndPrintFileContent(String fileName) {  Path filePath = Paths.get(fileName);  try {  List<String> fileLines = Files.readAllLines(filePath);  System.out.println("Содержимое файла " + fileName + ":");  for (String line : fileLines) {  System.out.println(line);  }  } catch (IOException e) {  System.err.println("Ошибка при чтении файла: " + e.getMessage());  }  } } |

В результате запуска программы получен ответ, представленный на рисунке 2.1.

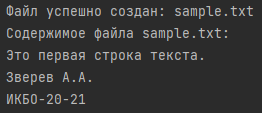


Рисунок 2.1 – Выполнение задания 1

**Задание 2**

В соответствии с заданием было реализовано копирование файлов различными средствами и проведено их сравнение. В листинге 2.2 представлен код реализации задания.

Листинг 2.2 – Код к заданию 2

|  |
| --- |
| package PR2;  import java.io.\*; import java.nio.channels.FileChannel; import java.nio.file.Files; import java.nio.file.Path; import java.nio.file.StandardCopyOption; import org.apache.commons.io.FileUtils; public class FileCopyComparison {  //Zverev  public static void main(String[] args) throws IOException {   String sourceFile = "source.txt"; // Имя исходного файла  String destinationFile = "destination.txt"; // Имя файла назначения  // Создание файла размером 100 МБ  createLargeFile(sourceFile, 100);   // Метод 1: FileInputStream/FileOutputStream  long startTime1 = System.currentTimeMillis();  copyUsingFileStreams(sourceFile, destinationFile);  long endTime1 = System.currentTimeMillis();  printTimeAndMemoryUsage("FileInputStream/FileOutputStream", startTime1,  endTime1);   // Метод 2: FileChannel  long startTime2 = System.currentTimeMillis();  copyUsingFileChannel(sourceFile, destinationFile);  long endTime2 = System.currentTimeMillis();  printTimeAndMemoryUsage("FileChannel", startTime2, endTime2);   // Метод 3: Apache Commons IO  long startTime3 = System.currentTimeMillis();  copyUsingApacheCommonsIO(sourceFile, destinationFile);  long endTime3 = System.currentTimeMillis();  printTimeAndMemoryUsage("Apache Commons IO", startTime3, endTime3);   // Метод 4: Files class  long startTime4 = System.currentTimeMillis();  copyUsingFilesClass(sourceFile, destinationFile);  long endTime4 = System.currentTimeMillis();  printTimeAndMemoryUsage("Files class", startTime4, endTime4);  System.out.println("Автор: " + "Зверев А.А. ИКБО-20-21");  }  //Zverev  private static void createLargeFile(String fileName, int sizeInMB) throws  IOException {  byte[] data = new byte[1024 \* 1024]; // 1 МБ буфер  FileOutputStream fos = new FileOutputStream(fileName);  for (int i = 0; i < sizeInMB; i++) {  fos.write(data);  }  fos.close();  }  //Zverev  private static void copyUsingFileStreams(String source, String destination)  throws IOException {  FileInputStream fis = new FileInputStream(source);  FileOutputStream fos = new FileOutputStream(destination);  byte[] buffer = new byte[1024];  int bytesRead;  while ((bytesRead = fis.read(buffer)) != -1) {  fos.write(buffer, 0, bytesRead);  }  fis.close();  fos.close();  }  //Zverev  private static void copyUsingFileChannel(String source, String destination)  throws IOException {  FileInputStream fis = new FileInputStream(source);  FileOutputStream fos = new FileOutputStream(destination);  FileChannel sourceChannel = fis.getChannel();  FileChannel destinationChannel = fos.getChannel();  sourceChannel.transferTo(0, sourceChannel.size(), destinationChannel);  sourceChannel.close();  destinationChannel.close();  fis.close();  fos.close();  }  //Zverev  private static void copyUsingApacheCommonsIO(String source, String  destination) throws IOException {  File sourceFile = new File(source);  File destFile = new File(destination);  FileUtils.copyFile(sourceFile, destFile);  }  //Zverev  private static void copyUsingFilesClass(String source, String destination)  throws IOException {  Path sourcePath = Path.of(source);  Path destinationPath = Path.of(destination);  Files.copy(sourcePath, destinationPath,  StandardCopyOption.REPLACE\_EXISTING);  }  //Zverev  private static void printTimeAndMemoryUsage(String method, long startTime,  long endTime)  {  long elapsedTime = endTime - startTime;  System.out.println("Метод " + method + ":");  System.out.println("Время выполнения: " + elapsedTime + " мс");  Runtime runtime = Runtime.getRuntime();  long memoryUsed = runtime.totalMemory() - runtime.freeMemory();  System.out.println("Использование памяти: " + memoryUsed / (1024 \* 1024)  + " МБ");  System.out.println();  } } |

В результате запуска программы получен ответ, представленный на рисунке 2.2.

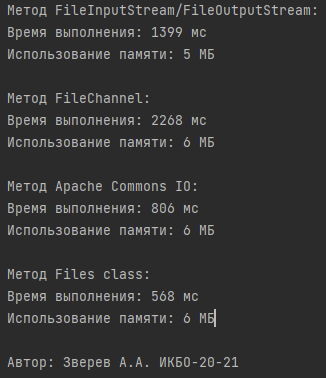


Рисунок 2.2 – Выполнение задания 2

**Задание 3**

В соответствии с заданием была реализована функция нахождения 16-битной контрольной суммы файла с использованием бинарных операций и ByteBuffer. В листинге 2.3 представлен код решения задания.

Листинг 2.3 – ChecksumCalculator.java

|  |
| --- |
| package PR2;  import java.io.FileInputStream; import java.io.IOException; import java.nio.ByteBuffer; import java.nio.channels.FileChannel; public class ChecksumCalculator {  //Zverev  public static short calculateChecksum(String filePath) throws IOException {  try (FileInputStream fileInputStream = new FileInputStream(filePath);  FileChannel fileChannel = fileInputStream.getChannel()) {  ByteBuffer buffer = ByteBuffer.allocate(2); // Создаем буфер для хранения 2 байт  short checksum = 0; //short = 2 байта по памяти  while (fileChannel.read(buffer) != -1) {  buffer.flip(); // Переключаем буфер в режим чтения  while (buffer.hasRemaining()) {  checksum ^= buffer.get(); // Выполняем XOR над байтами по одному  }  buffer.clear(); // Очищаем буфер для следующего чтения  }  return checksum;  }  }  //Zverev  public static void main(String[] args) {  String filePath = "sample.txt"; // Путь к файлу, для которого нужно вычислить контрольную сумму  try {  short checksum = calculateChecksum(filePath);  System.out.printf("Контрольная сумма файла %s: 0x%04X%n", filePath,  checksum);  System.out.printf("Контрольная сумма файла %s: ", filePath);  System.out.print(checksum + "\n");  } catch (IOException e) {  e.printStackTrace();  }  System.out.println("Автор: " + "Зверев А.А. ИКБО-20-21");  } } |

В результате запуска программы получен ответ, представленный на рисунке 2.3.

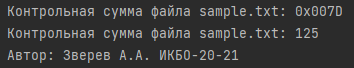


Рисунок 2.3 – Выполнение задания 3

**Задание 4**

В соответствии с заданием было реализовано наблюдение за каталогом с использованием WatchService. В листинге 2.4 представлен код решения задания.

Листинг 2.4 – Код к заданию 4

|  |
| --- |
| package PR2; import java.io.\*; import java.nio.file.\*; import java.security.DigestInputStream; import java.util.ArrayList; import java.util.HashMap; import java.util.List; import java.util.Map; import java.security.MessageDigest; import java.security.NoSuchAlgorithmException; public class DirectoryWatcher {  //Zverev  private static Map<Path, List<String>> fileContentsMap = new HashMap<>();  private static Map<Path, String> fileHashes = new HashMap<>();  //Zverev  public static void main(String[] args) throws IOException,  InterruptedException {  System.out.println("Автор: " + "Зверев А.А. ИКБО-20-21");  Path directory = Paths.get("./pr4\_4");  WatchService watchService = FileSystems.getDefault().newWatchService(); //Класс отслеживания на дефолтной файловой системе  directory.register(watchService, StandardWatchEventKinds.ENTRY\_CREATE,  StandardWatchEventKinds.ENTRY\_MODIFY, StandardWatchEventKinds.ENTRY\_DELETE); // Регистрация кактолога с указанием типов операций  firstObserve(directory); // Первичная проверка полсе начала работы (для выдачи ответа при первом изменении)  while (true) {  WatchKey key = watchService.take();// Ключ, сигнализирующий о событии  for (WatchEvent<?> event : key.pollEvents()) { // Ждем ивенты  WatchEvent.Kind<?> kind = event.kind(); // Тип ивента  if (kind == StandardWatchEventKinds.ENTRY\_CREATE) {  Path filePath = (Path) event.context();  System.out.println("Создан новый файл: " + filePath);  fileContentsMap.put(filePath,  readLinesFromFile(directory.resolve(filePath))); //В хешмапе ключ - файл, значение - список строк (содержание)  calculateFileHash(directory.resolve(filePath)); //Считаем хеш, чтобы при удалении вывести  } else if (kind == StandardWatchEventKinds.ENTRY\_MODIFY) {  Path filePath = (Path) event.context();  System.out.println("Файл изменен: " + filePath);  detectFileChanges(directory.resolve(filePath)); //resolve объединяет пути (относительный файла + текущий)  } else if (kind == StandardWatchEventKinds.ENTRY\_DELETE) {  Path filePath = (Path) event.context();  System.out.println("Удален файл: " + filePath);  String hash = fileHashes.get(directory.resolve(filePath));  if (hash != null) {  System.out.println("Хеш-сумма удаленного файла: " + hash);  }  }  }  key.reset(); //Сброс сохраненных событий  }  }  //Zverev  private static void firstObserve(Path directory) throws IOException {  try (DirectoryStream<Path> stream = Files.newDirectoryStream(directory)) //Перебор каталога по файлам  {  for (Path filePath : stream) {  if (Files.isRegularFile(filePath)) {  fileContentsMap.put(filePath, readLinesFromFile(filePath));  calculateFileHash(filePath);  }  }  }  }  //Zverev  private static void detectFileChanges(Path filePath) throws IOException {  List<String> newFileContents = readLinesFromFile(filePath);  List<String> oldFileContents = fileContentsMap.get(filePath);  if (oldFileContents != null) {  List<String> addedLines = newFileContents.stream()  .filter(line -> !oldFileContents.contains(line))  .toList(); //Обработка через stream для операции над каждой строкой  List<String> deletedLines = oldFileContents.stream()  .filter(line -> !newFileContents.contains(line))  .toList();  if (!addedLines.isEmpty()) {  System.out.println("Добавленные строки в файле " + filePath +  ":");  addedLines.forEach(line -> System.out.println("+ " + line));  }  if (!deletedLines.isEmpty()) {  System.out.println("Удаленные строки из файла " + filePath +  ":");  deletedLines.forEach(line -> System.out.println("- " + line));  }  }  calculateFileHash(filePath);  fileContentsMap.put(filePath, newFileContents);  }  //Zverev  private static List<String> readLinesFromFile(Path filePath) throws  IOException {  List<String> lines = new ArrayList<>();  try (BufferedReader reader = Files.newBufferedReader(filePath)) {  String line;  while ((line = reader.readLine()) != null) {  lines.add(line);  }  }  return lines;  }  //Zverev  private static void calculateFileHash(Path filePath) {  try {  MessageDigest md = MessageDigest.getInstance("MD5"); //Алгоритм хеширования для вычисления контрольной суммы  try (InputStream is = Files.newInputStream(filePath);  DigestInputStream dis = new DigestInputStream(is, md)) {  while (dis.read() != -1) ;  String hash = bytesToHex(md.digest()); //Вычисляем хеш и преобразуем в 16-ое число  fileHashes.put(filePath, hash);  } catch (IOException e) {  e.printStackTrace();  }  } catch (NoSuchAlgorithmException e) { //Если не нашел MD5  e.printStackTrace();  }  }  //Zverev  private static String bytesToHex(byte[] bytes) {  String hashStr = "";  for (byte b : bytes) {  hashStr += String.format("%02x", b);  }  return hashStr;  }} |

В результате запуска программы получен ответ, представленный на рисунках 2.4 – 2.5.

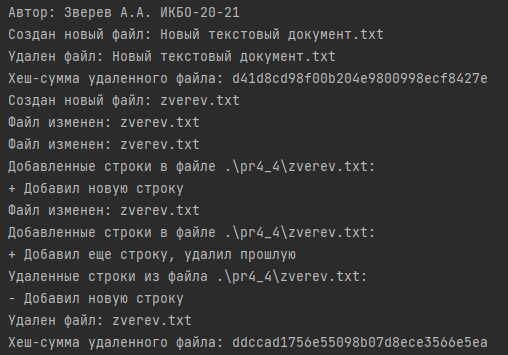


Рисунок 2.4 – Выполнение задания 4, часть 1

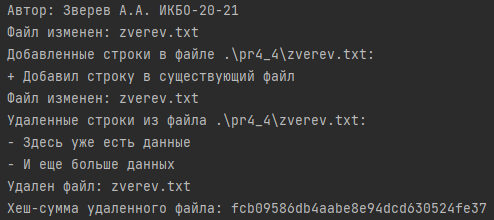


Рисунок 2.5 – Выполнение задания 4, часть 2

**Практическая работа 3**

**Задание 1**

В соответствии с заданием была реализована программа сигнализации на основе показаний датчиков. В листинге 3.1 представлен код реализации всего задания.

Листинг 3.1 – Код к заданию 1

|  |
| --- |
| package PR3; import io.reactivex.Observable; import io.reactivex.Observer; import io.reactivex.disposables.Disposable; import io.reactivex.subjects.PublishSubject; import java.util.Random; // Создаем класс для датчика температуры class TemperatureSensor extends Observable<Integer> {  //Zverev  private final PublishSubject<Integer> subject = PublishSubject.create(); //Создаем датчик температуры  @Override  protected void subscribeActual(Observer<? super Integer> observer) //Любой тип, родительский для int  { //Переопределяем абстрактный класс подписки - бизнес-логика подключения  subject.subscribe(observer); // Создаем подписку на события датчика температуры  }  //Zverev  public void start() {  new Thread(() -> { // В отдельном потоке  while (true) {  int temperature = new Random().nextInt(15, 31); // Генерируем случайное значение температуры  subject.onNext(temperature); // Отправляем значение температуры подписчикам  try {  Thread.sleep(1000); // Пауза 1 секунда  } catch (InterruptedException e) {  e.printStackTrace();  }  }  }).start(); // Запускаем поток для симуляции работы датчика  } } //Zverev // Создаем класс для датчика CO2 class CO2Sensor extends Observable<Integer> {  private final PublishSubject<Integer> subject = PublishSubject.create();  @Override  protected void subscribeActual(Observer<? super Integer> observer) {  subject.subscribe(observer); // Создаем подписку на события датчика CO2  }  public void start() {  new Thread(() -> {  while (true) {  int co2 = new Random().nextInt(30, 101); // Генерируем случайное значение CO2  subject.onNext(co2); // Отправляем значение CO2 подписчикам  try {  Thread.sleep(1000); // Пауза 1 секунда  } catch (InterruptedException e) {  e.printStackTrace();  }  }  }).start(); // Запускаем поток для симуляции работы датчика  } } //Zverev // Создаем класс для сигнализации class Alarm implements Observer<Integer> {  private final int CO2\_LIMIT = 70;  private final int TEMP\_LIMIT = 25;  private int temperature = 0;  private int co2 = 0;  @Override  public void onSubscribe(Disposable d) { //На вход переменная управления подпиской  System.out.println(d.hashCode() + " has subscribed");  }  @Override  public void onNext(Integer value) {  System.out.println("Next value from Observable = " + value);  if (value <= 30){ //На основании неперекрытия возможных значений источников  temperature = value;  } else {  co2 = value;  }  checkingSystem();  }//Zverev  public void checkingSystem(){  if (temperature >= TEMP\_LIMIT && co2 >= CO2\_LIMIT){  System.out.println("ALARM!!! Temperature/CO2: " + temperature + "/"  + co2);  } else if (temperature >= TEMP\_LIMIT){  System.out.println("Temperature warning: " + temperature);  } else if (co2 >= CO2\_LIMIT){  System.out.println("CO2 warning: " + co2);  }  }  @Override  public void onError(Throwable e) {  e.printStackTrace();  }  @Override  public void onComplete() {  System.out.println("Completed");  } } //Zverev class Main {  public static void main(String[] args) {  System.out.println("Автор: " + "Зверев А.А. ИКБО-20-21");  TemperatureSensor temperatureSensor = new TemperatureSensor(); // Создаем датчик температуры  CO2Sensor co2Sensor = new CO2Sensor(); // Создаем датчик CO2  Alarm alarm = new Alarm(); // Создаем сигнализацию  temperatureSensor.subscribe(alarm); // Подписываем сигнализацию на датчик температуры  co2Sensor.subscribe(alarm); // Подписываем сигнализацию на датчик CO2  temperatureSensor.start(); // Запускаем датчик температуры  co2Sensor.start(); // Запускаем датчик CO2}} |

В результате запуска программы получен ответ, представленный на рисунке 3.1.

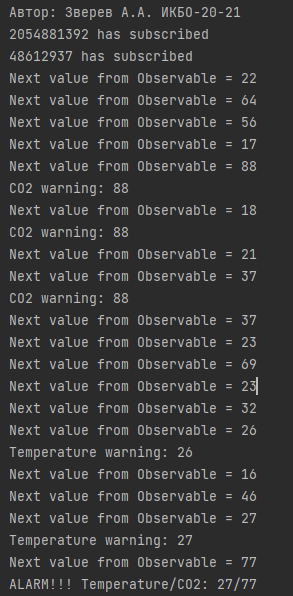


Рисунок 3.1 – Выполнение задания 1

**Задание 2**

В соответствии с вариантом были выполнены задачи 2.1.2, 2.2.2 и 2.3.2, в рамках которых реализовано преобразование потоков чисел. В листинге 3.2 представлен код реализации задания.

Листинг 3.2 – Код к заданию 2

|  |
| --- |
| package PR3;  import io.reactivex.Observable; import java.util.Random;  //Zverev public class Task2\_1\_2 { //Номер в списке 10, Вариант 2 (10 % 3 + 1 = 2)   public static void main(String[] args) {   Random random = new Random();  System.out.println("Автор: " + "Зверев А.А. ИКБО-20-21");  System.out.println("Номер в списке: 10; " + "Вариант: 2");   //Задача 2.1.2  Observable<Integer> source = Observable.range(0, 1000) // 1000 значений в потоке  .map(i -> random.nextInt(1001)) // Генерируем случайные числа от 0 до 1000  .filter(number -> number > 500); // Применяем оператор filter для фильтрации чисел больше 500   source.subscribe(num -> System.out.println("Число > 500: " + num));   } } //Zverev class Task2\_2\_2 {   public static void main(String[] args) {   Random random = new Random();  System.out.println("Автор: " + "Зверев А.А. ИКБО-20-21");  System.out.println("Номер в списке: 10; " + "Вариант: 2");   //Задача 2.2.2  Observable<Integer> stream1 = Observable.range(0, 1000)  .map(i -> random.nextInt(101)); //Малые числа для 1 потока  Observable<Integer> stream2 = Observable.range(0, 1000)  .map(i -> random.nextInt(901, 1001)); //Большие числа для 2 потока  Observable<Integer> mergedStream = Observable.concat(stream1, stream2); //Объединяем оба потока последовательно   mergedStream.subscribe(num -> System.out.println("Объединенный поток: " + num));   } } //Zverev class Task2\_3\_2 {   public static void main(String[] args) {   Random random = new Random();  System.out.println("Автор: " + "Зверев А.А. ИКБО-20-21");  System.out.println("Номер в списке: 10; " + "Вариант: 2");   //Задача 2.3.2  Observable<Integer> source = Observable.range(0, 10); //10 последовательных чисел в потоке  Observable<Integer> result = source.take(5); // Берем только первые пять элементов    result.subscribe(num -> System.out.println("Первые 5 чисел: " + num));   } } |

В результате запуска программы получен ответ, представленный на рисунках 3.2 – 3.5.

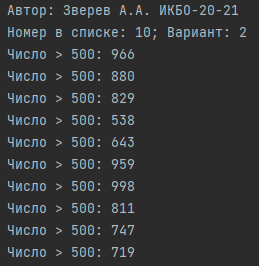


Рисунок 3.2 – Выполнение задания 2.1.2

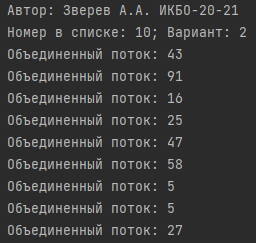


Рисунок 3.3 – Выполнение задания 2.2.2, часть 1

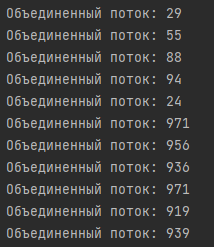


Рисунок 3.4 – Выполнение задания 2.2.2, часть 2

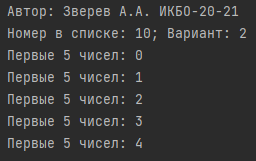


Рисунок 3.5 – Выполнение задания 2.3.2

**Задание 3**

В соответствии с заданием был реализован класс UserFriend, функция получения друзей getFriends и алгоритм преобразования потока чисел в поток объектов UserFriend. В листинге 3.3 представлен код решения задания.

Листинг 3.3 – UserFriend.java

|  |
| --- |
| package PR3;  import io.reactivex.Observable;  public class UserFriend {  //Zverev  private final int userId;  private final int friendId;  public UserFriend(int userId, int friendId) {  this.userId = userId;  this.friendId = friendId;  }  public int getUserId() {  return userId;  }  public int getFriendId() {  return friendId;  }  public static Observable<UserFriend> getFriends(int userId) { //Получение друзей  //Берем данные из массива  return Observable.fromArray(  new UserFriend(userId, (int) (Math.random() \* 100)),  new UserFriend(userId, (int) (Math.random() \* 100)),  new UserFriend(userId, (int) (Math.random() \* 100))  );  }  //Zverev  public static void main(String[] args) {  System.out.println("Автор: " + "Зверев А.А. ИКБО-20-21");  // Создаем массив случайных userId  Integer[] userIdArray = {1, 2, 3, 4, 5};  // Создаем поток userId из массива  Observable<Integer> userIdStream = Observable.fromArray(userIdArray);  // Преобразуем поток userId в поток объектов UserFriend  Observable<UserFriend> userFriendStream = userIdStream  .flatMap(userId -> getFriends(userId)); // Используем функцию getFriends  // Подписываемся на поток и выводим результат  userFriendStream.subscribe(userFriend -> {  System.out.println("User: " + userFriend.getUserId() + ", Friend: "  + userFriend.getFriendId());  });  } } |

В результате запуска программы получен ответ, представленный на рисунке 3.6.

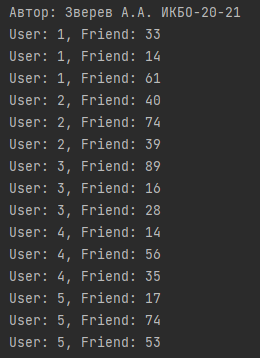


Рисунок 3.6 – Выполнение задания 3

**Задание 4**

В соответствии с заданием была реализована программа обработки файлов с применением инструментов RxJava. В листинге 3.4 представлен код решения задания.

Листинг 3.4 – FileProcessingSystem.java

|  |
| --- |
| package PR3;  import io.reactivex.Completable; import io.reactivex.Observable; import io.reactivex.schedulers.Schedulers;  //Zverev // Класс File представляет файлы с типом и размером class File {  private final String fileType;  private final int fileSize;  public File(String fileType, int fileSize) {  this.fileType = fileType;  this.fileSize = fileSize;  }  public String getFileType() {  return fileType;  }  public int getFileSize() {  return fileSize;  } } //Zverev // Генератор файлов class FileGenerator {  // Генерирует файлы асинхронно с задержкой  public Observable<File> generateFile() {  return Observable  .fromCallable(() -> { //Из функции c возвратом значения  try {  String[] fileTypes = {"XML", "JSON", "XLS"};  String fileType = fileTypes[(int) (Math.random() \* 3)];  int fileSize = (int) (Math.random() \* 91) + 10;  Thread.sleep((long) (Math.random() \* 901) + 100); // Имитация генерации файла  return new File(fileType, fileSize);  } catch (InterruptedException e) {  throw new RuntimeException(e);  }  })  .repeat() // Повторяем бесконечно  .subscribeOn(Schedulers.io()) // Выполняется в фоновом потоке  .observeOn(Schedulers.io()); // Результаты наблюдаются в фоновом потоке  } } //Zverev // Очередь файлов class FileQueue {  private final int capacity;  private final Observable<File> fileObservable;  // Создает очередь с заданной вместимостью и подключается к генератору файлов  public FileQueue(int capacity) {  this.capacity = capacity;  this.fileObservable = new FileGenerator().generateFile()  .replay(capacity) // При подключении обработчика выдает набор последних файлов  .autoConnect(); // Включается автоматически при подписке обработчика  }  // Получает наблюдаемый поток файлов  public Observable<File> getFileObservable() {  return fileObservable;  } } //Zverev // Обработчик файлов class FileProcessor {  private final String supportedFileType;  // Создает обработчик файлов для определенного типа файлов  public FileProcessor(String supportedFileType) {  this.supportedFileType = supportedFileType;  }  // Обрабатывает файлы асинхронно с задержкой  public Completable processFiles(Observable<File> fileObservable) {  return fileObservable  .filter(file -> file.getFileType().equals(supportedFileType)) // Фильтрует файлы по типу  .flatMapCompletable(file -> { // Преобразует в Completable  long processingTime = file.getFileSize() \* 7; // Вычисляет время обработки  return Completable  .fromAction(() -> {  Thread.sleep(processingTime); // Имитация обработки файла  System.out.println("Processed " +  supportedFileType + " file with size " + file.getFileSize());  }).subscribeOn(Schedulers.io()) // Выполняется в фоновом потоке  .observeOn(Schedulers.io()); // Результаты наблюдаются в фоновом потоке  }).onErrorComplete(); // Игнорирует ошибки и завершает успешно  } } //Zverev // Основной класс системы обработки файлов public class FileProcessingSystem {  public static void main(String[] args) {  System.out.println("Автор: " + "Зверев А.А. ИКБО-20-21");  int queueCapacity = 5;  FileQueue fileQueue = new FileQueue(queueCapacity);  String[] supportedFileTypes = {"XML", "JSON", "XLS"};  for (String fileType : supportedFileTypes) {  new FileProcessor(fileType)  .processFiles(fileQueue.getFileObservable()) // Подключаем обработчик  .subscribe(() -> {}, // Обработка успешного завершения (OnNext)  throwable -> System.err.println("Error processing file: " + throwable)); //OnError  }  // Даем системе время для работы  try {  Thread.sleep(10000); // Работает 10 секунд  } catch (InterruptedException e) {  e.printStackTrace();  }  } } |

В результате запуска программы получен ответ, представленный на рисунке 3.7.

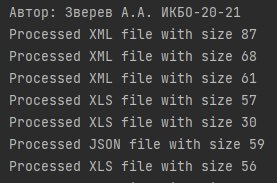


Рисунок 3.7 – Выполнение задания 4

В результате работы сформирован GitHub-репозиторий, в котором представлен исходный код решения практических работ. Репозиторий находится по ссылке: <https://github.com/B-A-Jl-E-N-O-K/rksp_2.git>