|  |  |
| --- | --- |
|  | Министерство образования и науки Российской Федерации |
| Федеральное государственное автономное образовательное учреждение высшего образования |
| «Уральский федеральный университет имени первого Президента России Б. Н. Ельцина» |
| Институт радиоэлектроники и информационных технологий – РтФ |
|  |
|  | |

**ОТЧЕТ**

О ВЫПОЛНЕНИИ

ЛАБОРАТОНОЙ РАБОТЫ № 7

Руководитель ст. пр. Н. А. Архипов

Студент гр. РИМ-130971 Е.В. Треглазов

Екатеринбург 2024

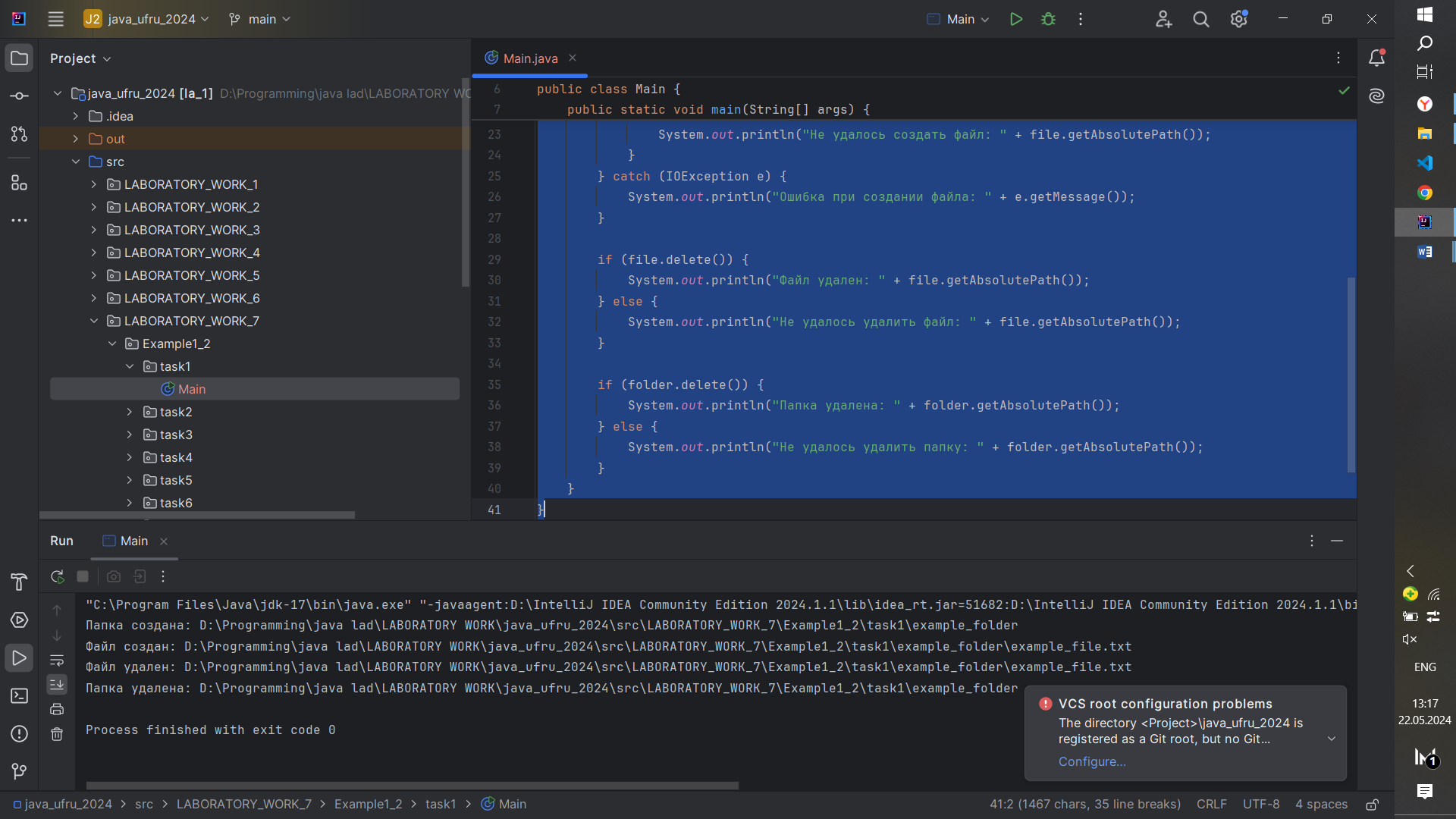
Репозиторий: https://github.com/kuk86kuk/java\_ufru\_2024.

Example1\_2

Task1

public class Main {  
 public static void main(String[] args) {  
 File folder = new File("src/LABORATORY\_WORK\_7/Example1\_2/task1/example\_folder");  
 if (!folder.exists()) {  
 if (folder.mkdir()) {  
 System.*out*.println("Папка создана: " + folder.getAbsolutePath());  
 } else {  
 System.*out*.println("Не удалось создать папку: " + folder.getAbsolutePath());  
 }  
 } else {  
 System.*out*.println("Папка уже существует: " + folder.getAbsolutePath());  
 }  
 File file = new File(folder.getAbsolutePath() + File.*separator* + "example\_file.txt");  
 try {  
 if (file.createNewFile()) {  
 System.*out*.println("Файл создан: " + file.getAbsolutePath());  
 } else {  
 System.*out*.println("Не удалось создать файл: " + file.getAbsolutePath());  
 }  
 } catch (IOException e) {  
 System.*out*.println("Ошибка при создании файла: " + e.getMessage());  
 }  
  
 if (file.delete()) {  
 System.*out*.println("Файл удален: " + file.getAbsolutePath());  
 } else {  
 System.*out*.println("Не удалось удалить файл: " + file.getAbsolutePath());  
 }  
  
 if (folder.delete()) {  
 System.*out*.println("Папка удалена: " + folder.getAbsolutePath());  
 } else {  
 System.*out*.println("Не удалось удалить папку: " + folder.getAbsolutePath());  
 }  
 }  
}

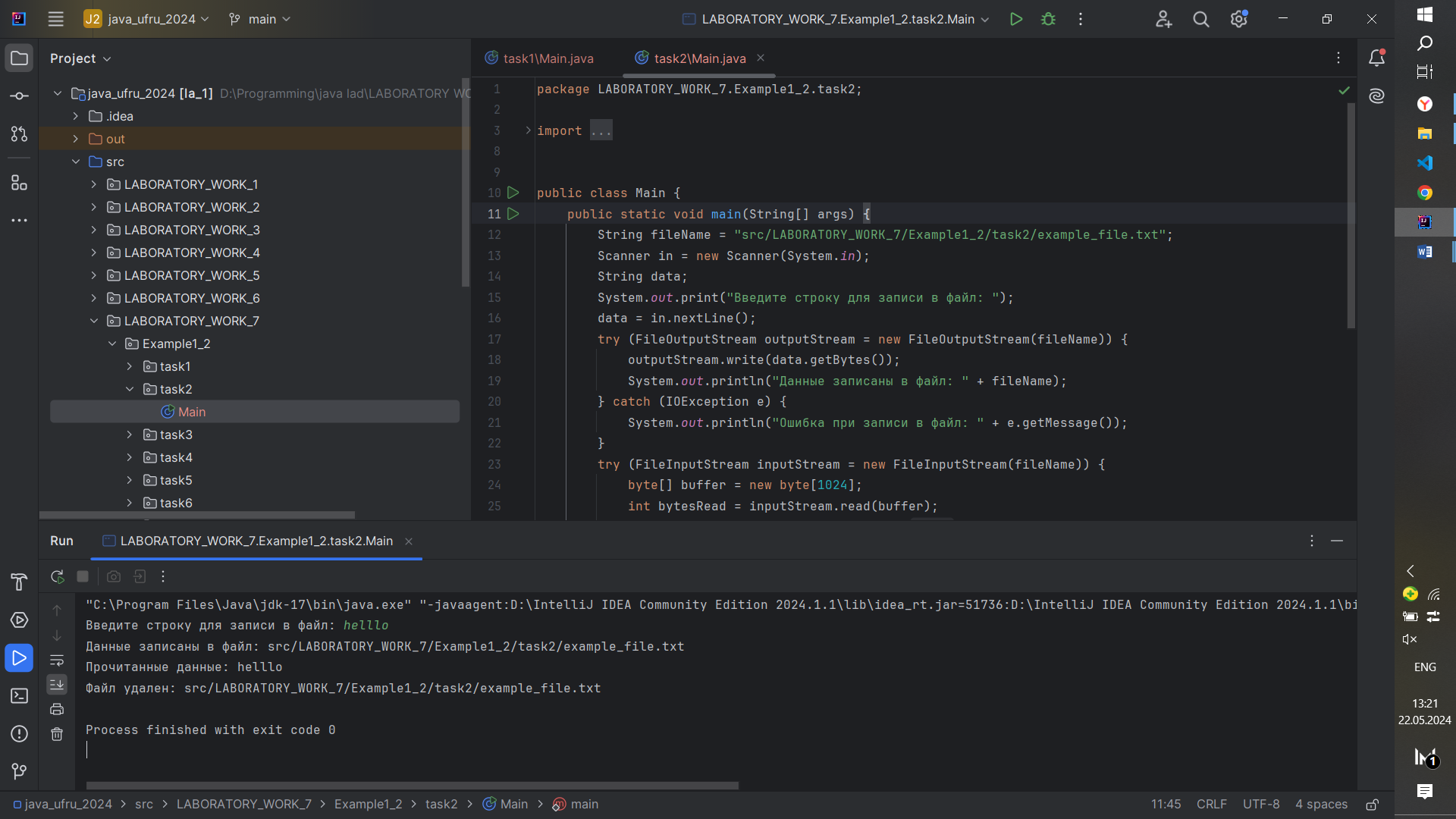
Программа создает папку **example\_folder** в директории src/LABORATORY\_WORK\_7/Example1\_2/task1/example\_folder, а затем создает файл **example\_file.txt** внутри этой папки. Затем программа удаляет файл и папку и выводит ход выполнения программы в консоль. Обратите внимание, что программа проверяет, существует ли папка или файл, прежде чем пытаться их создать или удалить.



Task2

public class Main {  
 public static void main(String[] args) {  
 String fileName = "src/LABORATORY\_WORK\_7/Example1\_2/task2/example\_file.txt";  
 Scanner in = new Scanner(System.*in*);  
 String data;  
 System.*out*.print("Введите строку для записи в файл: ");  
 data = in.nextLine();  
 try (FileOutputStream outputStream = new FileOutputStream(fileName)) {  
 outputStream.write(data.getBytes());  
 System.*out*.println("Данные записаны в файл: " + fileName);  
 } catch (IOException e) {  
 System.*out*.println("Ошибка при записи в файл: " + e.getMessage());  
 }  
 try (FileInputStream inputStream = new FileInputStream(fileName)) {  
 byte[] buffer = new byte[1024];  
 int bytesRead = inputStream.read(buffer);  
 String readData = new String(buffer, 0, bytesRead);  
 System.*out*.println("Прочитанные данные: " + readData);  
 } catch (IOException e) {  
 System.*out*.println("Ошибка при чтении файла: " + e.getMessage());  
 }  
 if (new File(fileName).delete()) {  
 System.*out*.println("Файл удален: " + fileName);  
 } else {  
 System.*out*.println("Не удалось удалить файл: " + fileName);  
 }  
 in.close();  
 }  
}

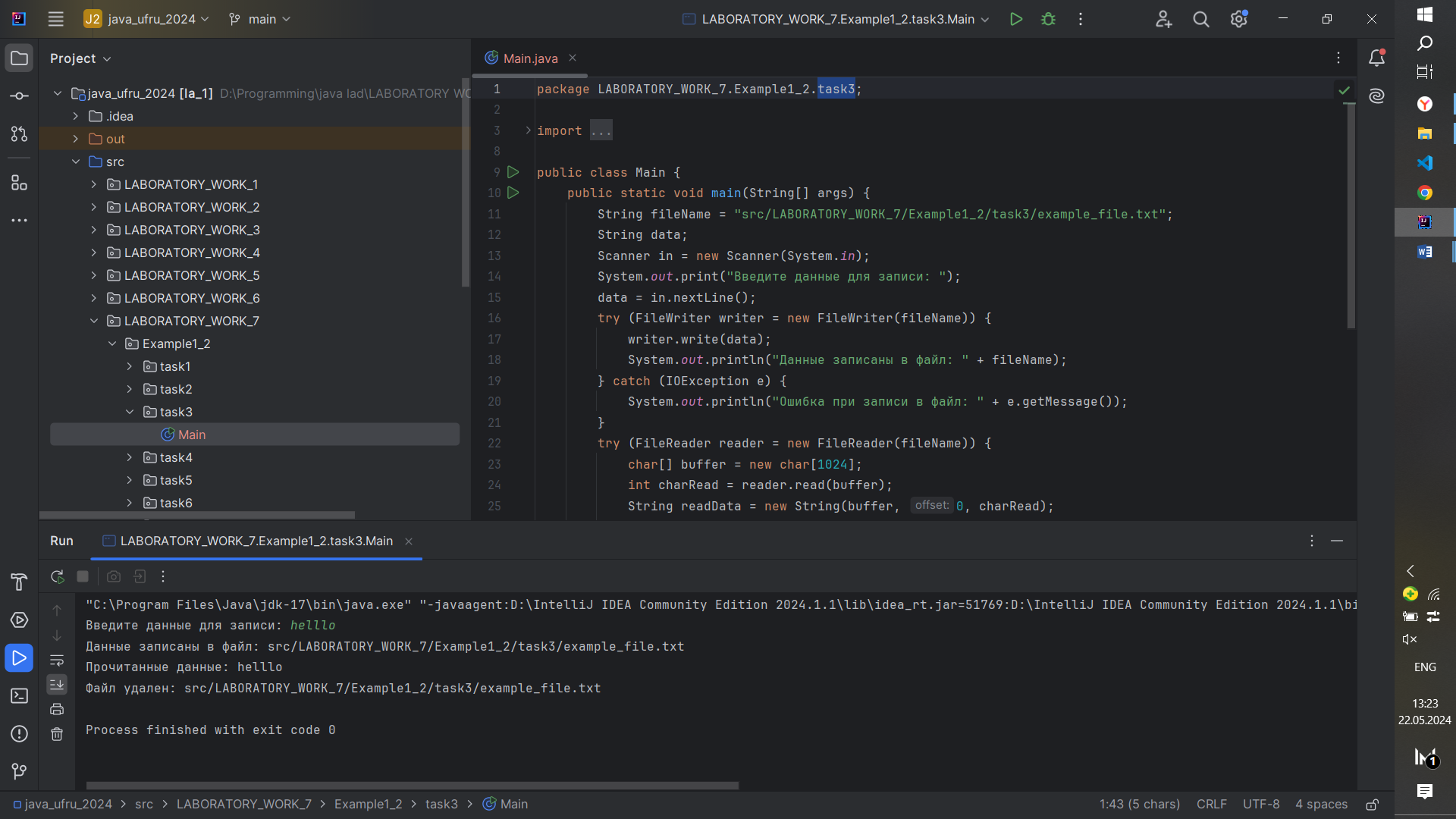
Программа записывает строку **data** в файл **example\_file.txt**, используя байтовый вывод **FileOutputStream**. Затем программа читает данные из файла, используя байтовый ввод **FileInputStream**, и выводит их в консоль. Затем программа удаляет файл и выводит ход выполнения программы в консоль. Обратите внимание, что программа использует конструкцию try-with-resources для автоматического закрытия потоков после их использования.



Task3

public class Main {  
 public static void main(String[] args) {  
 String fileName = "src/LABORATORY\_WORK\_7/Example1\_2/task3/example\_file.txt";  
 String data;  
 Scanner in = new Scanner(System.*in*);  
 System.*out*.print("Введите данные для записи: ");  
 data = in.nextLine();  
 try (FileWriter writer = new FileWriter(fileName)) {  
 writer.write(data);  
 System.*out*.println("Данные записаны в файл: " + fileName);  
 } catch (IOException e) {  
 System.*out*.println("Ошибка при записи в файл: " + e.getMessage());  
 }  
 try (FileReader reader = new FileReader(fileName)) {  
 char[] buffer = new char[1024];  
 int charRead = reader.read(buffer);  
 String readData = new String(buffer, 0, charRead);  
 System.*out*.println("Прочитанные данные: " + readData);  
  
 } catch (IOException e) {  
 System.*out*.println("Ошибка при чтении файла: " + e.getMessage());  
 }  
 if (new File(fileName).delete()) {  
 System.*out*.println("Файл удален: " + fileName);  
 } else {  
 System.*out*.println("Не удалось удалить файл: " + fileName);  
 }  
 in.close();  
 }  
}

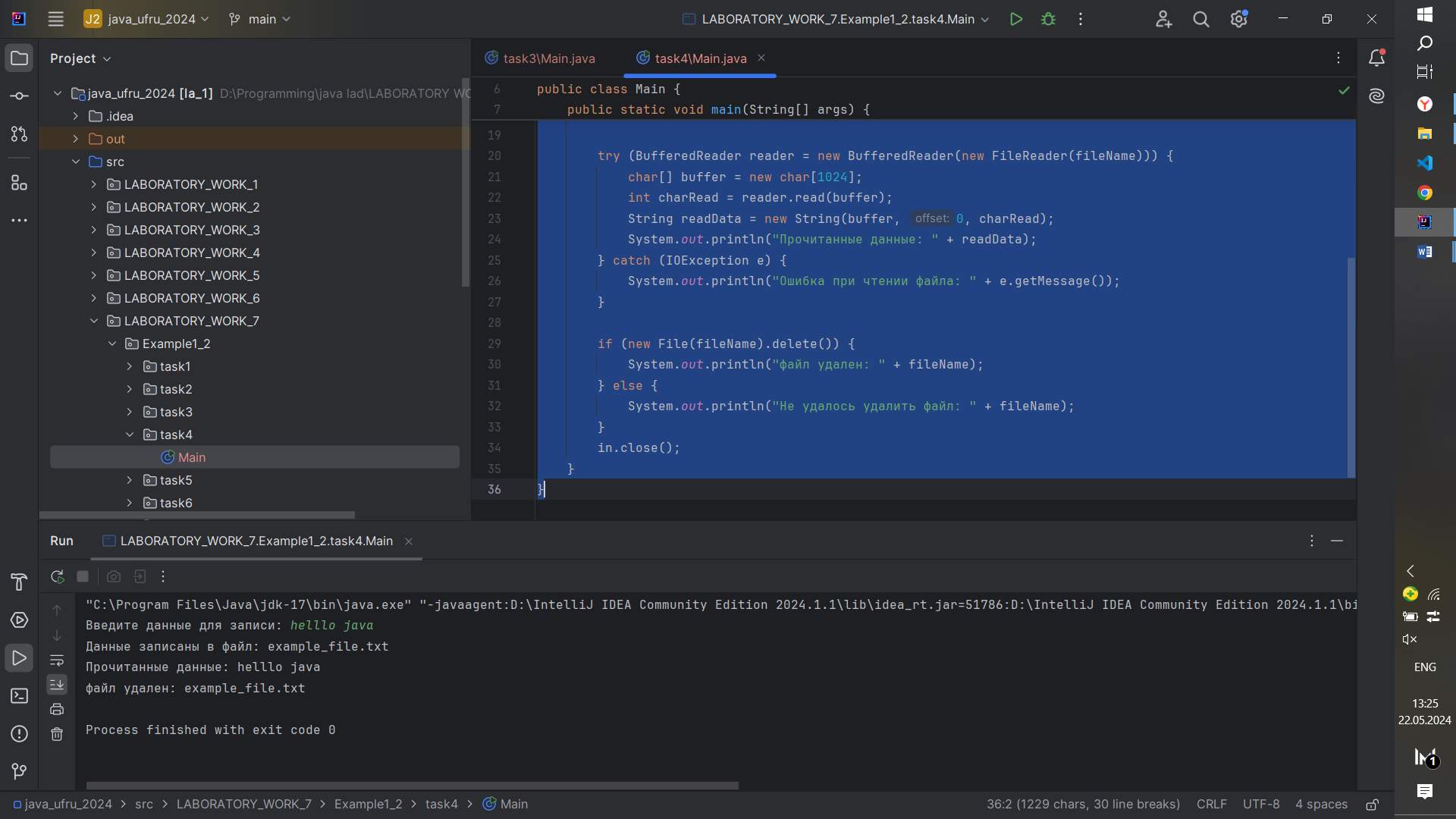
Программа записывает строку **data** в файл **example\_file.txt**, используя **FileWriter**. Затем программа читает данные из файла, используя **FileReader**, и выводит их в консоль. Затем программа удаляет файл и выводит ход выполнения программы в консоль. Обратите внимание, что программа использует конструкцию try-with-resources для автоматического закрытия потоков после их использования.



Task4

public class Main {  
 public static void main(String[] args) {  
 String fileName = "example\_file.txt";  
 String data;  
 Scanner in = new Scanner(System.*in*);  
 System.*out*.print("Введите данные для записи: ");  
 data = in.nextLine();  
 try (BufferedWriter writer = new BufferedWriter(new FileWriter(fileName))) {  
 writer.write(data);  
 System.*out*.println("Данные записаны в файл: " + fileName);  
 } catch (IOException e) {  
 System.*out*.println("Ошибка при записи в файл: " + fileName);  
 }  
  
 try (BufferedReader reader = new BufferedReader(new FileReader(fileName))) {  
 char[] buffer = new char[1024];  
 int charRead = reader.read(buffer);  
 String readData = new String(buffer, 0, charRead);  
 System.*out*.println("Прочитанные данные: " + readData);  
 } catch (IOException e) {  
 System.*out*.println("Ошибка при чтении файла: " + e.getMessage());  
 }  
  
 if (new File(fileName).delete()) {  
 System.*out*.println("файл удален: " + fileName);  
 } else {  
 System.*out*.println("Не удалось удалить файл: " + fileName);  
 }  
 in.close();  
 }  
}

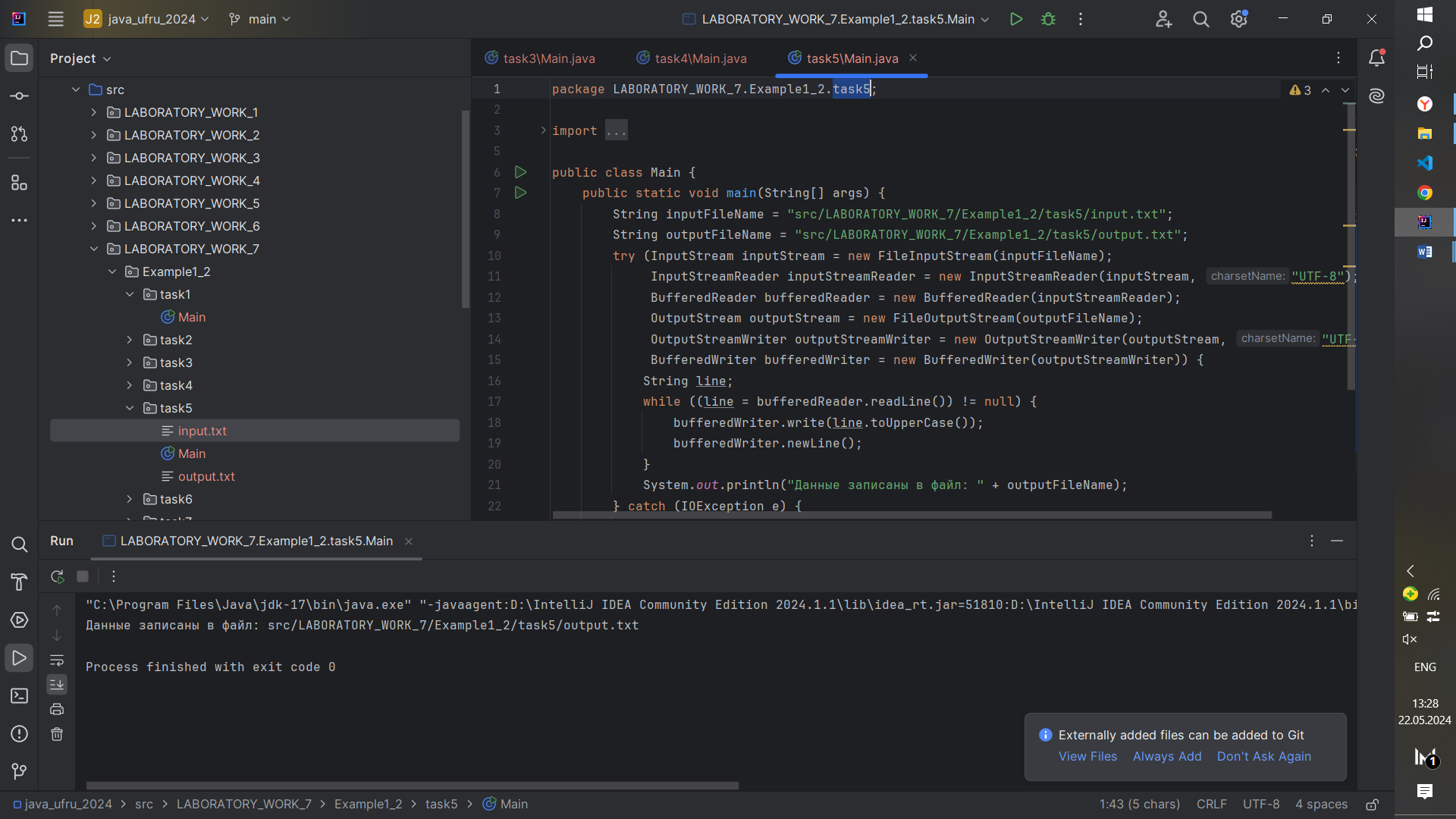
Программа использует буферизацию ввода/вывода, передавая ссылку на **FileReader** и **FileWriter** в качестве параметров в конструкторы **BufferedReader** и **BufferedWriter**. Затем программа записывает строку **data** в файл **example\_file.txt**, используя **BufferedWriter**. Затем программа читает данные из файла, используя **BufferedReader**, и выводит их в консоль. Затем программа удаляет файл и выводит ход выполнения программы в консоль. Обратите внимание, что программа использует конструкцию try-with-resources для автоматического закрытия потоков после их использования.



Task5

public class Main {  
 public static void main(String[] args) {  
 String inputFileName = "src/LABORATORY\_WORK\_7/Example1\_2/task5/input.txt";  
 String outputFileName = "src/LABORATORY\_WORK\_7/Example1\_2/task5/output.txt";  
 try (InputStream inputStream = new FileInputStream(inputFileName);  
 InputStreamReader inputStreamReader = new InputStreamReader(inputStream, "UTF-8");  
 BufferedReader bufferedReader = new BufferedReader(inputStreamReader);  
 OutputStream outputStream = new FileOutputStream(outputFileName);  
 OutputStreamWriter outputStreamWriter = new OutputStreamWriter(outputStream, "UTF-8");  
 BufferedWriter bufferedWriter = new BufferedWriter(outputStreamWriter)) {  
 String line;  
 while ((line = bufferedReader.readLine()) != null) {  
 bufferedWriter.write(line.toUpperCase());  
 bufferedWriter.newLine();  
 }  
 System.*out*.println("Данные записаны в файл: " + outputFileName);  
 } catch (IOException e) {  
 System.*out*.println("Ошибка при чтении/записи файла: " + e.getMessage());  
 }  
 }  
}

Программа читает данные из файла **input.txt**, используя **FileInputStream**, **InputStreamReader** и **BufferedReader**, которые позволяют прочитать данные в правильной кодировке (в данном случае UTF-8). Затем программа записывает преобразованные данные в файл **output.txt**, используя **FileOutputStream**, **OutputStreamWriter** и **BufferedWriter**. При записи данных программа преобразует каждую строку в верхний регистр с помощью метода **toUpperCase()**, а затем добавляет новую строку с помощью метода **newLine()**. Как только запись завершена, программа выводит ход выполнения программы в консоль.



Task6

public class Main {  
 public static void main(String[] args) {  
 String inputFileName = "src/LABORATORY\_WORK\_7/Example1\_2/task6/input.txt";  
 String outputFileName = "src/LABORATORY\_WORK\_7/Example1\_2/task6/output.txt";  
 try (BufferedReader bufferedReader = new BufferedReader(new FileReader(inputFileName));  
 PrintWriter printWriter = new PrintWriter(outputFileName, "UTF-8")) {  
 String line;  
 while ((line = bufferedReader.readLine()) != null) {  
 printWriter.println(line.toUpperCase());  
 }  
 System.*out*.println("Данные записаны в файл: " + outputFileName);  
 } catch (IOException e) {  
 System.*out*.println("Ошибка при чтении/записи файла: " + e.getMessage());  
 }  
 }  
}

Программа, выполняет следующие действия:

Чтение данных из файла:

* Используется FileReader для открытия файла input.txt.
* BufferedReader применяется для эффективного чтения данных из файла по одной строке за раз. Это позволяет уменьшить количество операций ввода-вывода, что повышает производительность.

Преобразование данных:

* Каждая прочитанная строка преобразуется в верхний регистр с помощью метода toUpperCase(). Это делается для изменения всех символов строки на их верхний регистр аналог.

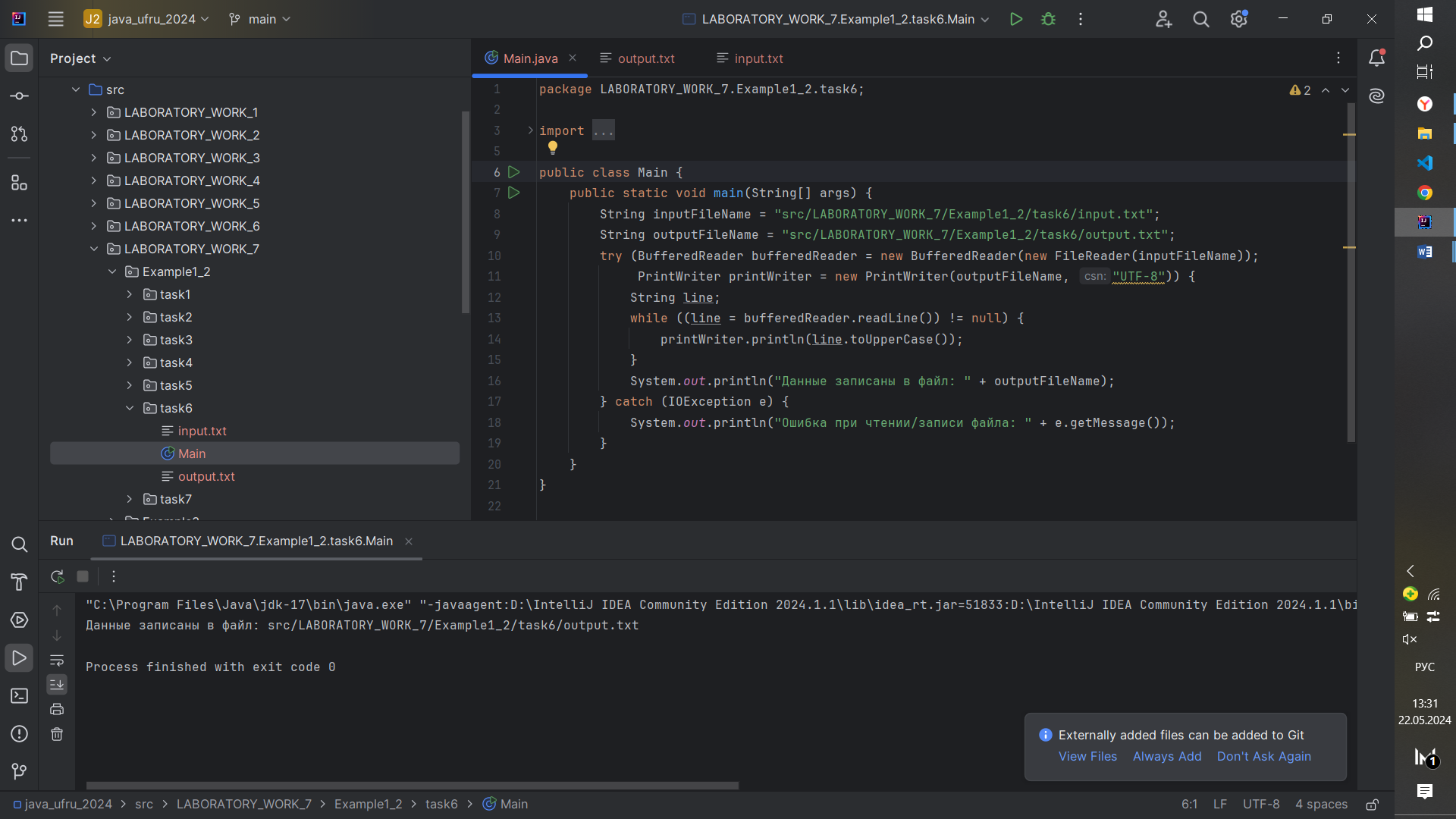
Запись данных в файл:

* PrintWriter используется для записи преобразованных данных в файл output.txt. Этот класс удобен для записи текстовых данных и поддерживает автоматическое закрытие потока при использовании в конструкции try-with-resources.
* При записи каждая строка завершается символом новой строки (println()).

Обработка исключений:

* Программа обрабатывает возможные исключения IOException, которые могут возникнуть при чтении или записи файла. В случае ошибки выводится сообщение, содержащее детали ошибки.

После успешной записи данных в файл, программа выводит сообщение в консоль, указывающее на успешное выполнение операции и имя выходного файла.



Task7

public class Main {  
 public static void main(String[] args) {  
 Scanner in = new Scanner(System.*in*);  
 System.*out*.print("Введите имя: ");  
 String name = in.nextLine();  
 System.*out*.print("Введите возраст: ");  
 int age = in.nextInt();  
 Person person = new Person(name, age);  
 try {  
 FileOutputStream fileOut = new FileOutputStream("src/LABORATORY\_WORK\_7/Example1\_2/task7/person.json");  
 ObjectOutputStream out = new ObjectOutputStream(fileOut);  
 out.writeObject(person);  
 out.close();  
 fileOut.close();  
 System.*out*.println("Serialized data is saved in person.ser");  
 } catch (  
 IOException e) {  
 e.printStackTrace();  
 }  
 }  
}

Код, выполняет следующие функции:

Ввод данных пользователем:

* Использует Scanner для ввода данных с консоли. Пользователю предлагается ввести имя и возраст.

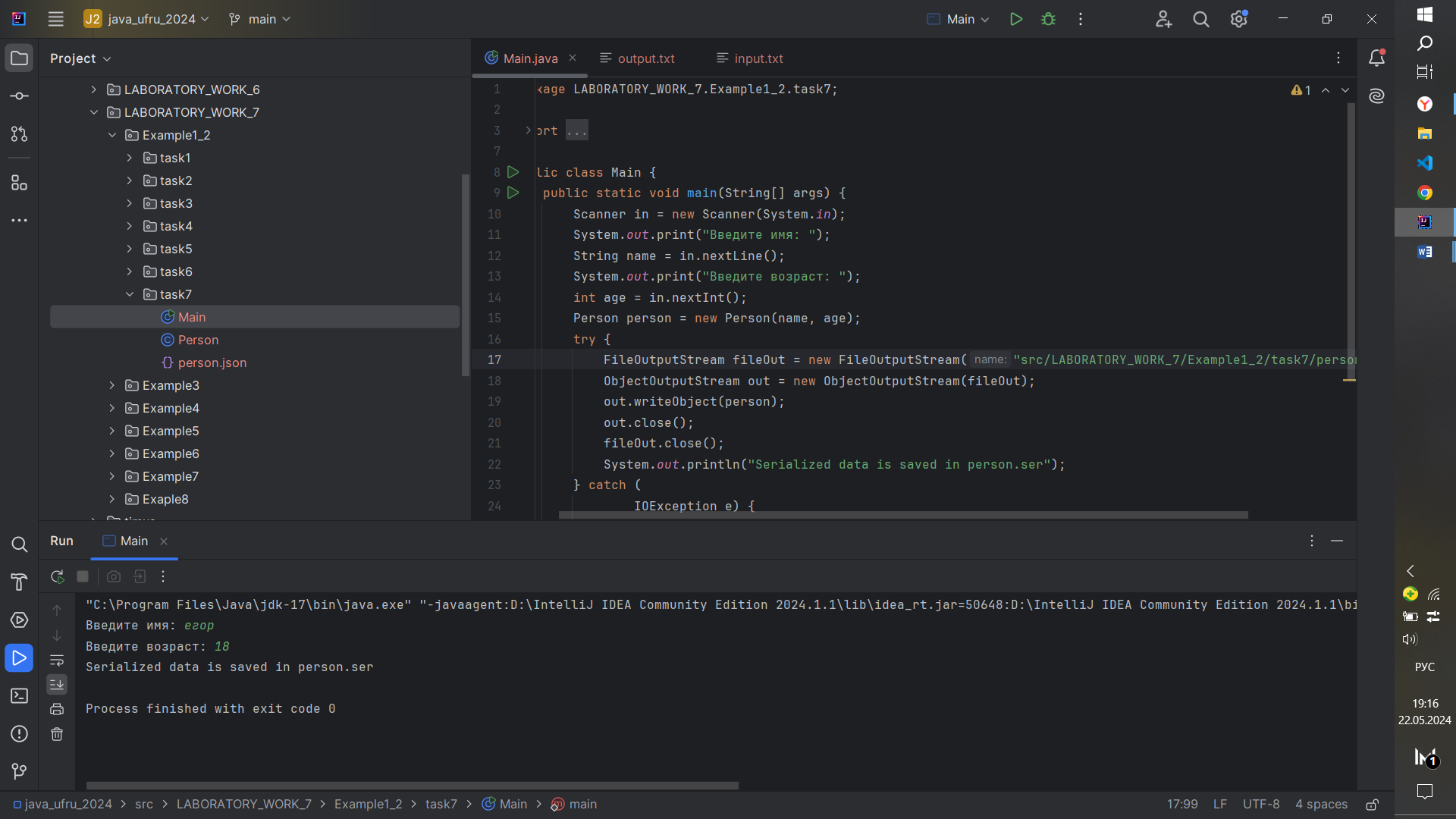
Создание объекта класса Person:

* Создается экземпляр класса Person с использованием введенных данных (имя и возраст).

Сериализация объекта:

* Открывается файл person.json для записи данных.
* Используется FileOutputStream для создания потока вывода в файл.
* ObjectOutputStream используется для сериализации объекта Person и записи его в файл. Сериализация — это процесс преобразования состояния объекта в поток байтов для сохранения или передачи.
* После завершения сериализации, поток вывода закрывается.

После успешной сериализации и сохранения данных в файл, программа выводит сообщение в консоль, уведомляя о том, что сериализованные данные сохранены в файле person.json.



Example3

public class Main {  
 public static void main(String[] args) throws IOException {  
 String fileName = "src/LABORATORY\_WORK\_7/Example3/input.txt";  
 BufferedReader reader =  
 new BufferedReader(  
 new FileReader(fileName));  
 int lineCount = 0;  
 while (reader.readLine() != null) {  
 lineCount++;  
 }  
 reader.close();  
 System.*out*.println("Количество строк в файле: " + lineCount);  
 }  
}

Код, выполняет следующие функции:

Чтение файла:

* Открывает файл с именем input.txt, расположенный в указанной директории.

Подсчет строк:

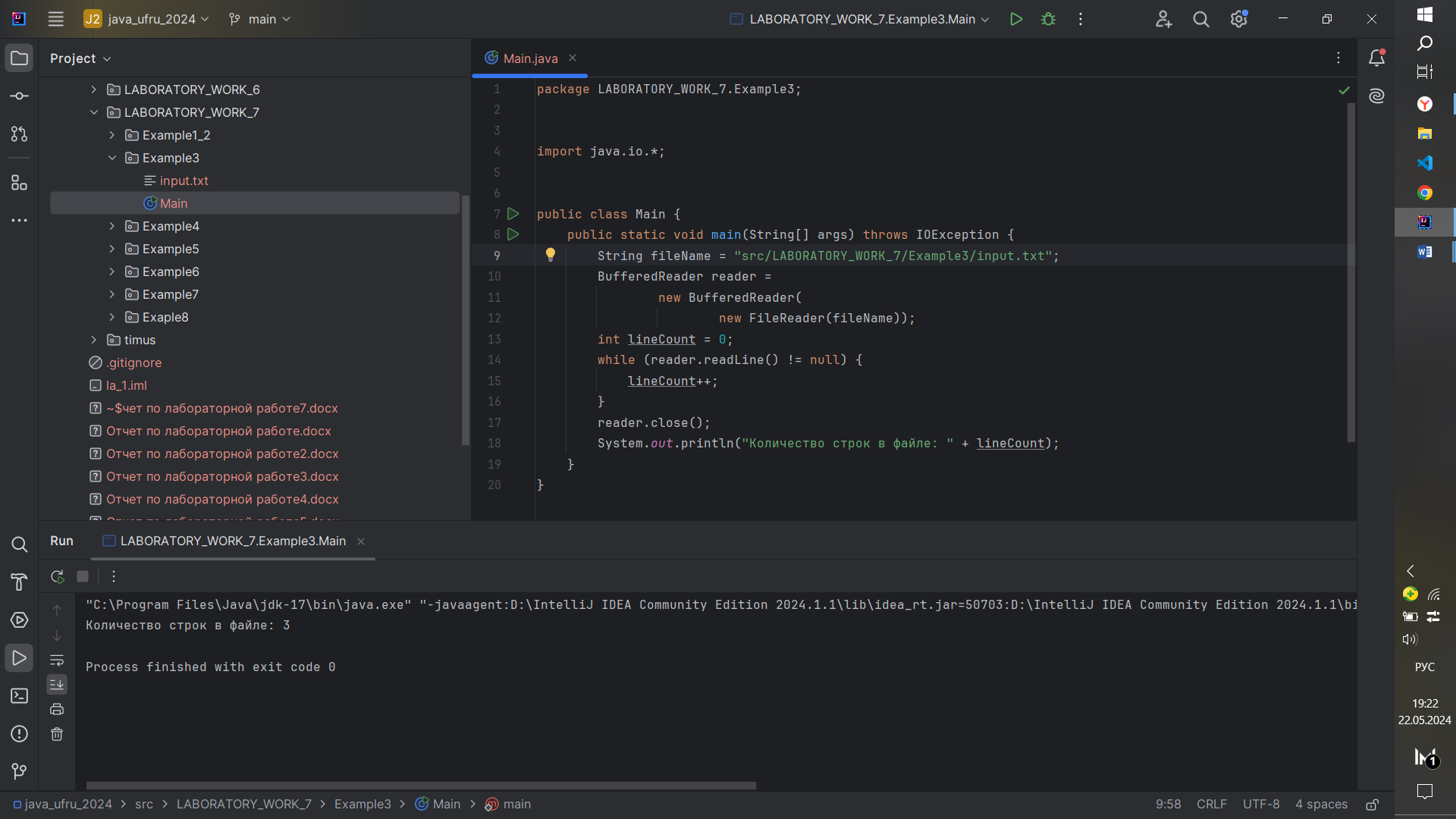
* Используется цикл while для подсчета количества строк в файле. Каждая итерация цикла читает одну строку из файла с помощью метода readLine(). Если метод возвращает null, это означает, что достигнут конец файла, и цикл завершается.

Закрытие потока:

* После завершения чтения файла, поток чтения BufferedReader закрывается с помощью метода close(). Это важно для освобождения системных ресурсов и предотвращения утечек памяти.

Вывод результата:

* После подсчета строк, программа выводит в консоль количество найденных строк в файле.



Example4

public class Main {  
 public static void main(String[] args) throws IOException {  
 FileReader reader = new FileReader("src/LABORATORY\_WORK\_7/Example4/input.txt");  
 FileWriter writer = new FileWriter("src/LABORATORY\_WORK\_7/Example4/output.txt");  
 int c;  
 while ((c = reader.read()) != -1) {  
 writer.write(c);  
 }  
 reader.close();  
 writer.close();  
 System.*out*.println("Файл скопирован успешно!");  
 }  
}

Код, выполняет следующие функции:

Чтение из файла:

* Открывает файл с именем input.txt, расположенный в указанной директории, с помощью FileReader.
* Запись в файл:
* Открывает файл с именем output.txt в той же директории для записи данных с помощью FileWriter.

Копирование данных:

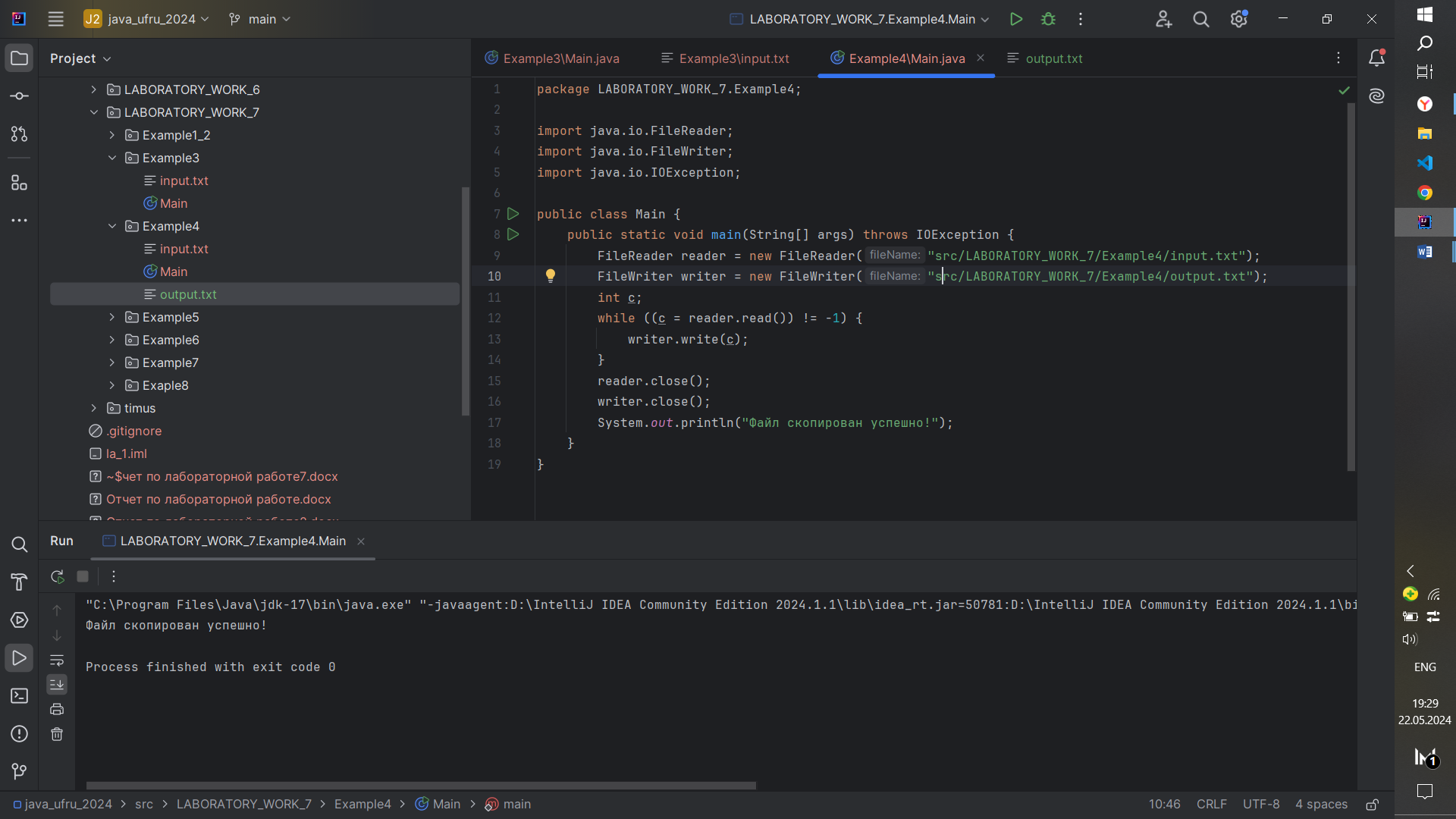
* Используется цикл while для чтения символов из входного файла и их немедленной записи в выходной файл. Метод read() класса FileReader читает один символ из файла и возвращает его в виде целого числа. Если метод возвращает -1, это означает, что достигнут конец файла, и цикл завершается.
* Каждый прочитанный символ сразу записывается в выходной файл с помощью метода write() класса FileWriter.

Закрытие потоков:

* После завершения копирования данных, поток чтения FileReader и поток записи FileWriter закрываются с помощью методов close(). Это важно для освобождения системных ресурсов и предотвращения утечек памяти.

Вывод сообщения:

* После успешной копировки файла, программа выводит в консоль сообщение о том, что файл был скопирован успешно.



Example5

public class Main {  
 public static void main(String[] args) {  
 String myPath = "src/LABORATORY\_WORK\_7/Example5";  
 Scanner in = new Scanner(System.*in*);  
 System.*out*.print("Введите название файла: ");  
 String fileName = in.nextLine();  
 String finalFileName = myPath + File.*separator* + '/' + fileName + ".txt";  
 File file = new File(finalFileName);  
 if (file.exists()) {  
 long fileSize = file.length();  
 System.*out*.println("Размер файла " + fileName + " составляет " + fileSize + " байт.");  
 } else {  
 System.*out*.println("Файл не найден.");  
 }  
 }  
}

Код, выполняет следующие функции:

Ввод данных пользователем:

* Использует Scanner для ввода данных с консоли.

Формирование полного пути к файлу:

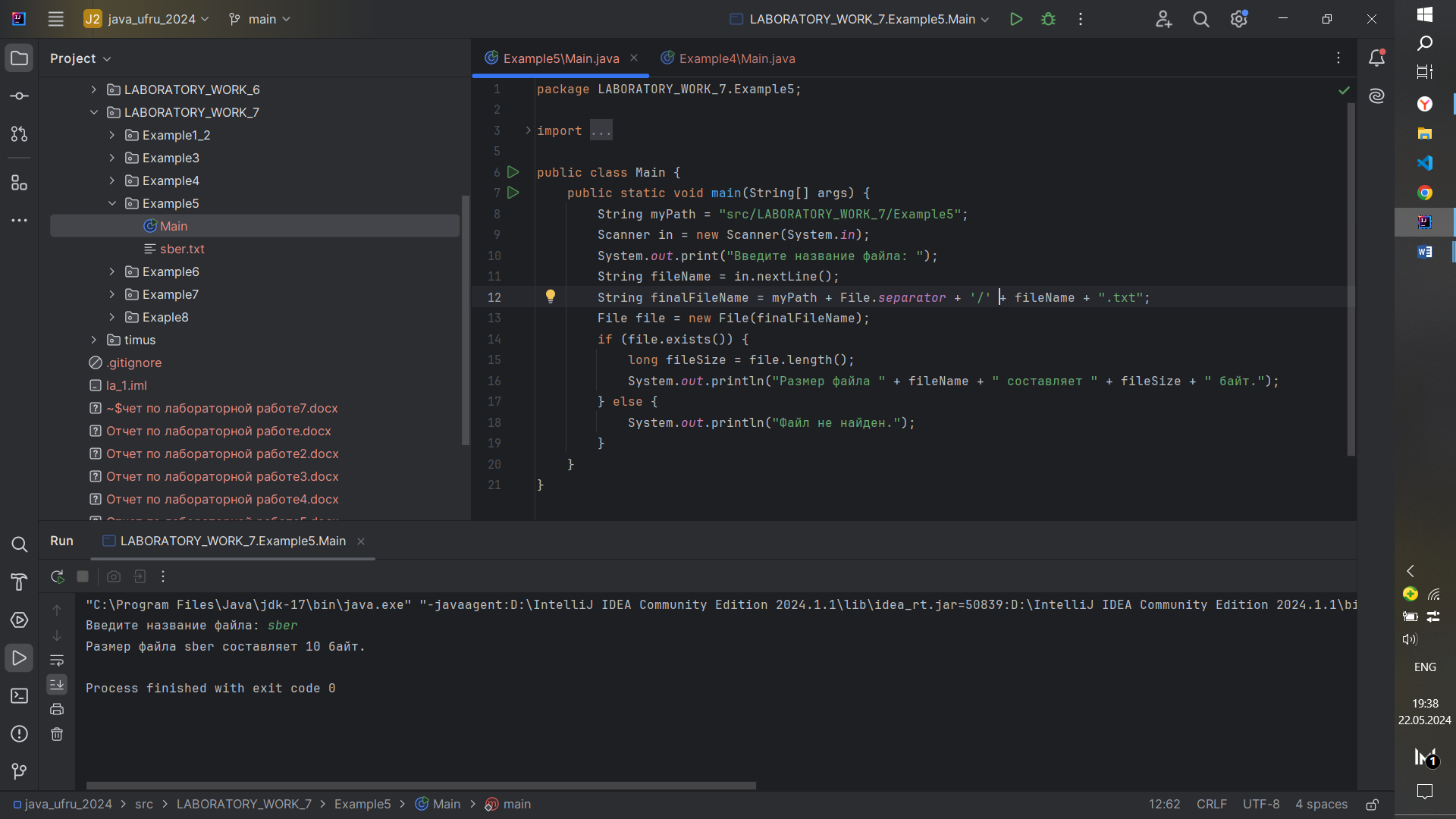
* Создается строка finalFileName, которая содержит полный путь к файлу. Путь состоит из базовой директории myPath, и введенного названия файла с добавлением расширения .txt.

Проверка существования файла:

* Проверяется, существует ли файл, используя метод exists() класса File.

Определение размера файла:

* Если файл существует, его размер определяется с помощью метода length() класса File. Размер файла возвращается в байтах.
* Выводится сообщение в консоль, указывающее на размер файла.
* Если файл не существует, выводится сообщение в консоль о том, что файл не найден.



Example6

public class Main {  
 public static void main(String[] args) throws IOException {  
 String myPath = "src/LABORATORY\_WORK\_7/Example6";  
 Scanner in = new Scanner(System.*in*);  
 System.*out*.print("Введите название файла: ");  
 String fileName = in.nextLine();  
 System.*out*.print("Введите слово для поиска: ");  
 String wordToSearch = in.nextLine();  
 String finalFileName = myPath + File.*separator* + '/' + fileName + ".txt";  
 try (BufferedReader reader = new BufferedReader(new FileReader(finalFileName))) {  
 String line;  
 while ((line = reader.readLine()) != null) {  
 if (line.contains(wordToSearch)) {  
 System.*out*.println(line);  
 }  
 }  
 } catch (IOException e) {  
 System.*out*.println("Ошибка при чтении файла: " + e.getMessage());  
 }  
 }  
}

Код, выполняет следующие функции:

Ввод данных пользователем:

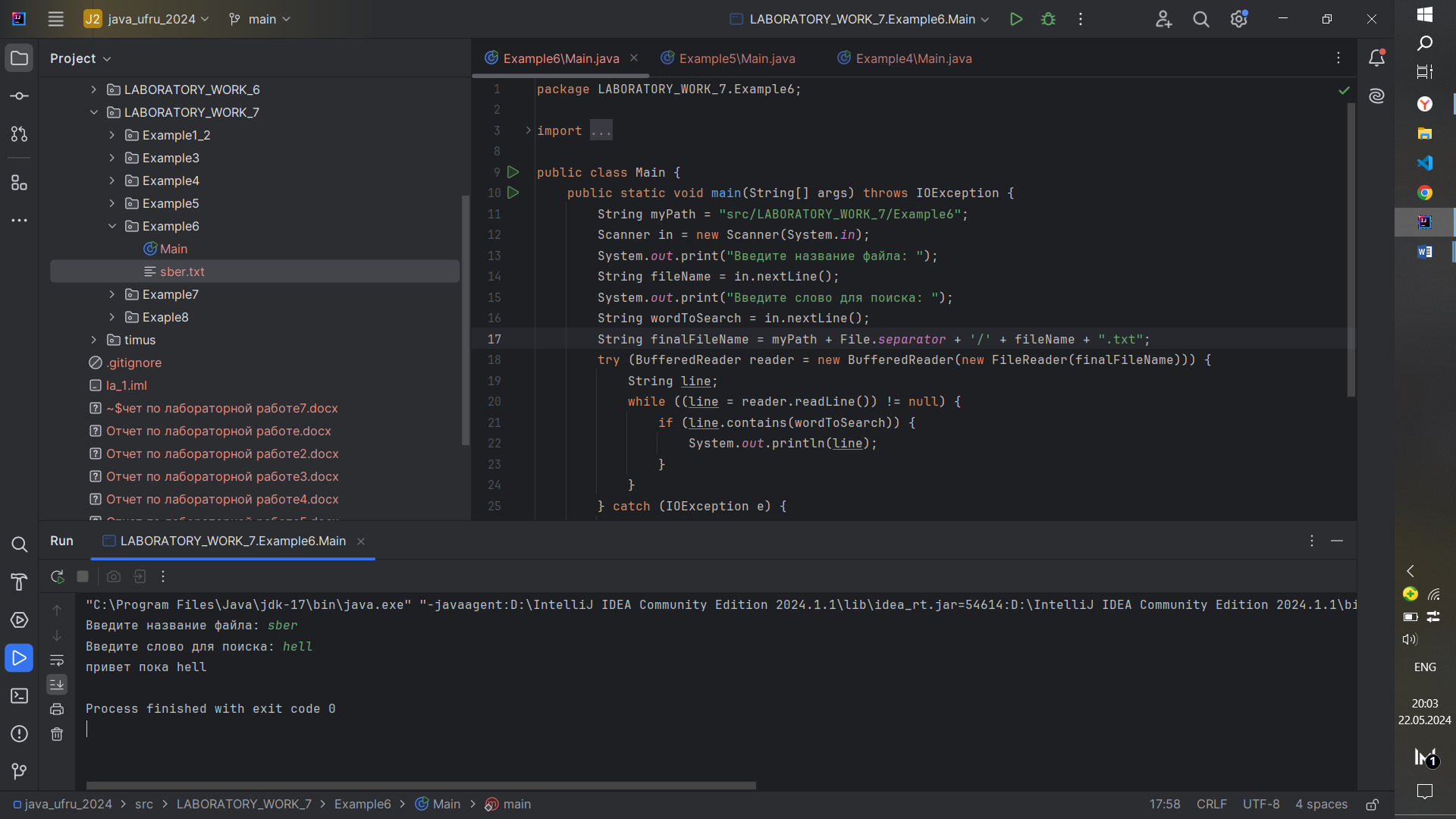
* Использует Scanner для ввода данных с консоли.

Формирование полного пути к файлу:

* Создается строка finalFileName, которая содержит полный путь к файлу. Путь состоит из базовой директории myPath, и введенного названия файла с добавлением расширения .txt.

Чтение файла и поиск слова:

* Открывается файл для чтения с помощью BufferedReader.
* Используется цикл while для чтения строк из файла. Каждая прочитанная строка проверяется на наличие введенного слова с помощью метода contains().
* Если строка содержит искомое слово, она выводится в консоль.



Example7

public class Main {  
 public static void main(String[] args) throws IOException {  
 String myPath = "src/LABORATORY\_WORK\_7/Example7";  
 Scanner in = new Scanner(System.*in*);  
 System.*out*.print("Введите название файла для записи: ");  
 String fileName = in.nextLine();  
 System.*out*.print("Введите текст для записи в файл: ");  
 String textToWrite = in.nextLine();  
 String finalFileName = myPath + File.*separator* + fileName + ".txt";  
 try (BufferedWriter writer = new BufferedWriter(new FileWriter(finalFileName))) {  
 writer.write(textToWrite);  
 } catch (IOException e) {  
 System.*out*.println("Ошибка при записи в файл: " + e.getMessage());  
 return;  
 }  
 System.*out*.println("Текст успешно записан в файл.");  
 System.*out*.println("Количество записанных символов: " + textToWrite.length());  
 }  
}

Код, выполняет следующие функции:

Ввод данных пользователем:

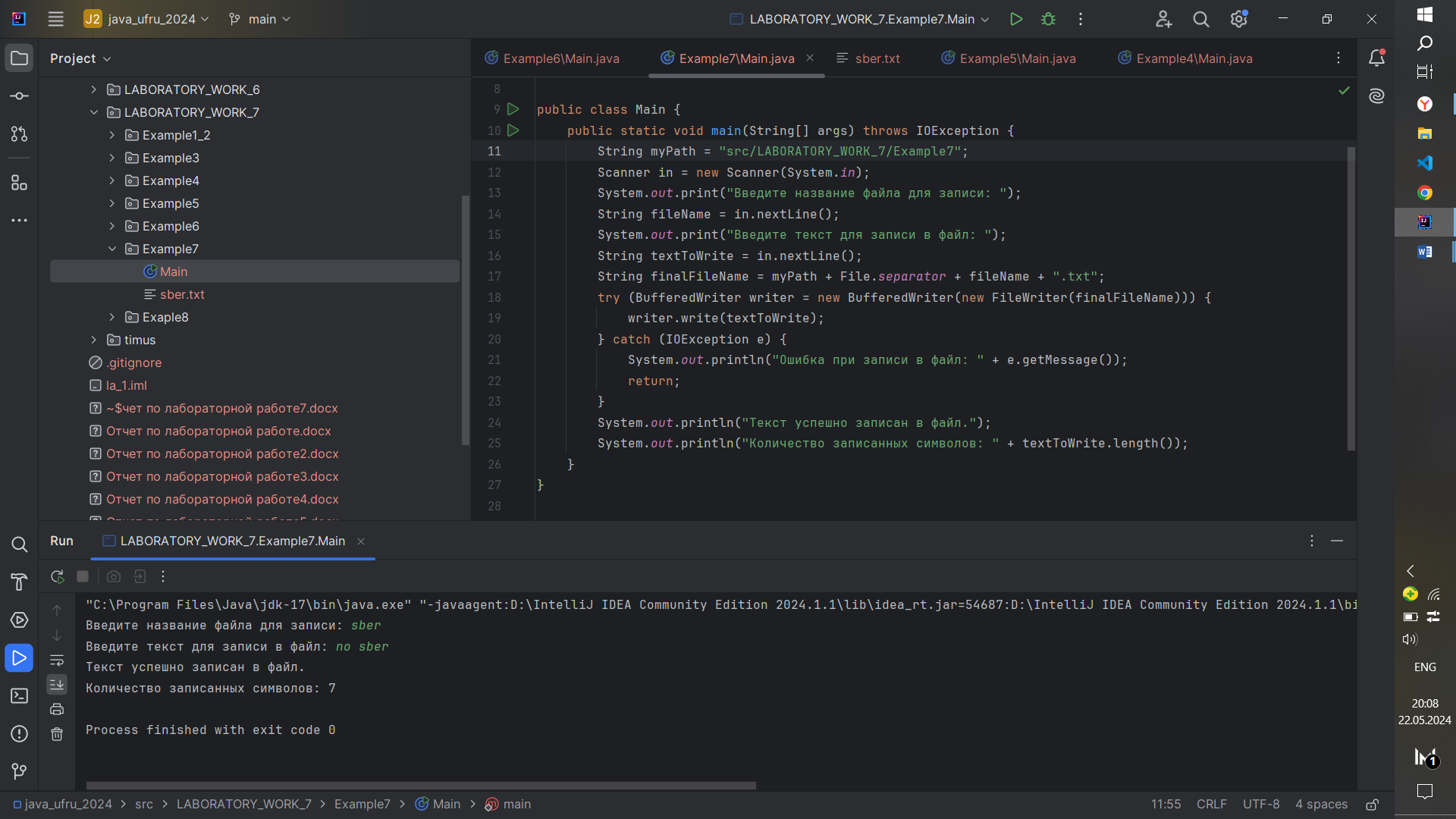
* Использует Scanner для ввода данных с консоли.

Формирование полного пути к файлу:

* Создается строка finalFileName, которая содержит полный путь к файлу. Путь состоит из базовой директории myPath, разделителя и введенного названия файла с добавлением расширения .txt.

Запись текста в файл:

* Открывается файл для записи с помощью BufferedWriter. Это позволяет эффективно записывать данные в файл, используя буфер.
* Введенный текст записывается в файл с помощью метода write().



Example8

public class Main {  
 public static void main(String[] args) {  
 // Создание объекта Person  
 Person person = new Person("Alice", 30);  
  
 // Путь к файлу  
 String filePath = "person.json";  
  
 // Сериализация объекта в файл  
 try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(filePath))) {  
 oos.writeObject(person);  
 System.*out*.println("Объект сериализован и сохранен в файл: " + filePath);  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
  
 // Десериализация объекта из файла  
 try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(filePath))) {  
 Person restoredPerson = (Person) ois.readObject();  
 System.*out*.println("Объект десериализован из файла:");  
 System.*out*.println(restoredPerson);  
 } catch (IOException | ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
 }  
}

Person: Реализует интерфейс Serializable, что позволяет сериализовать объекты этого класса. Используется ObjectOutputStream для записи объекта Person в файл. Используется ObjectInputStream для чтения объекта Person из файла. Исключения IOException и ClassNotFoundException обрабатываются для корректной работы с файлами и объектами.

Этот код позволяет сохранить объект класса Person в файл и восстановить его, а затем вывести информацию о восстановленном объекте на экран.

