**Министерство науки и высшего образования 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

Центр ускоренного обучения

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №8**

**По дисциплине «Программирование»**

Студент группы РИЗ-140938у Бабенко М.А.

Преподаватель: Архипов Н.А.

**Екатеринбург, 2025 г.**

**Лабораторная работа N8**

**««Система ввода/вывода в java. Работа с файлами через байтовые**

**потоки»»**

Цель: получение навыков работы с каталогами и файлами операционной системы, а также с классами ввода/вывода, получение навыков ввода/вывода данных файла через символьные потоки.

**Задания для самостоятельной работы**

**Задание 1.** В отдельных проектах выполнить примеры 1…10 лабораторной работы. Протестировать программы с помощью отладчика. Выявить различие в работе программ в примерах 7 и 8.

В proj81.java записывается программа, которая позволяет создавать файлы. Сначала, инициализируется переменная типа File – **f1**, затем **new File** – вызов **конструктора класса File** для создания объекта, а в скобках указывается название создаваемого файла – команда **«File f1 = new File("MyFile1.txt");»**. Но пока происходит инициализация нового файла, а не его создание. Чтобы его создать, требуется ввести команду «f1.createNewFile();» – тогда будет создан файл, если его еще нет. Затем, при помощи условия if «(f1.exists())», проверяется, существует ли заданный файл.

Также, существует возможность создать файлы в других директориях, включая различные диски. Создается другая переменная типа **File – f2**, со строкой-названием файла, содержащим путь к нему «E:\\MyFile2.txt» – команда **«File f2 = new File("E:\\MyFile2.txt");»**. После инициализации переменной файла, нам требуется его создать – вводим команду **«f2.createNewFile();»**. Затем, вводим команду, которая выведет полный путь второго файла – **« System.out.println("Полный путь 2: " + f2.getAbsolutePath());»**, на путь конкретно будет указывать **«f2.getAbsolutePath()»**, где **f2** – переменная, к которой относится файл.

Также, существует возможность создания вложенных папок. Для этого вводится по примеру прошлых команд команда **«File f3 = new File("E:\\Papka1\\Papka2\\Papka3");»**; в ней не фигурируют названия файлов, а только названия папок, которые будут созданы. Тем не менее, переменная **f3** будет ссылаться на эту цепочку файлов, после их создания. Даже пусть работа происходит с папками, а не файлами – переменная, используемая нами – **File**. Чтобы папки создались, команда будет отличаться – **«f3.mkdirs();»**, то есть будут созданы папки по пути, который принадлежит переменной **f3**. Затем, как и до этого, будет выведен путь к папкам, относящимся к переменной **f3**.

Команда **catch** нужна, чтобы оповестить пользователя об ошибках, если таковые появятся при создании файлов.

Результат работы программы и ее код показан на рисунке 1. Созданные файлы и папки показаны на рисунке 2.

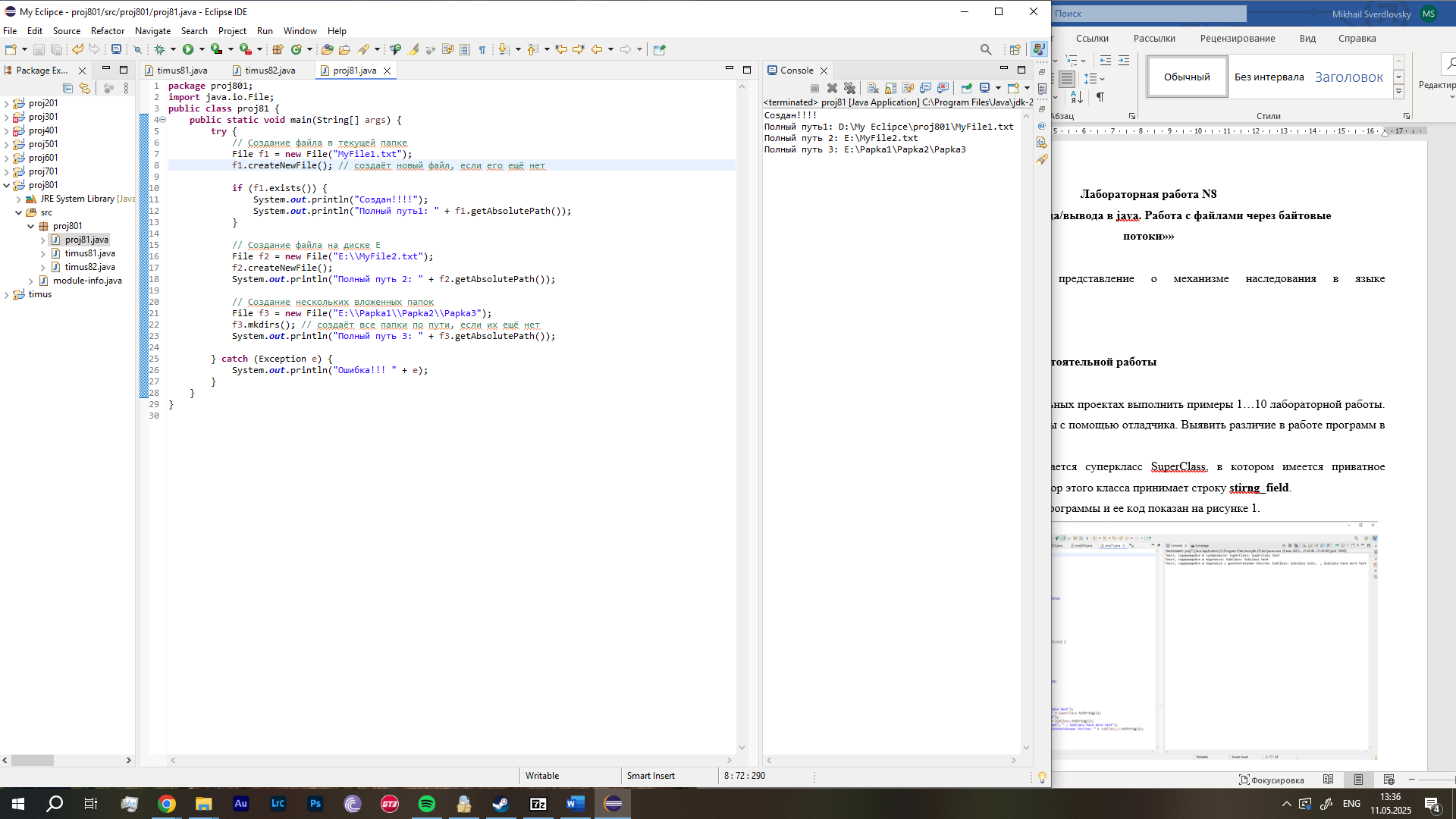


Рисунок 1 – Код и результат работы программы proj81

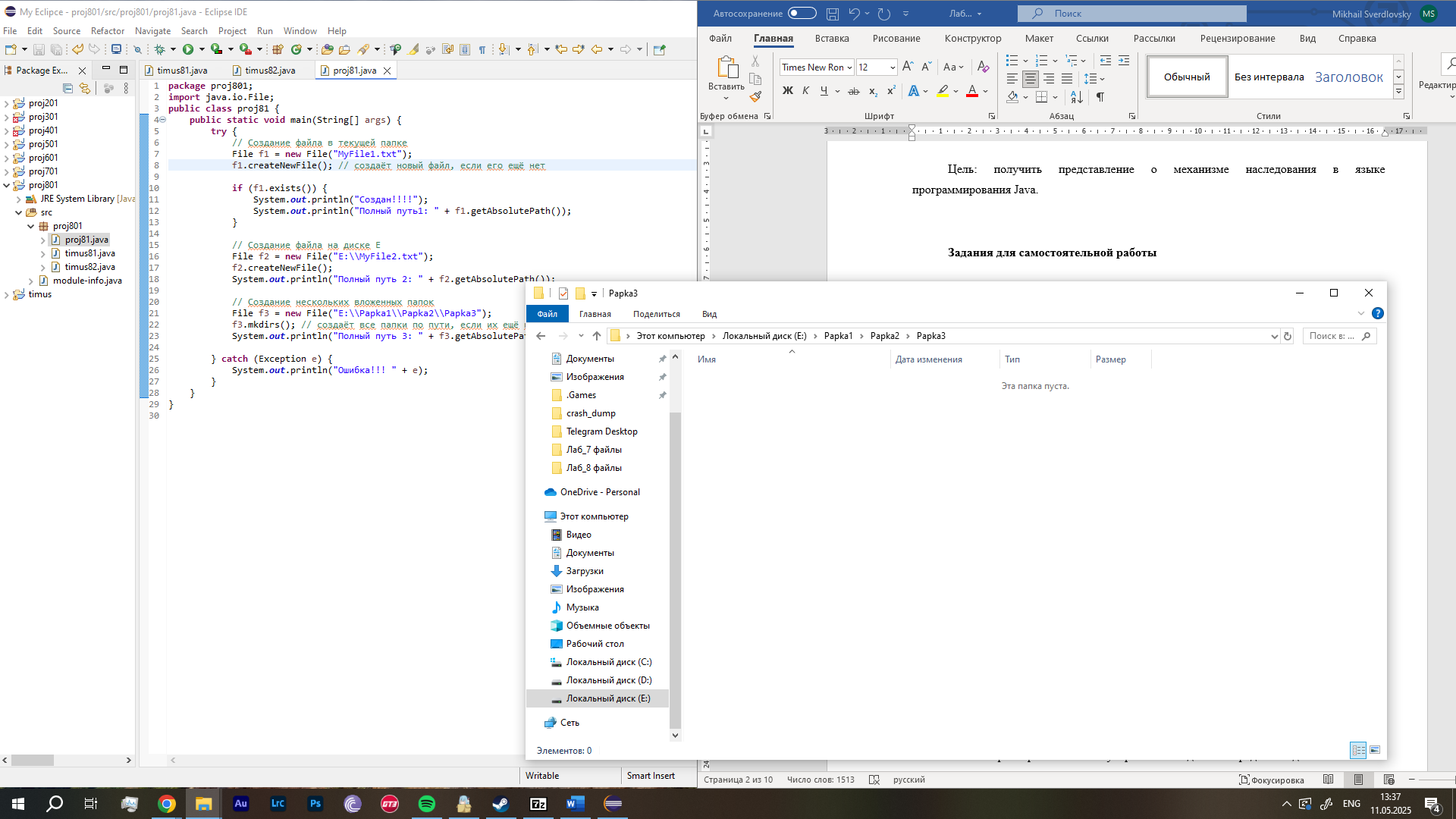


Рисунок 2 – Созданные программой proj81 папки

Создается публичный класс **proj82**, в нем – метод **readAllByByte**. Он читает данные из потока ввода побайтово. Объявляется публичный статический метод, возвращающий ничего, принимающий параметр-объект **InputStream**; метод, который может выбросить исключение **IOException**. Далее, реализуется цикл **while**, который будет реализовывать чтение содержимого файла, пока оно не закончится (в этом случае встретится команда **break**). Объявляется переменная типа int с названием oneByte, в которую при помощи метода **in.read** будет записываться один байт из потока (целое число, представляющее собой код символа); если достигнут конец потока (конец данных) – метод **in.read** возвращает «-1». С этим связана проверка на конец потока – далее идет команда **«if (oneByte != -1)»**, это будет означать что данные закончились. Пока данные не закончились, код символов преобразуется в символ и выводится пользователю на экран. Если данные закончились, на экран выводится **«end»**, прерывается цикл и метод.

В главном методе реализуется блок **try-catch**. Все действия, которые могут привести к ошибке записаны в блоке **try**, если ошибка происходит, то она перехватывается блоком **catch**, и выводится сообщение об ошибке.

С помощью команды создаем переменную типа **InputStream** с названием **inFile**, создается объект для чтения файла **FileInputStream** по указанному пути, далее, вызывается описанный выше написанный нами метод **readAllByByte**, которому передаются данные файла как аргумент, после чего поток закрывается с помощью **inFile.close()**.

Далее, с создается переменная **FileInputStream** с названием inUrl и объект URL, содержащий ссылку на веб-страницу поисковика Google, с помощью метода openStram() создается потом, с помощью которого можно читать данные с веб-страницы. Используется описанный выше написанный нами метод **readAllByByte**, которому передаются данные файла как аргумент, после чего поток закрывается с помощью **inUrl.close()**.

Создается переменная **FileInputStream** с названием **inArray** и объект **ByteArrayInputStream**, содержащий массив данных типа byte, с помощью метода. Используется описанный выше написанный нами метод **readAllByByte**, которому передаются данные файла как аргумент, после чего поток закрывается с помощью **inArray.close()**.

После, в коде встречается перехватчик ошибок типа **IOException e**, который прерывает исполнение основного кода, в случае, если возникают ошибки ввода-вывода, на который ссылается тип ошибок **IOException**, e – объект этого исключения (в нем хранятся данные об ошибке). Результат работы программы и ее код показан на рисунке 3.

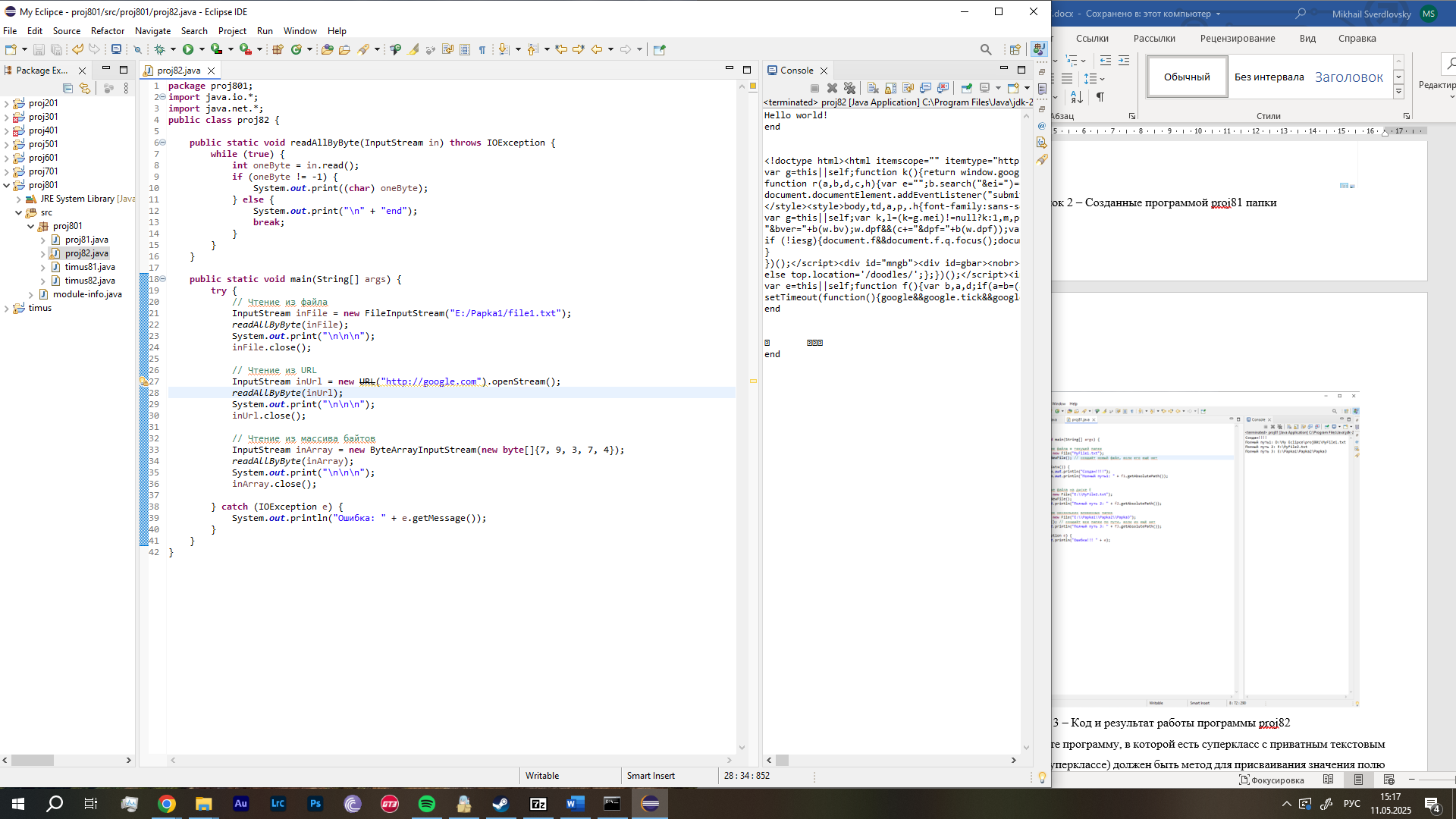


Рисунок 3 – Код и результат работы программы proj82

В proj83.java записывается программа которая читает содержимое файла по 5 байт. Создается публичный статический метод **readAllByArray**, который может выбросить исключение **IOException;** он читает данные из потока (**InputStream**). Создается массив типа byte, содержащий 5 элементов, затем в него загружаются данные. С помощью цикла реализуется запись из потока (in.read возвращает количество реально прочитанных байт, меньше 5, если файл закончился – уже все прочитано) и из прочитанных байт создается строка в кодировке cp1251.

В главном методе создается строка filename, содержащая путь до фала, который требуется прочитать. Далее, используется конструкция try-catch, предназначенная для остановки программы в случае, если во время работы произойдут ошибки. Далее, объявляется переменная **inFile** типа **FileInputStream**, которая ссылается на созданную ранее строку **filename**, в которой записан путь до требуемого файла; после этого вызывается метод **readAllByArray**, в качестве аргумента для которого используется переменная **inFile**. В случае ошибки, программой будет выведено сообщение о ней, а также будет выведен код самой ошибки. Код программы и ее работа показаны на рисунке 4.

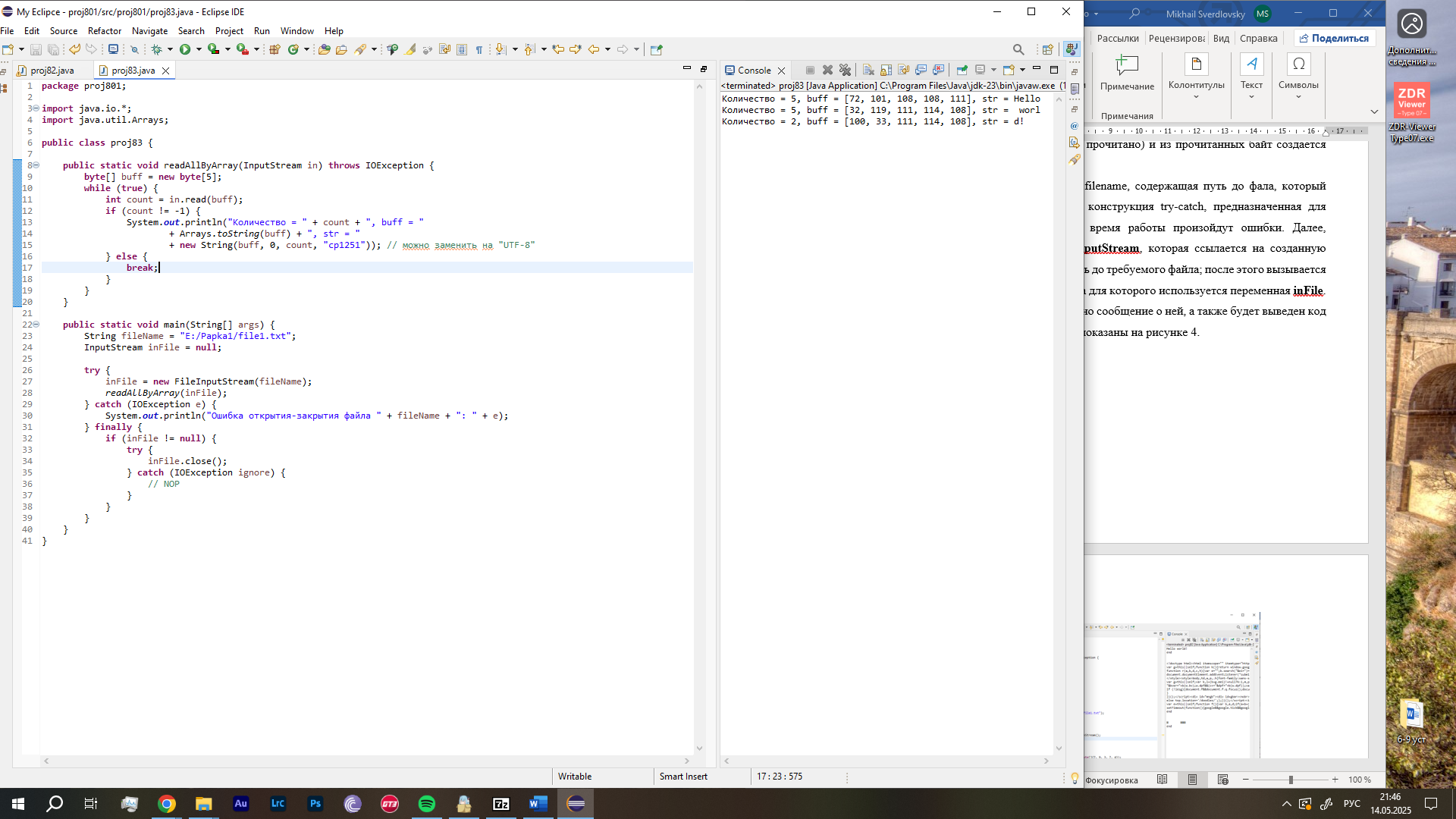


Рисунок 4 – Код и результат работы программы proj83

В программе **proj84.java** в главном методе создается переменная **f1** типа **File**, в которой указана строка с путем файла. Далее, команда **f1.createNewFile()**; отвечает за создание файла, если такового нет. Далее, открывается чтение с клавиатуры в кодировке pc1251. Строкой **«DataOutputStream wr = new DataOutputStream(new FileOutputStream(f1.getAbsolutePath()));»** создается поток для бинарной записи данных в файл: **f1.getAbsolutePath()** – возвращает полный путь к файлу, «**new FileOutputStream(...)»** создает поток байтов в бинарном виде, направленный в файл по указанному пути, **«new DataOutputStream(...)»** - преобразует числа в байты, является оберткой для прошлой команды. Этот поток данных записывается в переменную **wr** типа DataOutputStream.

Далее, пользователь вводит в консоль количество чисел, которые хочет записать и сами числа, командой **«wr.flush();»** записывается в файл то, что при чтении с клавиатуры хранилось в буфере.

Затем, аналогично создается второй файл – **f2**. Аналогичным образом в него записываются данные из первого файла, пока не возникнет ошибка **«EOFException e»** из-за окончания файла.

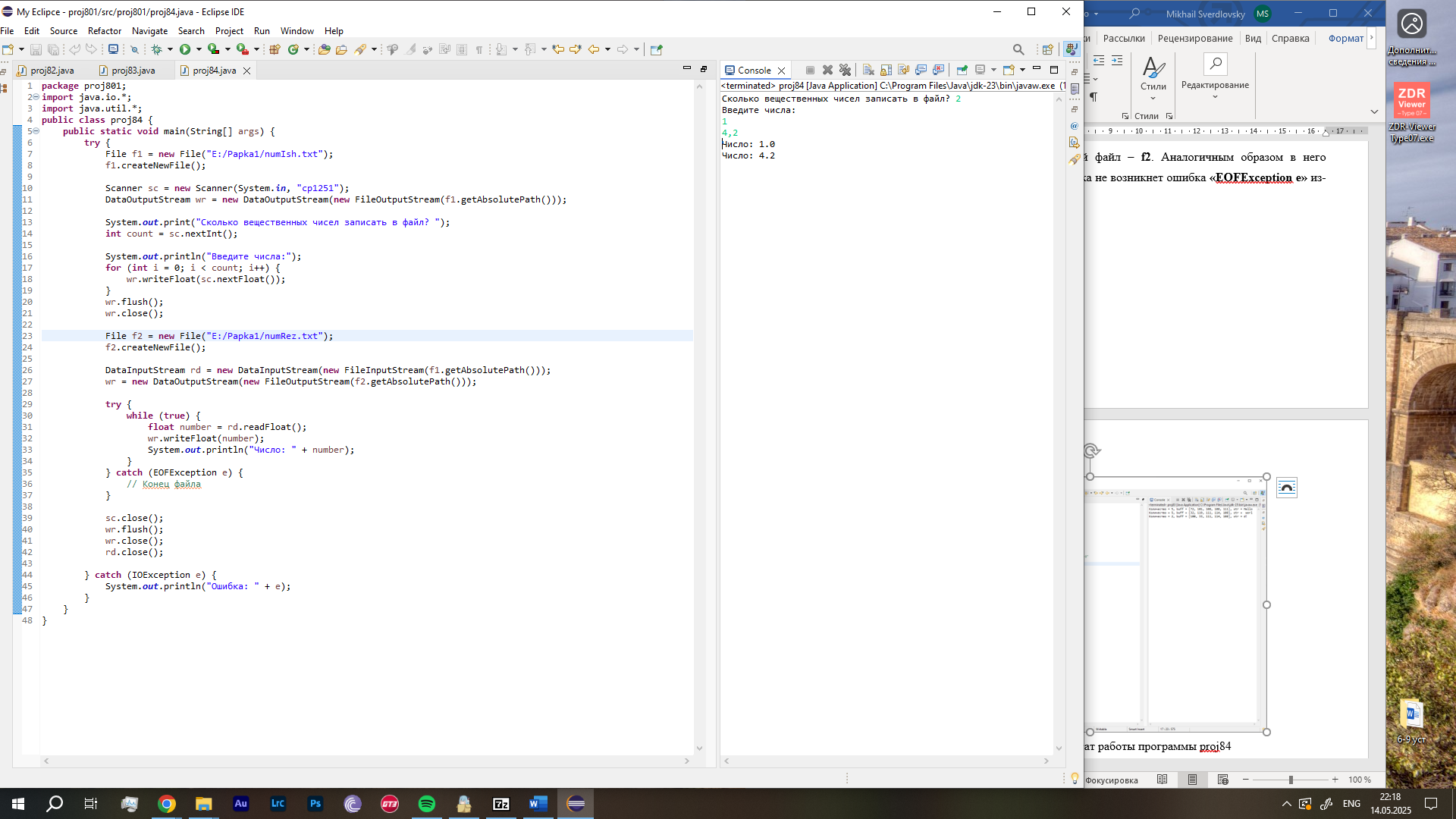


Рисунок 5 – Код и результат работы программы proj84

В программе proj85 в главном методе происходит ввод пользователем имени файла, который будет создан позже. Через конструкцию try-catch создается переменная типа File – **f1**, в которой будет содержаться строка с введенным именем файла; также он будет выведен на экран. Далее, пользователь вводит количество строк, которые будут вводиться далее, после чего командой **«sc.nextLine()»** очищается буфер. Открывается поток **dOut** для записи строк в бинарном формате, и с помощью цикла for строки записываются в в буфер через **«writeUTF()»**, а затем командой **«dOut.Flush()»** – из буфера в файл.

Затем происходит открытие поток для чтения из файла в бинарном формате с форматом **UTF**. Цикл реализован так, что строки читаются пока не наступит **«EOFException»** - конец файла. Код программы и ее работа показаны на рисунке 6.

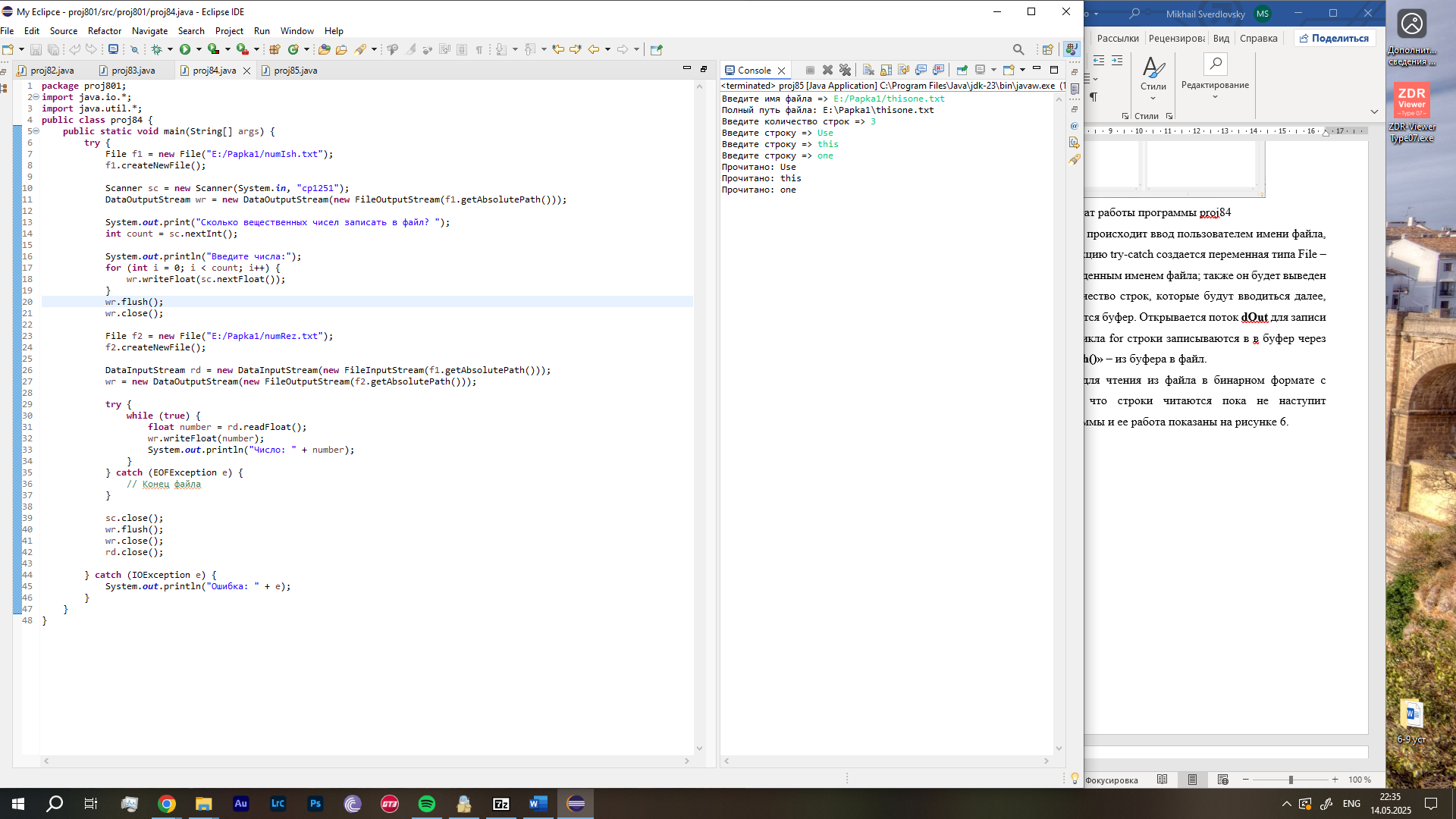


Рисунок 6 – Код и результат работы программы proj85

В программе proj86 в главном методе создаются две переменных – **in** класса Reader (чтение) и **out** класса Writer (запись). Далее используется структура try-catch, переменной **in** присваивается строка с путем назначения файла, откуда будет производиться запись, а **out** – куда (добавляем true, чтобы содержимое файла не перезаписывалось, а добавлялось к тому что уже есть. Затем, считывается каждый символ из файла по пути **in** (.read), записывается в файл по пути **out** (.append), и записанное выводится пользователю в консоль, пока команда .read не вернет -1, что будет означать конец файла. Затем, выводится сообщение о том, что копирование завершено. **Catch** перехватывает код в случае возникновения ошибки, а в **finally** происходит закрытие потоков. Код программы и ее работа показаны на рисунке 7.

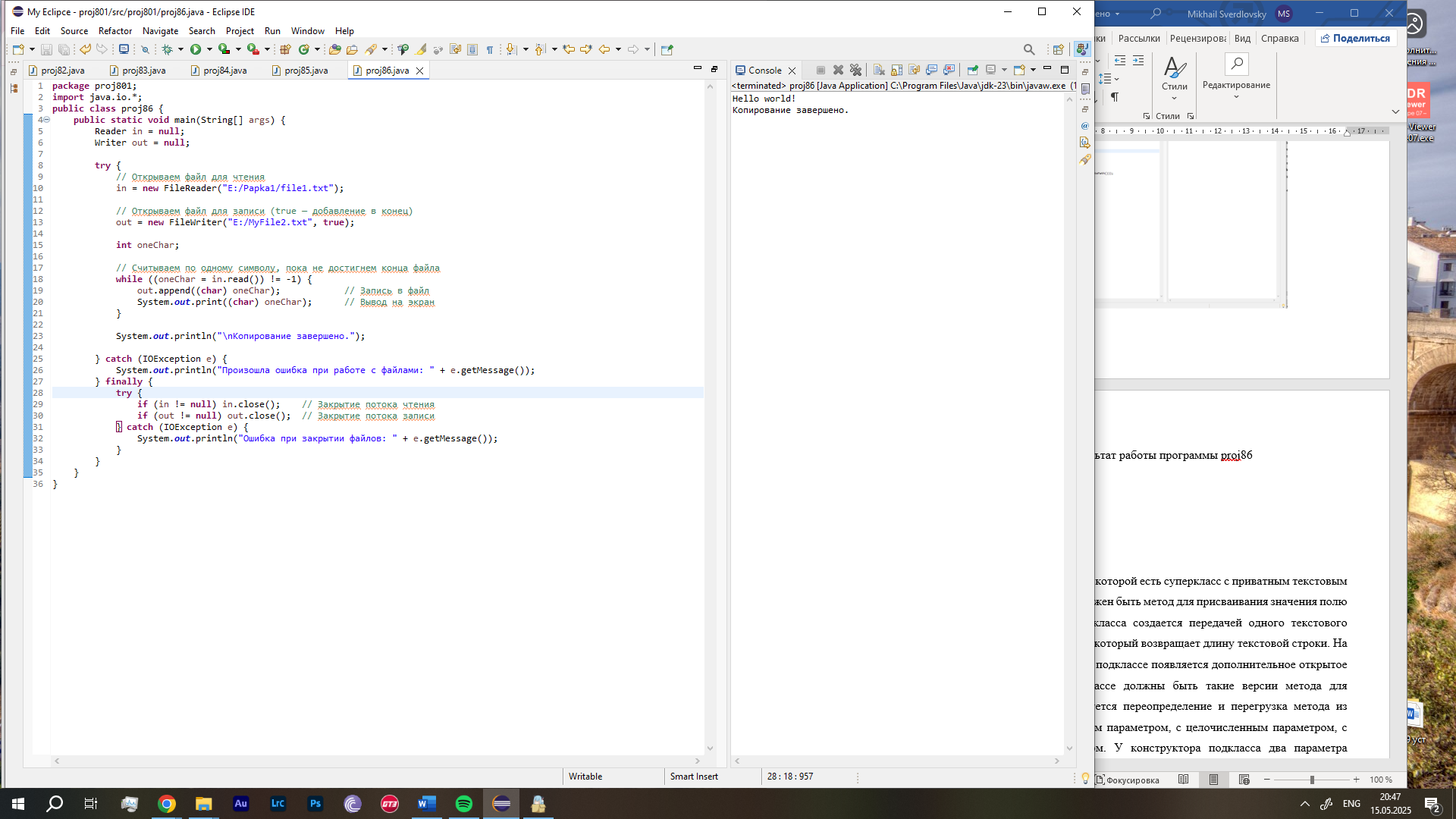


Рисунок 7 – Код и результат работы программы proj86

В программе proj87 в главном методе создаются две переменных – **br** типа BufferedReader (чтение) и **out** типа BufferedWriter (запись). Далее создается конструкция try-catch, переменной **br** присваивается строка с путем назначения файла, откуда будет производиться запись (добавляем 1024 – размер буфера ), а **out** – куда. Затем, создается счетчик, и циклом производится чтение каждой строки (.readLine из переменной br, пока не найдется пустая строка, не содержащая ни одного символа), прочитанное выводится на экран, строка записывается в другой файл, после чего добавляется переход на новую строку.

Код программы и ее работа показаны на рисунке 8.

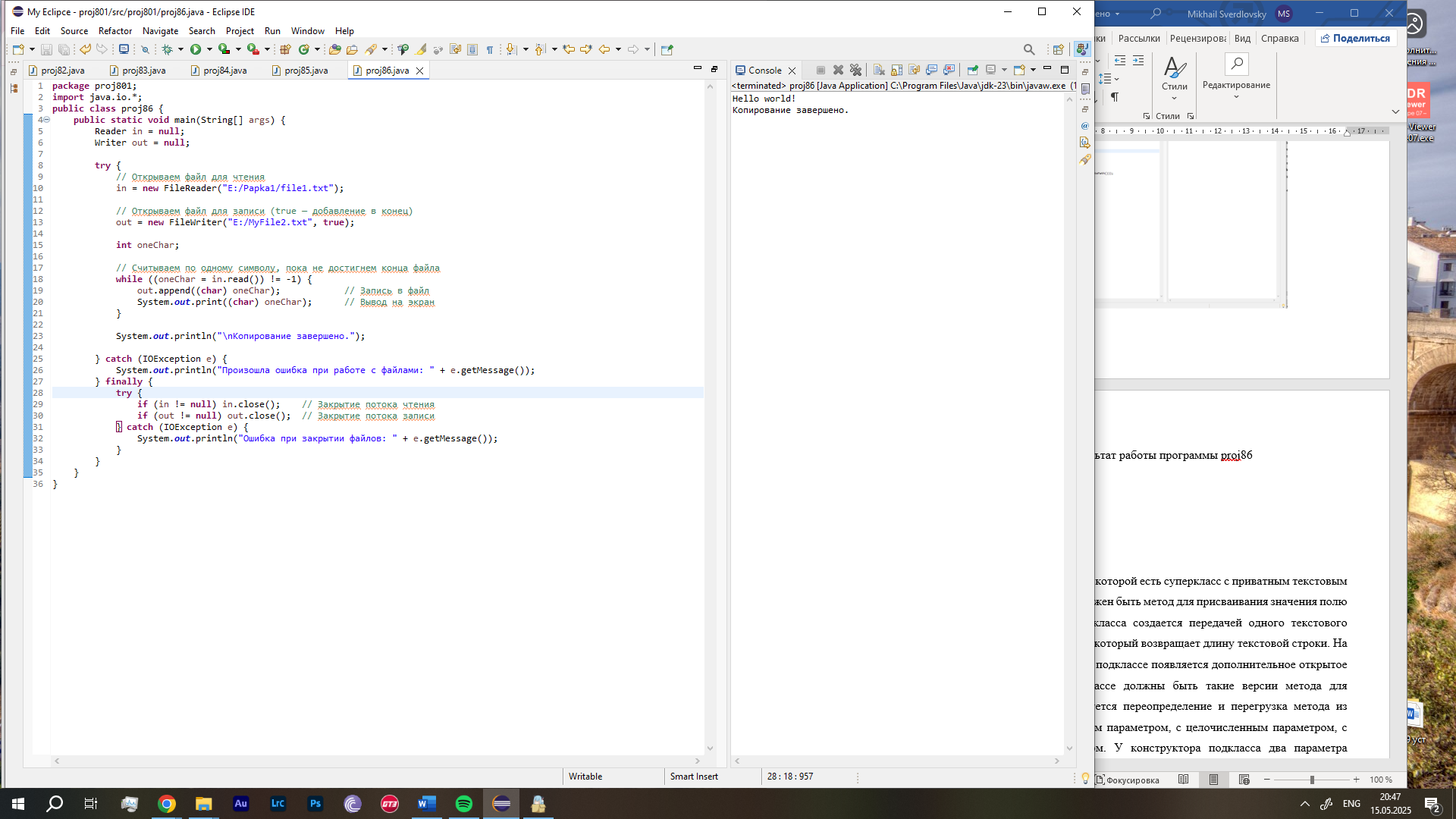


Рисунок 8 – Код и результат работы программы proj87

В программе proj88 создается метод **readAllByChar**. В нем осуществляется чтение из источника по символам. В главном методе сначала осуществляется чтение из файла: создается переменная **inFile** базового класса InputStream, в которую записан путь к читаемому файлу; создается переменная **rFile** в качестве аргумента которой присвоен путь из inFile в кодировке cp1251; далее вызывается метод **readAllByChar** с **rFile** в качестве аргумента. Далее осуществляется чтение из веб-страницы: создается переменная **inURL** базового класса InputStream, в которую записан адрес google.com, создается переменная **rUrl** в качестве аргумента которой присвоен путь из inUrl в кодировке cp1251; далее вызывается метод **readAllByChar** с **rUrl** в качестве аргумента. Далее осуществляется чтение из массива байт: создается массив байтов из пяти элементов, создается переменная **inArray** базового класса InputStream, в которую записан массив данных, создается переменная **rArray** в качестве аргумента которой присвоен путь из inArray в кодировке cp1251; далее вызывается метод **readAllByChar** с **rUrl** в качестве аргумента. Код программы и ее работа показаны на рисунке 9.

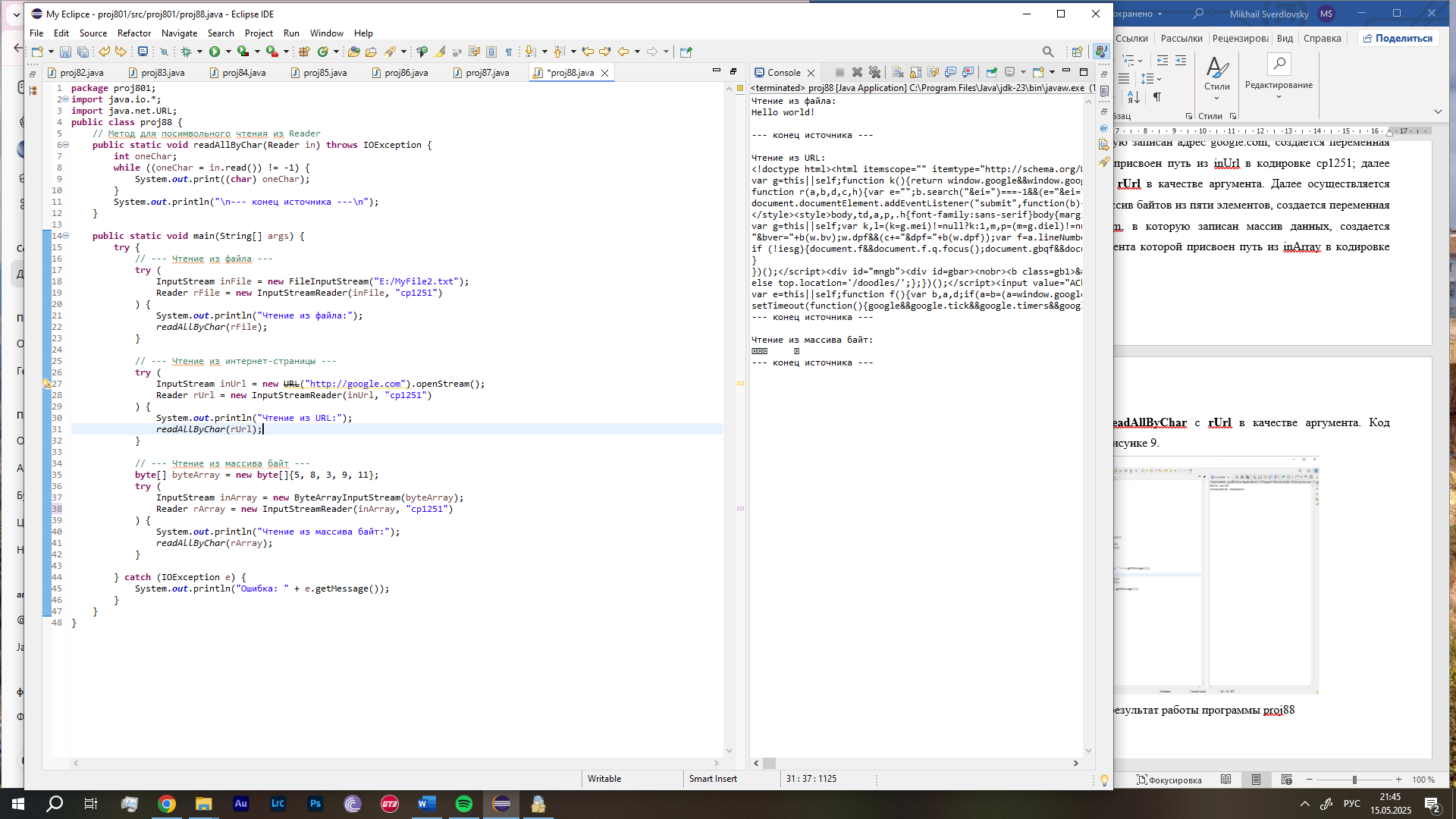


Рисунок 9 – Код и результат работы программы proj88

Программа отличается от прошлой тем, что читает текст посимвольно, а не построчно через буфер, как прошлая; используются совершенно другие классы. Построчную работу удобнее делать если требуется работать со строками; посимвольную работу удобнее делать, если требуется читать данные из сети или памяти.

В программе proj89 создается главный метод, в котором объявляются две строки, содержащие два файла – источник и назначения. Далее, в **try** переменная класса BufferedReader, внутри которой обертка для строки с файлом-источником (байты из переменной преобразуются в символы, которые преобразуются в строки), а также еще одна с файлом назначения. Далее, в цикле реализуется построковое чтение из файла-источника в файл назначения. Код программы и ее работа показаны на рисунке 10.

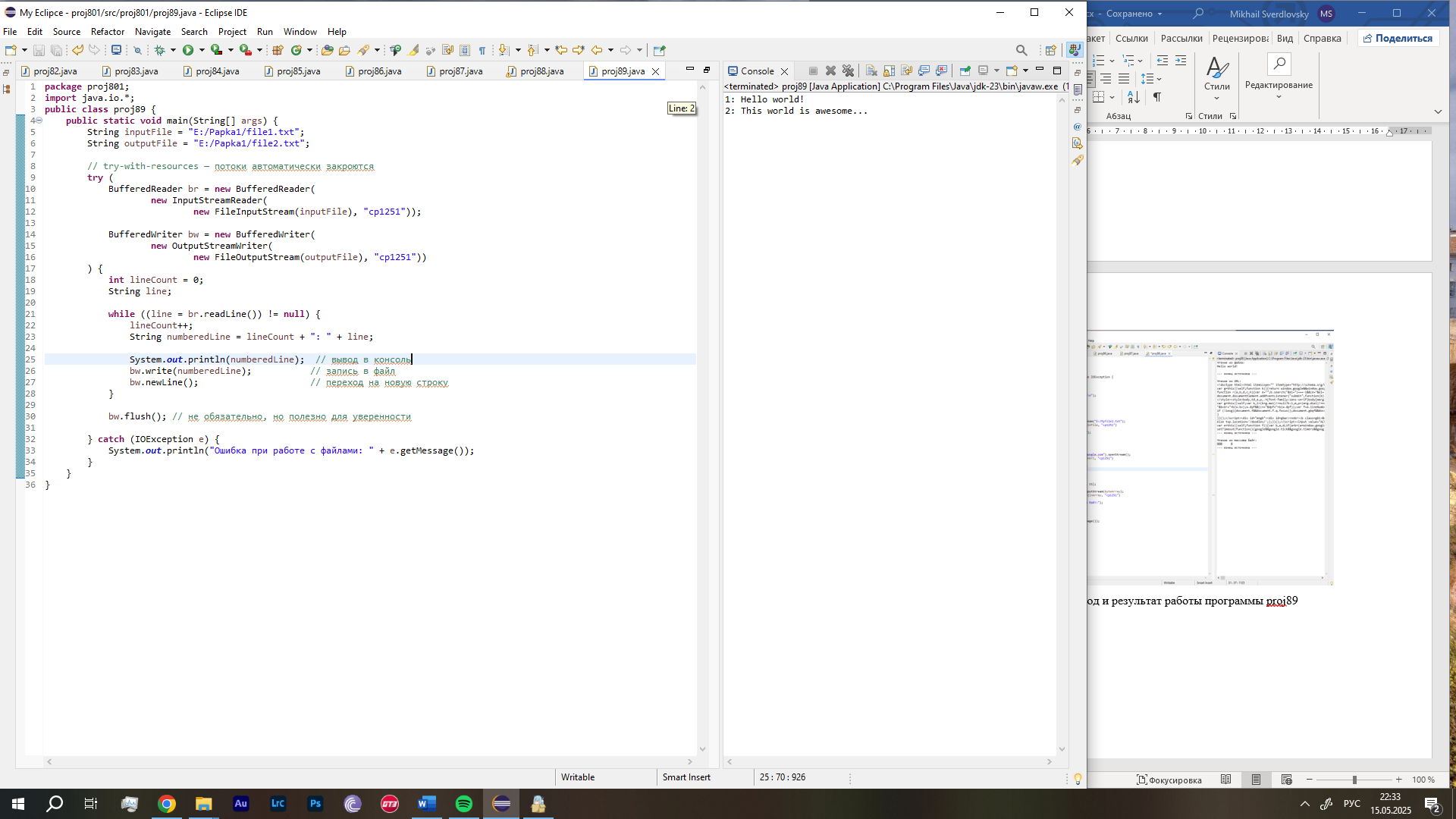


Рисунок 10 – Код и результат работы программы proj89

В программе proj810 создается главный метод, в котором создается переменная br с классом BufferedReader, внутри которой обертка для строки с файлом-источником (байты из переменной преобразуются в символы, которые преобразуются в строки). Далее, аналогичное проделывается для файла, в который будет вестись запись, только обертка предполагается для переменной **out** класса высокоуровневого символьного потока записи PrintWriter. Далее, реализуется цикл, который в качестве условия проверяет ранее инициализированную строку, сверяя ее со считываемой строчкой файла, пока не будет достигнут конец файла. В теле цикла происходит поочередная запись в конечный файл строчек из исходного при помощи команды **out.println**, являющейся экземпляром PrintWriter. Код программы и ее работа показаны на рисунке 11.

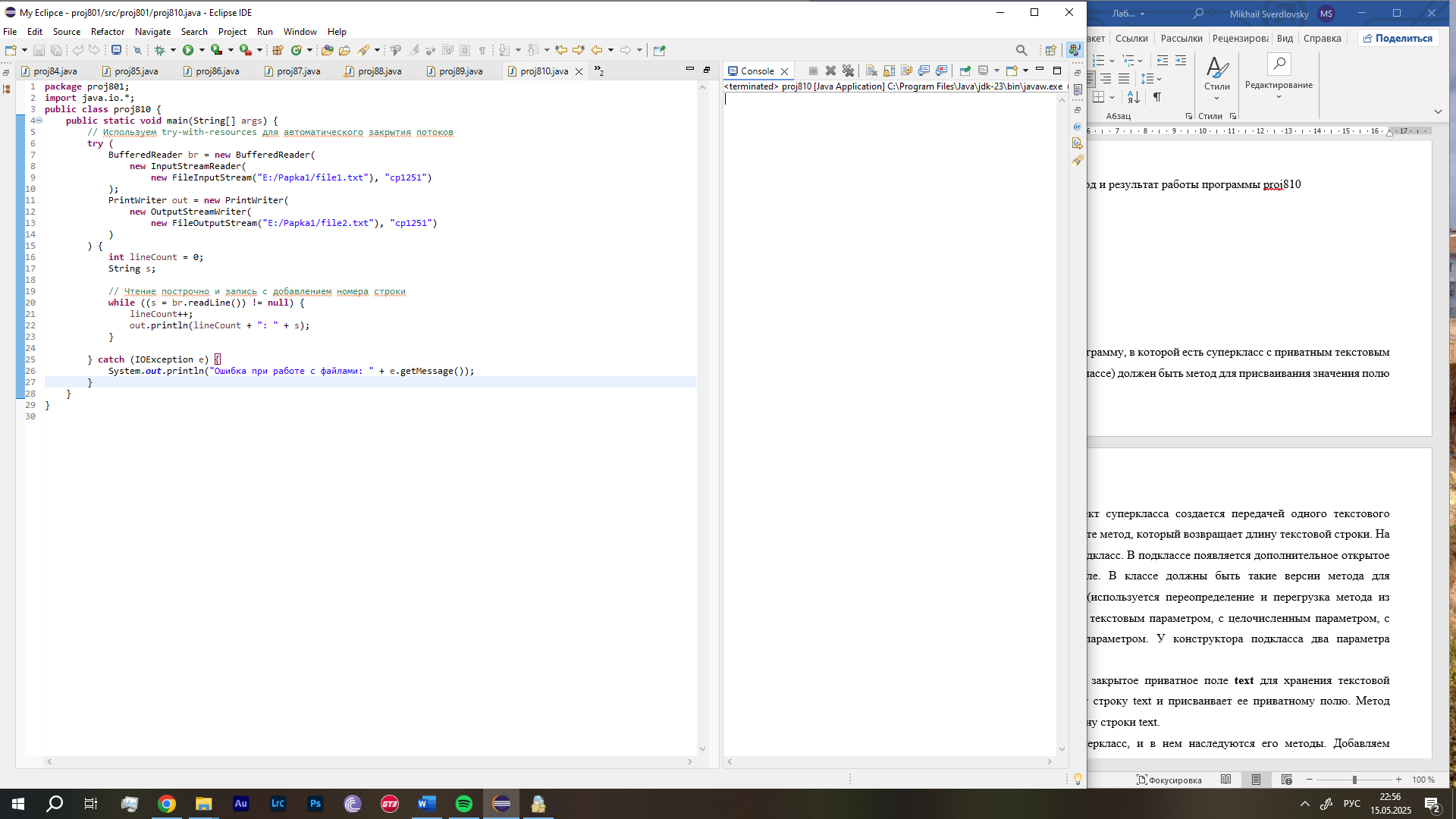


Рисунок 11 – Код и результат работы программы proj810

**Задание 2**. Создать проект, позволяющий из одного, предварительно созданного программными средствами файла, переписать данные, соответствующие условию - в исходном файле содержится две строки в формате UTF-8 и 5 чисел типа double. В результирующий файл переписать вторую строку и положительные числа.

В главном методе создается две строки – одна с файлом, в который будем записывать данные (**file1Path**), другая – в который будем копировать (**file2Path**). Создается пять вещественных чисел. Происходит создание двух файлов – **FileLW1** и **FileLW2**. Затем, с помощью буфера в них записываются строки и пять вещественных чисел. Затем, из первого файла (**FileLW1**) мы читаем нужное нам содержимое, и записываем так же через буфер во второй – **FileLW2**. Для всего процесса используем кострукцию try-catch, чтобы избежать ошибок во время выполнения команды

Результат работы программы и ее код показан на рисунке 12.



Рисунок 12 – Код и результат работы программы proj811

**Задание 3**. Создать проект, позволяющий из одного текстового файла, содержащего несколько строк (тип String) заранее подготовленного текста на русском языке (Пушкин, Лермонтов или другой российский классик на Ваш вкус), построчно переписать в другой текстовый файл слова начинающиеся с согласных букв.

– слова из предложения выделять методом split();

– в новом файле следует указать номер строки, в которой искомые

слова находились в исходном файле;

– для каждой строки указать количество выбранных слов.

Для начала создается метод, который проверяет первую букву каждого слова. В нем программой символ верхнего регистра распознается как и нижнего регистра, и его код сравнивает с записанными в массив согласными буквами; если есть совпадение – программа возвращает индекс первого символа

Далее, в главном методе, объявляем два файла – источник и файл назначения. Чтение и запись реализуем через буфер с помощью **try-with-recourses**. Вводятся переменные строки и ее номера. Также вводится массив символов, которые разделяют между собой слова, пропускаются пустые строчки и вызывается описанный выше метод. Затем, записывается в файл номер строки и слово, которое начинается на согласную букву, а также количество слов в строке.

Результат работы программы и ее код показан на рисунке 13.

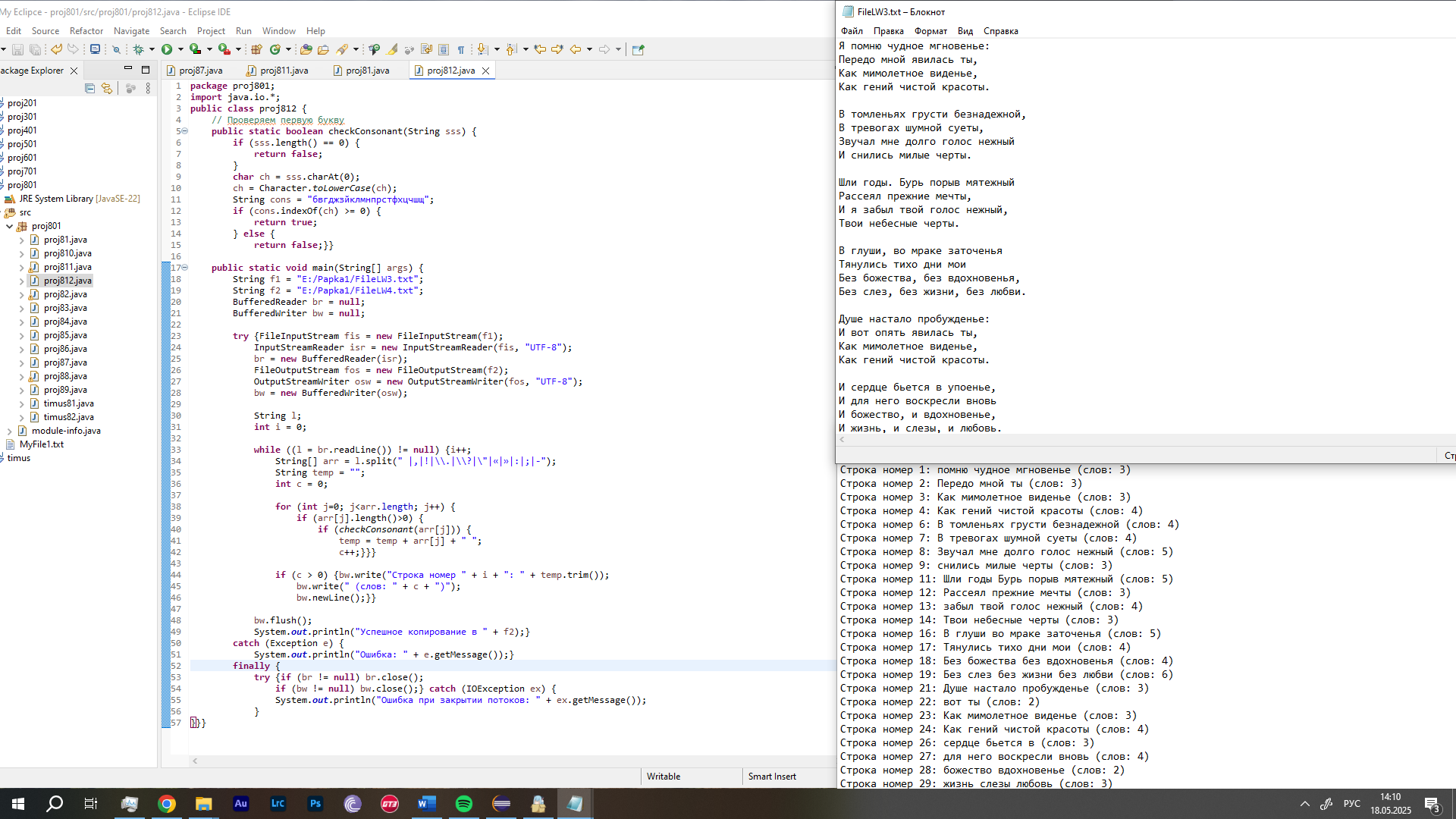


Рисунок 13 – Код и результат работы программы proj812

**Задачи из Timus**

**Задание 1286. Космолёт марки ВАЗ.**

**Задача:**

Общеизвестно (См. «Астронавигация для чайников», под общ. ред. чл.-кор. АН Ассоциации Свободных Миров (АСМ) проф. Г.Бейтса, с. 147-148.), что космолёт, оснащённый гипердвигателем класса B, способен выполнять переход с любой планеты на любую другую. Однако, на Вашем корабле марки ВАЗ-2106 (Вселенский Астромобильный Завод, цифры обозначают год начала выпуска), заклинило рычаг переключения коробки передач. В результате корабль способен перемещаться из точки с координатами (i, j) лишь в одну из точек из следующего списка: (i+q, j+p), (i−q, j+p), (i+q, j−p), (i−q, j−p), (i+p, j+q), (i−p, j+q), (i+p, j−q), (i−p, j−q) (все координаты — целочисленные, в стандартной межгалактической системе — Idem, с. 214). Помогите капитану Вашего корабля выяснить, сможет ли он своими силами добраться до планеты назначения, или ему следует вызывать ремонтную бригаду.

**Исходные данные.**

В первой строке даны два числа p и q (передаточные числа первой гиперпередачи сломанного корабля), разделённые пробелами. Во второй строке находится пара чисел x и y — координаты планеты, на которой произошла поломка. В третьей строке находится пара чисел x1 и y1 — координаты планеты назначения. Числа во второй и третьей строках также разделены пробелами. Все числа целые, по модулю не превосходят 2·109.

**Результат.**

Если капитан сможет довести корабль до планеты назначения, выведите YES. Если же нужно вызывать ремонтную бригаду, выведите NO.

**Решение:**

В публичной статической переменной **findBig** содержатся для вычисления большего общего делителя чисел.

Определяем переменные для параметров корабля – p и q, осуществляем их ввод с клавиатуры и даем им тип данных long (исходя из условий задания), а также создаем переменные для координат текущей планеты и целевой планеты. Далее, идет вычисление, насколько далеко находится цель, если координаты делятся на найденный общий делитель, то значит, что добраться можно. Затем, программой это проверяется – происходит возврат остатка от деления (проверка на ноль). Затем, выводится результат согласно заданию – «Yes» или «No».

Результат работы программы и ее код показан на рисунке 14.

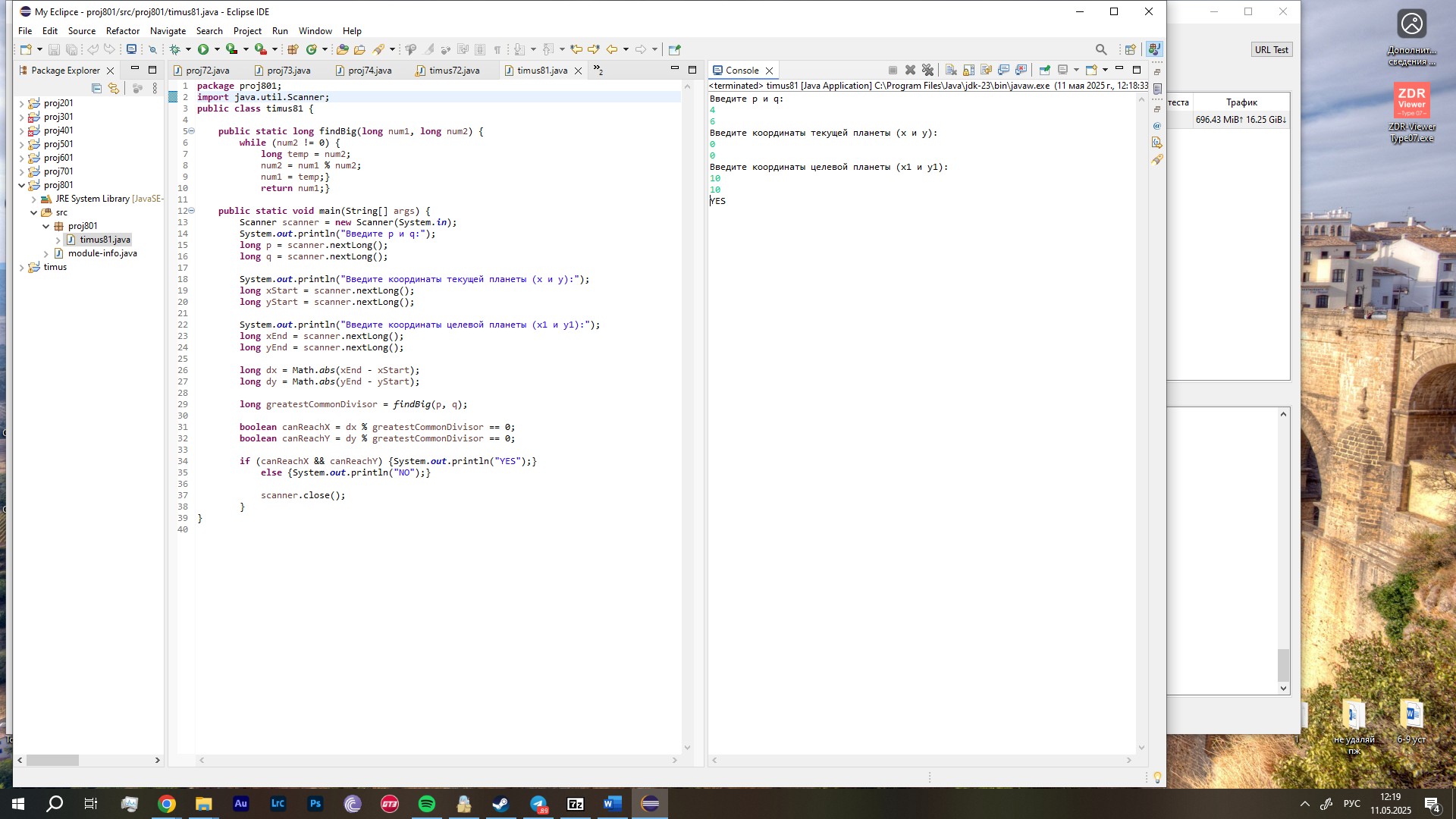


Рисунок 14 – Код и результат работы программы timus81

**1225. Флаги.**

**Задача:**

В День флага России владелец магазина решил украсить свою витрину полосками ткани белого, синего и красного цветов. Он хочет, чтобы выполнялись следующие условия:

Полоски одного цвета не должны располагаться рядом друг с другом.

Синяя полоска может быть расположена только между белой и красной или между красной и белой.

Определите количество способов выполнить желание владельца магазина.

Например, для N = 3 возможно 4 варианта.

**Исходные данные.**

N — количество полосок, 1 ≤ N ≤ 45.

**Результат.**

M — количество способов украсить витрину.

**Решение:**

Создается целочисленная переменная N, в которой пользователь задает количество полосок; затем создается двухмерный массив dp, который будет использоваться для счета способов. Затем, прописываются условия, которые были заданы в самом задании: расположение синей полоски, и, затем, при помощи цикла for, происходит проверка на еще одно условие – порядок расположения цветов. После, вводится переменная, в которую записываются способы украсить витрину, и она выводится пользователю на экран.

Результат работы программы и ее код показан на рисунке 15.

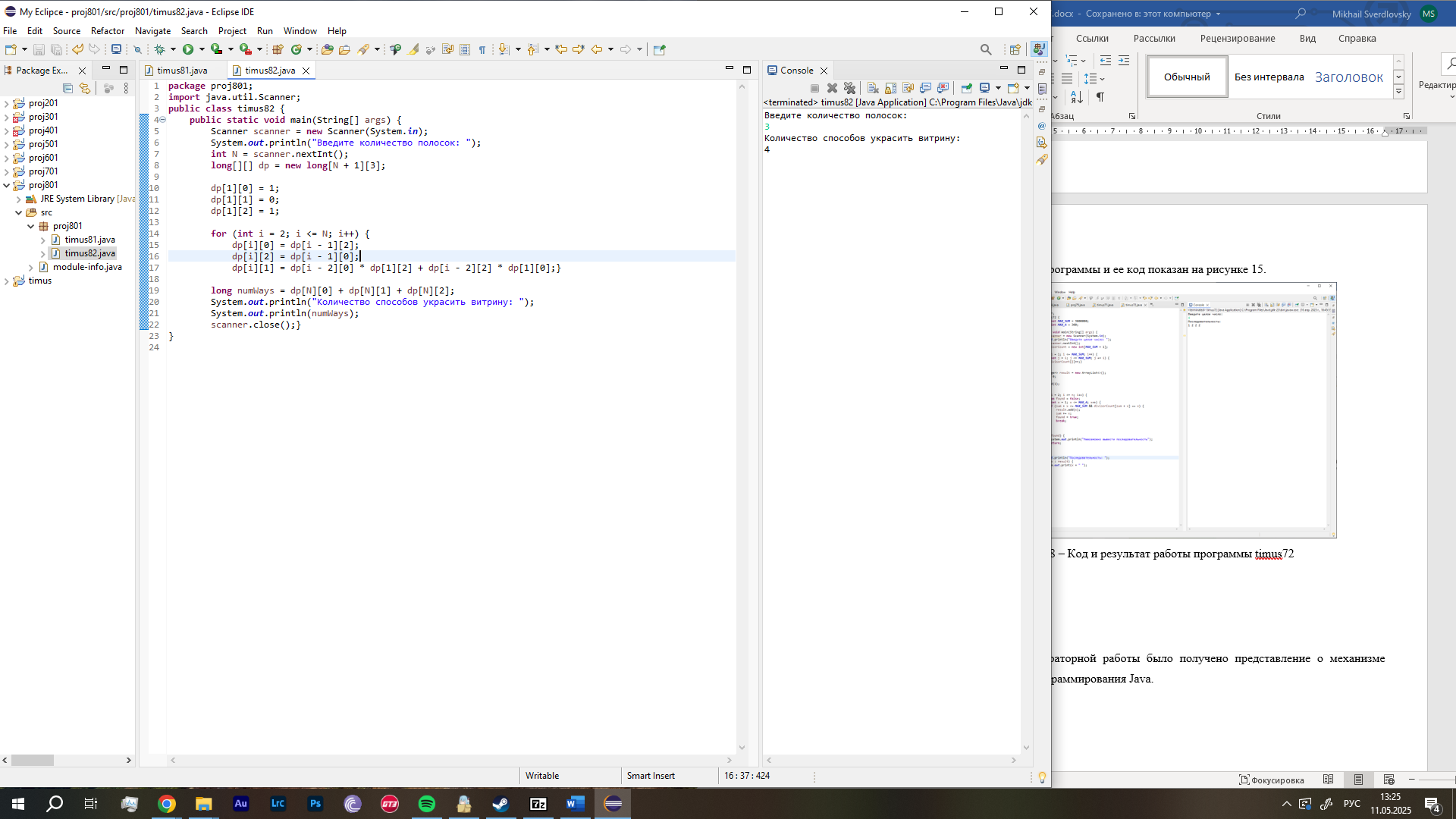


Рисунок 15 – Код и результат работы программы timus82

**Вывод**

В результате лабораторной работы были получены навыки работы с каталогами и файлами операционной системы, а также с классами ввода/вывода, получение навыков ввода/вывода данных файла через символьные потоки.