

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное
учреждение высшего образования
«СЕВАСТОПОЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

Кафедра информационных технологий и компьютерных систем

ПРОГРАММИРОВАНИЕ. ЯЗЫКИ ВЫСОКОГО УРОВНЯ

Методические указания
к лабораторным работам по дисциплине
для студентов направления
44.03.05 - Педагогическое образование
(с двумя профилями подготовки. Профиль: математика и информатика)
дневной формы обучения

В пяти частях

Часть 4

Использование классов библиотеки Java

Севастополь
СевГУ
2023

УДК 004.432(076.5)
ББК 32.973-018.1я73
П784

Р е ц е н з е н т :

В. Ю. Карлусов - канд. техн. наук, доцент кафедры информационных систем

Составители: Т. В. Волкова, Е. С. Владимирова

П784 Программирование. Языки высокого уровня [В 5-ти частях]. Часть 4. Использование классов библиотеки Java: методические указания к лабораторным работам по дисциплине для студентов направления 44.03.05 – Педагогическое образование (с двумя профилями подготовки. Профиль: математика и информатика) дневной формы обучения. / Севастопольский государственный университет, Институт информационных технологий, кафедра информационных технологий и компьютерных систем ; сост. Т. В. Волкова, Е. С. Владимирова. – Севастополь : СевГУ, 2023. – 97 с. – Текст : электронный.

Целью методических указаний является оказание помощи в подготовке к лабораторным работам по дисциплине «Программирование. Языки высокого уровня», предусматривающим разработку и выполнение программ, использующих классы библиотеки Java.

Предназначены для студентов первого курса дневной формы обучения направления 44.03.05 – Педагогическое образование (с двумя профилями подготовки. Профиль: математика и информатика).

УДК 004.432(076.5)
ББК 32.973-018.1я73

Рассмотрены и рекомендованы к изданию на заседании кафедры информационных технологий и компьютерных систем, протокол № 2 от 14 сентября 2022 г.

© Волкова Т. В., Владимирова Е. С., сост., 2023
© ФГАОУ ВО «Севастопольский
государственный университет», 2023

СОДЕРЖАНИЕ

1. ОБЩИЕ ТРЕБОВАНИЯ К ВЫПОЛНЕНИЮ ЛАБОРАТОРНЫХ РАБОТ	4
2. ЛАБОРАТОРНАЯ РАБОТА № 8. ОБРАБОТКА СИМВОЛЬНЫХ СТРОК. ИСПОЛЬЗОВАНИЕ КЛАССА STRING ПАКЕТА JAVA.LANG. ОБРАБОТКА ИСКЛЮЧЕНИЙ В JAVA	6
3. ЛАБОРАТОРНАЯ РАБОТА № 9. ВВОД-ВЫВОД. БАЙТОВЫЕ И СИМВОЛЬНЫЕ ПОТОКИ. ИСПОЛЬЗОВАНИЕ КЛАССОВ ПАКЕТА JAVA.ИО. ОБРАБОТКА ИСКЛЮЧЕНИЙ В JAVA	34
4. ЛАБОРАТОРНАЯ РАБОТА № 10. ВВОД-ВЫВОД СИМВОЛЬНЫХ СТРОК. ИСПОЛЬЗОВАНИЕ КЛАССОВ ПАКЕТА JAVA.ИО. ОБРАБОТКА ИСКЛЮЧЕНИЙ В JAVA.....	49
5. ЛАБОРАТОРНАЯ РАБОТА № 11. ИСПОЛЬЗОВАНИЕ КЛАССОВ ПАКЕТА JAVA.ИО. РАБОТА С ФАЙЛАМИ С ПОМОЩЬЮ КЛАССА FILE. ВВОД-ВЫВОД БАЙТОВ СРЕДСТВАМИ КЛАССА RANDOMACCESSFILE.....	74
БИБЛИОГРАФИЧЕСКИЙ СПИСОК.....	97

1. ОБЩИЕ ТРЕБОВАНИЯ К ВЫПОЛНЕНИЮ ЛАБОРАТОРНЫХ РАБОТ

Для освоения основ программирования на языке Java предлагается серия лабораторных работ. Каждая работа требует изучения методических указаний и рекомендованной литературы, подготовку текста программы, компиляцию программы некоторое ее исследование.

Все действия, указанные в методических указаниях к отдельной работе (освоение практических приемов использования системы разработки программ, объяснение примененных элементов языка Java, ответы на вопросы, задания) обязательно должны быть выполнены до следующего лабораторного занятия.

По выполненной лабораторной работе составляется отчет. Оформление отчета должно соответствовать требованиям к текстовым учебным документам соответствующих ГОСТов [Электронный фонд правовой и нормативно-технической документации <http://docs.cntd.ru>].

Отчет представляется на листах формата А4 и в электронном виде (файл Microsoft Word). Первый лист отчета – титульный. На нем указывается название вуза, название выпускающей кафедры, название работы, ФИО и группа исполнителя, ФИО принимающего работу, город (Севастополь) и год. Пример оформления титульного листа приведен в приложении А. Следующие листы являются собственно отчетом и должны содержать следующие разделы:

- 1) цель работы
- 2) постановка задачи;
- 3) анализ задачи, выявляющий связи между элементами задачи (обоснование типов входных и выходных данных, описание реализуемых функций);
- 4) схема и описание алгоритма решения задачи;
- 5) тестовые примеры и результаты их обработки вручную;
- 6) текст Java-программы, заданной вариантом задания;
- 7) сведения об отладке программы и проверке ее работоспособности (описание ошибок (синтаксических и логических), выявленных на этапе отладки программы, результаты работы (в виде скриншотов), сравнение результатов работы программы на тестовых примерах с результатами ручных расчетов);
- 8) выводы (констатирует решение всех задач, описанных в разделе «Постановка задачи»).

Студент обязан завести рабочую тетрадь, в которой должны фиксироваться инструкции к работе, вопросы, возникающие при выполнении работы и ответы на них, черновики и фрагменты программ. В рабочей тетради можно представлять отчеты по работам. При этом титульный лист каждого отчета должен находиться на правой странице разворота. Рабочую тетрадь обязательно иметь на каждом лабораторном занятии – будут оцениваться качество и полнота выполненных заданий.

Студент допускается к защите отчета по лабораторной работе после того, как он продемонстрирует преподавателю выполнение поставленной задачи на компьютере (результаты работы программы и/или другое задание, например, работу с окном кода BlueJ) и предоставит папку с разработанным java-проектом (в электронном виде). Рекомендуемое имя папки: Лаб_n_m_Фамилия_Группа_Год, где n – номер лабораторной работы, m – номер проекта (используется если задание по лабораторной работе предусматривает разработку нескольких проектов). Текстовый файл с отчетом по работе рекомендуется поместить в папку проекта.

Защита лабораторных работ состоит из доклада студента о проделанной работе с объяснением содержания отчета. Студент должен ответить на контрольные вопросы к соответствующей лабораторной работе, приведенные в методических указаниях, а также другие вопросы преподавателя, касающиеся поставленной задачи, показать работоспособность подготовленной программы и навыки работы с программной системой. При защите текущей работы возможны вопросы по темам предыдущих лабораторных работ. Результат защиты оценивается по шкале 0 – 100 баллов (ESTC).

Для подготовки программ студентам рекомендуется установить на своих компьютерах последнюю версию системы разработки программ BlueJ и соответствующую версию комплекта разработчика Java Development Kit (<http://www.bluej.org>).

Выполнив несложную настройку BlueJ, можно выбрать наиболее удобный для Вас язык интерфейса.

Рекомендуется иметь съемный носитель информации (флэш-диск), на котором будут храниться проекты лабораторных работ и соответствующие отчеты.

2. ЛАБОРАТОРНАЯ РАБОТА № 8. ОБРАБОТКА СИМВОЛЬНЫХ СТРОК. ИСПОЛЬЗОВАНИЕ КЛАССА STRING ПАКЕТА JAVA.LANG. ОБРАБОТКА ИСКЛЮЧЕНИЙ В JAVA

2.1. Цель работы

Целью данной работы является:

- 1) ознакомление с возможностями библиотечного класса String, получение навыков создания и обработки текстовых строк (объектов класса String) с помощью методов класса String;
- 2) получение навыков создания собственных классов, предназначенных для обработки текстовых строк (переменных экземпляра типа String);
- 3) исследование возможностей предотвращения или обработки исключительных ситуаций, которые могут возникнуть в процессе выполнения Java-программы.

2.2. Постановка задачи

Разработать программу, осуществляющую обработку предложения, заданного текстовой строкой. Программа должна разбивать исходное предложение на слова, обрабатывать слова способом, предусмотренным вариантом задания, собирать новое предложение из обработанных слов. В программе должны быть предусмотрены средства для предотвращения или обработки возможных исключительных ситуаций.

2.3. Внеаудиторная подготовка

Для подготовки к лабораторной работе следует изучить краткие теоретические сведения, приведенные в пункте 2.4 методических указаний. Рекомендуется также ознакомиться с [1] (с.331-350, 243-262).

2.4. Краткие теоретические сведения

2.4.1. Работа со строками. Класс String

Как и в большинстве других языков программирования, строка в Java – это последовательность символов. Но, в отличие от многих языков, которые реализуют строки как *символьные массивы*, в Java строки реализуются, как *объекты* типа String.

Реализация строк в виде встроенных объектов обеспечивает полный комплект свойств, которые делают обработку строк очень удобной. Например, в Java существуют методы для сравнения двух строк, поиска подстроки в строке, конкатенации (сцепления) двух строк, изменения регистра символов строки.

Имеется несколько способов создания String-объектов, что облегчает их построение.

После создания String-объекта символы, входящие в строку, нельзя изменять. Чтобы изменить версию существующей строки, нужно создать новый объект типа String, который содержит необходимую модификацию. При этом исходная строка останется неизменной. Этот подход используется потому, что фиксированные (неизменяемые) строки можно реализовать более эффективно, чем изменяемые.

Для тех случаев, когда желательна изменяемая строка, существует компаньон класса String – класс StringBuffer. Объекты этого класса могут изменяться после их создания.

Классы String и StringBuffer определены в пакете java.lang. Таким образом, они доступны всем программам автоматически. Оба класса объявлены как final, т.е. ни один из этих классов не может иметь подклассов (наследников).

Когда говорят, что строки в объектах типа String являются неизменяемыми, это означает, что содержимое String-объекта не может быть изменено после того, как он был создан. Однако переменную, объявленную, как String-ссылка можно в любое время изменить так, чтобы она указывала на другой String-объект.

2.4.1.1. Конструкторы класса String

Класс String поддерживает 11 конструкторов: создать пустую строку, создать строку из массива байтов, создать строку из массива символов, создать копию строки из String-объекта, создать копию строки из объектов StringBuffer или StringBuilder [2]. Ниже приведены заголовки некоторых конструкторов и примеры создания строк.

Конструктор **String()** – участвует в создании пустой строки:

String S1=new String();

(инициализировался String-объект с пустой строкой "", его адрес был присвоен ссылочной переменной S1).

Конструктор **String (char [] chars)** – создает строку из массива символов:

Char [] chars={'a','b','c'};

String S2= new String (chars);

(инициализировался String-объект со строкой "abc", его адрес был присвоен ссылочной переменной S2).

Конструктор **String (String strObj)** – создает String-объект, содержащий такую же символьную строку, как и в String-объекте, переданном в качестве параметра:

String S3=new String ("Hello");

Здесь в качестве параметра в конструктор передан String-объект – литерал. В результате был создан String-объект со строкой "Hello", его адрес был присвоен ссылочной переменной S3.

String S4=new String (S3);

Здесь в качестве параметра в конструктор передана ссылка S3 на String-объект, содержащий значение “Hello”. В результате был создан еще один String-объект со строкой “Hello”, на него ссылается переменная S4). Мы получили два экземпляра одной и той же строки.

2.4.1.2. Длина строки

Длина строки определяется количеством входящих в нее символов. Для получения этого значения вызовите от имени строки метод **int length ()**, определенный в классе String.

После выполнения участка кода

String S=new String("OK");

System.out.println (S.length());

в окно терминала будет выведено число 2.

2.4.1.3. Строковые литералы

Для каждого строкового литерала в программе Java автоматически создает String-объект, поэтому можно создать String-объект, не используя операцию new:

String S1="ABC";

аналогично

String S1=new String ("ABC");

Поскольку объект типа String создается для каждого строкового литерала, то этот литерал можно применять в любом месте, где указывается String-объект, например выполнение оператора

System.out.println ("abc".length());

повлечет за собой вывод в окно терминала числа 3.

2.4.1.4. Операция конкатенации

Операция конкатенации – единственная операция, которую можно применять к String-объектам. Обозначается знаком «+».

В результате выполнения следующего участка кода

String age="30";

String S="Ему "+age+" лет.”;

System.out.println (S);

произойдет конкатенация трех строк и на экран будет выведена строка

"Ему 30 лет." .

2.4.1.5. Методы класса String

Извлечение символов из вызывающей строки.

1) Извлечение одиночного символа.

char charAt (int j) – извлечение одиночного символа из позиции j строки (позиции строки нумеруются, начиная с нуля).

В результате выполнения оператора

char ch="abc".charAt(1);

значение ch будет равно 'b'.

2) Извлечение подстроки:

**void getChars(int sourceStart,
 int sourceEnd,
 char target [],
 int targetStart)**

Подстрока в строке-источнике находится в позициях с sourceStart по (sourceEnd-1). Эта подстрока копируется в символьный массив target, начиная с индекса targetStart. Массив-приемник target должен быть достаточно большим, чтобы вместить все символы подстроки.

3) Извлечение всех символов строки

char [] toCharArray() – возвращает массив символов всей вызывающей строки.

Методы для сравнения строк

boolean equals (String str) – сравнивает вызывающий String-объект со String-объектом str. Возвращает true при полном совпадении строк. Метод чувствителен к регистру.

boolean equalsIgnoreCase (String str) – сравнивает строки без учета регистра.

Важно понимать, что операция, выполняемая методом equals() и операция == – это две различные операции. Метод equals() сравнивает строки внутри String-объектов, а операция == сравнивает две объектные ссылки, чтобы узнать, указывают ли они на один и тот же экземпляр (String-объект).

На рисунке 2.1 изображена конфигурация объектов и ссылочных переменных в оперативной памяти, полученная после выполнения следующего участка кода:

```
String S1="Привет, друзья!";
String S2=new String (S1);
String S3=S2;
```

Для данного случая операции сравнения вернут следующие результаты:

if (S1==S2)... значение логического условия равно **false**;
if (S3==S2)... значение логического условия равно **true**;
if (s3.equals(s1))... значение логического условия равно **true**;

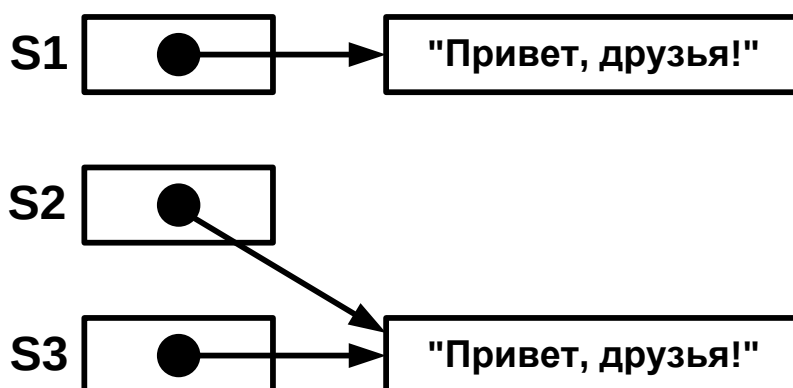


Рисунок 2.1 – Ссылочные переменные и String-объекты

Часто не достаточно просто знать идентичны ли две строки. Для приложений сортировки нужно знать, какая из них меньше, равна или больше, чем другая.

Одна строка считается меньше, чем другая, если она расположена перед другой в словарном (упорядоченном по алфавиту) списке. Строка считается больше, чем другая, если она расположена после другой в словарном списке.

Такое сравнение выполняет String-метод `compareTo()`, имеющий следующую сигнатуру:

int compareTo (String str);

Здесь `str` – ссылка на String-объект, сравниваемый с вызывающим String-объектом. Метод возвращает целочисленное значение – результат сравнения, который интерпретируется так:

меньше нуля – вызывающая строка меньше строки-параметра;

больше нуля – вызывающая строка больше строки-параметра;

равен нулю – вызывающая строка равна строке-параметру.

В ASCII коды прописных букв меньше, чем коды строчных. Метод

int compareToIgnoreCase (String str);

сравнивает две строки, игнорируя различия в регистре.

Метод `endsWith()` проверяет, заканчивается ли запускающая строка подстрокой, указанной в качестве параметра:

boolean endsWith(String suffix);

Методы, возвращающие позицию символа или подстроки в строке

int indexOf (int ch) – ищет первое вхождение символа ch в вызывающую строку и возвращает номер соответствующей позиции;

int lastIndexOf (int ch) – ищет последнее вхождение символа ch в вызывающую строку и возвращает номер соответствующей позиции;

int indexOf (String str) – ищет первое вхождение подстроки str в вызывающую строку и возвращает номер соответствующей позиции;

int lastIndexOf (String str) – ищет последнее вхождение подстроки str в вызывающую строку и возвращает номер соответствующей позиции.

Если искомый символ или подстрока не найдены, методы `indexOf` и `lastIndexOf` возвращают отрицательное значение.

Методы, возвращающие ссылку на новую строку, приведены в таблице 2.1. Нестатические методы обрабатывают образ (копию) строки вызывающего String-объекта (строка в вызывающем String-объекте не изменяется).

Таблица 2.1 – Методы, меняющие образ строки

String substring (int startIndex)	Извлечение подстроки из вызывающей строки
String substring (int startIndex, int endIndex)	
String concat (string str)	Конкатенация строк
String replace (char original, char replacement)	Замена одного символа другим в возвращаемой строке (вызывающая не меняется)
String trim()	Удаляет все пробелы в начале и конце вызывающей строки
Static String valueOf (double num)	Преобразует значение в строковое (используется, например, когда надо выполнить операцию конкатенации). Любой объект, передаваемый в метод <code>valueOf</code> возвращает результат обращения к методу <code>toString ()</code>
Static String valueOf (long num)	
Static String valueOf (object ob)	
Static String valueOf (char [] chars)	
String toLowerCase ()	Приведение символов вызывающей строки к верхнему регистру
String toUpperCase ()	Приведение символов вызывающей строки к нижнему регистру

2.4.2. Обработка исключений в Java

Во время выполнения программы на некоторых наборах исходных данных могут возникать ошибки. Например, пользователь вводит в программу данные, обработка которых вызывает исключительную ситуацию «деление на ноль», что приводит к остановке выполнения программы и передаче управления исполнительной системе.

Большинство пользователей было бы смущено, если бы ваша программа прекращала свою работу и печатала трассу стека (обратную последовательность вызова методов, начиная с метода, в котором произошло прерывание) каждый раз, когда произошла ошибка.

Обнаружив возможность подобной ситуации в своей программе, программист может решить проблему двумя способами.

Первый способ: предотвратить ошибку. Программист может добавить в программу участок кода, проверяющий входные данные.

Предположим, что фрагмент программы использует данные a , b , c , и для нормального выполнения должны соблюдаться условия:

$$(a0 \leq a \leq a1) \text{ и } (b0 \leq b \leq b1) \text{ и } (c0 \leq c \leq c1)$$

Возможный вариант кода:

```
if ((a0<=a)&&(a<=a1)&&(b0<=b)&&(b<=b1)&&(c0<=c)&&(c<=c1))
    {выполнить фрагмент}
else {как-то сообщить о невозможности
      выполнения и решить, что делать
      дальше}
```

Второй способ: можно использовать специальные возможности языка Java. Java предлагает более изящный метод обработки исключительных ситуаций, переводя управление обработкой ошибок времени выполнения в объектно-ориентированное русло.

Исключение (исключительная ситуация) – это *аварийное состояние*, которое возникает во время выполнения кодовой последовательности (**ошибка времени выполнения**).

Когда исключительное событие возникает, исполнительной системой создается объект, представляющий это исключение, который «вбрасывается» в метод, вызвавший ошибку (выбросивший исключение). Метод может выбрать, обрабатывать ли исключение самому, или передать куда-то еще. **В любом случае, в некоторой точке исключение «захватывается» и обрабатывается.**

Выбрасываемые исполнительной системой исключения касаются фундаментальных ошибок, которые нарушают ограничения среды выполнения или правила языка Java (деление на ноль, значение индекса массива находится вне допустимого диапазона и т.п.).

Для того, чтобы отслеживать и обрабатывать ошибку времени выполнения, включите код, который нужно контролировать, внутрь блока **try** (защищенного блока). Сразу после блока **try** укажите **catch**-блок, определяющий тип исключения, которое нужно перехватить, и его обработчик.

Общая форма защищенного блока:

```
try{
    // блок кода для контроля над ошибками
}
catch (ExceptionType1 exOb) {
    // обработчик исключений типа ExceptionType1
}
catch (ExceptionType2 exOb) {
    // обработчик исключений типа ExceptionType2
}
//...
[finally {
    // блок кода, выполняющийся после того, как блок try/catch
    // завершился и перед кодом, следующим за блоком try/catch
}]
```

Здесь **ExceptionType** – тип (класс) исключения, которое возникло;
exOb – объект этого исключения;
finally – блок заключен в квадратные скобки, так как он не является обязательным. Если он все-таки есть, то он выполняется независимо от того, было выброшено исключение или нет.

Все классы исключений являются потомками класса **Throwable**

Подклассы **Throwable**:

1) Класс **Exception** – используется для исключительных ситуаций, которые должны перехватывать программы пользователя.

У **Exception** имеется важный подкласс **RuntimeException**, Исключения этого класса для ваших программ определены автоматически и включают такие события, как деление на ноль, недопустимая индексация массива и т.п.

В подклассах этого класса пользователь может создавать и свои собственные заказные типы исключений.

2) Класс **Error** определяет исключения, перехват которых вашей программой не ожидается. Они вызываются не кодом вашей программы и применяются исполнительной системой для указания ошибок, имеющих отношение к самой среде времени выполнения (ошибки Java-машины). Пример – переполнение системного стека.

Внимание! Защищенный блок не является средством, позволяющим исключить или облегчить процесс отладки программы. Он используется для обеспечения реакции полностью отлаженной программы на некоторые наборы входных данных в **ПРЕДУСМОТРЕННЫХ** программистом случаях.

Ниже будет рассмотрен пример обработки исключительной ситуации типа **ArrayIndexOutOfBoundsException** (индекс массива вышел за пределы).

2.4.3. Пример программы обработки символьных строк

Требуется разработать программу, позволяющую разбивать исходную строку на слова, производить обработку слов заданными методами (изменять или удалять некоторые слова) и собирать из обработанных слов новую строку. Слова в строке отделены друг от друга одним или несколькими пробелами. Число слов в предложении ограничено.

Разработаем класс Sentence (Предложение), объект которого представляет собой «интеллектуальную строку», поведение которой определяется методами класса. Поля и методы класса описаны в таблице 2.2.

Таблица 2.2 – Описание класса Sentence (Предложение)

Обозначение	Описание
Именованная константа	
private final static int MAX_NUM_OF_WORDS	ограничение на максимальное число слов в предложении (исходной строке);
Поля (переменные экземпляра):	
private String sourceStr	ссылка на исходную строку (String-объект);
private String [] words;	ссылка на массив слов исходной строки;
private int counter;	число слов в исходной строке;
Методы:	
public Sentence ()	конструктор: поле sourceStr инициализируется ссылкой на пустую строку ("");
public Sentence (String str)	конструктор: поле sourceStr инициализируется значением ссылки str;
public void setSourceStr(String str)	сеттер: sourceStr инициализируется значением ссылки str;
public String[] getWords()	геттер: возвращает значение поля words (ссылку);
public int getCounter()	геттер: возвращает значение поля counter;
public String toString()	возвращает строку описания Sentence-объекта, совпадающую с исходной строкой (sourceStr);
private void splitIntoWords()	Заполняет массив, на который указывает поле words словами, найденными в исходной строке;
public String [] wordsLengthAboveN(int N)	возвращает ссылку на массив слов, длина которых больше N, полученных из исходной строки;
public String [] wordsLetterFirstEqvLast ()	возвращает ссылку на массив слов, у которых первая буква равна последней, полученных из исходной строки;
public String capitalFirstLetterInWords()	возвращает строку, полученную из исходной: первая буква каждого слова делается заглавной, остальные – строчными;

Продолжение таблицы 2.2

Обозначение	Описание
public String deleteWordsLengthAboveN(int N)	возвращает строку, полученную из исходной: удаляются все слова, длина которых меньше или равна N.

На рисунке 2.2 приведен алгоритм метода splitIntoWords().

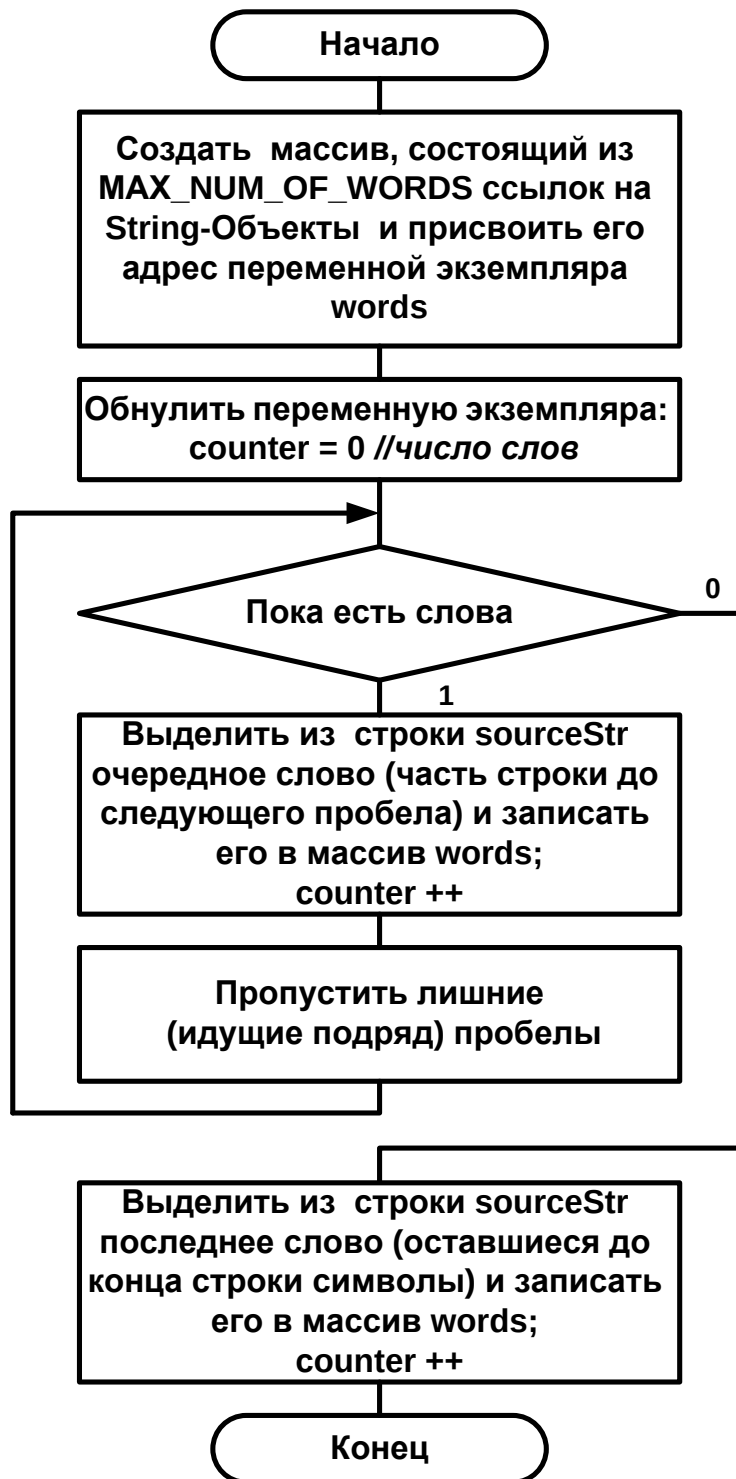


Рисунок 2.2 – Алгоритм метода splitIntoWords (разбить на слова)

//разделить на слова

```
private void splitIntoWords(){
    //Создает массив слов исходной строки, его адрес присваивает
    //переменной экземпляра words, в переменной экземпляра
    //counter вычисляет число слов
    words = new String[ MAX_NUM_OF_WORDS ];
    int start, end;
    counter = 0; //счетчик слов - слов еще не нашли
    start = 0; //начало первого слова - самый левый символ строки
    end = sourceStr.indexOf(" "); //найти конец первого слова
    while (end > 0){ //пока есть слова
        words[counter] = sourceStr.substring(start, end); //очередное слово
        counter++; //увеличиваем для следующего слова
        while (sourceStr.charAt(end) == ' ') end++;
        start = end; //теперь лишние пробелы будут пропущены
        end = sourceStr.indexOf(" ", start);
    } //while
    words[counter] = sourceStr.substring(start); //последнее слово
    counter++; //число слов
} //splitIntoWords()
```

...

Очевидно, что если исходная строка содержит более **MAX_NUM_OF_WORDS** = 4 слов, при выполнении кода одной из строк

S[counter] = sourceStr.substring(start,end); *//очередное слово* или

S[counter] = sourceStr.substring(start); *//последнее слово*

может возникнуть исключение класса **ArrayIndexOutOfBoundsException** (индекс массива вышел за пределы). Данная ситуация изображена на рисунке 2.4.

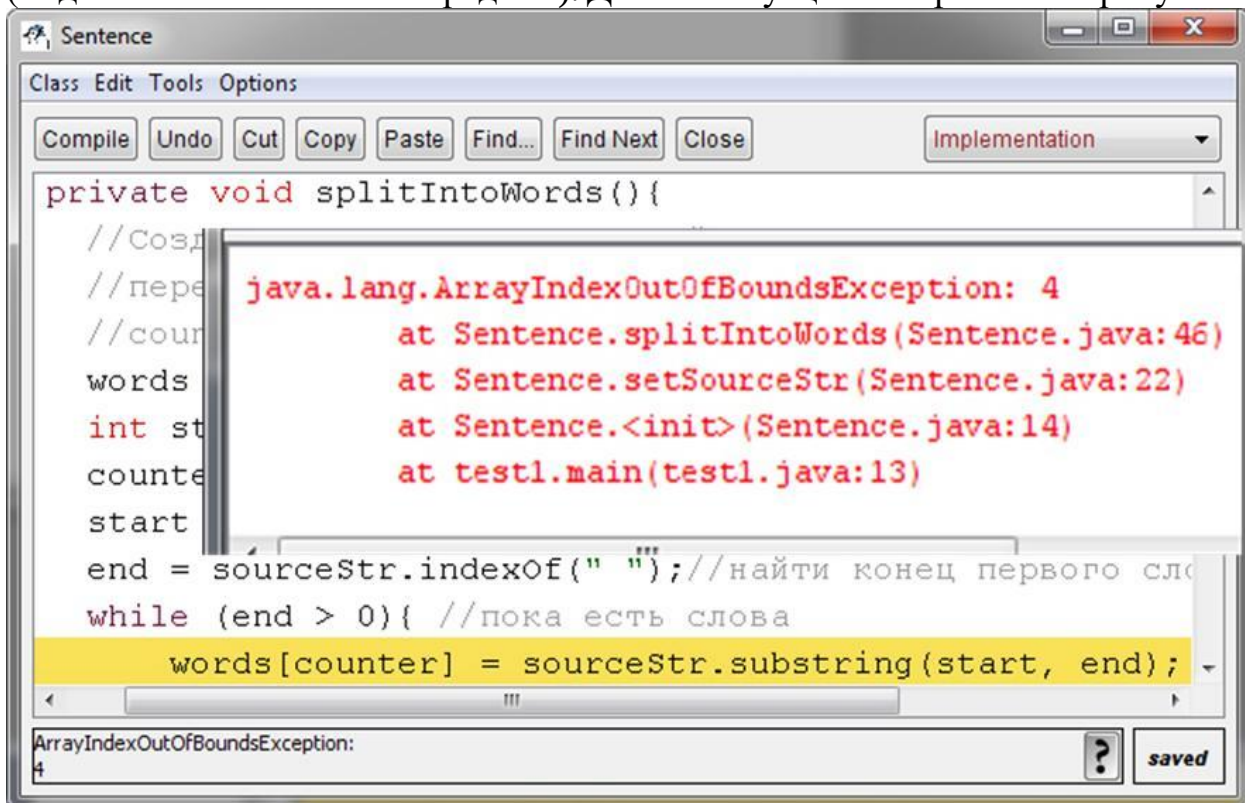


Рисунок 2.4 – Исключительная ситуация при выполнении программы

Чтобы метод `splitIntoWords()`, выбрасывающий исключение, мог перехватить и обработать это исключение, нужно заключить «опасный» участок кода в защищенный блок:

//разделить на слова

```
private void splitIntoWords(){
    words = new String[ MAX_NUM_OF_WORDS ];
    int start, end;
    counter = 0; //счетчик слов - слов еще не нашли
    start = 0; //начало первого слова - самый левый символ строки
    end = sourceStr.indexOf(" "); //найти конец первого слова
    try { //а вдруг индекс выйдет за границу массива?
        while (end > 0){ //пока есть слова
            words[counter] = sourceStr.substring(start, end); //очередное слово
            counter++; //увеличиваем для следующего слова
            while (sourceStr.charAt(end) == ' ') end++;
            start = end; //теперь лишние пробелы будут пропущены
            end = sourceStr.indexOf(" ", start);
        } //while
        words[counter] = sourceStr.substring(start); //последнее слово
    } //try
    catch (ArrayIndexOutOfBoundsException e){
        System.out.println(
            "В предложении слишком много слов, лишние придется обрезать: "
            + "\n"+e);
        System.out.println();
        counter--;
    }
    counter++; //число слов
} //splitIntoWords()
```

Выполнение следующего кода в классе `Test` обеспечит вывод в окно терминала результатов, изображенных на рисунке 2.5.

```
public class Test {
    // вывод массива строк
    public static void putArrayOfStrings (String [ ] strings){
        if (strings == null) return;
        for (int i = 0; i < strings.length; i++){
            if (strings[i] == null) break;
            System.out.println (strings[i]);
        }
    }
    public static void main (String[ ] args){
        // Создание и обработка объекта-предложения
        Sentence sent = new Sentence(
            " А Алиса спрятала ключик под коврик ");
        System.out.println (sent);
        System.out.println ();
        System.out.println ("Слова исходной строки:");
        putArrayOfStrings(sent.getWords());
    }
}
```



```

//геттеры:
//возвращает ссылку на массив слов исходной строки
public String[ ] getWords() {return words;}
//возвращает число слов в исходной строке
public int getCounter() {return counter;}

// возвращает исходную строку
public String toString(){
    return sourceStr;
}
//разделить на слова
private void splitIntoWords(){
    //Создает массив слов исходной строки, его адрес присваивает
    //переменной экземпляра words, в переменной экземпляра
    //counter вычисляет число слов
    words = new String[ MAX_NUM_OF_WORDS ];
    int start, end;
    counter = 0; //счетчик слов - слов еще не нашли
    start = 0; //начало первого слова - самый левый символ строки
    end = sourceStr.indexOf(" "); //найти конец первого слова
    try { //а вдруг индекс выйдет за границу массива?
        while (end > 0){ //пока есть слова
            words[counter] = sourceStr.substring(start, end); //очередное слово
            counter++; //увеличиваем для следующего слова
            while (sourceStr.charAt(end) == ' ') end++;
            start = end; //теперь лишние пробелы будут пропущены
            end = sourceStr.indexOf(" ", start);
        } //while
        words[counter] = sourceStr.substring(start); //последнее слово
    } //try
    catch (ArrayIndexOutOfBoundsException e){
        System.out.println(
            "В предложении слишком много слов, лишние придется обрезать: "
            + "\n"+e);
        System.out.println();
        counter--;
    }
    counter++; //число слов
} //splitIntoWords()

// возвращает ссылку на массив слов исходной строки,
// длина которых больше N
public String [ ] wordsLengthAboveN(int N){
    if (counter == 0) return null;
    // создадим новый массив строк
    String [ ] arr = new String[counter];
    int j = 0; // индекс нового массива
    for (int i = 0; i < counter; i++){
        if (words[i].length() > N){
            arr[ j ] = words[ i ];
            j = j + 1;
        }
    }
}

```

```

return arr;
}

// возвращает ссылку на массив слов исходной строки,
// у которых первая буква равна последней
public String [ ] wordsLetterFirstEqvLast (){
    if (counter == 0) return null;
    char c1, cN;
    // создадим новый массив строк
    String [ ] arr = new String[counter];
    int j = 0; // индекс нового массива
    for (int i = 0; i < counter; i++){
        c1 = words[ i ].charAt(0);
        cN = words[ i ].charAt (words[ i ].length() - 1);
        if (c1 == cN){
            arr[ j ] = words[ i ];
            j = j + 1;
        }
    }
    return arr;
}

```

```

// Возвращает строку, полученную из исходной:
// первая буква каждого слова делается заглавной,
// остальные - строчными
public String capitalFirstLetterInWords(){
    if (sourceStr.equals(""))return "";
    String str = "";
    for (int i = 0; i < counter; i++){
        str = str + " " + words[i].substring(0,1).toUpperCase() +
            words[i].substring(1).toLowerCase();
    }
    // удаляем образовавшийся в начале строки пробел
    //и возвращаем строку
    return str.trim();
}

```

*Конкатенация → «замусоривание»
памяти при больших размерах
массива words!*

```

// Возвращает строку, полученную из исходной:
// первая буква каждого слова делается заглавной,
// остальные - строчными
public String capitalFirstLetterInWords(){
    if (sourceStr.equals(""))return "";
    //Создаем объект класса StringBuilder, который позволяет
    //работать с изменяемой строкой:
    StringBuilder result = new StringBuilder();
    for (int i = 0; i < counter; i++){
        result.append(" ");
        result.append(words[i].substring(0,1).toUpperCase());
        result.append(words[i].substring(1).toLowerCase());
    }
    // удаляем образовавшийся в начале строки пробел:
    result.delete(0, 1);
    return result.toString();
}

```

```

// Возвращает строку, полученную из исходной:
// удаляются все слова,
// длина которых меньше или равна N
public String deleteWordsLengthAboveN(int N){
    if (sourceStr.equals("")) return "";
    StringBuilder result = new StringBuilder();
    for (int i = 0; i < counter; i++){
        result.append(" ");
        if (words[i].length() > N) result.append(words[i]);
    }
    // удаляем образовавшийся в начале строки пробел
    result.delete(0, 1);
    return result.toString();
}
} // Sentence

public class Test {
    // вывод массива строк
    public static void putArrayOfStrings (String [ ] strings){
        if (strings == null) return;
        for (int i = 0; i < strings.length; i++){
            if (strings[i] == null) break;
            System.out.println (strings[i]);
        }
    }
    public static void main (String[ ] args){
        int N = 4;
        // Создание и обработка первого объекта-предложения
        Sentence sent = new Sentence(
            " А Алиса спрятала ключик под коврик ");
        System.out.println (sent); //sent.toString()
        System.out.println ();
        System.out.println ("Слова исходной строки:");
        putArrayOfStrings(sent.getWords());
        System.out.println ();
        System.out.printf ("Вывести слова длиной больше %d символов\n", N);
        putArrayOfStrings (sent.wordsLengthAboveN(N));
        System.out.println ();
        System.out.println (
            "Вывести слова, у которых первая и последняя буква одинаковые");
        putArrayOfStrings(sent.wordsLetterFirstEqvLast ());
        System.out.println ();
        System.out.println (
            "Сделать первую букву каждого слова в предложении заглавной");
        System.out.println (sent.capitalFirstLetterInWords());
        System.out.println ();
        System.out.println (
            "Убрать из предложения слова, длина которых меньше или равна "+N);
        System.out.println (sent.deleteWordsLengthAboveN(N));
        System.out.println ();
    }
}

```

```

//Создание и обработка второго предложения
System.out.println (
    "Предложение, полученное из исходного следующим образом: \n"+
    "сначала удалены слова, длина которых меньше или равна "+N+", \n"+
    "а затем первые буквы слов сделаны заглавными:");
System.out.println ();
//Создаем новую ссылку sent1 на объект-предложение и сам объект,
// который инициализируется строкой,
// возвращенной методом DeleteWordsLengthAboveN,
// запущенным от имени первого объекта-предложения
// (на него указываем sent)
Sentence sent1 = new Sentence(sent.deleteWordsLengthAboveN(N));
//Создаем новый объект-предложение, который
// инициализируется строкой, возвращенной методом
// CapitalFirstLetterInWords, запущенным от имени объекта,
// на который указывает sent1,
// затем ссылке sent1 присваивается адрес этого нового объекта.
// Связь с объектом, на который ранее указывала
// ссылочная переменная sent1, теряется. Он будет «освобожден»
// «Сборщиком мусора»
sent1 = new Sentence(sent1.capitalFirstLetterInWords());
System.out.println (sent1); //sent1.toString()
System.out.println ();
// Применяем к объекту, на который указывает sent1,
// методы класса Sentence
System.out.println ("Слова исходной строки нового предложения:");
putArrayOfStrings(sent1.getWords());
System.out.println ();
N = 5;
System.out.printf ("Вывести слова длиной больше %d символов\n",N);
putArrayOfStrings(sent1.wordsLengthAboveN(N));
System.out.println ();
System.out.println (
    "Вывести слова, у которых первая и последняя буква одинаковые");
putArrayOfStrings(sent1.wordsLetterFirstEqvLast ());
System.out.println ();
//местуем работу методов, если исходная строка - пустая
System.out.println (
    "Создаем предложение с пустой строкой и поочередно");
System.out.println ("запускаем методы обработки предложения ");
Sentence sent2 = new Sentence();
putArrayOfStrings(sent2.getWords());
putArrayOfStrings (sent2.wordsLengthAboveN(N));
putArrayOfStrings(sent2.wordsLetterFirstEqvLast ());
System.out.println (sent2.capitalFirstLetterInWords());
System.out.println (sent2.deleteWordsLengthAboveN(N));
}
} //Test

```

Результаты работы программы представлены на рисунках 2.6, 2.7.

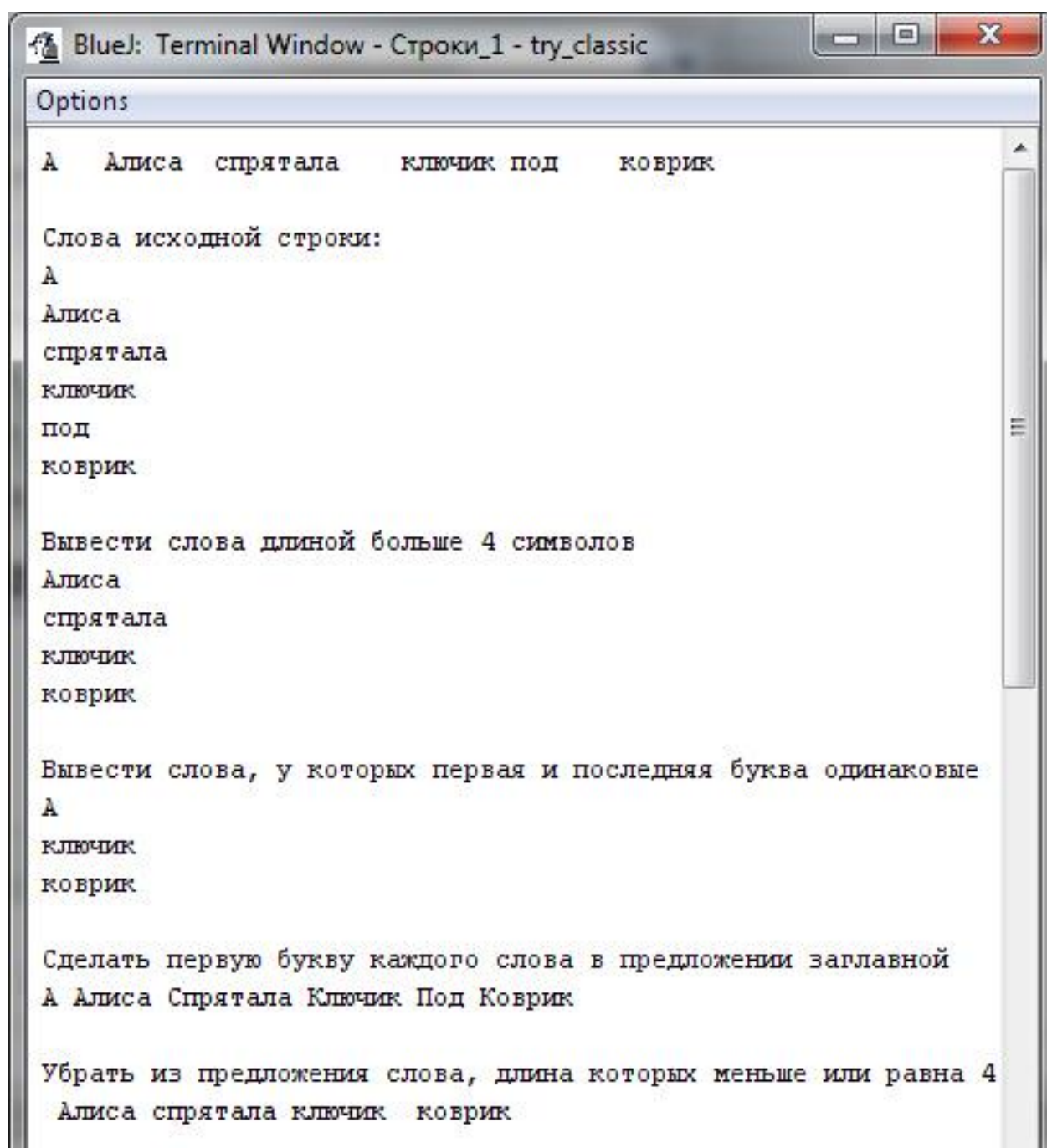


Рисунок 2.6 – Результаты работы проекта

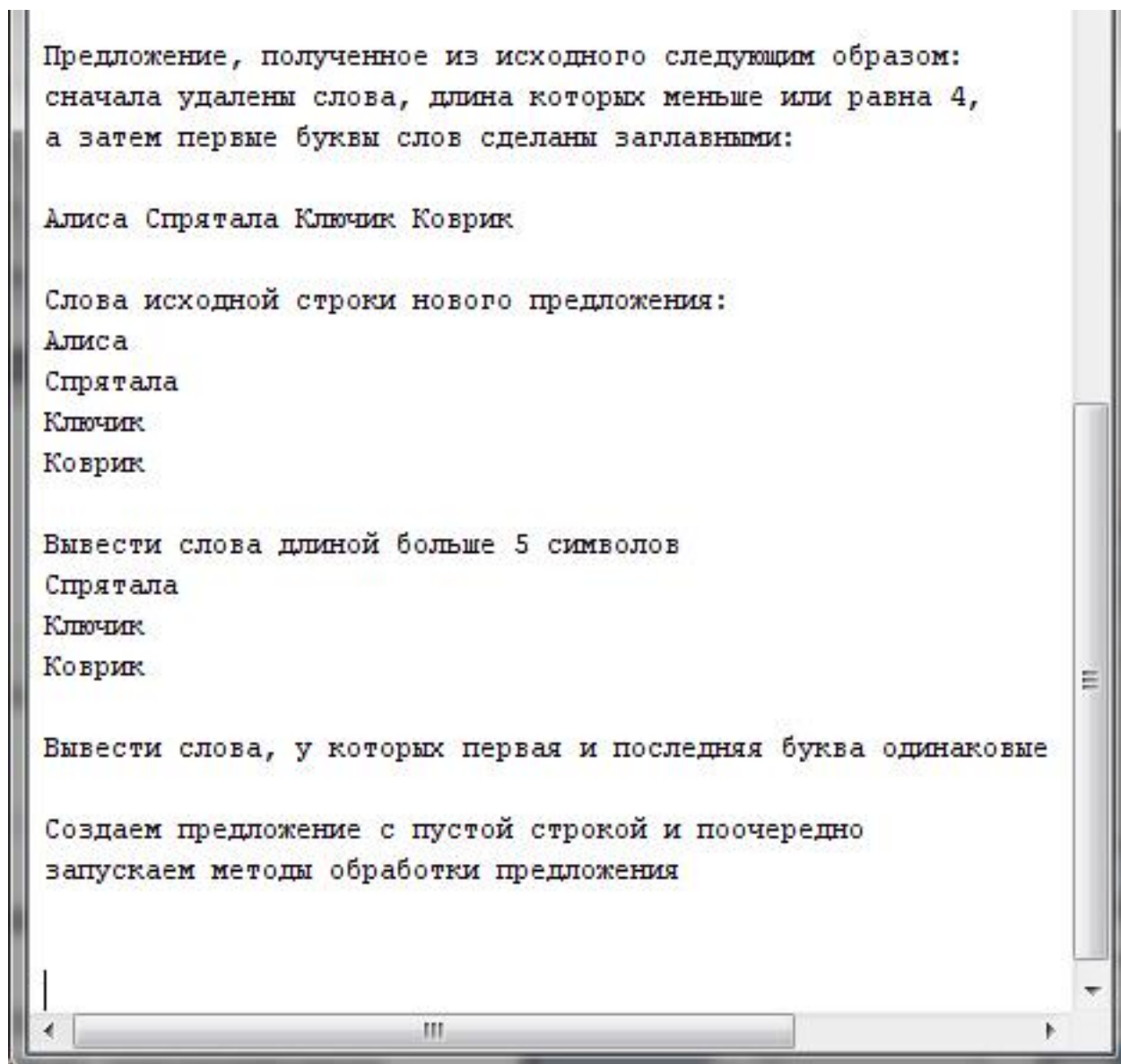


Рисунок 2.7 – Результаты работы проекта (продолжение)

Следует отметить, что ограничение на количество слов в предложении (**MAX_NUM_OF_WORDS**) введено, главным образом, с целью смоделировать возникновение исключения класса **ArrayIndexOutOfBoundsException** и показать использование защищенного блока (try-catch) в программе. Поскольку в Java массив является объектом (создается во время выполнения программы), можно программно создавать «растущие» массивы, т.е. после заполнения некоторого массива элементами создавать новый массив большей величины, копировать в него элементы из старого массива, после чего ссылочной переменной, указывающей на старый массив, присваивать значение ссылки на новый массив.

В заключении сделаем еще одно важное замечание, касающееся применения рекурсии при определении данных. Если пользователь хочет, чтобы методы **CapitalFirstLetterInWords()** и **DeleteWordsLengthAboveN(int N)** возвращали не простые строки (ссылки на объекты типа String), а «интеллектуальные», т.е. умеющие выполнять методы класса **Sentence**, можно изменить код указанных методов так, чтобы они возвращали ссылки на объекты класса **Sentence**:

```
class Sentence { //Предложение
...
// Возвращает строку, полученную из исходной:
// первая буква каждого слова делается заглавной,
// остальные - строчными
public Sentence capitalFirstLetterInWords(){
    if (sourceStr.equals(""))return "";
    StringBuilder result = new StringBuilder();
    for (int i = 0; i < counter; i++){
        result.append(" ");
        result.append(words[i].substring(0,1).toUpperCase());
        result.append(words[i].substring(1).toLowerCase());
    }
    // удаляем образовавшийся в начале строки пробел
    result.delete(0, 1);
    return new Sentence (result.toString());
}

// Возвращает строку, полученную из исходной:
// удаляются все слова,
// длина которых меньше или равна N
public Sentence deleteWordsLengthAboveN(int N){
    if (sourceStr.equals("")) return "";
    StringBuilder result = new StringBuilder();
    for (int i = 0; i < counter; i++){
        result.append(" ");
        if (words[i].length() > N)result.append(words[i]);
    }
    // удаляем образовавшийся в начале строки пробел
    result.delete(0, 1);
    return new Sentence (result.toString());
}
} // Sentence

public class Test {
...
//Создание и обработка второго предложения
System.out.println (
    "Предложение, полученное из исходного следующим образом: \n"+
    "сначала удалены слова, длина которых меньше или равна "+N+", \n"+
    "а затем первые буквы слов сделаны заглавными:");
System.out.println ();
```

```

//Создаем новую ссылку sent1 на объект-предложение и сам объект.
// В следующем операторе результатом каждой
// операции разыменования (.) является адрес нового
// объекта класса Sentence, к которому можно
// применять методы класса Sentence
Sentence sent1=sent.DeleteWordsLengthAboveN(N).CapitalFirstLetterInWords();
System.out.println (sent1);
...
}
} // Test

```

2.5. Порядок выполнения работы

1) Внимательно изучите текст классов проекта-примера **Строки_1-try_classic** (приведен выше). Особое внимание уделите тому, как метод «Разбить на слова» (**splitIntoWords()**) обрабатывает исключение, которое может возникать на некоторых наборах исходных данных.

2) Разработайте алгоритмы методов, определенных вариантом задания.

3) Создайте новый проект с названием **Фамилия_Группа_lab6_1_2sem**. Разработайте исходный текст класса «Предложение». Возьмите за основу соответствующий код из проекта-примера и включите в него методы обработки предложения, определенные вариантом задания (методы обработки предложения, которые были даны в примере, можете исключить).

4) Разработайте класс Test, проверяющий работу заданных методов подобно тому, как это сделано в проекте-примере.

5) Проведите отладку программы, добейтесь того, чтобы она выдавала ожидаемые результаты.

6) Смоделируйте исключительную ситуацию и проверьте, как ваша программа на нее реагирует.

7)* Создайте новый проект с названием **Фамилия_Группа_lab6_2_2sem**. Разработайте статические методы обработки символьных строк (не менее двух), поместите их в класс-библиотеку статических методов **StringProcessLib**. Создайте класс Test для тестирования разработанных методов. Проведите отладку программы, добейтесь того, чтобы она выдавала ожидаемые результаты *(выполнение этого задания обязательно для студентов, претендующих на отличную оценку)*.

2.6. Варианты заданий

Вариант задания определяется таблицей 2.3, в которой заданы методы класса «Предложение», подлежащие программированию. Исходная строка, определенная в классе «Предложение» состоит из слов, разделенных одним или несколькими пробелами. Словом считается любая совокупность символов, находящаяся между двумя соседними сериями пробелов.

Номер варианта задания V необходимо вычислить по формуле

$$\text{int } V = (N \leq 14) ? \text{variant} : N \% 14;$$

где N – номер студента в списке группы,

variant – номер варианта задания в таблице 2.1.

В таблице 2.4 перечислены статические методы обработки текстовых строк для выполнения задания 7)* п. 2.5. Студент должен разработать метод, соответствующий номеру варианта задания и еще несколько (не менее одного) методов (по согласованию с преподавателем). При создании указанных методов можно использовать методы класса String, методы класса Character и собственные (разработанные студентом) методы.

Таблица 2.3 – Методы класса «Предложение»

Номер варианта	Содержание методов
1	1) Возвращает массив слов, начинающихся на заданную букву; 2) Возвращает строку, в которой оставлены слова, начинающиеся на заданную букву и длина которых меньше заданной.
2	1) Возвращает массив слов, в которых есть заданное буквосочетание (подстрока); 2) Возвращает строку, в которой оставлены слова, включающие заданное буквосочетание и длина которых больше заданной.
3	1) Возвращает массив слов, в которых есть заданная буква; 2) Возвращает строку, в которой оставлены слова, включающие заданную букву и оканчивающиеся на 'ь'.
4	1) Возвращает массив слов, оканчивающихся на заданную букву; 2) Возвращает строку, в которой оставлены слова, оканчивающиеся на заданную букву и начинающиеся на букву 'а'.
5	1) Возвращает массив слов, начинающихся с заданного буквосочетания (подстроки); 2) Возвращает строку, в которой оставлены слова, начинающихся с заданного буквосочетания и длина которых меньше заданной.
6	1) Возвращает массив слов, оканчивающихся заданным буквосочетанием (подстрокой); 2) Возвращает строку, в которой оставлены слова, оканчивающиеся заданным буквосочетанием и длина которых больше заданной.

Продолжение таблицы 2.3

Номер варианта	Содержание методов
7	1) Возвращает массив слов, содержащих специальные символы '@', '&', '#'; 2) Возвращает строку, в которой оставлены слова, содержащие специальные символы '@', '&', '#', длина которых находится в заданных пределах.
8	1) Возвращает массив слов, включающих цифры; 2) Возвращает строку, в которой оставлены слова, включающие цифры, длина которых находится в заданных пределах.
9	1) Возвращает массив слов, длина которых больше заданной; 2) Возвращает строку, в которой оставлены слова, начинающиеся на заданную букву и длина которых больше заданной.
10	1) Возвращает массив слов, длина которых меньше заданной. 2) Возвращает строку, в которой оставлены слова, включающие заданное буквосочетание и длина которых меньше заданной.
11	1) Возвращает массив слов, оканчивающихся на 'ь'; 2) Возвращает строку, в которой оставлены слова, включающие заданную букву и оканчивающиеся на 'ь'.
12	1) Возвращает массив слов, начинающихся на букву 'а'; 2) Возвращает строку, в которой оставлены слова, оканчивающиеся на заданную букву и начинающиеся на букву 'а'.
13	1) Возвращает массив слов, начинающихся на заданную букву; 2) Возвращает строку, в которой оставлены слова, начинающиеся на заданную букву и оканчивающиеся на заданную букву
14	1) Возвращает массив слов, длина которых находится в заданных пределах; 2) Возвращает строку, в которой оставлены слова, содержащие специальные символы '@', '&', '#', длина которых находится в заданных пределах.

Таблица 2.4 – Статические методы обработки символьных строк, подлежащие разработке (класс StringProcessLib)

Номер метода	Содержание метода
1	public static int compSubStr (String s, String p), подсчитывающий, сколько раз подстрока p входит в строку s.
2	public static String compareAndCreateStr(String s1, String s2), создающий из строк s1 и s2 строку, в которой строки s1 и s2 расположены в порядке: меньшая, большая (сравнение строк лексикографическое).
3	public static int compWords(String s), возвращающий количество слов в строке s. Слова разделены одним или несколькими пробелами.
4	public static int wordsNumStartLetter (String[] words, char letter), возвращающий количество строк в массиве строк words, начинающихся с буквы letter.
5	public static int wordsNumWithLength (String[] words, int lg), возвращающий число строк в массиве строк words, длина которых меньше lg.
6	public static int wordsNumWithSubstr (String[] words, String s), возвращающий число элементов массива строк words, включающих буквосочетание, заданное строкой s.
7	public static String revers (String s), возвращающий строку, в которой порядок следования символов противоположен порядку следования символов в строке s
8	public static String [] palindromes(String s), возвращающий массив слов строки s, которые одинаково читаются слева направо и справа налево
9	public static String replacement1 (String s, char oldChar, char newChar), возвращающий строку, являющуюся образом строки s, в которой все символы, равные oldChar, заменены на символ newChar.
10	public static String replacement2 (String s), возвращающий строку, являющуюся образом строки s, в которой все буквы заменены на символ '#', все цифры заменены на символ '&'.
11	public static String lowerSubstr(String s, String substr), возвращающий строку, являющуюся образом строки s, в которой все буквосочетания, совпадающие с substr (регистр, в котором набраны буквы не учитывается) приводятся к нижнему регистру. Все остальные буквы должны быть приведены к верхнему регистру.

Продолжение таблицы 2.4

Номер метода	Содержание метода
12	<code>public static String[] upperSubstr (String s, String substr, char letter),</code> возвращающий массив слов строки s, в котором все слова, начинающиеся на заданную букву, приведены к верхнему регистру.
13	<code>public static String concat1(String s, String str),</code> возвращающий строку, в которой к строке s присоединена строка str, если строки не равны, и '!', если строки равны.
14	<code>public static String loverUpper(String s, char ch),</code> возвращающий строку, являющуюся образом исходной строки, в которой все символы, встретившиеся до первого вхождения символа ch, приведены к нижнему регистру (сам символ ch остается без изменения), а остальные символы – к верхнему регистру.
15	<code>public static String[] stringSort1(String[] words),</code> возвращающий отсортированный по возрастанию массив строк. Исходный массив может содержать null-ссылки и ссылки на пустые строки (""). В отсортированном массиве их быть не должно.
16	<code>public static String[] stringSort2(String[] words),</code> возвращающий отсортированный по убыванию массив строк. Исходный массив может содержать null-ссылки и ссылки на пустые строки (""). В отсортированном массиве их быть не должно.
17	<code>public static String stringWordsReplace (String s, int typeSort),</code> возвращающий строку, являющуюся образом строки s, в которой слова переставлены в порядке typeSort: если typeSort<0 – по убыванию, если typeSort>0 – по возрастанию. Если typeSort=0, метод возвращает образ исходной строки без изменения.
18	<code>public static String doublingSubstr(String s, String substr),</code> возвращающий строку, являющуюся образом строки s, в которой удвоены все найденные подстроки substr.
19	<code>public static String collectWords (String [] words),</code> возвращающий строку, полученную путем объединения через пробел слов массива words. Массив words может содержать null-ссылки, сосредоточенные в конце массива.

2.7. Содержание отчета

- 1) Цель работы.
- 2) Постановка задачи с описанием варианта задания.
- 3) Алгоритмы разработанных методов.
- 4) Структура проекта.
- 5) Текст классов.
- 6) Протокол отладки (описание найденных синтаксических и логических ошибок в программе и способов их устранения).
- 7) Результаты выполнения программы, в том числе, для случая, когда в программе возникает исключение.
- 8) Постановка задачи, алгоритмы разработанных методов, структура проекта, текст классов, протокол отладки и результаты выполнения программы для задания 7)* п. 2.5.
- 9) Выводы.

2.8. Контрольные вопросы

1) Как реализуются строки в Java? Изобразите схематично состояние оперативной памяти после выполнения следующего участка кода:

```
String s1="Hello!";
String s2= new String (s1);
String s3=s2;
```

Какого типа переменные s1, s2 и s3?

2) Как можно создать строку в Java? Какие конструкторы класса String вам известны?

3) Как определить длину строки для String-переменной и String-литерала? Приведите пример.

4) Какая операция определена для String-объектов? Приведите пример.

5) Какие методы класса String позволяют извлечь одиночный символ из строки, преобразовать строку в массив символов?

6) Приведите фрагмент кода, который создает массив символов на основе строки, удаляет из этого массива все пробелы, а затем создает новую строку на основе полученного массива символов.

7) Чем отличается операция == от операции, выполняемой методом equals, для строк? Объясните на примере.

8) Как сравнить две строки на равенство без учета регистра?

9) При помощи каких методов можно сравнить строки лексикографически (на больше, меньше, равно)?

10) Какие методы класса String позволяют определить номер позиции символа или подстроки в строке?

11) При помощи каких методов можно извлечь подстроку из строки? Чем отличаются эти методы?

12) Какой метод удаляет пробелы в начале и конце строки?

13) Какой метод позволяет осуществить замену символа на другой в строке?

14) Для чего предназначен метод `valueOf()` класса `String`. Какие его модификации вам известны? Почему все методы `valueOf` в классе `String` объявлены статическими?

15) Для чего предназначены методы `toLowerCase()` и `toUpperCase()`? Приведите пример использования.

16) Объясните, когда и почему следует избегать операции конкатенации для объектов класса `String`? Приведите примеры из разработанного проекта.

17) Какие классы позволяют создавать изменяемые строки символов?

18) Чем похожи и чем отличаются `Sentence`-объекты, описанные в разделе 2.4.3, и `String`-объекты. Изобразите схематично состояние оперативной памяти после выполнения следующего участка кода:

```
String str = "Hello!";
```

```
Sentence sent = new Sentence (str);
```

19) Какие методы в вашем проекте определяют поведение `Sentence`-объектов?

20) Какие преимущества получит проект, описанный в разделе 2.4.3, если методы класса `Sentence`, возвращающие строки, изменить так, чтобы они возвращали объекты класса `Sentence`?

21) Какие способы борьбы с ошибками, возникающими на некоторых наборах исходных данных вам известны?

22) Опишите структуру и механизм работы защищенного блока

23) Какая исключительная ситуация может возникнуть в вашей программе и почему? Каким способом вы ее предотвратили? Какой метод выбрасывает и обрабатывает исключение?

24) Объясните алгоритм работы методов «Разбить на слова» и «Собрать слова», используемых в вашем проекте.

25) Объясните алгоритмы работы созданных вами методов.

26) Как программно создать «растущий» массив в Java?

27) Измените код метода «Разбить на слова» так, чтобы можно было снять ограничение на число слов в предложении.

9. ЛАБОРАТОРНАЯ РАБОТА № 9. ВВОД-ВЫВОД. БАЙТОВЫЕ И СИМВОЛЬНЫЕ ПОТОКИ. ИСПОЛЬЗОВАНИЕ КЛАССОВ ПАКЕТА JAVA.IO. ОБРАБОТКА ИСКЛЮЧЕНИЙ В JAVA

3.1. Цель работы

Целью данной работы является:

- 1) ознакомление с основными принципами ввода-вывода данных и библиотечными классами Java, предназначенными для осуществления операций ввода-вывода;
- 2) получение навыков работы с байтовыми и символьными потоками Java;
- 3) исследование возможностей обработки исключительных ситуаций, которые могут возникнуть в процессе выполнения операций ввода-вывода Java-программой.

2.2. Постановка задачи

- 1) Разработать программу, осуществляющую вывод данных из массива байтов в байтовый поток, связанный с файлом `byte1_фамилия_группа.txt` и в стандартный поток `out`, определенный в библиотечном классе `System`.
- 2) Разработать программу, осуществляющую ввод данных из байтового потока, связанного с файлом `byte1_фамилия_группа.txt`, обработку вводимых байтов, предусмотренную вариантом задания, вывод обработанных байтов в байтовый поток, связанный с файлом `byte2_фамилия_группа.txt`.
- 3) Разработать программу, осуществляющую ввод данных из символьного потока связанного с файлом `char1_фамилия_группа.txt`, обработку вводимых данных, предусмотренную вариантом задания, вывод данных в символьный поток, связанный с файлом `char2_фамилия_группа.txt`.

3.3. Внеаудиторная подготовка

Для подготовки к лабораторной работе следует изучить краткие теоретические сведения, приведенные в пункте 3.4 методических указаний. Рекомендуется также ознакомиться с [1] (Глава 12). Дополнительные сведения по работе с файлами можно получить из [2–5].

3.4. Краткие теоретические сведения

3.4.1. Поточковый ввод-вывод

Операции ввода-вывода – это операции передачи данных от внешнего устройства программе (ввод) или из программы внешнему устройству (вывод). Устройство ввода-вывода является источником при вводе данных или приемником при выводе данных.

Простейшие средства ода-вывода определены в классе System. Широкий набор классов для работы с файлами содержится в пакетах java.io, java.nio.

Ввод-вывод в Java основан на понятии «поток». Понятие потока позволяет с различными устройствами работать единообразно, используя одни и те же действия.

Под потоком данных понимается последовательность байтов, поступающих друг за другом на устройство ввода вывода или принимаемых от него. Схемы алгоритмов чтения из потока (записи в поток) приведены на рисунке 3.1.

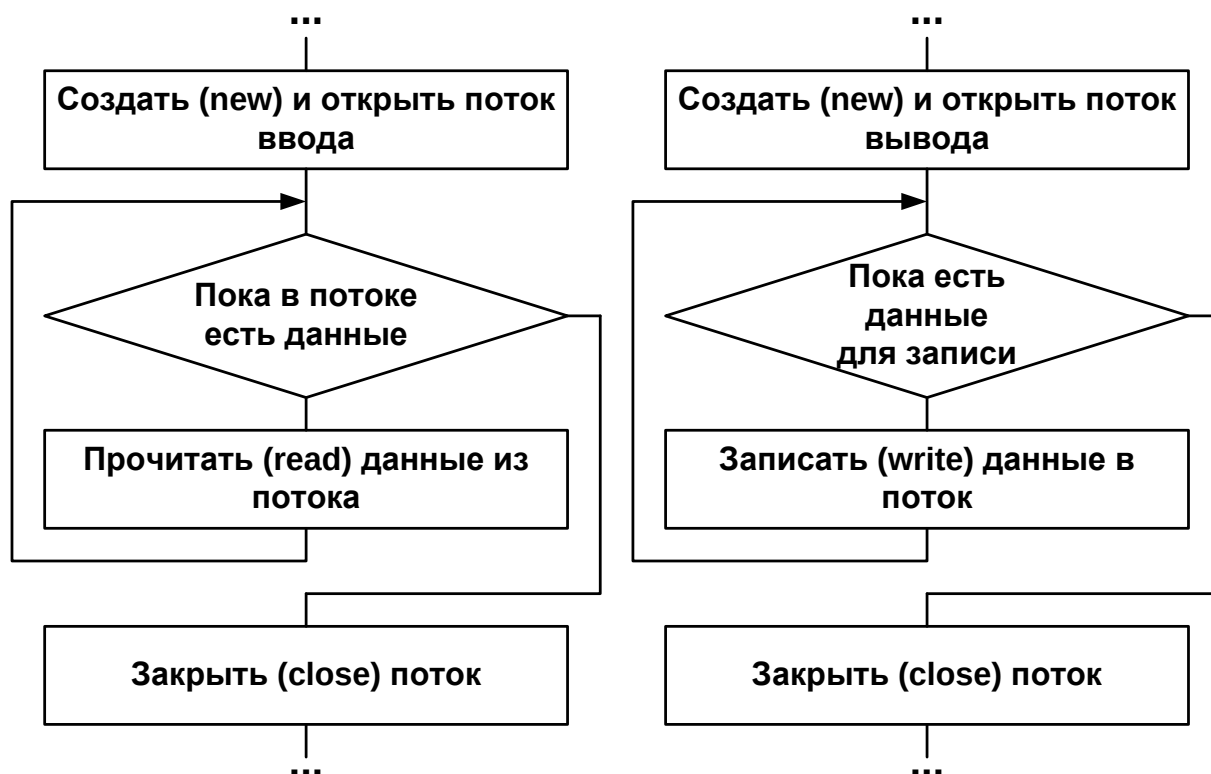


Рисунок 3.1 – Схемы алгоритма чтения из потока и записи в поток

Функциональность, связанная с перемещением данных по потоку, определена в абстрактных классах InputStream, OutputStream, Reader и Writer пакета java.io. Они используются для создания нескольких конкретных поточных подклассов.

Классы InputStream и OutputStream предназначены для байтовых потоков, а Reader и Writer для символьных. Байтовые и символьные поточные классы формируют отдельные иерархии.

Классы символьных потоков используем для работы с символами или строками, а классы байтовых потоков – при работе с байтами или другими двоичными объектами.

Иерархия потоков:

InputStream (абстрактный) → FileInputStream (поток байтов) →
BufferedInputStream (буфер+поток байтов)

Reader (абстрактный) → FileReader (поток символов) → BufferedReader
(буфер+поток символов)

Буфер – это служебная область памяти (обычный размер – 2048 байтов), выделенная потоку. Буфер повышает быстродействие операций ввода-вывода. Например, вместо ввода данных по одному байту, вводится объем, заполняющий буфер. Далее в программу передаются байты из буфера, пока он не исчерпается. После этого снова заполняется буфер и т.д.

Аналогичная иерархия имеет место и для выводных потоковых классов (в имени класса Input заменяется на Output, Reader на Writer).

Наиболее простым является поток байтов. Он представляет «сырые» данные. Вся работа и ответственность по интерпретации байтов – в программе. Работа с сырыми данными выполняется, например, при копировании файлов.

Классы байтовых потоков имеют имена вида ...Stream.

На основе потока байтов строится поток символов. В операционной системе конкретного компьютера может быть принят способ кодирования (например, Unicode), в котором один символ кодируется несколькими байтами. Поток символов преобразует цепочки байтов в символы и символы в цепочки байтов, согласно способу кодирования. Так, для ОС Windows типичным является использование кодовой таблицы 1251, в которой латинские и русские буквы представляются одиночными байтами. Виртуальная машина JVM использует Unicode, в котором русские символы кодируются парами байтов. Преобразование делается автоматически, если JVM выполняет ввод-вывод для символьных потоков.

Имена символьных потоков для ввода – это ...Reader, для вывода – ...Writer. В символьном потоке могут быть специальные символы – \t, \r, \n. В частности, комбинации \r или \n или \r\n могут означать переход к новой строке. В случае символьных потоков обнаружение окончания строки текста – обязанность программы.

Системные потоки

Системный входной поток (поток ввода) **in** типа InputStream объявлен в классе System как статический и используется для ввода с клавиатуры. Поток **in** доступен для чтения сразу после запуска программы. Выходные потоки (потоки вывода) **out** и **err** типа PrintStream объявлены в классе System как статические и используются для вывода на терминал, оба открыты и готовы для вывода данных. Обычно поток **in** используется как основа для вводного потока символов. Стандартный поток **out** – символьный поток, символы которого преобразуются в цепочки байтов, согласно системной таблице кодирования.

Содержит уже знакомые нам методы для вывода данных стандартных типов: `print`, `println`, `printf`, `format`.

Связь потоков с устройствами ввода-вывода

Ввод-вывод для жестких дисков поддерживается классами `FileInputStream`, `FileOutputStream` (байты), `FileReader`, `FileWriter` (символы). Каждый класс содержит конструктор, в котором указывается имя файла как `String`-литерал или `String`-переменная. Например,

```
FileInputStream fis=new FileInputStream(имя-файла); .
```

Аргумент **имя-файла** может задавать файл в текущем каталоге проекта: `"my-dat.by"`, или задавать абсолютный путь к файлу: `"c:\\javall\\projects\\lab1\\my-data.by"`.

Методы абстрактного класса – потока байтов `InputStream`:

`int available()` возвращает число байтов, доступных в настоящее время для чтения;

`void close()` закрывает поток и освобождает ресурсы, ранее полученные для потока;

`void mark(int readlimit)` отмечает текущую позицию входного потока специальным маркером, который можно будет использовать до тех пор, пока из потока не будет прочитано `readlimit` байтов;

`boolean markSupported()` проверяет, допустимы ли методы `mark` и `reset` (возвращает `true/false`);

`int read()` читает из потока очередной байт и возвращает его как правый байт `int` (последние 8 бит из 32 бит), или `-1`, если уже нет доступных байтов (встречается конец файла);

`int read(byte[] b)` читает порцию байтов, заполняя массив `b` (возвращает количество реально прочтенных байтов; если `b.length==0`, ничего не читает и возвращает `0`; если нет доступных байтов, возвращает `-1`);

`int read(byte[] b, int off, int len)` читает `len` байтов в массив `b` со смещением от начала массива `off` (возвращает количество реально прочтенных байтов; если `len==0`, ничего не читает и возвращает `0`; если нет доступных байтов, возвращает `-1`);

`void reset()` переустанавливает входной указатель на предварительно установленный маркер (перемещается по потоку в обратном направлении до маркера);

`void skip(long n)` пытается пропустить во входном потоке `n` байтов без чтения, возвращает число фактически проигнорированных байтов.

Все методы, кроме `mark` и `markSupported`, могут создавать исключение `IOException`.

Методы абстрактного класса – потока байтов `OutputStream`:

`void write(int b)` пишет правый байт из `int b` в поток, левые байты `int` не используются;

`void write(byte b[])` записывает в выходной поток массив байтов `b`;

void write(byte b[], int off, int len) записывает в поток len байтов из b, начиная с элемента b[off];

void flush() завершает операцию вывода, сохраняя данные из внутреннего буфера.

void close() закрывает выходной поток.

Все методы этого класса имеют тип void и создают исключение **IOException** в случае ошибки.

Случаи, при которых возникает исключение **IOException**:

- чтение после того, как данные в потоке закончились, или поток закрыт
- запись данных, для которых нет места на диске
- запись данных после того, как поток закрыт
- закрытие выводного потока, если нельзя записать оставшиеся в буфере данные.

При попытке связать поток с файлом, имя которого задано неверно, возникает исключение **FileNotFoundException**.

Борьба с исключениями

Каждое из действий (или группа действий), которые могут дать исключение, заключается в блок try. Тогда программист может обработать исключение в соответствующем блоке catch (т. е. метод, выбросивший исключение будет перехватывать и обрабатывать его самостоятельно).

Если метод может выбрасывать исключения, но сам их не обрабатывает (программист не предусмотрел соответствующих действий), типы выбрасываемых методом исключений нужно перечислить в заголовке этого метода с помощью ключевого слова throws для того, чтобы перехват исключения мог быть произведен вызывающим методом или исполняющей системой Java. Например, метод main, в котором обработку исключений мы поручаем системе, следует объявить так:

```
public static void main(String[] args) throws IOException{...}.
```

3.4.2. Примеры программ, осуществляющих потоковый ввод-вывод

Пример 1. Первая программа осуществляет вывод в файл byte1_Иванов_ИВТ11.txt, находящийся в папке проекта, массива байтов, значения которых соответствуют кодам символов '1', '2', 'z', 'я' в кодовой таблице win1251. Кроме того, каждый байт, интерпретированный программой как код символа в кодовой таблице unicode, выводится в окно терминала (в стандартный поток out, определенный в библиотечном классе System).

```

import java.io.IOException; //исключения ввода-вывода
import java.io.FileOutputStream; // байтовый поток вывода

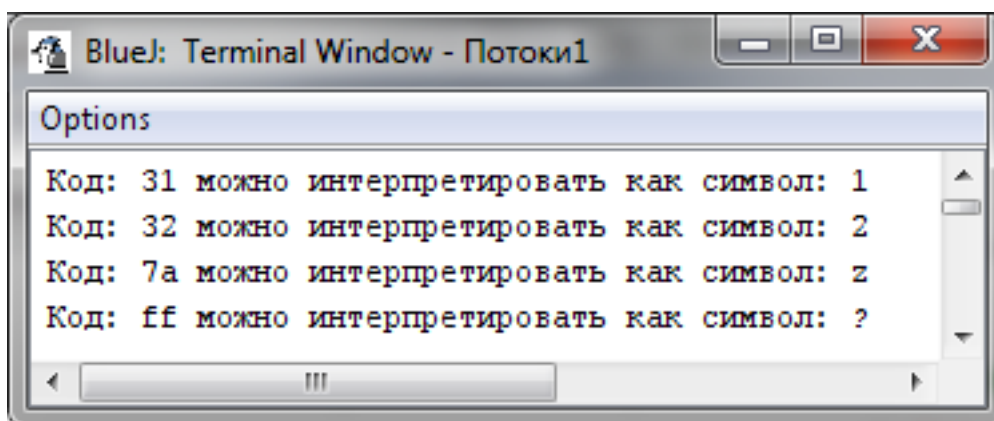
public class Main{
    public static void main(String[ ] args) throws IOException{
        byte[ ] b = { (byte)0x31, (byte)0x32, (byte)0x7a, (byte)0xff };
        FileOutputStream fo = null; //ссылка на выходной поток
        for(int i = 0; i < b.length; i++){
            System.out.printf("Код: %02x ", b[i]); //выводим в поток out
            System.out.println("Можно интерпретировать как символ: "+
                               (char)b[i]+" "); //выводим в поток System.out
        }
        try{
            // создаем выходной поток fo и связываем его с файлом
            fo = new FileOutputStream("byte1_Иванов_ИБТ11.txt");
            fo.write(b); //вывод массива байтов b в файл
        }
        catch (IOException e){System.out.println("error:\n"+e);} //ошибка
                                                //вывода
        finally{if(fo != null)fo.close();} // закрыть поток, если он открыт
    }
}

```

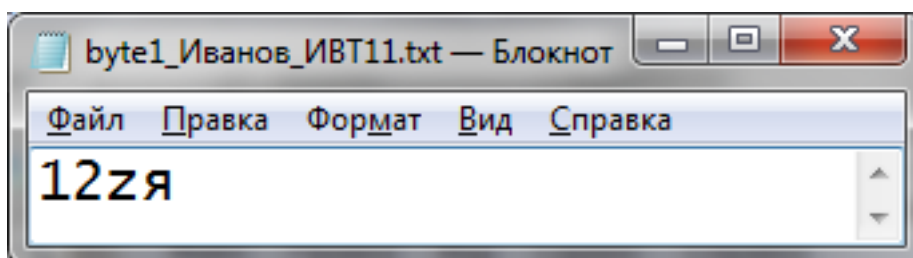
Результаты работы программы представлены на рисунке 3.2.

При выводе в поток out видно, что код 0xFF не опознается Java как код буквы 'я', т.к. Java использует кодовую таблицу unicode, где код символа занимает 16 битов, в отличие от 8 битов в кодовой таблице win1251.

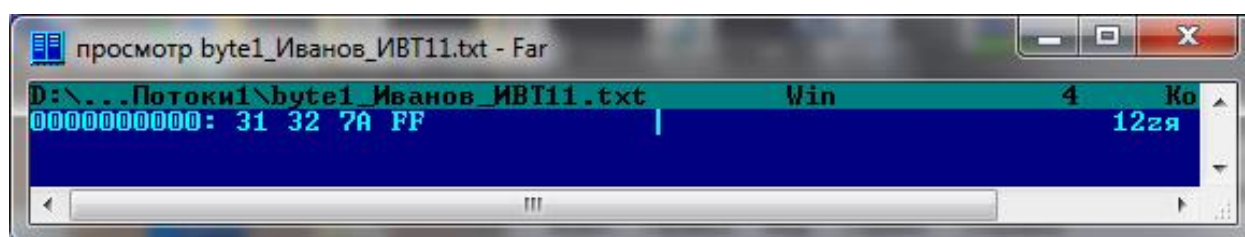
Цифры и символы английского алфавита интерпретируются верно, т.к. их коды меньше 256 (старший байт кода равен нулю). Напомним, что стандартный набор символов, известный как ASCII (включает буквы английского алфавита, цифры, символы пунктуации, специальные символы) располагается в интервале значений кодов от 0 до 127.



а) Вывод, осуществленный программой в стандартный поток out



б) Просмотр выходного файла в приложении «Блокнот»



в) Просмотр выходного файла в приложении Far

Рисунок 3.2 – Результаты работы программы для примера 1

Пример 2. Вторая программа последовательно вводит байты из входного потока, который связан с файлом, сформированным первой программой (файл **byte1_Иванов_ИВТ11.txt** скопирован в папку второго проекта). Строковый двоичный эквивалент каждого байта, полученный с помощью метода `byteToString`, выводится в стандартный поток `System.out` (окно терминала). Затем байт подвергается обработке, заданной методом `byteProcess`, и выводится в поток `System.out` (строковый двоичный эквивалент) и в выходной поток, связанный с файлом **byte2_Иванов_ИВТ11.txt**.

```

import java.io.IOException;
import java.io.FileInputStream;
import java.io.FileOutputStream;
public class Main{
    public static byte byteProcess(byte b){ //обработка байта
        //установка старшего бита в 1, если младший равен нулю
        int c = b; //почему удобно работать с int?
        if ((c & 1) == 0) c = c | 0x80;
        return (byte)c;
    }
    public static String byteToString(byte b){
        //возвращает строку символов - двоичный эквивалент байта
        String str = "";
        for (int i = 1; i <= 8; i++){
            str = b < 0 ? str + "1" : str + "0";
            b = (byte)(b << 1);
        }
        return str;
    }
    public static void main(String[] args) throws IOException{
        byte b = 0; int d = 0;
        FileInputStream fi = null; //ссылка на поток ввода
        FileOutputStream fo = null; //ссылка на поток вывода
        try{
            // создаем потоки fi и fo и связываем их с файлами
            fi = new FileInputStream("byte1_Иванов_ИВТ11.txt");
            fo = new FileOutputStream("byte2_Иванов_ИВТ11.txt");
            // пока в потоке ввода есть данные
            while((d = fi.read()) != -1){ //ввести очередной байт из потока fi
                b = (byte)d; //выделить младший байт из четырех (d – int)
                System.out.print(byteToString(b) + " --> "); // байт до изменения
                b = byteProcess(b); // обработка байта
                System.out.println(byteToString(b)); // байт после изменения
                fo.write(b); //вывести байт в поток fo
            }
        }
        catch (IOException e){System.out.println("error:\n"+e);} //ошибка ввода-вывода
        finally{ if(fi != null) fi.close(); if(fo != null) fo.close(); } //закрыть потоки, если они открыты
    }
}

```

Схема алгоритма метода преобразования байта в его строковый двоичный эквивалент (String byteToString(byte b)), определенного в программе, приведена на рисунке 3.3. Объясните, какой принцип используется в алгоритме для проверки очередного бита в байте b.

Метод обработки байта (byte byteProcess(byte b)), студент должен запрограммировать самостоятельно в соответствии с вариантом задания.

Результаты работы программы, приведенной выше, представлены на рисунке 3.4.

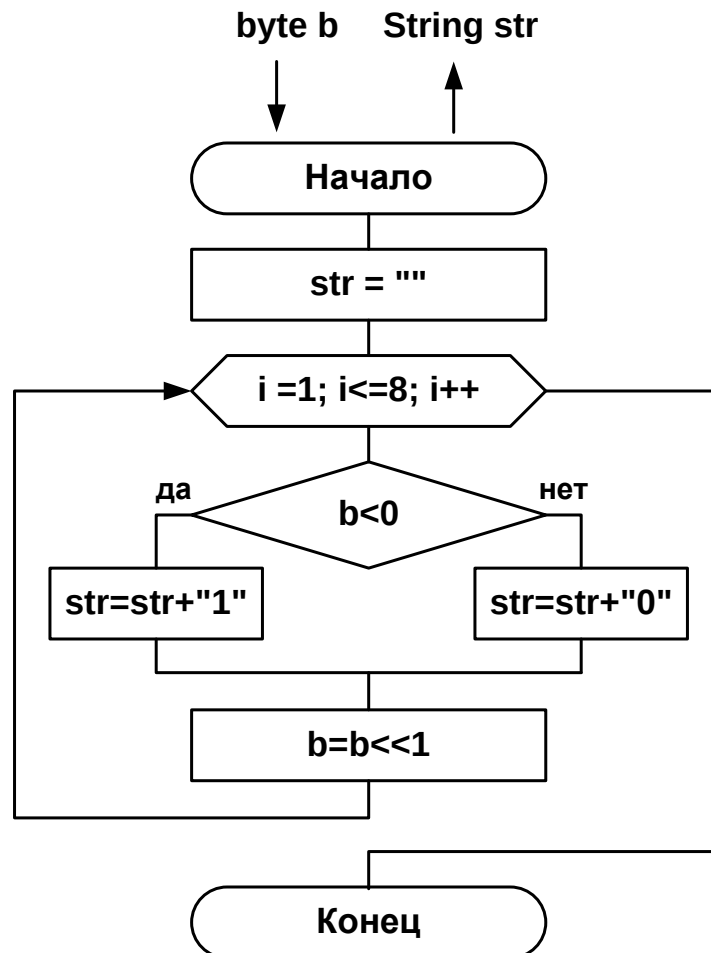


Рисунок 3.3 – Схема алгоритма получения строкового двоичного эквивалента байта b

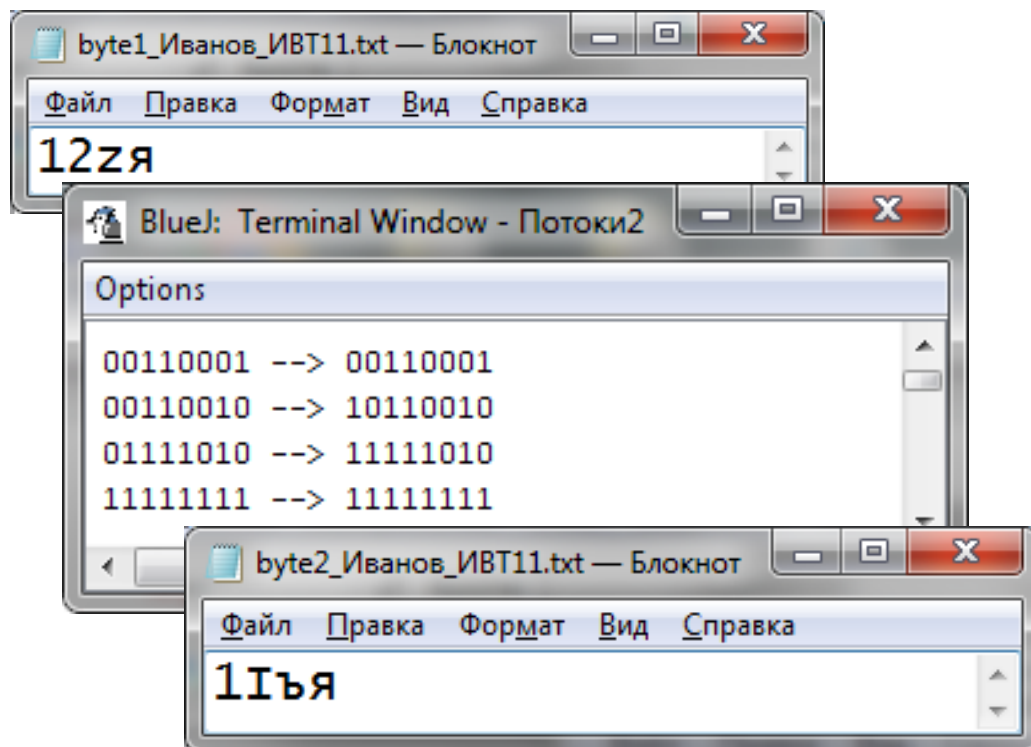


Рисунок 3.4 – Результаты работы программы для примера 2

Пример 3. Программа вводит данные из файла **char1_Иванов_ИБТ11.txt** посредством символьного потока ввода. Указанный файл предварительно создан в папке проекта с помощью текстового редактора «Блокнот». Каждый введенный из потока символ выводится для контроля в стандартный поток System.out (окно терминала). В поток вывода, связанный с файлом **char2_Иванов_ИБТ11.txt** выводятся только символы, являющиеся гласными буквами русского алфавита. Каждый вариант задания предусматривает некоторую подобную обработку входного потока символов перед направлением символов в выходной поток.

```
import java.io.IOException; //исключения ввода-вывода
import java.io.FileReader; //символьный поток ввода
import java.io.FileWriter; //символьный поток вывода
public class Main{
    public static void main(String[ ] args) throws IOException{
        char b = 0; int d = 0;
        FileReader fi = null; //ссылка на символьный поток ввода
        FileWriter fo = null; //ссылка на символьный поток вывода
        try{
            //создаем и связываем потоки с файлами
            fi = new FileReader("char1_Иванов_ИБТ11.txt");
            fo = new FileWriter("char2_Иванов_ИБТ11.txt");
            //пока есть данные в потоке ввода
            while((d = fi.read()) != -1){ //ввести код символа из fi (2 байта)
                b = (char)d; //преобразовать код в символ
                System.out.print(b); //вывести символ в System.out
                //преобразование b согласно варианту
                //вывод в файл только гласных букв русского алфавита
                if ("aeёиоуыэюяАЕЁИОУЫЭЮЯ".indexOf(b) >= 0) fo.write(b);
            } //while
        }
        catch (IOException e){System.out.println("\nerror:\n"+e);}
        finally{if(fi != null) fi.close(); if(fo != null) fo.close();}
    }
}
```

Результаты работы приведенной выше программы отображены на рисунке 3.5. Заметим, что символьные потоки позволяют корректно работать с символами русского алфавита в Java.

Создадим исключительную ситуацию при открытии потока ввода. Для этого удалим (временно) из папки проекта файл **char1_Иванов_ИБТ11.txt**. Результаты работы программы в этом случае приведены на рисунке 3.6.

Создадим исключительную ситуацию при выводе данных. Вернем в папку проекта файл **char1_Иванов_ИБТ11.txt**., а в текст программы после цикла **while** вставим строку **fo.close (); fo.write(b); :**

```

...
if ("aeёиоуыэюяАЕЁИОУЫЭЮЯ".indexOf(b)>=0)fo.write(b);
    } //while
    fo.close (); fo.write(b); //ошибка!
}
catch (IOException e){System.out.println("\nerror:\n"+e);}
...

```

Результаты работы программы в этом случае приведены на рисунке 3.7.

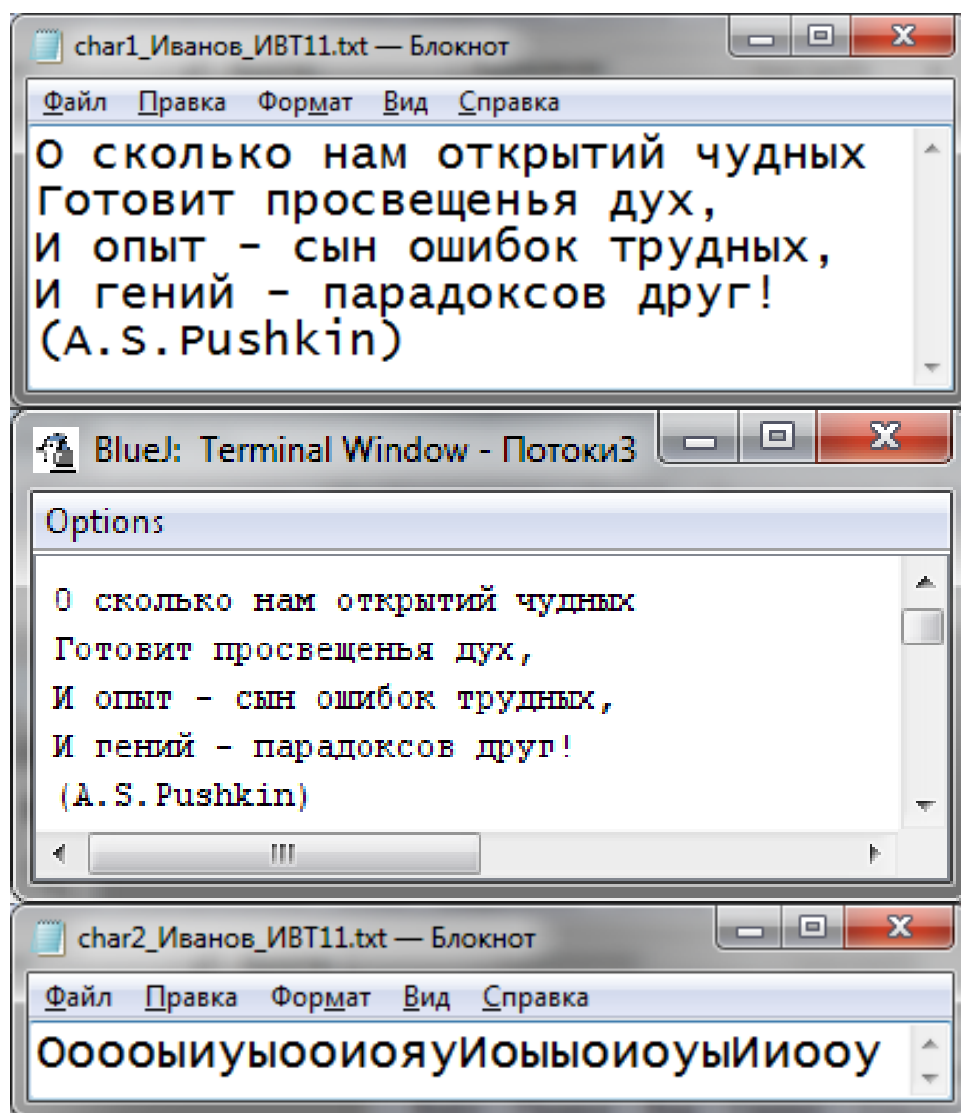


Рисунок 3.5 – Результаты работы программы для примера 3

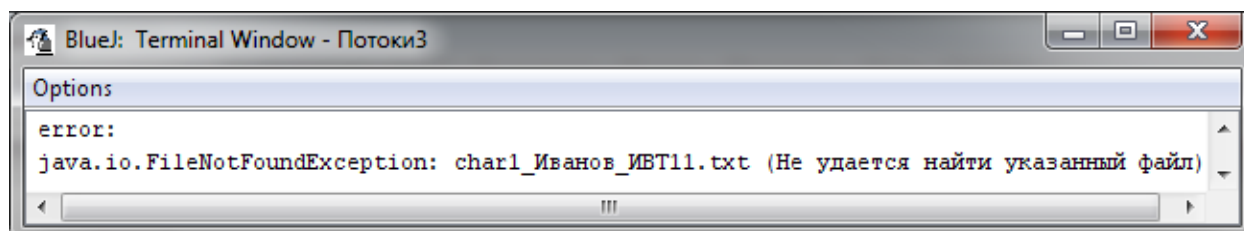


Рисунок 3.6 – Захват программой исключения, возникшего при открытии потока ввода

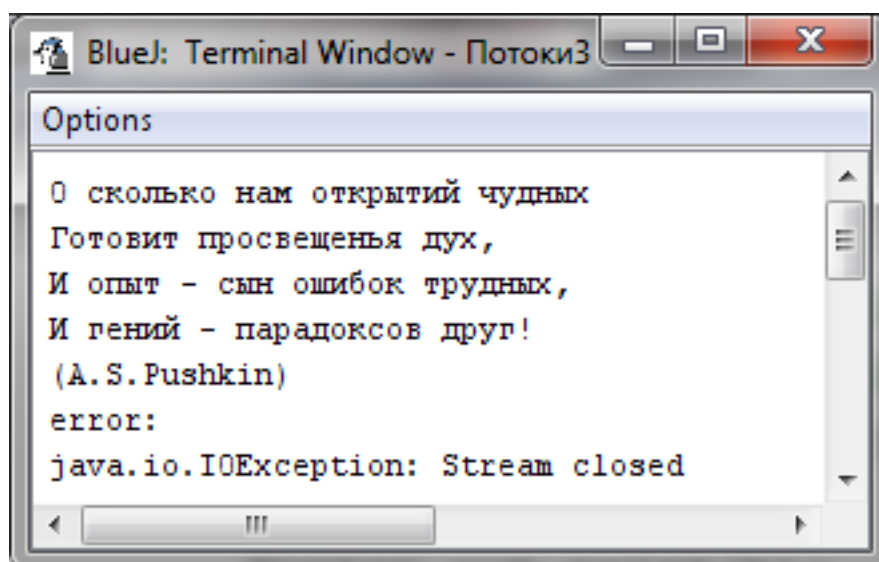


Рисунок 3.7 – Захват программой исключения, возникшего при попытке вывода в закрытый поток

3.5. Порядок выполнения работы

1) Внимательно изучите тексты программ, приведенные выше. Эти программы вы можете взять за основу.

2) Разработайте программу, осуществляющую вывод данных из массива байтов в байтовый поток, связанный с файлом `byte1_фамилия_группа.txt` и в стандартный поток `System.out`. Вывести не менее 5 байтов, являющихся кодами символов английского алфавита, и 5 байтов, являющихся кодами символов русского алфавита в кодировке `win1251`.

3) Согласно варианту задания (таблица 3.1), разработайте программу, осуществляющую следующие действия:

- ввод данных из байтового потока, связанного с созданным первой программой файлом `byte1_фамилия_группа.txt`;
- обработку вводимых байтов, предусмотренную вариантом задания;
- вывод обработанных байтов в байтовый поток, связанный с файлом `byte2_фамилия_группа.txt`, а их строковых двоичных эквивалентов (до и после обработки) – в поток `System.out`.

4) Согласно варианту задания (таблица 3.1), разработайте программу, осуществляющую следующие действия:

- ввод данных из символьного потока связанного с файлом `char1_фамилия_группа.txt`,
- вывод вводимых символов в поток `System.out` (для контроля);
- обработку вводимых данных, предусмотренную вариантом задания;
- вывод обработанных данных в символьный поток, связанный с файлом `char2_фамилия_группа.txt`.

5) Проведите отладку программ, добейтесь того, чтобы они выдавали ожидаемые результаты. Объясните, почему байтовые потоки Java не стоит применять для работы с символами.

6) Для третьей программы смоделируйте исключительные ситуации при вводе и выводе данных и проверьте, как ваша программа на них реагирует.

3.6. Варианты заданий

Вариант задания определяется таблицей 3.1.

Номер варианта задания V необходимо вычислить по формуле

$$\text{int } V = (N \leq 14) ? \text{variant} : N \% 14;$$

где N – номер студента в списке группы,

variant – номер варианта задания в таблице 3.1.

В качестве дополнительного задания организуйте в третьей программе подсчет и вывод в поток System.out числа вхождений любого выбранного вами символа (для исходного файла).

Таблица 3.1 – Варианты заданий

Номер варианта	Обработка вводимых байтов (для программы 2)	Обработка вводимых символов перед копированием в выходной файл (для программы 3)
1	Поменять местами левую правую половины байта.	Заменить каждый буквенный символ большой буквой.
2	Установить первый, третий, пятый биты байта в значение 0.	Заменить каждый пробел на символ '- '.
3	Поменять местами второй и третий биты байта.	Выбирать для копирования только латинские символы и цифры (коды латинских символов и цифр меньше 128).
4	Поменять местами второй и четвертый биты байта.	Выполнить кодирование текста: заменить каждый символ другим символом, код которого на 1 больше кода первоначального символа (использовать коды символов, не превышающие 0x7FFF).
5	Заменить все биты байта противоположными значениями.	После каждого символа, прочтенного из файла, вставить пробел.
6	Заменить все биты байта, кроме правого, нулями.	Выбирать для копирования только согласные буквы русского алфавита.

Продолжение таблицы 3.1

Номер варианта	Обработка вводимых байтов (для программы 2)	Обработка вводимых символов перед копированием в выходной файл (для программы 3)
7	Изменить значения первого, третьего, пятого битов на противоположные.	Выбирать для копирования только буквы и специальные символы (не буквы и не цифры).
8	Заменить левый бит нулем, правый – единицей, остальные не изменять.	Выбирать для копирования все символы, являющиеся буквами или цифрами.
9	Поменять местами пятый и шестой биты байта.	Выбирать для копирования только специальные символы (не являющиеся буквой или цифрой).
10	Изменить порядок правых трех битов на обратный.	Сделать все буквы латинского алфавита прописными.
11	Установить в 1 третий бит, если пятый бит равен 0.	Сделать все гласные буквы строчными.
12	Сбросить в 0 четвертый бит, если старший бит равен 1.	Заменить буквы латинского алфавита А, В, С, D, Е, F соответственно на сочетания символов 10, 11, 12, 13, 14, 15.
13	Сбросить в 0000 вторую тетраду, если первая равна 1111.	Заменить каждый символ 'Ё' на символ 'Е' и 'ё' на 'е'.
14	Установить в 1111 первую тетраду, если вторая равна 0000.	Вставить после каждого символа-цифры, кроме '0', символ '0'.

3.7. Содержание отчета

1) Цель работы.

Для каждого из трех заданий:

- 2) Постановка задачи с описанием варианта задания.
- 3) Текст разработанной программы.
- 4) Протокол отладки (описание найденных синтаксических и логических ошибок в программе и способов их устранения).
- 5) Результаты выполнения программы, в том числе, для случаев, когда в программе возникают исключения (исключения моделировать только в третьей программе).

6) Выводы.

3.8. Контрольные вопросы

- 1) В чем состоит и какие преимущества обеспечивает потоковая модель ввода-вывода Java?
- 2) Что понимается под потоком?
- 3) В каких абстрактных классах определены основные методы для работы с потоками? Какие это методы?
- 4) Сформулируйте алгоритмы чтения из потока и записи в поток.
- 5) Как объявить байтовые потоки для считывания и записи? Как их создать и связать с конкретными файлами?
- 6) Как обнаружить, что все данные из входного файла уже введены?
- 7) Чем байтовые потоки отличаются от символьных?
- 8) Объясните, какие проблемы могут возникнуть при попытках ввести символы русского алфавита при помощи байтового потока? Какие символы можно вводить с помощью байтового потока без проблем и почему?
- 9) Объясните разработанный вами алгоритм обработки байта (для программы 2).
- 10) Объясните, каким образом вы обрабатывали символы перед выводом их в выходной файл (для программы 3).
- 11) Как подсчитать число вхождений заданного символа во входном файле?
- 12) Какие исключения могут возникать при работе с потоками? Как использовать блоки `try`, `catch`, `finally` для работы с исключениями?

4. ЛАБОРАТОРНАЯ РАБОТА № 10. ВВОД-ВЫВОД СИМВОЛЬНЫХ СТРОК. ИСПОЛЬЗОВАНИЕ КЛАССОВ ПАКЕТА JAVA.IO. ОБРАБОТКА ИСКЛЮЧЕНИЙ В JAVA

4.1. Цель работы

Целью данной работы является:

- 1) ознакомление с библиотечными классами Java, предназначенными для осуществления операций ввода-вывода символьных строк;
- 2) получение навыков работы с буферизованными символьными потоками Java;
- 3) закрепление навыков работы с числовыми массивами с учетом их объектной природы в Java;
- 4) исследование возможностей обработки исключительных ситуаций, которые могут возникнуть в процессе выполнения операций ввода-вывода и преобразования введенной информации Java-программой.

4.2. Постановка задачи

- 1) Разработать программу, осуществляющую ввод матрицы из текстового файла, обработку матрицы заданным способом и вывод матрицы в другой текстовый файл.
- 2) В программе предусмотреть перехват и обработку исключений, которые могут возникнуть при вводе, выводе, преобразовании символьной информации в числовую.
- 3) Организовать в программе выброс собственного исключения при выполнении условия, заданного вариантом задания.

4.3. Внеаудиторная подготовка

Для подготовки к лабораторной работе следует изучить краткие теоретические сведения, приведенные в пункте 4.4 методических указаний. Рекомендуется также ознакомиться с [1] (с.302-312, с.249-262). Дополнительные сведения по работе с файлами можно получить из [2–5].

4.4. Краткие теоретические сведения

4.4.1. Ввод-вывод символьных строк. Буферизованные символьные потоки

Текстовые потоки в Java – это буферизованные потоки символов с именами `BufferedReader` и `PrintWriter`. Кроме обычных для символьных потоков операций ввода-вывода `read` и `write`, текстовые потоки имеют операции `readLine` (ввод строки) и `println` (вывод строки). Текстовые потоки используются для работы с

данными, которые состоят из символов. Такие данные получаются при использовании встроенных текстовых редакторов в системах разработки программ или файловых менеджеров. Такой текстовый файл можно получить также, если сохранить документ Microsoft Word, выбрав при сохранении тип файла «обычный текст».

В текстовых файлах можно хранить числовые данные, которые обычно готовятся вручную. При хранении таких данных рассматривают два варианта: «свободный формат», когда одно число отделено от другого цепочкой разделителей, а количество символов, представляющих число, выбирается при подготовке данных, и «жесткий формат», когда каждое число и каждая группа разделителей занимают заранее определенное число позиций. Пример задания жесткого формата: в каждой строке 10 чисел, каждое число занимает 5 позиций, из которых две правых – дробные, между числами – по два пробела, всего в строке $68=10*5+9*2$ символов.

Связь потоков с файлами

Объявление вводного потока input типа **BufferedReader**:

```
input = new BufferedReader(new FileReader("matrt.txt")); .
```

Конструктору класса **BufferedReader** требуется параметр – символьный файловый поток **FileReader**, который связывается с файлом обычным образом.

Объявление выводного потока output типа **PrintWriter**:

```
output= new PrintWriter("outputfile.txt");
```

Обработка строк текстового файла, не представляющих числовые данные, обычно состоит из следующих действий:

- ввод строки из файла при помощи метода `readLine` (метод возвращает объект типа `String`, содержащий введенную строку, или `null`, если строки в файле закончились);
- разделение строки на слова (методы класса `Tokenizer` или метод `split`, аргументом которого является регулярное выражение, соответствующее разделителям;
- обработка слов строки согласно поставленной задаче.

Обработка числовых данных в свободном формате отличается от обработки текстовых строк только тем, что после разделения строки на слова слово преобразуется в число подходящим методом. Методы находятся в классах-оболочках стандартных типов. Например, метод `parseDouble(word)` класса `Double` возвращает `double`-значение для строки `word`.

Обработка данных в жестком формате не требует разделения введенной строки на слова. Строка «нарезается» на цепочки байтов, длины которых

соответствуют формату данных. После этого каждая цепочка преобразуется в число, как и в предыдущем варианте.

3.4.2. Пример программы, осуществляющей ввод-вывод символьных строк

Программа выполняет ввод матрицы из текстового файла, обработку матрицы и вывод результата обработки в другой текстовый файл.

Текстовый файл, в котором находится матрица, имеет следующую структуру:

- в начале файла может быть любое число пустых строк или строк, состоящих из одних пробелов;
- в первой непустой строке файла записаны символьные эквиваленты значений числа строк и числа столбцов матрицы (первое число отделено от второго любым количеством пробельных символов);
- в последующих строках файла записаны символьные эквиваленты значений элементов соответствующих строк матрицы (отделяются друг от друга любым количеством пробельных символов);
- между непустыми строками файла, а также после последней непустой строки файла, может находиться любое число пустых строк или строк, состоящих из одних пробелов.

Если в файле задано больше строк или столбцов матрицы, чем это заявлено в первой непустой строке файла, то такая ситуация не считается ошибкой, а матрица при вводе должна быть «обрезана» до нужных размеров (игнорируются лишние строки и столбцы).

Примеры файлов, соответствующих вышеописанным правилам, приведены на рисунках 4.1-4.3.

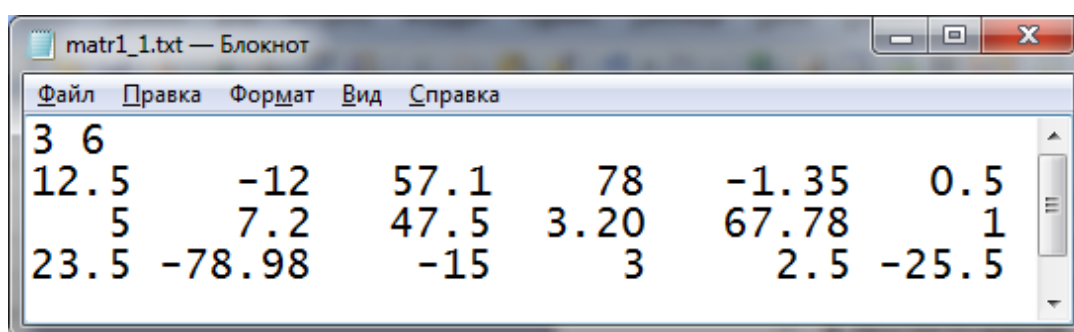


Рисунок 4.1 – Пример файла, в котором задана матрица 3x6 элементов (вещественных чисел)

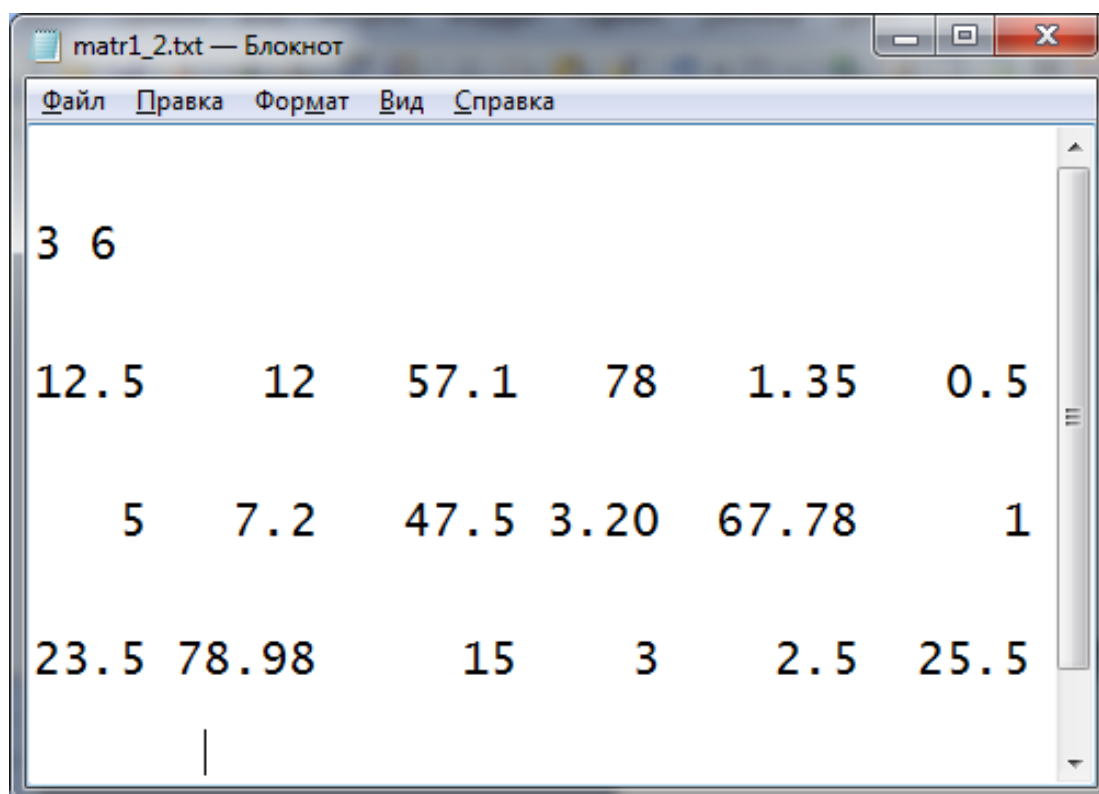
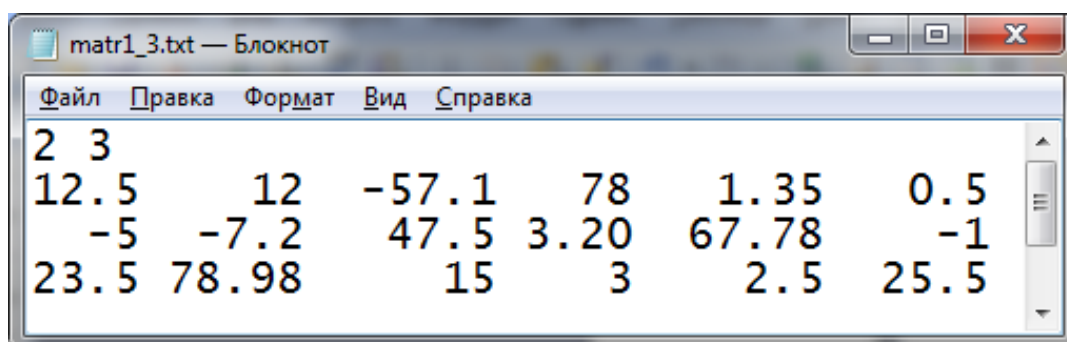


Рисунок 4.2 – Пример файла, в котором задана матрица 3х6 элементов (вещественных чисел), содержащего пустые строки и строки, состоящие из одних пробелов



Введена матрица:

```

12,50  12,00 -57,10
-5,00  -7,20  47,50
  
```

Рисунок 4.3 – Пример файла, в котором задана матрица 2х3 элемента (при вводе должны быть «обрезаны» лишние столбцы и строки)

Приведем список ошибок, которые могут возникать при вводе матрицы из файла.

- 1) файл не найден (файла с указанным именем нет в соответствующей папке) – исключение `FileNotFoundException` (библиотечный класс);
- 2) ошибка ввода данных – исключение `IOException` (библиотечный класс);
- 3) файл пуст (файл, используемый для ввода матрицы, пуст или состоит из одних пустых строк и пробелов) – рисунок 4.4;

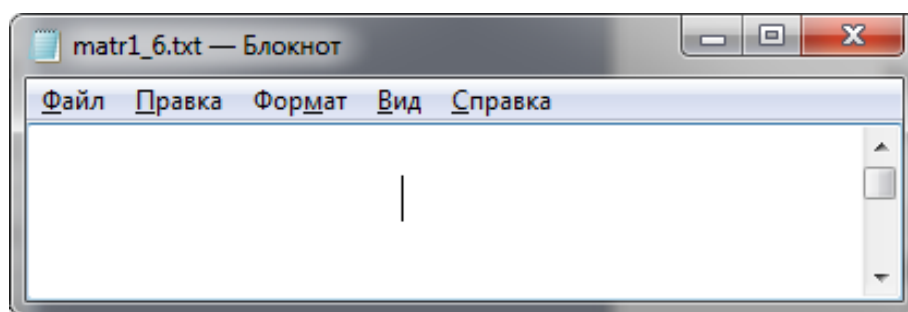


Рисунок 4.4 – Ошибка в структуре файла: файл пуст

4) не задано число столбцов матрицы (в первой непустой строке матрицы находится только одно число (слово) – рисунок 4.5.;

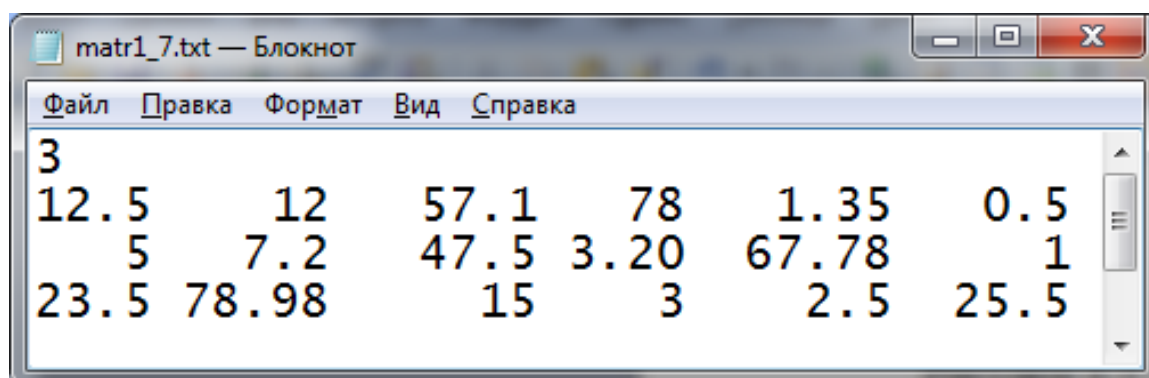


Рисунок 4.5 – Ошибка в структуре файла: не задано число столбцов матрицы

5) число строк и (или) столбцов матрицы меньше или равно нулю – рисунок 4.6;

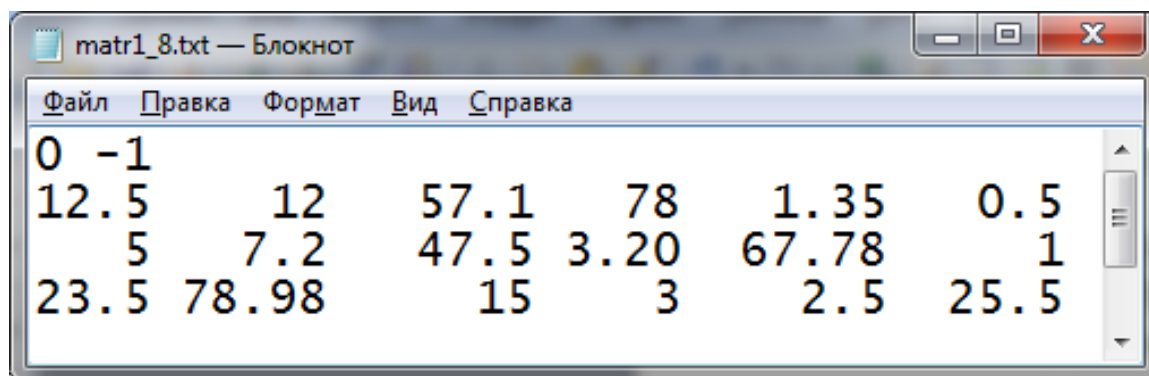


Рисунок 4.6 – Ошибка в данных файла: неправильно задано число строк и (или) столбцов матрицы

6) не хватает строк (число строк матрицы меньше заявленного в первой непустой строке файла) – рисунок 4.7;

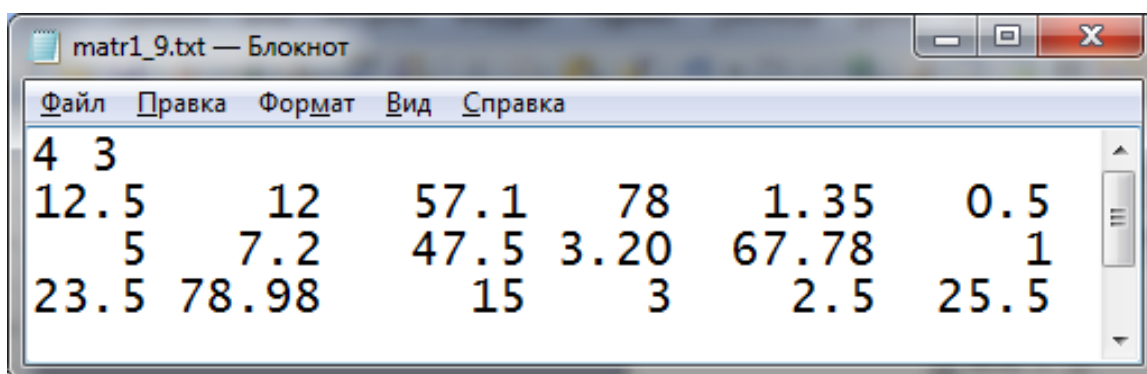


Рисунок 4.7 – Ошибка в структуре файла: не хватает строк матрицы (три вместо четырех заявленных)

7) в строке матрицы не хватает чисел – рисунок 4.8;

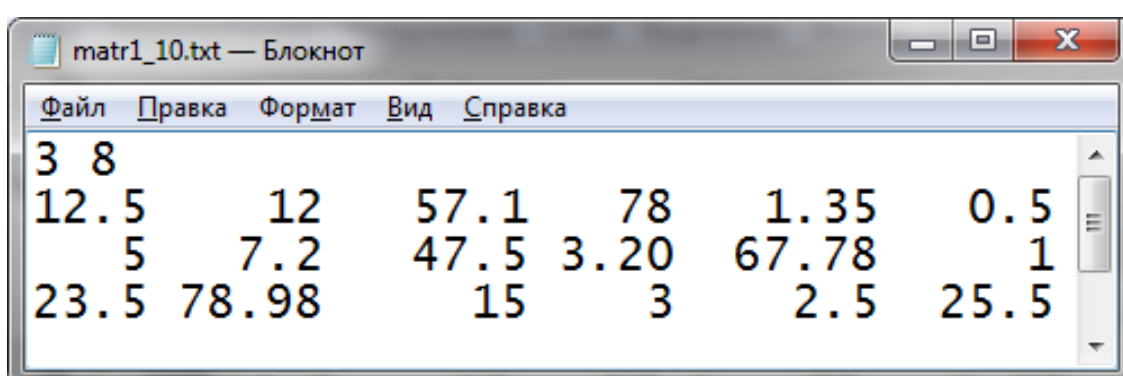


Рисунок 4.8 – Ошибка в структуре файла: не хватает чисел в первой же строке матрицы

8) недопустимая последовательность символов (невозможно преобразовать символьную подстроку в число) – исключение `NumberFormatException` (библиотечный класс) – рисунок 4.9.

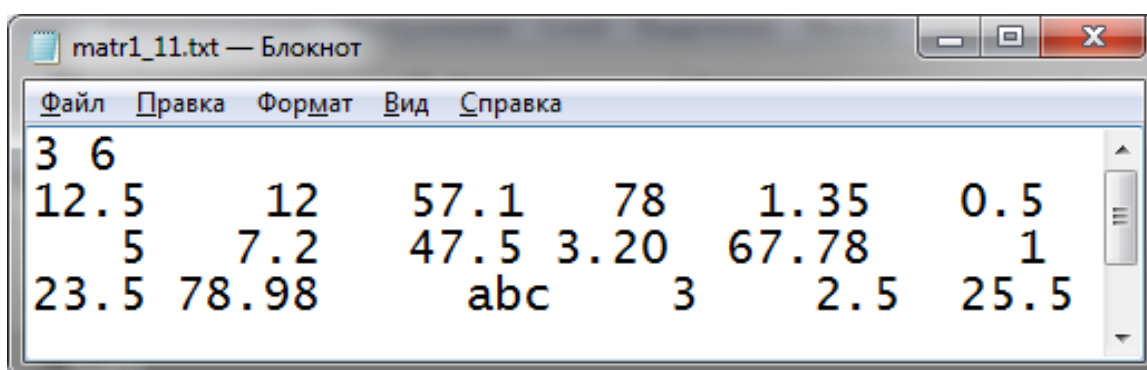


Рисунок 4.9 – Ошибка в данных файла: недопустимая последовательность символов в третьей строке матрицы

Отметим, что ошибки 3) – 7) появляются при выполнении программы в тех случаях, когда нарушена принятая **разработчиком** спецификация файла, из которого вводится матрица. Для обеспечения реакции программы на такие

ошибки разработчик может создать **собственный** класс исключений – MyException. Код класса MyException:

```
public class MyException extends Exception{
    private String msg; // переменная экземпляра: сообщение об ошибке
    public MyException (String s) { //конструктор
        msg = s;
    }
    //переопределим метод предка toString(),
    //возвращающий строку описания объекта-исключения
    public String toString() {
        return msg;
    }
}
```

После обработки введенной матрицы, программа должна вывести матрицу в файл. При выводе матрицы могут возникнуть описанные ниже исключительные ситуации.

1) Файл не найден (файла с указанным именем нет в соответствующей папке) – исключение FileNotFoundException (библиотечный класс). Если при выводе возникло такое исключение, но препятствий к созданию нового файла нет, то исключение игнорируется исполнительной системой, и новый файл создается. Поэтому нашей программе нет смысла перехватывать это исключение смысла.

2) Ошибка вывода данных – исключение IOException (библиотечный класс).

3) Попытка вывода в файл матрицы, представленной null-ссылкой. Для обеспечения реакции программы на эту ошибку разработчик может использовать класс MyException.

Структура проекта изображена на рисунке 4.10.

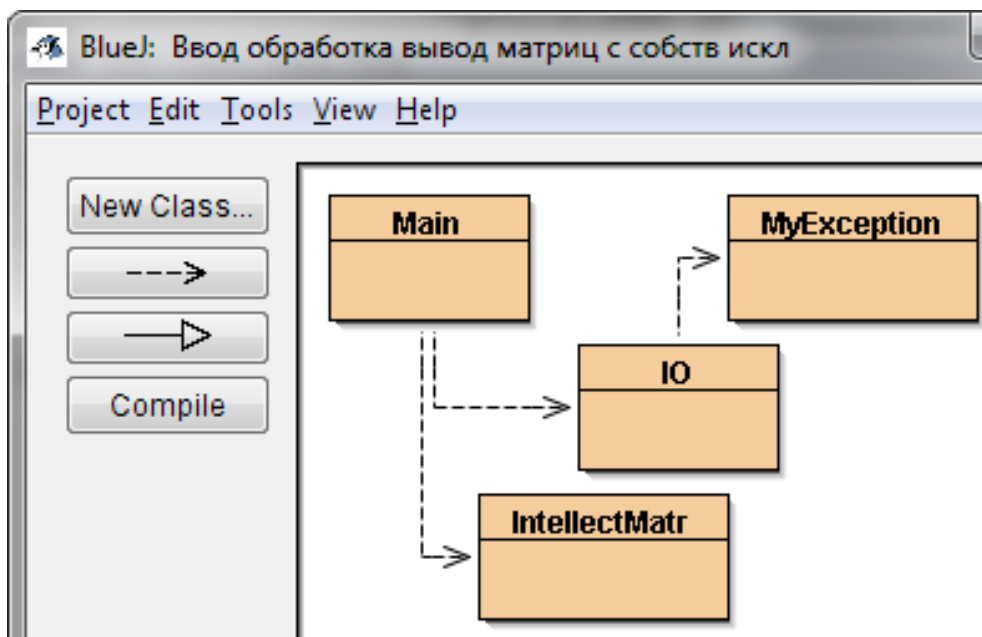


Рисунок 4.10 – Структура проекта

Класс IO содержит статические методы ввода и вывода информации (матрицы) в текстовый файл, а также статический метод вывода одномерного массива в окно терминала. Данный класс создан по принципу класса-библиотеки статических методов. Класс MyException (класс-шаблон для создания объектов), как было сказано выше, – это класс, описывающий пользовательское исключение. Класс IntellectMatr (интеллектуальная матрица – класс-шаблон для создания объектов) обеспечивает обработку матрицы. Класс Main содержит стартовую точку программы – метод main, который обеспечивает ввод с терминала имен входного и выходного текстовых файлов, ввод матрицы из входного файла, обработку матрицы и вывод матрицы в выходной файл.

Ниже приведен код классов IO, IntellectMatr и Main.

```
import java.io.*;
public class IO {
    // методы класса – статические (их можно использовать
    // без создания объектов этого класса)
    public static double[ ][ ] inpMatr (String fileName)
                                                throws IOException {
        // ввод матрицы из текстового файла с именем fileName
        double [ ][ ]matr; //матрица-приемник
        String line; //строка, считанная из файла
        int i = 0; //номер строки матрицы
        String[ ] numbers; //ссылка на массив слов, найденных в строке
        int n; //число строк матрицы
        int m; //число столбцов матрицы
        BufferedReader inp = null; // ссылка на входной поток
        try { //1 - контролируем исключения ввода-вывода
            //Открываем поток ввода
            //Могут возникнуть исключения FileNotFoundException,
            //IOException
            inp = new BufferedReader(new FileReader(fileName));
            //Далее - считываем матрицу – вводим строки из файла
            //При вводе может возникнуть исключение IOException
            //Пропускаем в начале файла пустые строки или
            //строки с одними пробелами
            while ((line = inp.readLine()) != null) if (!line.trim().equals("")) break;
            //Сначала нужно ввести число строк и число столбцов
            try { //2 - контролируем соответствие файла спецификации
                // (собственное исключение) и исключения, возникающие
                //при преобразовании из String в число
                if (line == null) // если считан конец файла
                    // генерируем исключение MyException
                    throw new MyException(String.format ("Файл %s пуст ", fileName));
                numbers = line.trim().split("\\s+"); //разбор строки на отдельные слова
                //(числа)
                if (numbers.length < 2) // Не задано число столбцов матрицы
                    // генерируем исключение MyException
                    throw new MyException(String.format(
                        "В файле %s не задано число столбцов матрицы", fileName));
                //может возникнуть исключение NumberFormatException:
            }
        }
    }
}
```

```

n = Integer.parseInt(numbers[0]); m = Integer.parseInt(numbers[1]);
if ((n <= 0) || (m <= 0)) //Неправильно задано число строк или
                           // столбцов матрицы

    // генерируем исключение MyException
    throw new MyException(String.format(
        "В файле %s неправильно задано число строк или столбцов матрицы",
        fileName));

    matr = new double[n][m]; //создаем матрицу-приемник
    // Считываем строки матрицы из файла
    // При вводе может возникнуть исключение IOException
    while ((line = inp.readLine()) != null) { // ввод очередной строки
        //если строка пустая, или состоит из одних пробелов
        // пропускаем ее
        line = line.trim();
        if (line.equals("")) continue;
        //разбор строки на отдельные слова (числа)
        numbers = line.split("\\s+");
        if (numbers.length < m) // в строке меньше m чисел, а
                               // должно быть m (или больше) чисел,
                               // (лишние игнорируются)
            // генерируем исключение MyException
            throw new MyException(String.format(
                "В строке %d матрицы из файла %s не хватает чисел", i, fileName));
            // далее может возникнуть исключение NumberFormatException:
            for(int j = 0; j < m; j++) matr[ i ][ j ] = Double.parseDouble(numbers[ j ]);
            i++; //увеличиваем номер введенной строки
            if (i == n) break; //введено указанное в файле число строк
            //если в файле еще есть строки – они игнорируются
        } //while
        if (i < n) //введено меньше строк, чем заявлено в первой
                  // непустой строке файла
            // генерируем исключение MyException
            throw new MyException(String.format(
                "В файле %s не хватает строк", fileName));
    } //try 2
    //Здесь порядок следования обработчиков catch роли
    // не играет, объясните, почему?
    catch (MyException e){
        // обработчик исключения для try 2 (неправильная структура файла)
        System.out.println (e); return null;}
    catch (NumberFormatException e){
        // обработчик исключения для try 2 (ошибка преобразования данных)
        System.out.printf ("В строке %d матрицы из файла %s", i, fileName);
        System.out.println (
            " обнаружена недопустимая последовательность символов");
        return null;}
    } //try 1
    //порядок следования блоков catch для try 1 определяется тем, что
    // класс FileNotFoundException является подклассом IOException

    catch (FileNotFoundException e){

```

```

//Файл не найден - обработчик для try 1
    System.out.printf ("Файл %s не найден\n", fileName);
    return null;}

catch (IOException e){
// Ошибка ввода данных - обработчик для try 1
    System.out.printf ("Ошибка при вводе данных из файла %s\n",
        fileName);

    return null;}
finally { if (inp != null) inp.close(); //закрываем поток
// При закрытии может возникнуть исключение IOException,
// которое в этой части программы не перехватывается и не
// обрабатывается, поэтому в заголовке метода inpMatr
// должно присутствовать предложение throws IOException
} // finally
return matr;
}
public static boolean outMatr (String fileName, double[ ][ ]matr)
    throws IOException {
// вывод матрицы matr в текстовый файл с именем fileName
    PrintWriter out = null; // ссылка на выходной поток
    try { //контролируем ошибки вывода и
        //входные данные метода (матрица не пуста)
        if (matr == null) throw new MyException("Матрица пуста");
        // открываем поток вывода
        out = new PrintWriter(new FileWriter(fileName));
        //выводим число строк и число столбцов матрицы
        out.println(matr.length + " " + matr[0].length);
        // выводим матрицу:
        for (int i = 0; i < matr.length; i++){
            // формируем i-ую строку для вывода
            //Создаем объект класса StringBuilder, который позволяет
            //работать с изменяемой строкой:
            StringBuilder line = new StringBuilder();
            for (int j = 0; j < matr[0].length; j++){
                line.append(" ");
                line.append(matr[ i ][ j ]);
            } // удаляем образовавшийся в начале строки пробел:
            line.delete(0, 1);
            //выводим строку в поток
            out.println(line.toString());
        }
    } //try
    catch (IOException e){
// Ошибка вывода данных - обработчик для try
        System.out.printf ("Ошибка при выводе данных в файл %s\n",
            fileName);

        return false;}
    catch (MyException e){
// обработчик исключения ( null-ссылка – матрица пуста)
        System.out.println (e); return false;}
    finally { if (out != null) out.close(); }
}

```

```

    return true;
}

public static void putArr (int X[ ]){
    //вывод одномерного массива X в окно терминала
    for (int i=0; i<X.length; i++)
        System.out.printf ("% 5d",X[i]);
    System.out.println();
} //putArr
} // class IO

public class IntellectMatr{
    //переменная экземпляра:
    private double [ ][ ] matr; //ссылка на прямоугольную матрицу
    //конструктор
    public IntellectMatr(double[ ][ ] matr){
        this.matr = matr;
    }
    //getter
    public double [ ][ ] getMatr() {return matr;}

    public int[ ] NegInRow (){
        //возвращает вектор, i-ый элемент которого равен
        //количеству отрицательных элементов в i-ой строке
        //матрицы
        int X[ ] = new int[matr.length]; // создает массив X,
        //состоящий из matr.length элементов типа double
        for (int i = 0; i < matr.length; i++){
            X[ i ] = 0; //начальное значение счетчика отр. элементов
            for (int j = 0; j < matr[0].length; j++)
                if (matr[ i ][ j ] < 0) X[ i ]++; //инкрементируем счетчик
        } //for i
        return X; // возвращаем ссылку на массив X
    } //NegInRow

    public boolean delCol(int n){
        //удаление из матрицы столбца с номером n
        if (matr == null) return false; //ссылка пуста
        if (matr[0] == null) return false; //нет столбцов
        if ((n < 0) || (n > matr[0].length-1)) return false; //неверный номер столбца
        if (matr[0].length-1 == 0) {matr = null; return true;} //не останется столбцов
        //создаем вспомогательную матрицу с уменьшенным на 1
        // (по сравнению с matr) числом столбцов
        double[ ][ ] X=new double [matr.length][matr[0].length-1];
        //для каждой строки переписываем в матрицу X из
        // матрицы matr элементы всех столбцов, кроме n-ного
        for (int i = 0; i < matr.length; i++){
            int k = 0; //номер столбца в массиве X
            for (int j = 0; j < matr[0].length; j++)
                if (j != n) {X[ i ][ k ] = matr[ i ][ j ]; k++;}
        }
        matr = X; //переприсваиваем ссылку
    }
}

```

```

        return true;
    } //delCol
    public boolean delRow (int n){
//удаление из матрицы строки с номером n
        if (matr == null) return false; //ссылка пуста
        if (matr[0] == null) return false; //нет столбцов
        if ((n < 0) || (n > matr.length-1)) return false; //неверный номер строки
        if (matr.length-1 == 0) {matr = null; return true;} //не останется строк
//создаем вспомогательную матрицу с уменьшенным на 1
// (по сравнению с matr) числом строк
        double [ ][ ] X = new double[matr.length-1][matr[0].length];
        int k = 0; //номер строки в массиве X
//все строки кроме n-ной переписываем из matr в X
        for (int i = 0; i < matr.length; i++)
            if(i != n) {X[k] = matr[ i ]; k++;} //работаем со ссылками
//на строки матрицы

        matr = X; //переприсваиваем ссылку
        return true;
    }

```

```

public void putMatr () { // вывод матрицы в окно терминала
    if ((matr == null) || (matr[0] == null)) return; //матрица пуста
    for (int i = 0; i < matr.length; i++) {
        for (int j = 0; j < matr[0].length; j++)
            System.out.printf ("% 7.2f", matr[ i ][ j ]);
        System.out.println();
    } //for i
} //putMatr
} //class IntellectMatr

```

```

import java.io.*;
public class Main {
    public static void main (String[ ] args) throws IOException {
        String fileName; //имя файла
        String line=""; //приемник строки
        String flag=""; //для выхода из цикла ввода с терминала
        double [ ][ ] matr; //ссылка на матрицу-приемник
//Определим поток для терминального ввода
        BufferedReader br = new BufferedReader (
            new InputStreamReader(System.in) );
        do {
            System.out.println("Введите имя файла для ввода матрицы");
            fileName = (br.readLine()).trim(); // после ввода имени файла в
// окно терминала читаем его и обрезаем пробелы в начале

```

```

// и в конце
matr = IO.inpMatr(fileName); //ввод матрицы из файла
if (matr != null){
    System.out.println ("Матрица введена успешно");
    // создаем объект класса Интеллектуальная матрица на базе
    // введенной матрицы
    IntellectMatr imatr = new IntellectMatr(matr);
    //выводим исходную матрицу в окно терминала
    System.out.println("Введена матрица:");
    imatr.putMatr();
    //обрабатываем матрицу методами класса
    // Интеллектуальная матрица и выводим промежуточные
    //результаты обработки в окно терминала
    System.out.println(
        "Количество отрицательных элементов в строках матрицы:");
    IO.putArr(imatr.NegInRow());
    System.out.println("Матрица после удаления второго столбца:");
    imatr.delCol(2);
    imatr.putMatr();
    System.out.println("Матрица после удаления второй строки:");
    imatr.delRow(2);
    imatr.putMatr();
    //Выводим результирующую матрицу в файл
    System.out.println("Введите имя файла для вывода матрицы");
    fileName = (br.readLine()).trim();
    if (IO.outMatr(fileName, imatr.getMatr()))
        System.out.println ("Успешный вывод");
    }
    System.out.println("Продолжить?: Да - <Enter>, Нет - <n>");
    flag = br.readLine().trim();
} while(!flag.equals("n")); // пока с терминала не введен символ n
}
//ввод с терминала буферизованный, т.е. происходит принажатии Enter
} //class Main

```

Результаты работы программы для входного файла, представленного на рисунке 4.1, приведены на рисунке 4.11.

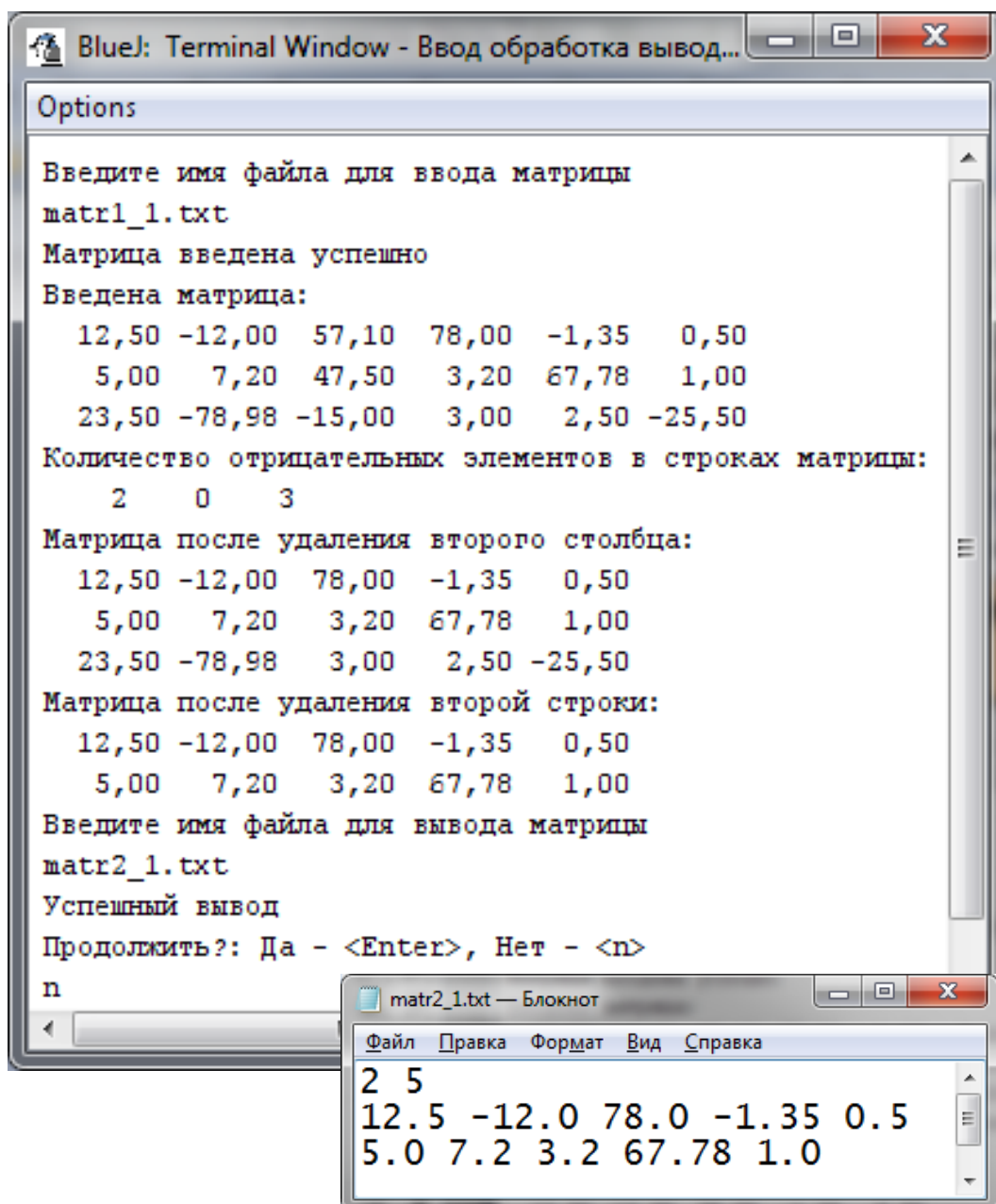


Рисунок 4.11 – Результаты работы программы для входного файла, представленного на рисунке 4.1

Реакцию программы на отсутствие входного файла, а также на ошибки спецификации входного файла, приведенные на рисунках 4.4-4.9 можно наблюдать на рисунке 4.12.

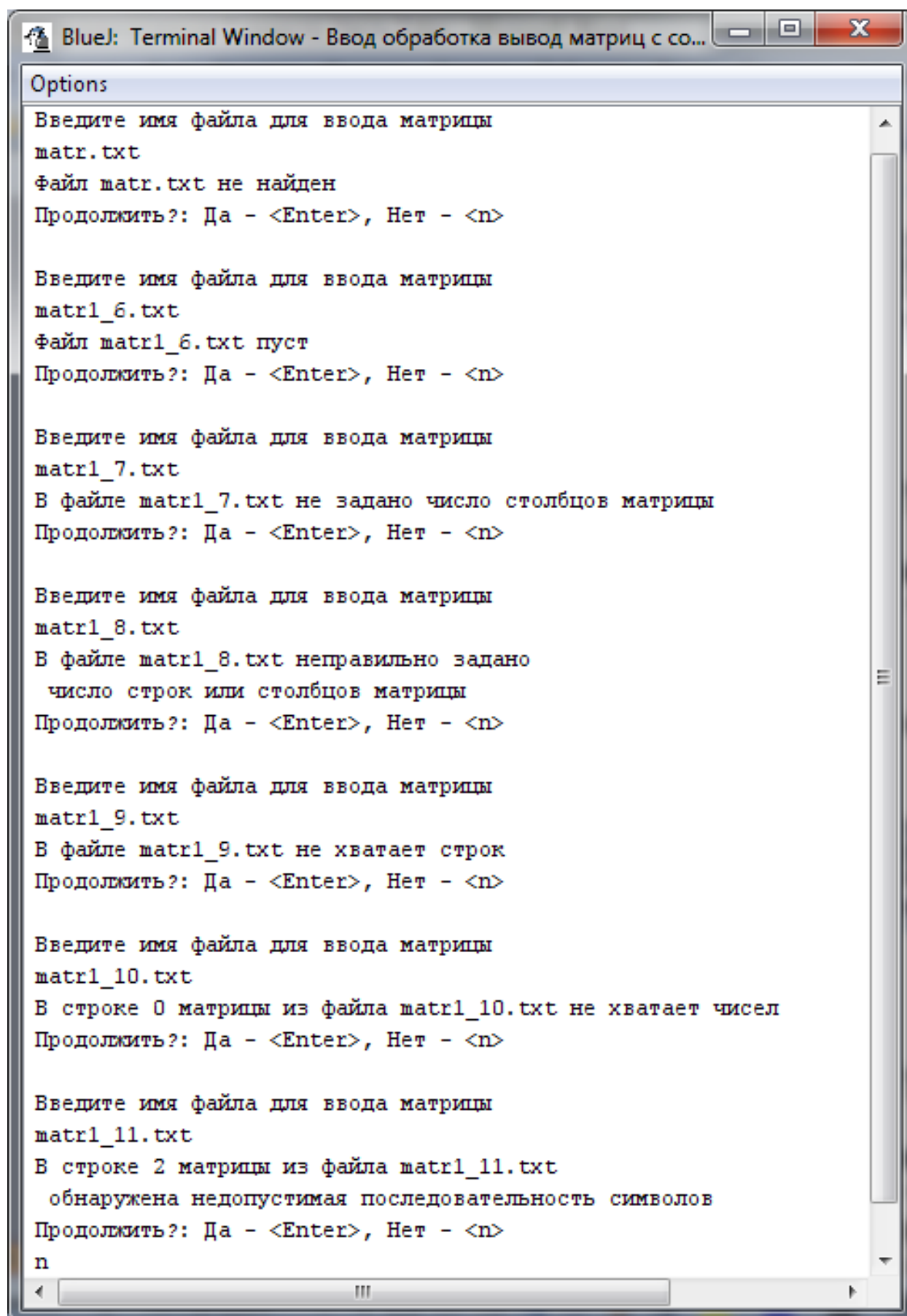


Рисунок 4.12 – Реакция программы на отсутствие входного файла, а также на ошибки спецификации входного файла, приведенные на рисунках 4.4–4.9

На рисунке 4.13 можно наблюдать реакцию программы на попытку вывода в файл пустой матрицы.

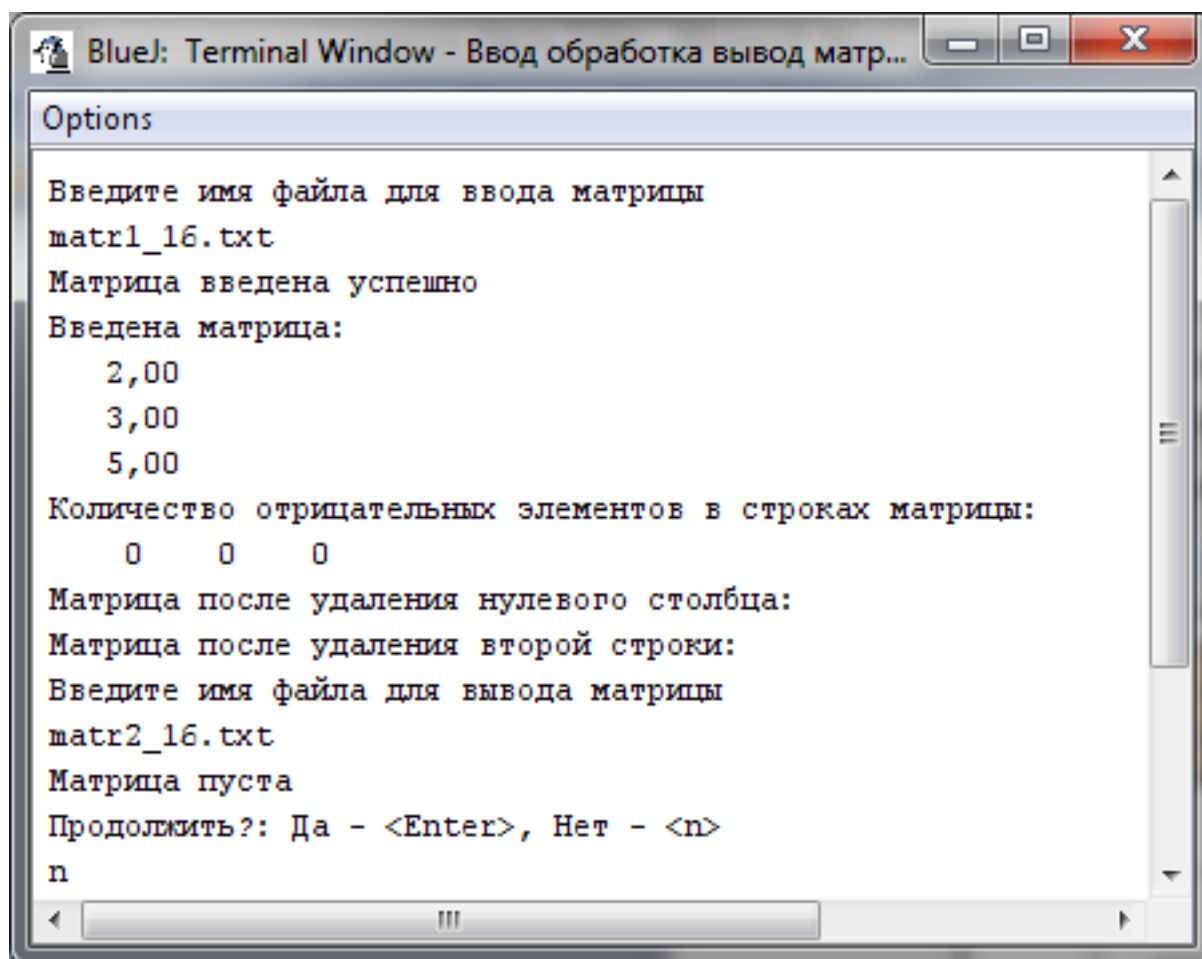


Рисунок 4.13 – Реакция программы на попытку вывода в файл пустой матрицы

Для того, чтобы обрабатывать описанные выше ошибки в методе **inpMatr** класса **IO** предусмотрено два защищенных блока **try** (один вложен в другой). Внешний блок **try(1)** контролирует исключения **FileNotFoundException** и **IOException**. Соответствующие обработчики должны быть выстроены именно в таком порядке, т.к. класс **IOException** является суперклассом (родительским классом) для класса **FileNotFoundException**. Вложенный блок **try(2)** контролирует исключения **MyException** (нарушение спецификации входного файла) и **NumberFormatException** (порядок следования соответствующих обработчиков не важен). В принципе, можно было обойтись и одним блоком **try** с четырьмя обработчиками.

В методе **outMatr** класса **IO** используется один защищенный блок **try** с двумя обработчиками **IOException** и **MyException**.

Заметим, что если постановка задачи не требует детализации сообщений об ошибках, код проекта можно упростить следующим образом: исключить класс **MyException**, в классе **IO** убрать обработчики исключений класса **MyException** и

вместо генерации этих исключений просто возвращать **null** в методе **inpMatr** и **false** в методе **outMatr**.

Код обработчиков **NumberFormatException** и **IOException** метода **inpMatr** в этом упрощенном варианте программы будет состоять из одного оператора **return null**;. Обработчик **FileNotFoundException** из метода **inpMatr** можно удалить, тогда это исключение будет обработано обработчиком родительского класса **IOException**. Код обработчика **IOException** метода **outMatr** будет состоять из одного оператора **return false**;. Возврат **null** методом **inpMatr** или **false** методом **outMatr** можно трактовать как ошибку.

Обратите внимание на то, что в заголовках методов **inpMatr** и **outMatr** присутствует фраза **throws IOException**, а все остальные исключения, которые могут возникать в этих методах, в заголовке методов не указаны. С помощью ключевого слова **throws** в заголовке метода перечисляются только те исключения, которые могут возникнуть в методе, но этим методом не перехватываются и не обрабатываются. Это делается для того, чтобы вызывающий метод «знал» о возможных необработанных в вызываемом методе исключениях и «предусмотрел» реакцию на них. Единственным исключением, которое не обрабатывается внутри методов **inpMatr** и **outMatr** является исключение **IOException**, которое может возникнуть в блоке **finally** при закрытии потоков, поэтому оно и указывается в заголовке данных методов.

4.5. Порядок выполнения работы

1) Внимательно изучите текст программы, приведенный выше. Методы и классы, определенные в этой программе, вы можете использовать в своей программе.

2) Разработайте программу, осуществляющую ввод матрицы из текстового файла, обработку матрицы обязательными методами, определенными вариантом задания, вывод матрицы в текстовый файл. В методе **main** используйте возможности терминального ввода для задания имен входных и выходных файлов, а также обеспечения возможности ввода, обработки, вывода любого числа матриц (опрос в цикле).

3) Проведите отладку программ, добейтесь того, чтобы они выдавали ожидаемые результаты.

4) Заготовьте несколько файлов с матрицами для демонстрации работы программы в ситуациях, изображенных на рисунках 4.1-4.9. Объясните, из каких файлов матрицы будут введены успешно, а в каких случаях возникнут исключения и какие.

5) Продемонстрируйте работу обязательных методов обработки матриц, определенных вариантом задания, на нескольких матрицах.

6)* Для студентов, претендующих на отличную оценку. Добавьте в программу дополнительные методы обработки матриц, предусмотренные вариантом задания, и продемонстрируйте их работу (используйте, где это целесообразно, методы удаления строки и столбца, приведенные в примере п. 3.4.2);

Дополнительные задания (выполняются по желанию студента):

7) Добавьте в программу метод ввода матрицы из файла, имеющего следующую спецификацию:

Первое слово в файле – число строк матрицы,
второе слово в файле – число столбцов матрицы,
каждое очередное слово – элемент матрицы.

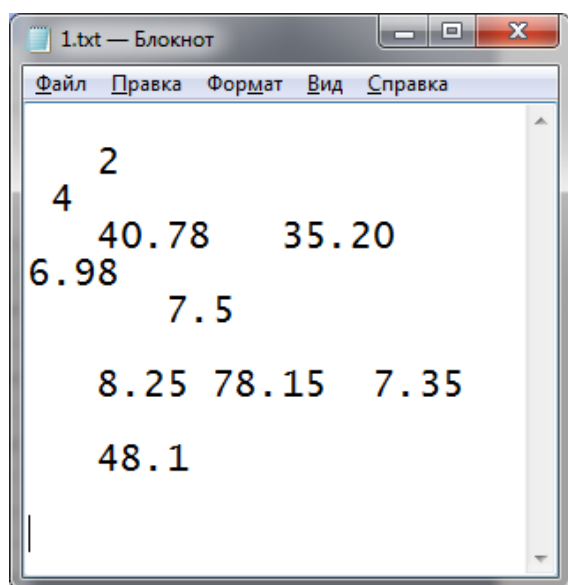
Элементы перечислены в следующем порядке: элементы первой строки, элементы второй строки и т.д.

Число слов (чисел) в строке файла может быть любым.

Слова разделены любым количеством пробельных символов.

До первого слова и до последнего слова файла может быть любое количество пробельных символов.

Исключения при нарушении спецификации файла: 1) файл пуст; 1) неправильно задано число строк и (или) столбцов (в файле меньше двух чисел или первые два числа отрицательные или равны нулю); 2) не хватает данных (количество слов не соответствует заявленному числу элементов матрицы); 3) ошибка преобразования данных (слово не преобразуется в число). Пример правильно организованного файла приведен на рисунке 4.14.



**В файле задана
следующая матрица:**

40.78	35.20	6.98	7.5
8.25	78.15	7.35	48.1

Рисунок 4.14 – Файл матрицы, соответствующий описанной в пункте 4.5 спецификации

Продемонстрируйте работу метода.

8) Напишите программу, обеспечивающую (в цикле):

- ввод очередной строки из текстового файла;
- обработку этой строки методами класса **Sentence** (Предложение), возвращающими значение String, разработанными вами в лабораторной работе №1 из данных методических указаний;
- вывод обработанной строки в другой текстовый файл.

9) Напишите программу подсчета и вывода в поток System.out числа вхождений любого выбранного вами символа (задается в качестве параметра) для введенного текстового файла.

4.6. Варианты заданий

Вариант задания определяется таблицей 4.1.

Номер варианта задания V необходимо вычислить по формуле

$$\text{int } V = (N \leq 14) ? \text{variant} : N \% 14;$$

где N – номер студента в списке группы,

variant – номер варианта задания в таблице 4.1.

Таблица 4.1 – Варианты заданий

Номер варианта	Обязательные члены класса Интеллектуальная матрица	Дополнительные* методы класса Интеллектуальная матрица (для студентов, претендующих на оценку «отлично»)
1	2	3
1	Поля: 1) ссылка на прямоугольную матрицу, тип элементов матрицы – float . Методы: 1) конструктор; 2) процедура вывода матрицы в окно терминала (без параметров); 3) функция, возвращающая вектор, i -ый элемент которого равен количеству нулевых элементов в i -ой строке матрицы (без параметров); 4) процедура, заменяющая все отрицательные элементы матрицы на нули (без параметров).	1) Вернуть номер первой из строк матрицы, в которой будут найдены отрицательные элементы (без параметров); 2) Удалить строки матрицы, содержащие отрицательные элементы (без параметров).
2	Поля: 1) ссылка на прямоугольную матрицу, тип элементов матрицы – double . Методы: 1) конструктор; 2) процедура вывода матрицы в окно терминала (без параметров); 3) функция, возвращающая вектор, i -ый элемент которого равен сумме положительных элементов i -ой строки матрицы (без параметров). 4) процедура, вычитающая A из всех положительных элементов матрицы (A – параметр).	1) Вернуть номер первого из столбцов, матрицы, в котором будут найдены отрицательные элементы (без параметров); 2) Удалить столбцы матрицы, содержащие отрицательные элементы (без параметров).

1	2	3
3	Поля: 1) ссылка на прямоугольную матрицу, тип элементов матрицы – int. Методы: 1) конструктор; 2) процедура вывода матрицы в окно терминала(без параметров); 3) функция, возвращающая вектор, <i>i</i>-ый элемент которого равен произведению положительных элементов <i>i</i>-го столбца матрицы (без параметров); 4) процедура, заменяющая на 0 все элементы матрицы, модуль которых меньше <i>A</i> (<i>A</i> – параметр).	1) Вернуть номер первой из строк, матрицы, сумма элементов в которой больше порогового значения <i>A</i> (<i>A</i> – параметр); 2) Удалить строки матрицы, сумма элементов в которых больше порогового значения <i>A</i> (<i>A</i> – параметр).
4	Поля: 1) ссылка на прямоугольную матрицу, тип элементов матрицы – int. Методы: 1) конструктор; 2) процедура вывода матрицы в окно терминала(без параметров); 3) функция, возвращающая вектор, <i>i</i>-ый элемент которого равен числу положительных элементов в <i>i</i>-ом столбце матрицы (без параметров); 4) процедура, заменяющая значения всех отрицательных элементов матрицы на значение минимального элемента матрицы (без параметров).	1) Вернуть номер первого из столбцов матрицы, сумма элементов в котором больше порогового значения <i>A</i> (<i>A</i> – параметр); 2) Удалить столбцы матрицы, сумма элементов в которых больше порогового значения <i>A</i> (<i>A</i> – параметр).
5	Поля: 1) ссылка на прямоугольную матрицу, тип элементов матрицы – int. Методы: 1) конструктор; 2) процедура вывода матрицы в окно терминала(без параметров); 3) функция, возвращающая вектор, <i>i</i>-ый элемент которого равен нулю, если в <i>i</i>-ом столбце матрицы есть положительные элементы, и единице, в противном случае; 4) процедура, заменяющая все отрицательные элементы матрицы на -1, а положительные – на 1 (без параметров).	1) Вернуть номер первого из столбцов матрицы, содержащего более <i>n</i> нулевых элементов (<i>n</i> – параметр); 2) Удалить столбцы матрицы, содержащие более <i>n</i> нулевых элементов (<i>n</i> – параметр).

1	2	3
6	Поля: 1) ссылка на прямоугольную матрицу, тип элементов матрицы – float. Методы: 1) конструктор; 2) процедура вывода матрицы в окно терминала (без параметров); 3) функция, возвращающая вектор, i-ый элемент которого равен количеству нулевых элементов в i-ой строке матрицы (без параметров); 4) процедура, заменяющая все отрицательные элементы матрицы на нули (без параметров).	1) Вернуть номер первой из строк матрицы, в которой будут найдены отрицательные элементы (без параметров); 2) Удалить строки матрицы, содержащие отрицательные элементы (без параметров).
7	Поля: 1) ссылка на прямоугольную матрицу, тип элементов матрицы – double. Методы: 1) конструктор; 2) процедура вывода матрицы в окно терминала (без параметров); 3) функция, возвращающая вектор, i-ый элемент которого равен сумме положительных элементов i-ой строки матрицы (без параметров). 4) процедура, вычитающая A из всех положительных элементов матрицы (A – параметр).	1) Вернуть номер первого из столбцов, матрицы, в котором будут найдены отрицательные элементы (без параметров); 2) Удалить столбцы матрицы, содержащие отрицательные элементы (без параметров).
8	Поля: 1) ссылка на прямоугольную матрицу, тип элементов матрицы – int. Методы: 1) конструктор; 2) процедура вывода матрицы в окно терминала (без параметров); 3) функция, возвращающая вектор, i-ый элемент которого равен произведению положительных элементов i-го столбца матрицы (без параметров); 4) процедура, заменяющая на 0 все элементы матрицы, модуль которых меньше A (A – параметр).	1) Вернуть номер первой из строк, матрицы, сумма элементов в которой больше порогового значения A (A – параметр); 2) Удалить строки матрицы, сумма элементов в которых больше порогового значения A (A – параметр).

1	2	3
9	Поля: 1) ссылка на прямоугольную матрицу, тип элементов матрицы – int. Методы: 1) конструктор; 2) процедура вывода матрицы в окно терминала(без параметров); 3) функция, возвращающая вектор, <i>i</i>-ый элемент которого равен числу положительных элементов в <i>i</i>-ом столбце матрицы (без параметров); 4) процедура, заменяющая значения всех отрицательных элементов матрицы на значение минимального элемента матрицы (без параметров).	1) Вернуть номер первого из столбцов матрицы, сумма элементов в котором больше порогового значения <i>A</i> (<i>A</i> - параметр); 2) Удалить столбцы матрицы, сумма элементов в которых больше порогового значения <i>A</i> (<i>A</i> - параметр).
10	Поля: 1) ссылка на прямоугольную матрицу, тип элементов матрицы – int. Методы: 1) конструктор; 2) процедура вывода матрицы в окно терминала(без параметров); 3) функция, возвращающая вектор, <i>i</i>-ый элемент которого равен нулю, если в <i>i</i>-ом столбце матрицы есть положительные элементы, и единице, в противном случае; 4) процедура, заменяющая все отрицательные элементы матрицы на -1, а положительные – на 1 (без параметров).	1) Вернуть номер первого из столбцов матрицы, содержащего более <i>n</i> нулевых элементов (<i>n</i> - параметр); 2) Удалить столбцы матрицы, содержащие более <i>n</i> нулевых элементов (<i>n</i> - параметр).
11	Поля: 1) ссылка на прямоугольную матрицу, тип элементов матрицы – float. Методы: 1) конструктор; 2) процедура вывода матрицы в окно терминала (без параметров); 3) функция, возвращающая вектор, <i>i</i>-ый элемент которого равен количеству нулевых элементов в <i>i</i>-ой строке матрицы (без параметров); 4) процедура, заменяющая все отрицательные элементы матрицы на нули (без параметров).	1) Вернуть номер первой из строк матрицы, в которой будут найдены отрицательные элементы (без параметров); 2) Удалить строки матрицы, содержащие отрицательные элементы (без параметров).

1	2	3
12	Поля: 1) ссылка на прямоугольную матрицу, тип элементов матрицы – double. Методы: 1) конструктор; 2) процедура вывода матрицы в окно терминала (без параметров); 3) функция, возвращающая вектор, i-ый элемент которого равен сумме положительных элементов i-ой строки матрицы (без параметров). 4) процедура, вычитающая A из всех положительных элементов матрицы (A – параметр).	1) Вернуть номер первого из столбцов, матрицы, в котором будут найдены отрицательные элементы (без параметров); 2) Удалить столбцы матрицы, содержащие отрицательные элементы (без параметров).
13	Поля: 1) ссылка на прямоугольную матрицу, тип элементов матрицы – int. Методы: 1) конструктор; 2) процедура вывода матрицы в окно терминала(без параметров); 3) функция, возвращающая вектор, i-ый элемент которого равен произведению положительных элементов i-го столбца матрицы (без параметров); 4) процедура, заменяющая на 0 все элементы матрицы, модуль которых меньше A (A – параметр).	1) Вернуть номер первой из строк, матрицы, сумма элементов в которой больше порогового значения A (A - параметр); 2) Удалить строки матрицы, сумма элементов в которых больше порогового значения A (A - параметр).
14	Поля: 1) ссылка на прямоугольную матрицу, тип элементов матрицы – int. Методы: 1) конструктор; 2) процедура вывода матрицы в окно терминала(без параметров); 3) функция, возвращающая вектор, i-ый элемент которого равен числу положительных элементов в i-ом столбце матрицы (без параметров); 4) процедура, заменяющая значения всех отрицательных элементов матрицы на значение минимального элемента матрицы (без параметров).	1) Вернуть номер первого из столбцов матрицы, сумма элементов в котором больше порогового значения A (A - параметр); 2) Удалить столбцы матрицы, сумма элементов в которых больше порогового значения A (A - параметр).

4.7. Содержание отчета

- 1) Цель работы.
- 2) Постановка задачи с описанием варианта задания.
- 3) Текст программы для основного задания (раздел 4.5, пункты 1 – 6).
- 4) Протокол отладки (описание найденных синтаксических и логических ошибок в программе и способов их устранения).
- 5) Результаты выполнения программы, в том числе, для случаев, когда в программе возникают исключения.
- 6) Необязательный раздел (выполняется по желанию студента). Постановки задач, тексты программ, протоколы отладки и результаты выполнения для дополнительных заданий (в случае их выполнения) – раздел 4.5, пункты 7 – 9.
- 7) Выводы.

4.8. Контрольные вопросы

- 1) Что такое буферизованный символьный поток?
- 2) Какие потоковые классы используются для ввода и вывода строк в текстовые файлы?
- 3) Как открыть буферизованный символьный поток и связать его с конкретным файлом?
- 4) Какие методы используются для ввода строки из текстового файла и вывода строки в текстовый файл?
- 5) Как определить, что строка в файле больше нет? Приведите пример из вашей программы.
- 6) Какие исключения могут возникать при исполнении операций ввода-вывода?
- 7) Почему в методе `IO.inpMatr`, осуществляющем ввод строк из файла, обработчик `catch` исключения `FileNotFoundException` расположен перед обработчиком `catch` исключения `IOException`? Что будет, если поменять их местами?
- 8) Как реагирует исполнительная система Java на появление исключения `FileNotFoundException` при открытии потока вывода?
- 9) Как организовать в программе ввод строк с терминала? Какие классы и как при этом надо использовать? Какой стандартный поток при этом используется? В каком классе он определен? Какие еще предопределенные потоки вам известны?
- 10) Как осуществить вывод строки на терминал? Какие известные вам методы можно использовать для этого? В каком классе они определены? Какой стандартный предопределенный поток используют эти методы?
- 11) Какой метод вы использовали в программе для разбиения считанной из файла строки на слова? Что является параметром метода. Какой тип возвращает метод?

- 12) Какое исключение и в каком случае может возникнуть в процессе преобразования слова (String) в число?
- 13) Для чего в программе вы использовали собственный класс исключений MyException. Какие ошибки вы контролируете при помощи этого класса? Покажите в программе, как и когда вы генерируете исключение класса MyException.
- 14) Можно ли обойтись без класса MyException в программе? Какие возможности при этом будут утрачены?
- 15) Чем отличается структура с внешним и вложенным блоками try, использованная в методе IO.inpMatr() программы-примера, от возможной структуры с одним блоком try и несколькими обработчиками (множественный try). Нарисуйте обе схемы.
- 16) Почему в заголовке методов IO.inpMatr и IO.outMatr во фразе throws указан только класс IOException, а классы других исключений, которые могут выбрасываться указанными методами, в этой фразе не упоминаются?
- 17) Опишите использованный вами в программе формат файла для хранения матрицы.
- 18) Для чего в методе IO.outMatr() использовался объект класса StringBuilder. Почему для такого случая не рекомендуется использовать тип String и операцию конкатенации?
- 19) Как удалить из матрицы строку с заданным номером? Как удалить из матрицы столбец с заданным номером?
- 20) Как в Java сделать (программно) одномерный массив с растущей при необходимости длиной?
- 21) Опишите разработанный вами класс «Интеллектуальная матрица».
- 22) Опишите класс IO, который вы используете в своей программе.
- 23) Опишите класс MyException, который вы используете в своей программе.
- 24) Опишите класс Main, который вы используете в своей программе.

5. ЛАБОРАТОРНАЯ РАБОТА № 11. ИСПОЛЬЗОВАНИЕ КЛАССОВ ПАКЕТА JAVA.IO. РАБОТА С ФАЙЛАМИ С ПОМОЩЬЮ КЛАССА FILE. ВВОД-ВЫВОД БАЙТОВ СРЕДСТВАМИ КЛАССА RANDOMACCESSFILE

5.1. Цель работы

Целью данной работы является:

- 1) ознакомление с библиотечными классами Java, предназначенными для работы с файлами;
- 2) освоение возможностей получения информации о файлах и папках с помощью методов класса File;
- 3) освоение возможностей ввода-вывода в файлы прямого доступа с помощью методов класса RandomAccessFile.

5.2. Постановка задачи

- 1) Разработать программу, позволяющую осуществлять следующие действия:
 - вывод информации (методами класса File) о файле и папке, в которой находится этот файл;
 - вывод списка файлов, находящихся в папке;
 - ввод и вывод данных из файла средствами класса RandomAccessFile (информация файла должна восприниматься программой как «сырые» байты).
- 2) Разработать программу, позволяющую выполнять следующие действия:
 - выводить данные в байтовый файл и вводить данные из байтового файла, интерпретируя последовательности байтов, как элементы заданного простого типа,
 - устанавливать указатель файла на нужную позицию для считывания элементов и создания из них матриц;
 - выполнять заданную операцию над матрицами и выводить элементы результирующей матрицы, начиная с заданной позиции файла.

5.3. Внеаудиторная подготовка

Для подготовки к лабораторной работе следует изучить краткие теоретические сведения, приведенные в пункте 5.4 методических указаний. Рекомендуется также ознакомиться с [1] (Глава 17. Класс RandomAccessFile. С.524). Полезно ознакомиться с содержанием всей главы 17, посвященной пакету java.io. Дополнительные сведения можно получить в [5, 6].

5.4. Краткие теоретические сведения

5.4.1. Библиотечный класс File

Класс File, определенный в пакете java.io, имеет дело непосредственно с файлами и файловой системой. Он не указывает, как нужно извлекать и сохранять информацию в файлах; в нем описываются свойства самого файла: права доступа, время и дата создания и изменения, путь в иерархии подкаталогов и т.п.

Для получения и управления информацией, связанной с дисковым файлом, используется объект класса File.

Каталог в Java трактуется как File-объект с одним дополнительным свойством – списком имен файлов, который можно просматривать с помощью метода list().

Для создания File-объектов можно использовать следующие конструкторы

File (String directoryPath) ;
File (String directoryPath, String filename) ;
File (File dirObj, String filename) .

Здесь directoryPath – путь к файлу (в дереве каталогов), filename – имя файла, dirObj – File-объект, который указывает каталог.

В Java принят Unix- и URL-стиль разделителей подкаталогов (прямой слэш – /). Java правильно воспринимает разделители подкаталогов, используемые в соглашениях по именованию файлов в операционных системах Unix и Windows/DOS. Если вы используете прямой слэш-символ(/), принятый в Unix, в Windows-версии Java, путь будет восприниматься правильно. Если вы используете соглашения Windows/DOS (обратный слэш-символ – \), то в строке пути нужно использовать escape-последовательность этого символа(\\).

В классе **File** определено много методов для получения стандартных свойств File-объекта, например:

getName() – возвращает имя файла, связанного с File-объектом;

getParent() – возвращает имя каталога предыдущего уровня;

exists() – возвращает true, если файл существует, и false, если нет;

isFile() – позволяет удостовериться, что файл будет вести себя как обычный файл, возвращает true, если запускающий File-объект связан с файлом, и false, если с каталогом или некоторому специальному файлу (типа драйвера устройства или именованного канала);

isAbsolute() – возвращает true, если запускающий File-объект был создан с указанием абсолютного пути к файлу, и false, если при создании был использован относительный путь;

lastModified() – возвращает Скоординированное Универсальное Время (UTC, Coordinated Universal Times) - количество миллисекунд с 1 января 1970 г.;

list() – возвращает список файлов для запускающего объекта-каталога.

Класс `File` не симметричен. Под этим подразумевается, что есть много методов, которые позволяют просмотреть свойства простого файлового объекта, но нет соответствующих функций для их изменения.

Класс `File` включает два полезных сервисных метода. Первый – **`renameTo()`** имеет следующую сигнатуру:

`boolean renameTo (File newName) .`

Здесь имя файла, указанное в `newName`, становится новым именем файла, связанного с вызывающим `File`-объектом. Метод возвратит `true` после успешного переименования и `false`, если файл не может быть переименован (например, если вы пытаетесь переименовать файл так, чтобы он переместился из одного каталога в другой, или используете существующее имя файла).

Второй сервисный метод – **`delete()`** удаляет дисковый файл, связанный с вызывающим `File`-объектом и возвращает `true`. Если файл не может быть удален, метод возвращает `false`. Его сигнатура:

`boolean delete() .`

Метод **`delete()`** можно также использовать для удаления каталога, если он пуст.

5.4.2. Библиотечный класс `RandomAccessFile`

В файлах можно хранить не только текст, но и «сырые» данные. Такие данные хранятся как наборы байтов, а для их обработки в Java используются потоки данных `DataInputStream` и `DataOutputStream`. Чтение данных стандартных типов из потока типа `DataInputStream` производят специальные методы, например `readDouble()` для чтения числа типа `double`. Этот метод считывает из потока 8 байтов и преобразует их в значение типа `double`.

Однако тип `DataInputStream` несколько неудобен, если при работе одновременно требуется несколько потоков этого типа, т.к. сложно определить, в каком потоке закончились данные.

Более удобно работать с сырыми данными, используя экземпляры класса `RandomAccessFile`.

Класс `RandomAccessFile` инкапсулирует файл прямого доступа. Он не является производным от классов `InputStream` или `OutputStream`. Вместо этого, указанный класс реализует интерфейсы `DataInput` и `DataOutput`, которые определяют базовые методы ввода/вывода. Он также поддерживает запросы позиционирования, т.е. можно устанавливать указатель (файла) на определенную позицию (байта) в файле.

Класс обладает двумя конструкторами:

RandomAccessFile (File fileObj, String access) throws IOException ;
RandomAccessFile (String filename, String access) throws IOException .

В первой параметр fileObj определяет имя файла, открываемого как File-объект. Во второй форме имя файла передается через параметр filename.

В обоих случаях параметр access определяет какой тип доступа разрешается к файлу. Если в вызове конструктора указано значение "r", то файл можно только читать, но не записывать в него информацию, если "rw", то файл открыт в режиме чтения-записи.

Метод **seek()** служит для установки текущей позиции указателя файла:

void seek (long newPos) throws IOException ,

где newPos определяет новую позицию (номер байта) указателя файла с начала файла. После вызова seek() следующая операция чтения или записи будет выполнена в указанной в вызове позиции файла.

RandomAccessFile реализует стандартные методы ввода и вывода, которые можно использовать для чтения и записи файлов произвольного доступа (ниже приведен неполный перечень):

void close() – закрывает поток, связанный с файлом произвольного доступа и освобождает все системные ресурсы, связанные с потоком;

long length() – возвращает длину файла в байтах;

int read() – читает очередной байт из файла, возвращает этот байт (последний байт в int-значении) или -1, если достигнут конец файла;

int read (byte [] b) – читает до b.length байтов из запускающего файла в массив b и возвращает число считанных байтов или -1, если достигнут конец файла;

int read (byte [] b, int off, int len) – читает до len байтов из запускающего файла в массив b (off – смещение в массиве b) и возвращает число считанных байтов или -1, если достигнут конец файла;

boolean readBoolean() – считывает логическое значение из файла (метод читает один байт из файла, начиная с текущего указателя файла, значение 0 представляет собой false, любое другое значение представляет собой true);

byte readByte() – считывает и возвращает следующий байт этого файла как знаковое byte-значение;

char readChar() – считывает и возвращает очередной символ из файла (2 байта);

double readDouble() – считывает и возвращает очередное double-значение из файла (8 байт);

float readFloat() – считывает и возвращает очередное float-значение из файла (4 байта);

void readFully(byte[] b) – метод читает b.length байт из файла в массив b, начиная с текущего указателя файла (метод считывает из файла несколько раз, пока требуемое количество байтов не будет достигнуто);

void readFully(byte[] b, int off, int len) – метод точно len байт из файла в массив b, начиная с текущего указателя файла (метод считывает из файла несколько раз, пока требуемое количество байтов не будет достигнуто; off – смещение данных в массиве b);

int readInt() – считывает и возвращает очередное int-значение из файла (4 байта);

String readLine() – считывает следующую строку текста из файла (метод последовательно читает байты из файла, начиная с текущего указателя файла, пока он не достигнет конца строки или конца файла; каждый байт преобразуется в символ, принимая значение байта, для младших восьми битов символа и, устанавливая старшие восемь битов символа в нули; метод возвращает следующую строку текста из файла, или нулевое значение, если конец файла встретился еще до того, один байт считался);

long readLong() – считывает и возвращает очередное long-значение из файла (8 байт);

short readShort() – считывает и возвращает очередное short-значение из файла (2 байта);

int readUnsignedByte() – считывает и возвращает очередной байт из файла, интерпретируемый как беззнаковое восьмибитное целое число;

int readUnsignedShort() – считывает и возвращает очередные два байта из файла, интерпретируемые как беззнаковое шестнадцатитбитное целое число;

void setLength (long len) – приводит длину файла к значению, указанному в len (метод может использоваться для удлинения или сокращения файла; если файл удлиняется, то добавленная часть имеет неопределенные значения);

int skipBytes(int n) – метод пытается пропустить n байт ввода, отбрасывая их (возвращает фактическое число пропущенных байтов, которых будет меньше, чем n, например, если достигнут конец файла).

Вышеперечисленным методам ввода значений из файла соответствуют следующие методы вывода:

```
void write(byte[] b),
void write(byte[] b, int off, int len),
void write(int b),
void writeBoolean(boolean v),
void writeByte(int v),
void writeBytes(String s),
void writeChar(int v),
void writeChars(String s),
void writeDouble(double v),
void writeFloat(float v),
void writeInt(int v),
void writeLong(long v),
void writeShort(int v).
```

Более подробное описание перечисленных методов можно получить в [5,6].

5.4.3. Генерация псевдослучайных значений. Класс `Random`

Экземпляр класса `java.util.Random` используется для генерации потока псевдослучайных значений. Класс использует 48-битное начальное значение, которое модифицируется с использованием линейной конгруэнтной формулы.

Конструкторы:

`Random()` – создает генератор псевдослучайных чисел, использующий для формирования начального значения текущее время;

`Random(long seed)` – создает генератор псевдослучайных чисел, использующий для начала работы число `seed`.

Два экземпляра, созданных на основе второго конструктора с одинаковым значением параметра `seed`, будут генерировать одинаковые последовательности псевдослучайных чисел.

Методы (перечень не полон):

`protected int next(int bits)` – генерирует следующее псевдо-случайное число;

`boolean nextBoolean()` – генерирует псевдослучайное `boolean`-значение;

`double nextDouble()` – генерирует псевдослучайное `double`-значение в диапазоне от 0 до 1;

`float nextFloat()` – генерирует псевдослучайное `float`-значение в диапазоне от 0 до 1;

`int nextInt()` – генерирует псевдослучайное `int`-значение;

`int nextInt(int n)` – генерирует псевдослучайное `int`-значение в диапазоне от 0 включительно до `n` исключительно;

`long nextLong()` – генерирует псевдослучайное `int`-значение.

Напомним, что классе `Math` есть метод `random()`, который также можно использовать для формирования последовательности псевдослучайных чисел.

5.4.4. Примеры программ

Пример 1. Разработать программу, которая выводит информацию о файле и папке, в которой находится файл, затем копирует информацию из этого файла в два других файла, после чего изменяет часть байтов во втором скопированном файле.

Создадим новый проект. Структура проекта изображена на рисунке 5.1.

В папке проекта создадим папку `Picture`, а в нее поместим несколько файлов с изображениями (форматы `.jpg`, `.tif`).

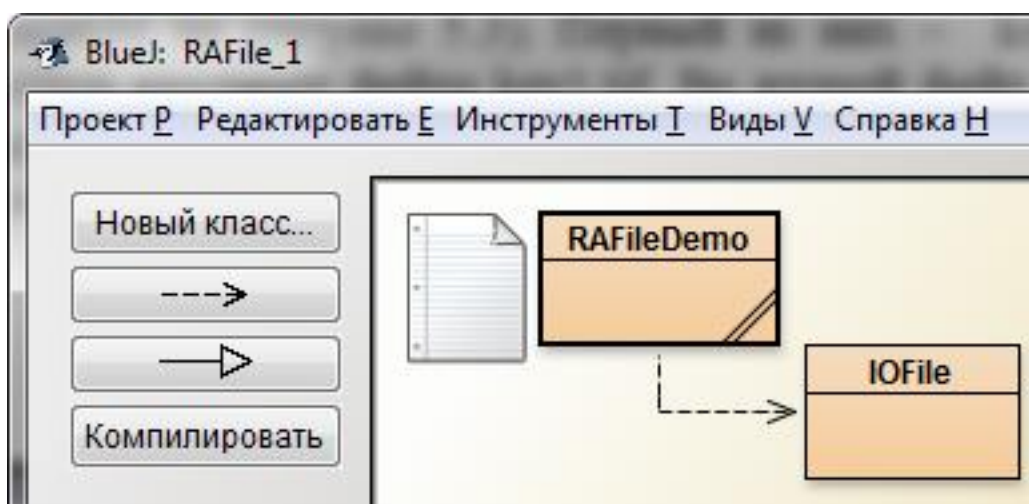


Рисунок 5.1 – Структура проекта для примера 1

В классе `IOFile` определено два статических метода. Первый вводит байты из файла в одномерный массив. Второй предназначен для вывода байтов из одномерного массива в файл. Данные методы используются запускающим классом `RAFileDemo` для выполнения поставленной задачи.

Текст программы приведен ниже.

```
import java.io.*;
public class IOFile{
    public static byte[] inputRAFile(File file) throws IOException {
        //Возвращает ссылку на массив байтов, введенных
        //из файла, связанного с File-объектом file, или null, если при вводе
        //произошла ошибка
        RandomAccessFile inpFile = null;
        int nextByte;
        int i = 0;
        byte [] arr = null;
        try //открываем файл только для чтения:
            inpFile = new RandomAccessFile(file,"r");
            // создаем массив байтов длиной, равной числу байтов в файле:
            arr = new byte[(int)inpFile.length()]; //используется явное
                                                    // преобразование из long в int
            // пока в файле есть байты, читаем очередной байт и
            // и записываем его в arr:
            while ((nextByte = inpFile.read()) != -1){
                arr[i] = (byte)nextByte; // read() возвращает int, преобразуем в byte
                i = i+1;
            }
        }
```

```

    }
    catch (Exception e) {arr = null;}
    finally{ if(inpFile != null) inpFile.close();}
    return arr;
}

public static boolean outputRAFile(File file, byte[ ] arr, int startInd, int num,
                                   long position) throws IOException {
    //Выводит num байтов, начиная с индекса startInd из массива arr
    // в файл, связанный с File-объектом file, начиная с позиции
    //position в файле
    RandomAccessFile outpFile = null;
    boolean f = true; //для возврата значения boolean
    int endInd = startInd + num; //граничный индекс в arr
    try { //открываем файл для чтения-записи:
        outpFile = new RandomAccessFile(file,"rw");
        outpFile.seek (position); //устанавливаем указатель файла на
                                   //заданную позицию
        // и выводим байты из массива arr файл:
        for (int i = startInd; i < endInd; i++) outpFile.write((int)arr[i]);
        // у метода write параметр типа int (используется
        // его последний байт)
    }
    catch (Exception e) {f = false;}
    finally{ if(outpFile != null) outpFile.close();}
    return f;
}
} // IOFile

```

```

import java.io.*;

public class RAFileDemo{
    static void p (String s){ // Выводит строку s
        System.out.println(s);
    }
    static void fileInfo(File f){ //Выводит свойства файла (папки)
        p("Имя файла: " + f.getName());
        p("Путь: " + f.getPath());
        p("Абсолютный путь: " + f.getAbsolutePath());
        p("Надкаталог: " + f.getParent());
        p(f.exists() ? "существует" : "не существует");
        p(f.canWrite() ? "для записи" : "не для записи");
    }
}

```

```

p(f.canRead() ? "для чтения" : "не для чтения");
p((f.isDirectory()? "каталог":"не каталог"));
p(f.isFile() ? "нормальный файл" : "возможно именованный канал");
p(f.isAbsolute() ? "абсолютный" : "не абсолютный");
p("последняя модификация файла: " + f.lastModified ());
p("размер файла: " + f.length() + " байтов");
System.out.println();
}
public static void main (String args[ ]) {
    File f1 = new File("Picture/key1.tif"); //файл
    fileInfo(f1); //вывести свойства файла
    File f2 = new File("Picture"); //папка
    fileInfo(f2); // вывести свойства папки
    String [ ] S = f2.list(); //получить список файлов папки
    // и вывести его:
    System.out.printf("Список файлов каталога %s:\n " , f2.getName());
    for (int i = 0; i < S.length; i++) System.out.println(S[i]);
    try{ //перехватываем исключения ввода-вывода
        //-----ввод-----
        byte [ ] arr = IOFile.inputRAFile(f1);
        if (arr != null) System.out.printf (
            "Информация из файла %s загружена успешно\n",
            f1.getAbsolutePath());
        else System.out.printf (
            "Ошибка ввода из файла %s\n", f1.getAbsolutePath());
        //-----вывод без замены-----
        File f3 = new File("Picture/key1_1.tif"); //файл
        if (IOFile.outputRAFile(f3, arr, 0, arr.length, 0))
            System.out.printf (
                "\nКопирование в файл %s произведено успешно\n",
                f3.getAbsolutePath());
        else System.out.printf ("\nОшибка вывода в файл %s\n",
            f3.getAbsolutePath());
        //-----вывод и замена-----
        f3 = new File("Picture/key1_2.tif"); //файл
        if (IOFile.outputRAFile(f3, arr, 0, arr.length, 0))
            System.out.printf (
                "\nКопирование в файл %s произведено успешно\n",
                f3.getAbsolutePath());
        else System.out.printf ("\nОшибка вывода в файл %s\n",

```

```

        f3.getAbsolutePath());

//Вносим "помехи" в файл
int n = 20000; // изменим 20000 байтов в файле на 0xFF
byte [ ] arr1 = new byte[n];
for (int i = 0; i < n; i++) arr1[i] = -1; // 0xFF= -1(в десятичной системе)
if (IOFile.outputRAFile(f3, arr1, 0, arr1.length, 10000)) // изменения
    // вносятся, начиная с позиции 10000 в файле
    System.out.printf (
        "\nВывод помехи в файл %s произведен успешно\n",
        f3.getAbsolutePath());
else System.out.printf ("\nОшибка вывода в файл %s\n",
    f3.getAbsolutePath());

//-----
//Получаем и выводим список файлов каталога
//после появления в нем новых файлов
S = f2.list();
System.out.printf("Список файлов каталога %s:\n", f2.getName());
for (int i = 0; i < S.length;i++) System.out.println(S[i]);
}
catch (Exception e) { } finally{ }
} //main
} // RAFileDemo

```

Результаты работы программы представлены на рисунках 5.2, 5.3. Анализируя эти результаты, можно увидеть, что в папке Picture, находящейся в текущем каталоге (в папке проекта), появились два новых файла (выделены красным на рисунке 5.3). Первый из них – key1_1.tif – является точной копией входного файла key1.tif. Во второй файл – key1_2.tif – программой были внесены изменения. На рисунке 5.4 представлены изображения, хранящиеся в указанных файлах. Следует отметить, что если изображение будет заporчено достаточно сильно, то программа-просмотрщик изображений просто откажется его открывать.

На рисунке 5.5 показаны значения байтов в файлах key1.tif и key1_2.tif, начиная с байта номер 10000 (0x2710). Чтобы увидеть значения байтов (в шестнадцатеричной системе) в любом файле, запустите программу Far Manager, для перехода на нужный диск нажмите комбинацию клавиш <ALT>+<F1>, найдите в дереве каталогов диска нужный файл, выделите его и нажмите <F3>, а затем <F4>.



```
BlueJ: Окно терминала - RAFile_1
Настройки

Имя файла: key1.tif
Путь: Picture\key1.tif
Абсолютный путь: D:\RAFile_1\Picture\key1.tif
Надкаталог: Picture
существует
для записи
для чтения
не каталог
нормальный файл
не абсолютный
последняя модификация файла: 1460629012000
размер файла: 245564 байтов

Имя файла: Picture
Путь: Picture
Абсолютный путь: D:\RAFile_1\Picture
Надкаталог: null
существует
для записи
для чтения
каталог
возможно именованный канал
не абсолютный
последняя модификация файла: 1461937858223
размер файла: 4096 байтов
```

Рисунок 5.2 – Результаты работы программы для примера 1

```
Список файлов каталога Picture:  
key1.tif  
key2.jpg  
key3.jpg  
key4.jpg  
student.jpg  
students.jpg  
Информация из файла D:\RAFile_1\Picture\key1.tif загружена успешно  
  
Копирование в файл D:\RAFile_1\Picture\key1_1.tif произведено успе  
  
Копирование в файл D:\RAFile_1\Picture\key1_2.tif произведено успе  
  
Вывод помехи в в файл D:\RAFile_1\Picture\key1_2.tif произведен ус  
Список файлов каталога Picture:  
key1.tif  
key1_1.tif  
key1_2.tif  
key2.jpg  
key3.jpg  
key4.jpg  
student.jpg  
students.jpg
```

Рисунок 5.3 – Результаты работы программы для примера 1 – продолжение



Рисунок 5.4 – Изображения в файлах

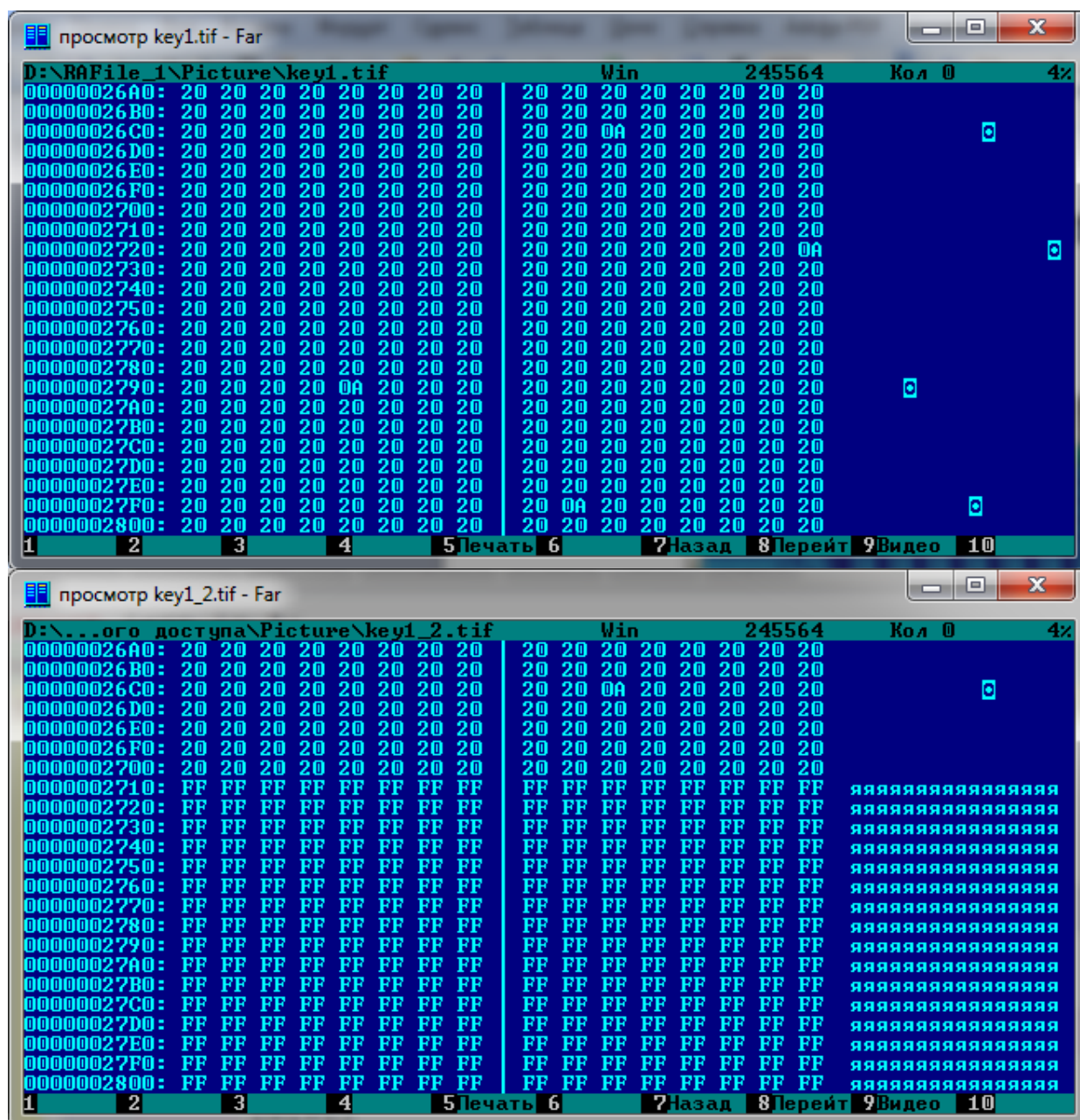


Рисунок 5.5 – Значения байтов в файлах key1.tif и key1_2.tif

Пример 2. Разработать программу, которая выполняет нижеперечисленные действия.

- 1) Выводит в байтовый файл 9 матриц размерностью 3x3. Тип элемента матрицы – double. Значения элементов задаются с помощью генератора псевдослучайных чисел.
- 2) Считывает из полученного файла две матрицы – третью и седьмую.
- 3) Выполняет операцию сложения матриц.
- 4) Результирующую матрицу выводит в тот же файл на место первого операнда (третьей матрицы).
- 6) Результаты вышеперечисленных действий выводит в окно терминала для контроля.

Структура проекта изображена на рисунке 5.6.

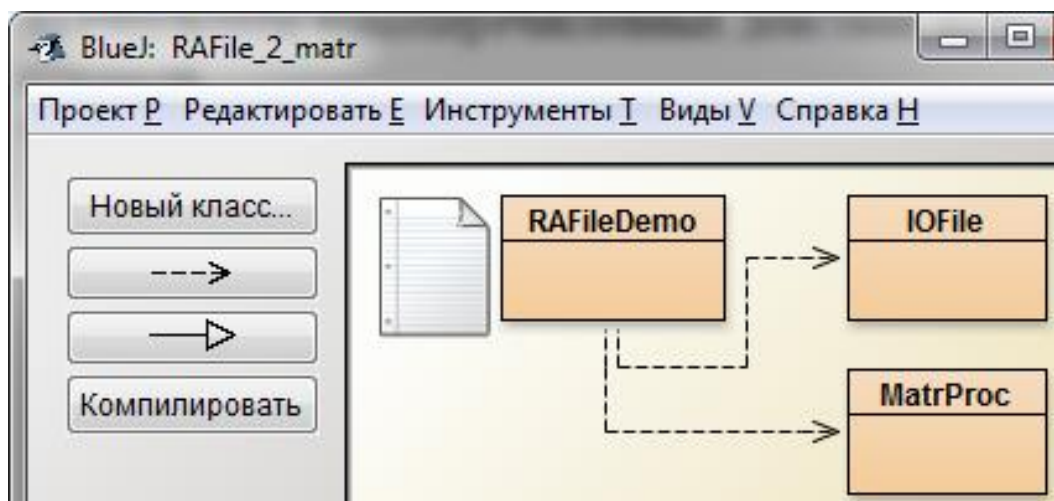


Рисунок 5.6 – Структура проекта для примера 2

Главный класс RAFileDemo обращается к статическим методам, определенным в классе IOFile, для выполнения операций ввода-вывода, и к статическим методам, определенным в классе MatrProc, для выполнения операций над матрицами.

Текст программы приведен ниже.

```
import java.io.*;
public class IOFile { //Функции ввода-вывода
    public static boolean outDoubleRAFile(File file, double[] arr, int pos)
        //Выводит числа double (8 байт каждое) из массива arr в файл file,
        //начиная с позиции pos в файле, возвращает true,
        // если вывод закончился успешно
        RandomAccessFile outpFile = null;
        boolean f = true; //для возврата
        try { //контролируем исключения при выводе данных
            //открываем файл для чтения и записи:
            outpFile = new RandomAccessFile(file, "rw");
            outpFile.seek(pos); // устанавливаем указатель файла
                // на заданную позицию
            for (int i = 0; i < arr.length; i++)
                outpFile.writeDouble(arr[i]); // выводим очередное число (8 байт)
            }
            catch (Exception e) {f = false;}
            finally{ if(outpFile != null) outpFile.close();} //закрываем файл
            return f;
        }
    }
    public static double[][] inpDoubleMatrRAFile(File file, int m, int pos)
        throws IOException {
        // Возвращает ссылку на матрицу с элементами double
        // размером m x m, введенную из файла file, начиная
        // с позиции pos; при ошибке ввода-вывода возвращает null
    }
}
```

```

RandomAccessFile inpFile = null;
long toRead; //счетчик байтов в файле
double [ ][ ] matr = new double[m][m]; // создаем матрицу m x m
try { //контролируем исключения при вводе данных
    //открываем файл для чтения:
    inpFile = new RandomAccessFile(file, "r");
    inpFile.seek (pos); //устанавливаем указатель файла в
        //заданную позицию
    toRead = inpFile.length() - pos + 1; //число байтов в файле, начиная
        //с позиции pos
    //Вводим числа из файла в матрицу:
    M1:{for (int i = 0; i < m; i++) // блок помечен меткой M1
        for (int j = 0; j < m; j++){
            matr[i][j] = inpFile.readDouble(); //введено очередное число
            toRead -= 8; // уменьшаем число оставшихся байт на 8
            if (toRead <= 0) break M1; //если данные в файле закончились,
                //переход на конец блока, отмеченного меткой M1
        }
    } //Конец блока, отмеченного меткой M1
} //try
catch (Exception e) {matr = null;} //если исключение, вернем null
finally{ if(inpFile != null) inpFile.close();} // закрываем файл
return matr; //возвращаем ссылку на матрицу или null
}

public static double[ ] inpDoubleArrRAFile(File file) throws IOException {
    //Вводит все данные из файла в одномерный double-массив,
    //возвращает ссылку на массив или null (если было исключение)
    //Число байтов в файле должно быть кратно 8,
    //иначе метод вернет null (объясните, почему)
    RandomAccessFile inpFile = null;
    long toRead; // счетчик байтов в файле
    double [ ] arr = null;
    try {inpFile = new RandomAccessFile(file, "r");
        toRead = inpFile.length(); //длина файла в байтах
        arr = new double[(int) toRead / 8]; // создаем double-массив
        int i = 0;
        while (toRead > 0){ //пока в файле есть числа
            arr[i] = inpFile.readDouble(); //прочли очередное число (8 байт)
            i = i + 1;
            toRead -= 8; // уменьшаем число оставшихся байтов на 8
        }
    }
    catch (Exception e) {arr = null;} //если исключение, вернем null
    finally{ if (inpFile != null) inpFile.close();} // закрываем файл
    return arr; //возвращаем ссылку на массив или null
}
} // IOFile

public class MatrProc{ //Работа с матрицами и массивами
    //Содержит статические методы для работы с матрицами
    //и одномерными массивами

```

```

public static void putMatr (double [ ][ ] matr){
    //выводит матрицу matr в окно терминала
    if (matr == null) return;
    for (int i = 0; i < matr.length; i++) {
        for (int j = 0; j < matr[0].length; j++)
            System.out.printf ("% 12.2f", matr[ i ][ j ]);
        System.out.println();
    } //for i
} //putMatr

public static void putArr (double X[ ], int n){
    //выводит массив X в окно терминала по n элементов в строке
    if (X == null) return;
    for (int i = 0; i < X.length; i++){
        if ((i != 0) && (i % n == 0)) System.out.println();
        System.out.printf ("% 12.2f", X[i]);
    }
    System.out.println();
} //putArr

public static double[ ] matrToArr (double matr[ ][ ]){
    //преобразует матрицу в одномерный массив:
    //данные первой строки, за ними данные второй строки и т.д.
    double[ ] arr = new double[matr.length * matr[0].length];
    int k = 0;
    for (int i = 0; i < matr.length; i++)
        for (int j = 0; j < matr[0].length; j++){
            arr[ k ] = matr[ i ][ j ];
            k = k+1;
        }
    return arr;
}

public static double sumElMatr(double[ ][ ] matr){
    //вычисляет сумму элементов матрицы
    if (matr == null) return 0;
    double sum = 0;
    for (int i = 0; i < matr.length; i++)
        for (int j = 0; j < matr[0].length; j++)
            sum += matr[ i ][ j ];
    return sum;
}

public static double[ ][ ] sumTwoMatr(double[ ][ ] m1, double [ ][ ] m2){
    //возвращает матрицу, являющуюся суммой двух матриц
    //m1 и m2, или null, если у матриц m1 и m2 разная размерность
    if ((m1 == null) || (m2 == null) || (m1.length != m2.length) ||
        (m1[0].length != m2[0].length)) return null;
    double [ ][ ] m = new double[m1.length][m1[0].length];
    for (int i = 0; i < m.length; i++)
        for (int j = 0; j < m[0].length; j++)
            m[ i ][ j ] = m1[ i ][ j ] + m2[ i ][ j ];
    return m;
}
} // MatrProc

```

```

import java.io.*;
import java.util.Random;
public class RAFileDemo{
    public static void main (String args[ ]) {
        int n = 9, m = 3; String t = "double"; //9 матриц 3 x 3, тип double
        //В папке проекта предварительно должна быть создана папка Matr
        File f1 = new File("Matr/matr_" + n + "_" + m + "_" + t + ".bin"); //файл
        System.out.printf (
            "В файл выводятся %d матриц %dx%d с элементами типа %s\n",
            n, m, m, t);
        //Формирование элементов матриц
        Random r = new Random(); //генератор псевдо-случайных чисел
        double [ ] arr = new double[n*m*m]; //всего чисел для вывода в файл
        for (int i = 0; i < arr.length; i++)
            arr[i] = r.nextDouble()*100; //получаем очередное число
        try{
            //-----вывод-----
            IOFile.outDoubleRAFile(f1, arr, 0);
            //-----ввод всего файла и контроль-----
            //очистка arr:
            for (int i = 0; i < arr.length; i++) arr[i] = 0;
            //ввод в одномерный массив всех чисел из файла
            System.out.printf (
                "Содержимое файла %s, интерпретированное как double-значения:\n",
                f1.getName());
            arr = IOFile.inpDoubleArrRAFile(f1);
            MatrProc.putArr (arr, m * m); //выводим массив arr в окно терминала
            //-----ввод из файла третьей и седьмой матрицы-----
            double[ ][ ] matr1= IOFile.inpDoubleMatrRAFile(f1, m, m * m * 8 * (3 - 1));
            System.out.println ("Введена матрица номер 3:");
            MatrProc.putMatr(matr1); //выводим матрицу в окно терминала
            System.out.printf ("Сумма элементов матрицы: %.2f\n",
                               MatrProc.sumElMatr(matr1));
            double[ ][ ] matr2 = IOFile.inpDoubleMatrRAFile(f1, m, m * m * 8 * (7 - 1));
            System.out.println ("Введена матрица номер 7:");
            MatrProc.putMatr(matr2); //выводим матрицу в окно терминала
            System.out.printf ("Сумма элементов матрицы: %.2f\n",
                               MatrProc.sumElMatr(matr2));

            //вычисление суммы матриц
            matr1 = MatrProc.sumTwoMatr(matr1, matr2);
            System.out.println ("Сумма матриц:");
            MatrProc.putMatr(matr1);
            //-----Вывод результирующей матрицы в файл-----
            //-----на место третьей матрицы-----
            double[ ] arr1 = MatrProc.matrToArr(matr1); //преобразуем матрицу
                                                         // в массив
            System.out.println ("После преобразования матрицы в массив:");
            MatrProc.putArr(arr1,m * m); //выводим на терминал
            IOFile.outDoubleRAFile(f1, arr1, m * m * 8 * (3 - 1)); //выводим в файл
            //-----ввод всего файла и контроль-----
            //очистка arr:

```

```

for (int i = 0; i < arr.length; i++) arr[i] = 0;
//ввод в одномерный массив всех чисел из файла
System.out.printf (
"Новое содержимое файла %s, интерпретированное как double-значения:\n",
f1.getName());
arr = IOFile.inpDoubleArrRAFile(f1);
MatrProc.putArr(arr,m * m); //вывод на терминал
} catch (Exception e) { } finally{ }
} //main
} // RAFileDemo

```

Перед запуском программы проверьте, чтобы в папке проекта была создана папка Matr.

Результаты работы программы представлены на рисунке 5.7

```

BlueJ: Окно терминала - RAFile_2_matr
Настройки
В файл выводятся 9 матриц 3x3 с элементами типа double
Содержимое файла matr_9_3_double.bin, интерпретированное как double-значения:
46,80    19,08    64,46    18,07    9,52    15,39    67,67    98,93    40,61
60,32    78,27    98,90    39,83    88,01    13,13    91,62    61,38    88,76
52,76    18,56    34,69    53,76    80,07    24,71    84,53    80,74    73,18
98,95    63,11    41,92    68,90    60,91    65,67    28,25    86,37    25,07
12,93    74,73    76,42    90,36    48,62    40,24    40,82    48,57    18,35
62,55    48,07    78,93    67,38    34,06    57,46    97,26    19,10    47,17
51,57    58,52    73,46    39,40    68,85    68,96    95,19    28,09    22,53
50,38    45,32    61,79    89,92    13,25    46,41    31,44    57,11    37,39
61,56    78,14    7,80    29,67    10,84    6,34    52,46    26,51    44,43
Введена матрица номер 3:
52,76    18,56    34,69
53,76    80,07    24,71
84,53    80,74    73,18
Сумма элементов матрицы: 498.0
Введена матрица номер 7:
51,57    58,52    73,46
39,40    68,85    68,96
95,19    28,09    22,53
Сумма элементов матрицы: 502.0
Сумма матриц:
104,33    77,08    108,15
93,16    148,92    93,67
179,72    108,83    95,71
После преобразования матрицы в массив:
104,33    77,08    108,15    93,16    148,92    93,67    179,72    108,83    95,71
Новое содержимое файла matr_9_3_double.bin, интерпретированное как double-значения:
46,80    19,08    64,46    18,07    9,52    15,39    67,67    98,93    40,61
60,32    78,27    98,90    39,83    88,01    13,13    91,62    61,38    88,76
104,33    77,08    108,15    93,16    148,92    93,67    179,72    108,83    95,71
98,95    63,11    41,92    68,90    60,91    65,67    28,25    86,37    25,07
12,93    74,73    76,42    90,36    48,62    40,24    40,82    48,57    18,35
62,55    48,07    78,93    67,38    34,06    57,46    97,26    19,10    47,17
51,57    58,52    73,46    39,40    68,85    68,96    95,19    28,09    22,53
50,38    45,32    61,79    89,92    13,25    46,41    31,44    57,11    37,39
61,56    78,14    7,80    29,67    10,84    6,34    52,46    26,51    44,43

```

Рисунок 5.7 – Результаты работы программы

5.5. Порядок выполнения работы

5.5.1. Задание 1

1) Внимательно изучите текст программы примера 1, приведенный в разделе 5.4.4. Методы и классы, определенные в этой программе, вы можете использовать в своей программе.

2) Создайте новый проект Lab9_1_Фамилия_Группа. В папке проекта создайте папку Picture и поместите в нее несколько файлов изображений (.tif, .jpg, .bmp и т.п.).

3) По аналогии с примером 1 разработайте программу, осуществляющую следующие действия:

- вывод информации (методами класса File) об одном из графических файлов и папке Picture, в которой находится этот файл;
- вывод списка файлов, находящихся в папке Picture изначально;
- ввод информации из выбранного графического файла средствами класса RandomAccessFile (информация файла должна восприниматься программой как «сырые» байты);
- копирование считанной информации в другой файл той же папки (при отсутствии указанного файла, он создается);
- копирование считанной информации в еще один файл той же папки (при отсутствии указанного файла, он создается) и изменение заданного количества байтов, начиная с заданного адреса в этом файле (внесение помех) средствами класса RandomAccessFile;
- вывод списка файлов папки Picture после создания в ней новых файлов.

4) Откройте полученные файлы программой-просмотрщиком изображений. Проанализируйте результат

5) Проверьте соответствие внесенных в файл изменений варианту задания (начальный адрес изменений, новое значение байтов, конечный адрес изменений), открыв исходный и измененный файлы редактором программы Far Manager, позволяющим представить содержимое файла в виде последовательности шестнадцатеричных значений байтов.

6)* Добавьте в свою программу блок кода, ограничивающий список файлов (возвращаемых методом **list()** папки) заданным (как параметр) расширением файла (например, чтобы в список включались только файлы с расширением .jpg) и *протестируйте* эту возможность (использование интерфейса **FilenameFilter** рассмотрено в лекции по вводу-выводу данных – проект **Фильтр файлов**).

5.5.2. Задание 2

1) Внимательно изучите текст программы примера 2, приведенный в разделе 5.4.4. Методы и классы, определенные в этой программе, вы можете взять за основу для разработки своей программы.

2) Создайте новый проект Lab9_2_Фамилия_Группа. Создайте в папке проекта папку Matr (в ней будет расположен файл с матрицами).

3) По аналогии с примером 2 разработайте программу, выполняющую следующие действия:

- формирование файла данных: вывод в файл средствами класса `RandomAccessFile` значений элементов заданного типа для заданного количества матриц заданной размерности (значения элементов сформировать с помощью генератора псевдослучайных чисел);
- ввод и контроль (вывод в окно терминала) всех значений из полученного файла данных;
- ввод из файла данных значений для формирования матриц с заданными номерами;
- выполнение над матрицами заданной операции;
- замену в файле данных первой матрицы-операнда на данные матрицы-результата выполненной операции;
- ввод и контроль (вывод в окно терминала) всех значений из измененного файла данных.

5.6. Варианты заданий

Вариант задания определяется таблицами 5.1 и 5.2.

Номер варианта задания V необходимо вычислить по формуле

$$\text{int } V = (N \leq 14) ? \text{variant} : N \% 14;$$

где N – номер студента в списке группы,

variant – номер варианта задания в таблице 5.1 (5.2).

В таблице 5.2 для обозначения логических операций использованы знаки $\vee$ – дизъюнкция (логическое ИЛИ), $\wedge$ – конъюнкция (логическое И), $\oplus$ – сложение по модулю 2 (исключающее ИЛИ). Инверсия элемента обозначена горизонтальной чертой над элементом.

5.7. Содержание отчета

1) Цель работы.

Для каждого из двух заданий:

2) Постановка задачи с описанием варианта задания.

3) Текст программы.

4) Протокол отладки (описание найденных синтаксических и логических ошибок в программе и способов их устранения).

5) Результаты выполнения программы. Для программы 1 кроме скриншота окна терминала предоставить скриншоты окон редактора `Far Manager` для начального и конечного адреса внесенных в файл изменений с интерпретацией этих адресов в десятичной системе счисления.

6) Выводы.

Таблица 5.1 – Варианты для выполнения задания 1 (п. 5.5.1)

Номер варианта	Адрес в файле, с которого вносятся изменения (Dec)	Новое значение байта	Число байтов
1	21000	ED	11000
2	5000	F7	15000
3	15000	AB	5000
4	25000	BD	7000
5	11000	8F	20000
6	14000	9E	15000
7	21000	BB	8000
8	12000	AF	9000
9	17000	FB	16000
10	11500	CD	11500
11	7000	DC	9500
12	9000	AD	10200
14	10500	DB	19000
15	18000	D7	14500

Таблица 5.2 – Варианты для выполнения задания 2 (п. 5.5.1)

Номер варианта	Число матриц	Размерность	Тип элемента	Номер первой матрицы (A)	Номер второй матрицы (B)	Операция (результат – матрица C)
1	5	4 x 4	int	2	4	$c_{ij} = a_{ij} \vee b_{ij}$
2	7	5 x 5	boolean	4	6	$c_{ij} = a_{ij} \wedge b_{ij}$
3	4	4 x 4	long	1	3	$c_{ij} = a_{ij} \oplus b_{ij}$
4	3	6 x 6	boolean	1	3	$c_{ij} = a_{ij} \vee \overline{b_{ij}}$
5	5	4 x 4	short	2	5	$c_{ij} = a_{ij} \wedge \overline{b_{ij}}$
6	6	5 x 5	int	2	4	$c_{ij} = a_{ij} \oplus \overline{b_{ij}}$
7	5	7 x 7	boolean	1	4	$c_{ij} = \overline{a_{ij}} \vee b_{ij}$
8	6	4 x 4	int	3	5	$c_{ij} = \overline{a_{ij}} \wedge b_{ij}$
9	5	5 x 5	float	2	5	$c_{ij} = a_{ij} - b_{ij}$
10	5	4 x 4	long	2	4	$c_{ij} = \overline{a_{ij}} \vee \overline{b_{ij}}$
11	4	6 x 6	short	2	3	$c_{ij} = \overline{a_{ij}} \wedge \overline{b_{ij}}$

Продолжение таблицы 5.2

Номер варианта	Число матриц	Размерность	Тип элемента	Номер первой матрицы (A)	Номер второй матрицы (B)	Операция (результат – матрица C)
12	4	4 x 4	float	1	4	$c_{ij} = a_{ij} / b_{ij}$
13	5	5 x 5	short	2	5	$c_{ij} = \overline{a_{ij}} \wedge \overline{b_{ij}}$
14	4	7 x 7	boolean	3	5	$c_{ij} = \overline{a_{ij}} \vee \overline{b_{ij}}$

5.8. Контрольные вопросы

1) Какие возможности предоставляет программисту класс File. Охарактеризуйте его конструкторы и методы.

2) Чем, с точки зрения Java, папка отличается от файла? Как получить список файлов, находящихся в папке?

3) Какие возможности инкапсулирует класс RandomAccessFile. Когда его удобно применять?

4) Охарактеризуйте конструкторы класса RandomAccessFile.

5) Назовите методы класса RandomAccessFile, которые позволяют устанавливать указатель файла произвольного доступа на нужную позицию и пропускать при вводе заданное количество байтов.

6) Назовите методы ввода и вывода, которые позволяют работать с данными в файле, как с «сырыми» байтами. Как определить, что все байты файла уже введены?

7) Какой метод позволяет определить длину файла произвольного доступа в байтах. Где и для чего в ваших программах вы использовали эту характеристику?

8) Какие методы позволяют вводить и выводить последовательности байтов, интерпретируя их как данные простых типов Java?

9) Какой метод позволяет закрыть файл произвольного доступа и освободить связанные с ним системные ресурсы?

10) Опишите алгоритм метода ввода «сырых» байтов из файла в одномерный byte-массив (метод использовался в программе 1).

11) Опишите алгоритм метода вывода байтов из одномерного массива в файл, начиная с заданной позиции файла (метод использован в программе 1).

12) Опишите алгоритм метода вывода в файл прямого доступа одномерного массива элементов заданного простого типа, начиная с определенной позиции файла. Как вы использовали этот метод для сохранения в файле заданного количества матриц определенной размерности; для вывода в файл матрицы-результата операции (метод использовался в программе 2)?

13) Опишите алгоритм метода ввода матрицы заданной размерности с заданным номером и типом элементов из файла произвольного доступа (метод использовался в программе 2).

14) Опишите алгоритм ввода байтов из файла произвольного доступа в одномерный массив элементов заданного простого типа (метод использовался в программе 2).

15) Опишите назначение и алгоритмы методов, предусмотренных в вашей программе для обработки матриц.

16) Почему методы классов IOFile и MatrProc в программах примеров объявлены статическими?

17) Как передать массив или матрицу методу как параметр? Приведите пример из своей программы.

18) Как метод может вернуть массив или матрицу? Приведите пример из своей программы.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Ноутон П. Java 2 / П. Ноутон, Г. Шилдт. – СПб. : БХВ-Петербург, 2007. – 1072 с. — Текст : непосредственный.

2. Вязовик, Н. А. Программирование на Java : учебное пособие / Н. А. Вязовик. — 2-е изд. — Москва : ИНТУИТ, 2016. — 603 с. — Текст : электронный // Лань : электронно-библиотечная система. — URL: <https://e.lanbook.com/book/100405> (дата обращения: 10.11.2022). — Режим доступа: для авториз. пользователей.

3. Зорина, Н. В. Программирование на языке Джава : учебное пособие / Н. В. Зорина. — Москва : РТУ МИРЭА, 2021 — Часть 1 — 2021. — 148 с. — Текст : электронный // Лань : электронно-библиотечная система. — URL: <https://e.lanbook.com/book/218423> (дата обращения: 10.11.2022). — Режим доступа: для авториз. пользователей.

4. Коузен, К. Современный Java: рецепты программирования / К. Коузен. — Москва : ДМК Пресс, 2018. — 275 с. — ISBN 978-5-97060-134-1. — Текст : электронный // Лань : электронно-библиотечная система. — URL: <https://e.lanbook.com/book/116121> (дата обращения: 10.11.2022). — Режим доступа: для авториз. пользователей.

5. Java™ 2 Platform Standard Edition 5.0 API Specification. – URL : <https://docs.oracle.com/javase/1.5.0/docs/api/index.html> (дата обращения 10.11.2022). — Текст : электронный.

6. Java.io.RandomAccessFile Class // Tutorials Point : [сайт]. – Хайдарабад. 2022. — URL : http://www.tutorialspoint.com/java/io/java_io_randomaccessfile.htm (дата обращения 10.11.2022). — Текст : электронный.

ПРОГРАММИРОВАНИЕ. ЯЗЫКИ ВЫСОКОГО УРОВНЯ

Методические указания

Часть 4

Использование классов библиотеки Java

Составители: **Волкова** Татьяна Викторовна,
Владимирова Елена Сергеевна

В авторской редакции

Изд. № 63/2023. Объем 6,25 п.л. Усл. печ. л. 5,81. Уч.-изд. л. 6,13.
РИИЦМ ФГАОУ ВО «Севастопольский государственный университет»