**ЛАБОРАТОРНАЯ РАБОТА №6**

# ПОТОКИ ВВОДА/ВЫВОДА. РАБОТА С ФАЙЛАМИ. РАБОТА СО СТРОКАМИ

Цель: разработка консольных приложений с возможностью записи текста в файл, чтения его из файла и выполнения действий над этим текстом.

# Класс RandomAccessFile

Класс *RandomAccessFile,* расположенный в библиотеке *java.io,* обеспечивает чтение и запись данных в любом месте файла. Экземпляры этого класса поддерживают и чтение, и запись в файл произвольного доступа. Файл произвольного доступа ведет себя как многочисленный массив байтов, сохраненных в файловой системе. Есть своего рода курсор, или индекс в подразумеваемом массиве, названный указателем файла. Данный способ чтения и записи необходим в тех случаях, когда требуется получить доступ к данным, не читая данные с начала файла. Кроме того, файлы с произвольным доступом могут открываться одновременно по чтению и записи, а также могут открываться только по чтению.

Этот класс реализует сразу два интерфейса – *DataInput* и *DataOutput* – следовательно, может производить запись и чтение всех примитивных типов *Java*. Эти операции, как следует из названия, производятся с файлом. В файле с произвольным доступом присутствует указатель файла (*file pointer*), который всегда указывает на положение новой записи. Метод *seek* позволяет произвольно устанавливать указатель на любой байт внутри файла. Аргумент метода *seek* имеет тип *long (int)* и принимает значение от 0 до максимальной длины файла, выраженной в байтах. А с помощью метода *getFilePointer* можно получить текущую позицию указателя файла.

При создании объекта этого класса конструктору в качестве параметров нужно передать два параметра: файл и режим работы.

*RandomAccessFile (String name, String mode)*

*RandomAccessFile (File file, String mode)*

Где *name* – имя файла, зависящее от системы;

*file* – объект класса File, инкапсулирующий системно-зависимое имя файла;

*mode* – режим открытия файла, может принимать значения *”r”,”rw”,”rws”,”rwd”.*

Файл, с которым будет проводиться работа, указывается либо с помощью *String* – название файла, либо объектом *File*, ему соответствующим.

Значения режимов открытия файла (*mode*):

*"r"* – (*reed*, т.е. чтение) Открывает файл только для чтения (т.е. открытый файл можно использовать только для чтения информации из него, изменять информацию в файле нельзя). Запуск любых методов записи данных приведет к выбросу исключения *IOException*.

*"rw"* – (*reed/write*, т.е. чтение/запись) Открывает файл для чтения и записи (т.е. открытый файл можно использовать как для чтения информации из него, так и для добавления информации в файл). Если файл еще не создан, то осуществляется попытка создать его.

*"rws"* – (*reed/write/synchronously*, т.е. чтение/запись/синхронно) Открывает файл для чтения и записи подобно *"rw"*, а также при каждом изменении содержимого файла или метаданных требует системе синхронно записывать эти изменения на основной носитель.

При использовании синхронного принципа с сохранением не только содержимого файла, но и его метаданных необходимо понимать значение самих метаданных. Метаданные (буквально переводится как «данные о данных» или информация о другом наборе данных) – это информация, описывающая непосредственно файловую систему. Метаданные файловой системы могут содержать информацию о каталоге файла, дату последней модификации, индексные дескрипторы (т.е. структуру данных в файловой системе) и еще ряд специфических элементов. Например, в файловой системе NTFS (New Technology File System – файловая система новой технологии – стандартная файловая система для семейства операционных систем [Microsoft](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D0%B9%D0%BA%D1%80%D0%BE%D1%81%D0%BE%D1%84%D1%82) [Windows NT](https://ru.wikipedia.org/wiki/Windows_NT)) метаданные файловой системы являются частью файла. При этом следует иметь в виду, что не всякая файловая система может поддерживать метаданные и, следовательно, сохранение метаданных может быть не нужно вообще.

*"rwd"* – (*reed/write/device*, т.е. чтение/запись/устройство) Открывает файл по чтению и записи подобно *"rws"*, но принуждает систему синхронно записывать изменения на основной носитель только при каждом изменении содержимого файла. Если изменяются метаданные, синхронная запись не осуществляется.

*"rws"* и *"rwd"* режимы гарантируют при запуске методов, что все изменения файла будут сразу же записываться на носитель. Этот метод пригоден для уверенности, что критически важная информация не будет потеряна во время сбоя системы.

Попытка открыть несуществующий файл только на чтение приведет к исключению *FileNotFoundException*. При открытии на чтение и запись он будет создан сразу (или же будет выброшено исключение *FileNotFoundException*, если это невозможно осуществить).

После создания объекта *RandomAccessFile* можно воспользоваться методами интерфейсов *DataInput* и *DataOutput* для проведения с файлом операций считывания и записи. По окончании работы с файлом его следует закрыть, вызвав метод *close()*.

## Наиболее употребляемые методы класса RandomAccessFile

*long getFilePointer()* – Возвращает текущую позицию указателя файла.

*void seek(long pos)* – Перемещает указатель файла на *pos* байтов от начала файла. Начиная с этой позиции, будет производиться следующая операция чтения или записи. В этом случае длина файла не изменится. Смещение может быть установлено за пределы файла. Длина файла изменится только в случае записи, при которой смещение установлено за пределы файла.

*int skipBytes(int n)* – Осуществляет попытку пропустить *n* байт. Этот метод может пропустить гораздо меньше байт, возможно даже 0 байт. Подобная ситуация может возникнуть в случае достижения конца файла до того как *n* байт будет пропущено – это одна из возможностей. Этот метод никогда не выбрасывает *EOFException*. Возвратится действительное количество пропущенных байтов. Если *n* отрицательно, то байты не будут пропускаться.

*long length()* – Возвращает длину файла, выраженную в байтах.

*void setLength(long newLength)* – Устанавливает длину заданного файла.

Если текущая длина файла, возвращаемая методом *length* больше чем аргумент *newLength*, то файл будет обрезан. В этом случае, если смещение, которое возвращается методом *getFilePointer* больше чем аргумент *newLength*, то после этого метода смещение будет равно *newLength*. Если текущая длина файла, возвращаемая методом *length* меньше чем аргумент *newLength*, то файл будет расширен. В этом случае содержимое расширяемой порции файла не определено, хотя часто заполняется символом с кодом 0.

**Некоторые методы для чтения (ввода) из файла**

*int read()* – читает байт данных из заданного файла;

*int read(byte b[])* – читает байты данных до *b.length()* из заданного файла в массив байтов *b[]*;

*final boolean readBoolean() –* читает данные типа *boolean* из заданного файла;

*final byte readByte()* – читает 8-битное значение из заданного файла;

*final char readChar()* – читает символ из заданного файла;

*final double readDouble()* – читает данные типа *double* из заданного файла;

*final float readFloat()* – читает данные типа *float* из заданного файла;

*final int readInt()* – читает данные типа *int* из заданного файла;

*final String readLine()* – читает строку текста из заданного файла.

**Некоторые методы для записи в файл с прямым доступом**

*void write(byte b[])* – запись *b.length()* байтов от указанного байтового массива *b[]* в заданный файл начиная с текущей позиции указателя файла;

*void write(int b)* – запись указанного байта *b* в заданный файл;

*final void writeBoolean(boolean v)* – запись данных типа *boolean* в заданный файл*;*

*final void writeBytes(String s)* – запись строки в заданный файл в виде последовательности байтов;

*final void writeChars(String s)* – запись строки в заданный файл в виде последовательности символов;

*final void writeDouble(double v)* – запись данных типа *double* в заданный файл*;*

*final void writeFloat(float v)* – запись данных типа *float* в заданный файл*;*

*final void writeInt(int v)* – запись данных типа *int* в заданный файл*;*

## Создание файлов

Если при создании экземпляра класса *RandomAccessFile* в конструкторе указать имя файла, который отсутствует в файловой системе, то он будет автоматически создан с нулевой длиной. Автоматическое создание файла гарантировано в том случае, если пользователь имеет права на создание файлов в заданном каталоге. Если такие права отсутствуют, то будет выброшено исключение. Перед тем как оно будет выброшено, следует закрыть поток, освободив системные ресурсы, связанные с этим потоком. В подобных случаях для гарантированного закрытия файловых потоков часто применяется блок *finally* после соответствующих *try* и *catch* блоков. В нижеприведенной программе, код в блоке *finally* гарантирует закрытие файлового потока непосредственно перед выбросом исключения. Внешние *try catch* блоки в данном случае присутствуют из-за того, что использование метода *close()* в блоке *finally* требует либо внесения в список исключений, которые будут возбуждаться методом с помощью ключевого слова *throws*, либо перехвата исключения. В некоторых случаях лучше перехватывать исключения с помощью *try catch* блоков, чем вносить в список исключений метода, поскольку дальнейшее использование подобного метода внутри других методов также требует перехвата или внесения в список исключений. Кроме того, блоками *try catch* удобно управлять выводом информации о перехваченном исключении и обхода исключений для дальнейшего выполнения программы.

Пример 7.1 Создание файла

import java.io.RandomAccessFile;

import java.io.IOException;

public class CreateFileDemo {

public static void main(String[] args) {

RandomAccessFile raf = null;

try {

try {

// Создание нового файла

raf = new RandomAccessFile("output.dat", "rw");

raf.close();

} catch (IOException e) {

} finally {

if (raf != null) {

raf.close();

}

}

} catch (IOException e) {

}

}

}

При создании потоков могут возникать исключения *FileNotFoundException*, *SecurityException*, *IOException*. Исключение *FileNotFoundException* возникает при попытке открыть входной поток данных для несуществующего файла, то есть когда файл не найден. Исключение *SecurityException* возникает при попытке открыть файл, для которого запрещен доступ. Например, если файл можно только читать, а он открывается для записи, возникнет исключение *SecurityException*. Если файл не может быть открыт для записи по каким-либо другим причинам, то возникает исключение *IOException*.

## Чтение и запись файлов

После того как файл открыт, вы можете использовать любые методы *readXXX()* и *writeXXX()* для ввода и вывода. В момент создания объекта класса *RandomAccessFile* файловый указатель устанавливается в начало файла и имеет значение 0. Вызовы методов *readXXX()* и *writeXXX()* обновляют позиции файлового указателя, сдвигая его на количество прочитанных (записанных) байтов. Для произвольного сдвига файлового указателя на некоторое количество байтов можно применить метод *skipBytes()*, или же установить файловый указатель в определенное место файла вызовом метода *seek()*. Для того, чтобы узнать текущую позицию, в которой находится файловый указатель, нужно вызвать метод *getFilePointer()*.

*RandomAccessFile* одновременно реализует интерфейсы *DataInput* и *DataOutput*. Поэтому класс может производить запись и чтение всех примитивных типов *Java*. При этом их можно производить поочередно, произвольным образом перемещаясь по файлу.

Следующий пример демонстрирует запись чисел типа *int* в файл. Запись производится четырьмя байтами, старший байт первый. Запись начинается с текущей позиции указателя файла. Чтение и запись в приведенной программе проходит по нескольку байт в зависимости от того, сколько отводится байт для примитивного типа данных. Перед запуском программы необходимо удалить файл *int.txt* если он существует.

Пример 7.2 Запись чисел типа *int* в файл

import java.io.RandomAccessFile;

import java.io.IOException;

public class writeIntDemo {

public static void main(String args[]) {

int numbers[] = {1, 8, 16, 1024, 311111111};

try {

try (RandomAccessFile raf = new RandomAccessFile("int.txt", "rw")) {

for (int i = 0; i < numbers.length; i++) {

raf.writeInt(numbers[i]);

}

}

} catch (IOException e) {

System.err.println(e.getMessage());

}

}

}

В результате выполнения программы можно ничего не увидеть в текстовом редакторе. Реально записанные данные можно просмотреть в Hex-редакторе.

Следующий пример демонстрирует запись строки в указанное место в файле. В отличие от предыдущего примера в данном случае в файл записывается строка, которую можно просмотреть в текстовом редакторе. Программа демонстрирует позиционирование для записи в указанное место файла. Позиционирование осуществляется с помощью метода *seek*. Запись затирает данные, которые находились в месте записи. Перед запуском программы необходимо удалить файл *seek.txt* если он существует.

Пример 7.3 Запись строки в указанное место в файле

import java.io.RandomAccessFile;

import java.io.IOException;

public class seekDemo {

public static void main(String[] args) {

RandomAccessFile raf = null;

try {

try {

// Создание нового файла

raf = new RandomAccessFile("seek.txt", "rw");

raf.writeBytes("It is a string");//запись строки в файл

raf.seek(8);//смещение на 8 позицию в файле

raf.writeBytes("surprise!");//запись новой строки начиная с 8 позиции

raf.close();

} catch (IOException e) {

} finally {

if (raf != null) {

raf.close();

}

}

} catch (IOException e) {

}

}

}

В результирующем файле будет содержаться строка "It is a surprise!". Нужно постоянно иметь в виду, что запись данных в указанное место в файле фактически не сдвигает последующее содержимое, а затирает его.

## Дописывание в файл

Чаще всего для добавления данных в конец файла используется *RandomAccessFile*. Однако для добавления небольших блоков данных можно воспользоваться альтернативным способом с помощью класса *FileOutputStream*. Этот класс имеет конструктор, который считывает параметр типа *boolean*. Установка этого параметра в значение *true* приведет к записи в конец файла, то есть, к добавлению в конец файла.

Ниже приведена одна и та же программа дописывания в конец файла с помощью разных классов. При запуске программ необходимо удалить файл *output.txt*, если он существует.

Пример 7.4 Дописывание в конец файла с помощью класса *FileOutputStream*

import java.io.PrintStream;

import java.io.BufferedOutputStream;

import java.io.FileOutputStream;

import java.io.IOException;

public class FileAppendDemo {

public static void main(String[] args) {

PrintStream out = null;

try {

// Создание нового файла

out = new PrintStream(new BufferedOutputStream(new FileOutputStream("output.txt")));

// Запись некоторых тестовых данных в него

out.println("Original data");

// Закрытие файла

out.close();

// Открытие файла в режиме добавления

out = new PrintStream(new BufferedOutputStream(new FileOutputStream("output.txt", true)));

// Добавление данных

out.println("Appended data");

} catch (IOException e) {

} finally {

if (out != null) {

out.close();

}

}

}

}

Пример 7.5 Дописывание в конец файла с помощью класса *RandomAccessFile*

import java.io.RandomAccessFile;

import java.io.IOException;

public class FileAppendDemo {

public static void main(String[] args) {

long l;

RandomAccessFile raf = null;

try {

try {

// Создание нового файла

// Открытие файла в режиме чтения и записи

raf = new RandomAccessFile("output.txt", "rw");

raf.writeBytes("Original data \r\n");//запись данных в файл

raf.close();//закрытие файла

raf = new RandomAccessFile("output.txt", "rw");//открытие файла

l = raf.length();//определение длины файла

raf.seek(l);//смещение в конец файла

raf.writeBytes("Appended data");// Добавление информации

raf.close();

} catch (IOException e) {

} finally {

if (raf != null) {

raf.close();

}

}

} catch (IOException e) {

}

}

}

Наиболее типичной ошибкой при дописывании является следующая последовательность действий: запись, закрытие потока, открытие потока и дописывание без установки указателя. Если открыть поток и не установить указатель в файле, то он по умолчанию будет указывать на начало файла, и любые операции записи будут затирать уже имеющиеся данные файла.

Следующий пример демонстрирует запись и чтение данных в/из файла, а также работу методов *seek()*, *length()*, *setLength()*.

Пример 7.6 Запись и чтение данных

import java.io.\*;

public class JavaInputOutput {

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

try (RandomAccessFile rf = new RandomAccessFile("test.txt", "rw")) {

String str = "0123456789";

rf.write(str.getBytes()); //записываем строку str в файл

} catch (IOException e) {

System.err.println(e.getLocalizedMessage());

}

try (RandomAccessFile rf2 = new RandomAccessFile("test.txt", "rw")) {

System.out.println("Пропускаем 5 байтов в файле");

rf2.skipBytes(5);//пропускаем 5 байтов в файле

System.out.println((char) rf2.readByte());//выводим 6ой символ строки

System.out.println((char) rf2.readByte());

System.out.println((char) rf2.readByte());

System.out.println("Ставим на 5 позицию в файле");

rf2.seek(5);

System.out.println((char) rf2.readByte());

System.out.println((char) rf2.readByte());

System.out.println((char) rf2.readByte());

System.out.println((char) rf2.readByte());

//устанавливаем новую длину файла в байтах и переводим каретку на начало файла

rf2.setLength(7);

rf2.seek(0);

for (int i = 0; i < rf2.length(); i++) {

System.out.print((char) rf2.readByte());

}

System.out.println();

//устанавливаем новую длину файла в байтах и переводим каретку на начало файла

rf2.setLength(3);

rf2.seek(0);

for (int i = 0; i < rf2.length(); i++) {

System.out.print((char) rf2.readByte());

}

System.out.println();

} catch (IOException e) {

System.err.println(e.getLocalizedMessage());

}

}

}

## Клонирование файлов.

*RandomAccessFile* можно использовать также для копирования содержимого файла в новый файл с другим именем (так называемого клонирования). Наиболее быстрым и удобным является способ копирования с помощью каналов *java.nio.channels.FileChannel*. Этот пакет введен в JDK 1.4 и хорошо зарекомендовал себя для перемещения содержимого между файлами больших размеров. Копирование с помощью каналов особенно подходит в тех случаях, когда файл имеет размер, выходящий за пределы нескольких десятков мегабайт. Для небольших файлов можно воспользоваться обычным способом: прочитать в массив байт, а затем записать их в файл под другим именем. Метод клонирования с помощью каналов предоставляет возможность последовательно читать файл небольшими блоками и записывать их в другой файл, что существенно упрощает запись файлов большого объема. Кроме того, количество строк кода минимально. Ниже приведен пример для такого метода. Перед запуском программы следует создать исходный файл 1.txt.

Пример 7.7 Клонирование файлов

import java.io.RandomAccessFile;

import java.io.IOException;

import java.nio.channels.FileChannel;

public class CloneFileDemo {

public static void main(String[] args) {

FileChannel srcChannel = null;

FileChannel dstChannel = null;

try {

try {

srcChannel = new RandomAccessFile("1.txt", "r").getChannel();

dstChannel = new RandomAccessFile("2.txt", "rw").getChannel();

// Копирование содержимого файла

dstChannel.transferFrom(srcChannel, 0, srcChannel.size());

// Закрытие каналов

srcChannel.close();

dstChannel.close();

} catch (IOException e) {

} finally {

if (srcChannel != null) {

srcChannel.close();

}

if (dstChannel != null) {

dstChannel.close();

}

}

} catch (IOException e) {

}

}

}

Класс *RandomAccessFile* является гибким и мощным средством для чтения, записи, дописывания и клонирования файлов. Кроме того, обладает несколькими режимами записи файлов, что позволяет сохранить данные в случае системных сбоев. В отличие от других подобных классов данный класс имеет возможность записи и позиционирования в любом месте файла, а также позволяет устанавливать длину файла. Дополнительное преимущество класса состоит в том, что можно производить запись в файл данные примитивных типов и типа *String*. В сочетании с классом *FileChannel* позволяет быстро клонировать файл и является альтернативой клонирования при сочетании других классов.

# 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*.

Каждая создаваемая строка в действительности является объектом типа *String*. Даже строчные константы – это фактически *String*-объекты. Например, в утверждении

*System.out.println(“This is a String, too”);*

строка “This is a String, too” является *String*-константой.

Для работы со строками определен класс *String* в стандартной библиотеке *Java* в пакете *java.lang* (этот пакет импортируется по умолчанию).

Конструкторы:

*public String();*  //создает пустую строку

*public String(char value[ ]);* //создает строку из массива символов

*public String(byte bytes[ ]);* //создает строку из массива байт

Например,

*String str = new String(“Какая-то строка”);*

можно также записать

*String str = “Какая-то строка”;*

Для строк определена операция сложения, которая означает конкатенацию строк (применение оператора *«+»*, который соединяет две строки, порождая в результате объект класса *String*). Определена операция сложения с числом: сначала число преобразуется в строку, а потом соединяются строка с числом. Определена операция сложения строки с любым объектом. Для объекта вызывается метод *toString*, затем – конкатенация строк.

## Методы класса String

В классе String существует масса полезных методов, которые можно применять к строкам (перед именем метода указан тип того значения, которое он возвращает):

*int length()* — возвращает длину строки (т.е. количество символов в ней);

*boolean isEmpty()* — проверяет, пустая ли строка;

*String replace(a, b)* — возвращает строку, где символ *a* (литерал или переменная типа char) заменён на символ *b*;

*String toLowerCase()* — возвращает строку, где все символы исходной строки преобразованы к строчным;

*String toUpperCase()* — возвращает строку, где все символы исходной строки преобразованы к прописным;

*boolean equals(s)* — возвращает истину, если строка к которой применён метод, совпадает со строкой *s* указанной в аргументе метода (с помощью оператора *==* строки сравнивать нельзя, как и любые другие объекты);

*int compareTo(String str)* – определяет, является ли строка больше, меньше или равной другой строке. Если возвращается значение меньше 0, то вызывающая строка меньше строки str; если больше 0, то вызывающая строка больше строки str; если 0, то две строки эквивалентны.

*int indexOf(ch)* — возвращает индекс символа *ch* в строке (индекс это порядковый номер символа, но нумероваться символы начинают с нуля). Если символ совсем не будет найден, то возвратит -1. Если символ встречается в строке несколько раз, то возвратит индекс его первого вхождения.

*int lastIndexOf(ch)* — аналогичен предыдущему методу, но возвращает индекс последнего вхождения, если символ встретился в строке несколько раз.

*int indexOf(ch,n)* — возвращает индекс символа *ch* в строке, но начинает проверку с индекса *n* (индекс это порядковый номер символа, но нумероваться символы начинают с нуля).

*char charAt(n)* — возвращает код символа, находящегося в строке под индексом *n* (индекс это порядковый номер символа, но нумероваться символы начинают с нуля);

*String trim()* – удаляет из строки начальный и конечный пробелы;

*toCharArray()* – преобразовывает все символы строки в символьный массив;

*String substring(int beginIndex, int endIndex)* – выделяет подстроку из строки.

Пример 7.8 Демонстрация работы методов класса *String*

public class StringWork {

public static void main(String[] args) {

String s1 = "firefox";

System.out.println(s1.toUpperCase()); // выведет «FIREFOX»

String s2 = s1.replace('o', 'a');

System.out.println(s2); // выведет «firefax»

System.out.println(s2.charAt(1)); // выведет «i»

int i;

i = s1.length();

System.out.println(i); // выведет 7

i = s1.indexOf('f');

System.out.println(i); // выведет 0

i = s1.indexOf('r');

System.out.println(i); // выведет 2

i = s1.lastIndexOf('f');

System.out.println(i); // выведет 4

i = s1.indexOf('t');

System.out.println(i); // выведет -1

i = s1.indexOf('r', 3);

System.out.println(i); // выведет -1

}

}

Результат работы программы:

FIREFOX

firefax

i

7

0

2

4

-1

-1

## Преобразование строки типа String в массив типа String[ ]

Для того чтобы разбить строку на слова, разделенные пробелами, символами табуляций '\t', перевода строки '\n' и возврата каретки '\r' нужно подключить библиотеку *java.util.StringTokenizer* и создать объект класса *StringTokenizer*. Этот класс предназначен для выделения из строки отдельных токенов (слов) и может быть полезен, если вам необходимо работать с отдельными словами.

Пример 7.9 Преобразование строки типа *String* в массив типа *String[ ]*

import java.io.\*;

import java.util.\*;

public class ConvertString {

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

String analizingStr = "word1 word2 word3";

System.out.println("Вывод строки:");

System.out.println(analizingStr);// выведет word1 word2 word3

StringTokenizer tokenizer = new StringTokenizer(analizingStr);

int n = tokenizer.countTokens();//подсчет количества слов

String[] tokens = new String[n];//создание массива для записи отдельных слов

for (int i = 0; i < n; i++) {

tokens[i] = tokenizer.nextToken();//запись слов в массив

}

System.out.println("Вывод второго слова из строки:");

System.out.println(tokens[1]);//выведет word2

}

}

# Пример выполнения индивидуального задания

Требуется определить количество гласных, пробелов и общее количество букв. Реализовать механизм записи информации в файл и считывания ее из файла, также возможность редактирования исходного текста в файле (добавление текста в начало файла, добавление текста в конец файла, добавление текста в произвольную позицию в файле). В программе используется меню, и все действия реализованы в виде соответствующих методов класса.

Пример 7.10 Работа с файлом и строкой с использованием меню и отдельных методов класса

import java.io.\*;

public class WorkWithString {

String data;

String filename;

String choice;

String choice2;

RandomAccessFile fio;

BufferedReader in;

public WorkWithString() throws UnsupportedEncodingException {

this.in = new BufferedReader(new InputStreamReader(System.in, "Cp1251"));//создание объекта класса BufferedReader с возможностью ввода данных на русском языке(кодировка "Cp1251")

}

public void runConsol() throws IOException {

while (true) {

printMenu();//вызов метода вывода меню на экран

choice = in.readLine();//ввод выбранного пункта меню

if (choice.compareTo("1") == 0) {//сравнение выбранного пользователем пункта меню

textIntoFile();//вызов метода ввода текста и записи его в файл

} else if (choice.compareTo("2") == 0) {

printRedactMenu();//вызов метода вывода возможных действий редактирования текста на экран

choice2 = in.readLine();//ввод выбранного пункта

if (choice2.compareTo("1") == 0) {

addStart();//вызов метода добавления текста в начало файла

} else if (choice2.compareTo("2") == 0) {

addEnd();//вызов метода добавления текста в конец файла

} else if (choice2.compareTo("3") == 0) {

addRandom();//вызов метода добавления текста в указаную позицию в файле

}

} else if (choice.compareTo("3") == 0) {

textFromfile();//вызов метода чтения текста из файла и определение количества гласных, пробелов и общего количества букв

} else if (choice.compareTo("4") == 0) {

return;//выход из программы

}

}

}

public void printMenu() {//метод вывода меню на экран

System.out.println("Введите ваш выбор:");

System.out.println("1.Ввести текст и записать его в файл");

System.out.println("2.Редактировать текст в файле");

System.out.println("3.Прочитать текст из файла и выполнить над ним действия");

System.out.println("4.Выход");

}

public void textIntoFile() throws IOException {//метод ввода текста и записи его в файл

System.out.println("Введите текст:");

data = in.readLine();//ввод текста

System.out.println("Введите имя файла:");

filename = in.readLine();//ввод имя файла, которое следует вводить учитывая расширение, например text.txt

fio = new RandomAccessFile(new File(filename), "rw");//создание файла с заданным именем

fio.writeBytes(data);//запись текста в заданный файл

fio.close();

System.out.println("Выш текст сохранен.");

}

public void printRedactMenu() {//метод вывода возможных действий редактирования текста на экран

System.out.println("Выберите действие:");

System.out.println("1 - добавление текста в начало файла");

System.out.println("2 - добавление текста в конец файла");

System.out.println("3 - добавление текста в произвольную позицию в файле");

}

public void addStart() throws IOException {//метод добавления текста в начало файла

System.out.println("Введите имя файла:");

filename = in.readLine();//ввод имени файла, которое следует вводить учитывая расширение, например text.txt

fio = new RandomAccessFile(new File(filename), "rw");

data = fio.readLine();//чтение информации из заданного файла

System.out.println("Введите строку для добавления в начало:");

String s;

s = in.readLine();//ввод строки

fio.seek(0);//переход в начало файла

fio.writeBytes(s);//запись введенной строки

fio.seek(s.length());//переход в конец записанной строки

fio.writeBytes(data);//запись исходного текста после введенной строки

fio.close();

System.out.println("Cтрока записана в начало файла.");

}

public void addEnd() throws IOException {//метод добавления текста в конец файла

System.out.println("Введите имя файла:");

filename = in.readLine();//ввод имени файла, которое следует вводить учитывая расширение, например text.txt

fio = new RandomAccessFile(new File(filename), "rw");

data = fio.readLine();//чтение информации из заданного файла

System.out.println("Введите строку для добавления в конец:");

String s;

s = in.readLine();//ввод строки

fio.seek(fio.length());//переход в конец файла

fio.writeBytes(s);//запись введенной строки в конец файла

fio.close();

System.out.println("Cтрока записана в конец файла.");

}

public void addRandom() throws IOException {//метод добавления текста в указаную позицию в файле

System.out.println("Введите имя файла:");

filename = in.readLine();//ввод имени файла, которое следует вводить учитывая расширение, например text.txt

fio = new RandomAccessFile(new File(filename), "rw");

System.out.println("Введите строку для добавления в указанную позицию в файле:");

String s;

s = in.readLine();//ввод строки

System.out.println("Введите необходимую позицию в файле:");

int n;

n = Integer.parseInt(in.readLine());//ввод позиции

fio.seek(n);//смещение на n позицию в файле

data = fio.readLine();//чтение файла начиная с позиции n

fio.seek(n);

fio.writeBytes(s);//запись введенной строки с позиции n

fio.writeBytes(data);//запись прочитанного с позиции n текста после введенной строки

fio.close();

System.out.println("Cтрока записана в файл.");

}

public void textFromfile() throws IOException {//метод чтения текста из файла и определение количества гласных, пробелов и общего количества букв

System.out.println("Введите имя файла:");

filename = in.readLine();//ввод имени файла, которое следует вводить учитывая расширение, например text.txt

fio = new RandomAccessFile(new File(filename), "r");

data = fio.readLine();//чтение информации из заданного файла

fio.close();

System.out.println("Информация из файла: " + data);//вывод информации из файла

int spaces = 0, glas = 0, lett = 0; //установление счетчика количества пробелов, гласных и всех букв на 0

char ch;

for (int i = 0; i < data.length(); i++) {

ch = Character.toLowerCase(data.charAt(i)); //преобразоваине всех символов строки к строчным для корректного подсчета

if (Character.isWhitespace(ch)) {//проверка, является и символ пробелом

spaces++;//увеличение счетчика количества пробелов

}

if ((ch == 'a') || (ch == 'e') || (ch == 'i') || (ch == 'o') || (ch == 'u') || (ch == 'y')) {//проверка символа на то, является ли он гласной буквой

glas++;//увеличение счетчика гласных букв

}

lett++;//подсчет общего количества символо в строке

}

System.out.println("Количество пробелов - " + spaces + "\nКоличество гласных - " + glas + "\nКоличество букв - " + (lett - spaces));//вывод количества пробелов, гласных, общего количества букв, которое равно разнице между всеми символами строки и количеством пробелов

}

public static void main(String args[]) throws UnsupportedEncodingException, IOException {

WorkWithString n = new WorkWithString();//создание объекта класса WorkWithString

n.runConsol();//вызов метода, выполняющего действия над файлом и строкой

}

}

Результат работы программы:

Введите ваш выбор:

1. Ввести текст и записать его в файл
2. Редактировать текст в файле
3. Прочитать текст из файла и выполнить над ним действия
4. Выход

Выбрав 1-й вариант, введем строку и имя файла, где необходимо сохранить строку. После этого будет предложено снова осуществить выбор.

Теперь выберем 3-й вариант. Введем имя нашего файла. В результате выведется исходная строка и итоги ее анализа: количество пробелов, гласных и общее количество букв. И снова будет предложен выбор.

Теперь выберем 2-й вариант. Будут предложены возможности редактирования текста.

Выберите действие:

1. добавление текста в начало файла
2. добавление текста в конец файла
3. добавление текста в произвольную позицию в файле

Теперь выберем 4-й вариант. После этого программа завершит свою работу.

# Задания для самостоятельного выполнения

Дан исходный символьный файл, в котором находится текст, состоящий не менее чем из 30 слов. Написать программу, которая выполняла бы следующие действия над этим текстом. Читала его из файла. По необходимости, редактировала его в файле. Делала над ним действия, указанные ниже в вашем варианте заданий. Результат выполнения действий записывала в байтовый файл. При этом все указанные возможности необходимо реализовать через меню:

а) Чтение текста из файла (исходного).

б) Редактирование текста исходного файла (добавление текста в начало файла, добавление текста в конец файла, добавление текста в произвольную позицию в файле).

в) Выполнение действий над текстом.

г) Запись текста в файл.

Указанные действия необходимо реализовать в виде соответствующих методов класса. **Запрещается** помещать весь код в метод *public static void* ***main****(String args[])*. **Запрещается** использовать класс *Scanner*.

Варианты заданий:

1. Найти слова текста, начинающиеся с гласной буквы.
2. Найти слова текста, для которых последняя буква одного слова совпадает с первой буквой следующего слова.
3. Найти в тексте все цифры, идущие подряд.
4. Найти количество согласных букв.
5. Найти в тексте все символы «?» и «!», идущие подряд.
6. Найти в тексте слова одинаковой длины.
7. Поменять местами четные и нечетные слова текста.
8. Удалить в тексте все повторяющиеся слова.
9. Удалить в тексте все цифры.
10. Определить, содержит ли текст слова «Java» и «Hello», и определить их количество.
11. Найти в тексте слово максимальной длины.
12. Найти в тексте слова, помещенные между символами « ».
13. Удалить все «лишние» пробелы, то есть из нескольких подряд идущих пробелов оставить только один.
14. Поменять местами первое с последним словом в тексте.
15. Найти все символы «#», «\*» и «@» в тексте и заменить символом «&».

# Контрольные вопросы

1. Какие преимущества у класса *RandomAccessFile*?
2. Какие интерфейсы одновременно реализует класс *RandomAccessFile*?
3. Какие существуют конструкторы класса *RandomAccessFile*?
4. Какие существуют режимы открытия файлов и для чего они предназначены?
5. Какие существуют методы класса *RandomAccessFile*?
6. Какой метод перемещает указатель файла на определенное количество байтов от начала файла?
7. Как создать новый файл с помощью *RandomAccessFile*?
8. Какие существуют методы для работы со строками?
9. Как работает метод *equals()* и почему необходимо использовать его при работе со строками?
10. Как преобразовать строку *String* в массив *String[]*?