

ЛАБОРАТОРНАЯ РАБОТА № 11

«Работа с текстовыми файлами. Файлы с произвольным доступом и файлы для записи объектов»

Цель: Получение навыков ввода/вывода данных файла через символьные потоки. Получение навыков работы с файлами с произвольным доступом, содержащими данные какого-то одного примитивного типа или данные разных примитивных типов. Знакомство с механизмом сериализации и десериализации объектов собственных разработанных классов.

Учебные вопросы:

1. Классы иерархии символьных потоков;
2. Посимвольный ввод/вывод;
3. Буферизованный ввод/вывод данных текстового файла;
4. Преобразование байтовых потоков в символьные;
5. Использование класса `PrintWriter`;
6. Файлы с произвольным доступом;
7. Понятия сериализации и десериализации;
8. Пример записи объектов в файл и чтения из файла;
9. Задания для самостоятельной работы;
10. Описание результата выполнения лабораторной работы.

1. Классы иерархии символьных потоков

Потоки, ориентированные на байтовые последовательности, неудобны для обработки информации, записанной в формате **Unicode**, где один символ записан двумя байтами. Именно с учетом этой особенности для обработки символов используются классы, выделенные в отдельную иерархию, – это **Reader** и **Writer** (Рис. 1). Их основные классы-наследники описаны в табл. 1.

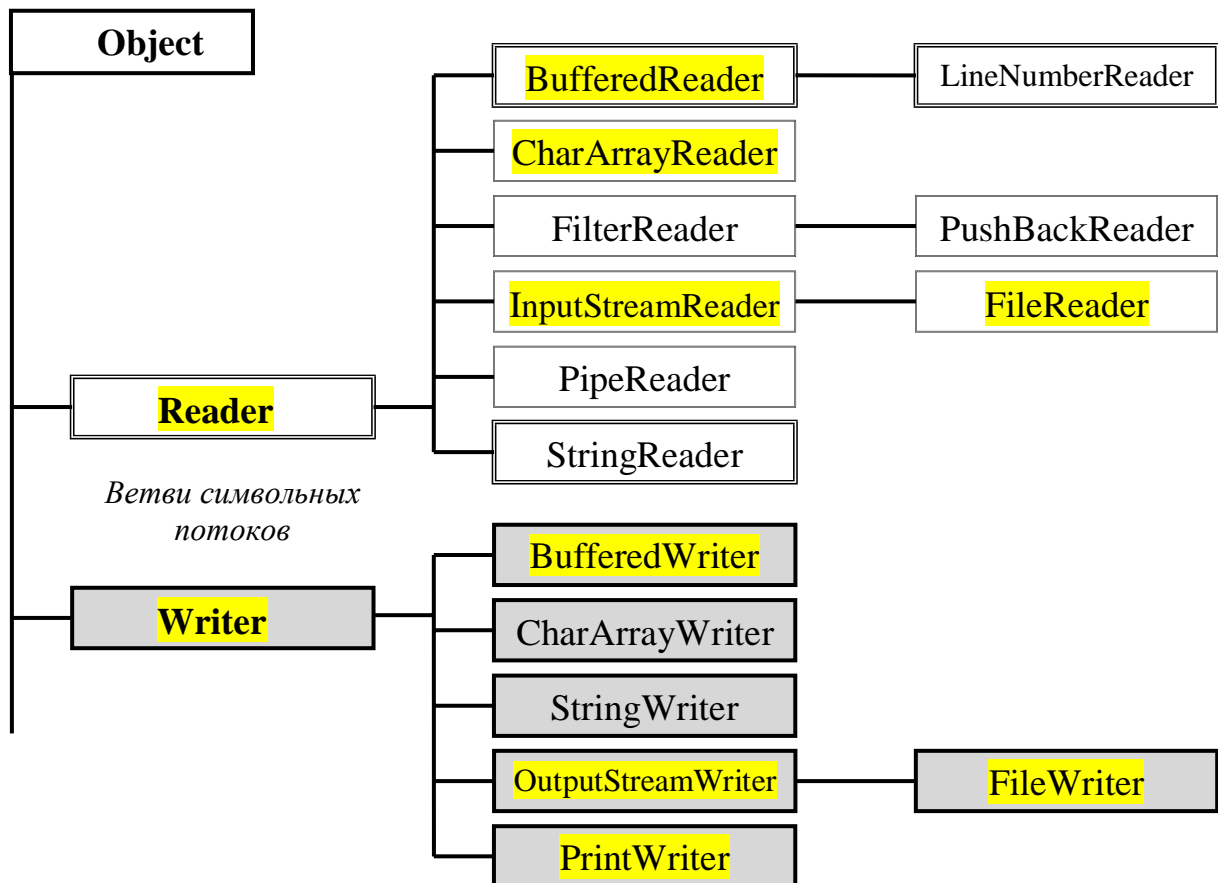


Рис. 1

Таблица 1

Символьные классы	Назначение
BufferedReader/ BufferedWriter	<p>Буферизированные символьные потоки ввода/вывода, позволяющие обрабатывать целые строки. Имеют методы: readLine()/writeLine()</p> <p>Принимают в качестве параметров: символьный поток и размер буфера.</p> <p>Если буфер не указан – используется буфер по умолчанию 8192 байта</p>
FileReader / FileWriter	<p>Поток ввода/вывода символов файла.</p> <p>Принимают в качестве параметров: файл или путь к файлу.</p> <p>Кроме того, есть конструктор FileWriter(String fileName, boolean append), где параметр append=true позволяет использовать файл на добавление информации.</p> <p>Требует обязательной обработки исключительной ситуации FileNotFoundException</p>
CharArrayReader/ CharArrayWriter	Потоки ввода/вывода символов, принимающие в качестве параметров массивы символов
StringReader/ StringWriter	Поток ввода/вывода указанных строк. StringReader принимает строку в качестве параметра
InputStreamReader/ OutputStreamWriter	Конвертируют потоки InputStream/OutputStream и их наследников в символьные потоки, позволяя задать нужную кодировку текста
PrintWriter	<p>Поток вывода, который поддерживает методы print() и println(). Работает быстрее, чем System.out.println().</p> <p>Конструкторы класса:</p> <p style="text-align: right;">77</p> <p>PrintWriter(OutputStream out);</p> <p>PrintWriter(OutputStream out, boolean autoFlush);</p> <p>Второй конструктор создает поток с автоматическим сбросом (очисткой) буфера для параметра autoFlush=true</p>

Следует помнить, что экземпляры абстрактных классов **Reader** и **Writer** создавать нельзя, а можно только объявлять переменные данного типа. Создаются экземпляры классов, ориентированных на определенные источники информации (файлы, массивы символов и т.д.).

Из классов библиотеки ввода/вывода, реализующих потоки, можно составить множество разнообразных конфигураций. Далее приведены примеры наиболее употребляемых конструкций (типичных сочетаний классов) для символьных потоков, встречающихся при решении практических задач. В примерах используется упрощенная обработка исключений.

Все потоки после обработки следует закрывать методом **close()**.

2. Посимвольный ввод/вывод

Самая простая, но неэффективная (медленная) реализация чтения/записи данных файла может быть продемонстрирована с использованием классов **Reader – FileReader**, **Writer – FileWriter**.

Основные методы:

- **read()** для чтения;
- **write()** для записи с удалением ранее имеющейся информации в файле;
- **append()** для добавления данных в файл (существующая ранее информация в файле не уничтожается).

Пример 1. Чтение из одного файла и запись в другой файл данных посимвольно.

```
public class File_RW_byByte {  
  
    public static void main(String[] args) throws IOException {  
        Reader in=null;           // можно сразу записать FileReader in=null;  
        Writer out=null;          // можно сразу записать FileWriter out =null;  
        try {  
            in = new FileReader("E:\\MyFile1.txt");           // файл для чтения  
            out= new FileWriter("E:\\MyFile2.txt", true); // файл для записи  
        }  
    }  
}
```

Метод main генерирует исключение
↓
разрешено добавление

// Данные считываются и записываются побайтно, как и для

// InputStream/OutputStream

```
int oneByte;           // переменная, в которую считываются данные
while ((oneByte = in.read()) != -1) {
    // out.write((char)oneByte);      // запись с уничтожением ранее
                                     // существующих данных
    out.append((char)oneByte);      // запись с добавлением данных в конец
    System.out.print((char)oneByte);
}
} catch (IOException e) {
    System.out.println("Ошибка!!!! ");
}
finally{
    in.close();
    out.close();
} }
```

3. Буферизованный ввод/вывод данных текстового файла

Значительно ускорить процесс чтения/записи помогает буферизация ввода/вывода, для этого полученная ссылка на экземпляр класса **FileReader/FileWriter** передается в качестве параметра в конструкторы класса **BufferedReader/BufferedWriter**.

Ниже приведены примеры записи возможных конструкций при создании объектов.

Размер буфера по умолчанию 8192 байта

`BufferedReader br1 = new BufferedReader(new FileReader("E:\\ File1.txt"));`

`BufferedReader br1 = new BufferedReader(new FileReader("E:\\ File1.txt"), 1024);`

`BufferedWriter out = new BufferedWriter(new FileWriter("E:\\ File2.txt"));`

Недостаток вышеприведенных конструкций — нет возможности управлять кодировкой.

Класс **BufferedReader** позволяет использовать метод **readLine()** для чтения отдельных строк. При достижении конца файла метод **readLine()** возвращает ссылку **null**.

Метод **writeLine()** класса **BufferedWriter** позволяет производить построчную запись.

При буферизированной записи данных в файл перед закрытием потока следует выполнять операцию очистки (сбрасыванием) буфера с дописыванием данных на диск **flush()**.

Пример 2. Чтение из одного файла и запись в другой файл данных построчно с использованием буфера в 1 килобайт.

```
public class Buf_RW_3 {
    public static void main(String[] args) throws IOException{
        BufferedReader br = null;
        BufferedWriter out=null;
        try {
            // Создание файловых символьных потоков для чтения и записи
            br = new BufferedReader( new FileReader("E:\\MyFile1.txt" ), 1024);
            out = new BufferedWriter( new FileWriter( "E:\\MyFile2.txt" ));
            // ← размер буфера

            int lineCount = 0;          // счетчик строк
            String s;
            // Переписывание информации из одного файла в другой
            while ((s = br.readLine()) != null) {
                lineCount++;
                System.out.println(lineCount + ": " + s);
                out.write(s);
                out.newLine();          // переход на новую строку
            }

        } catch (IOException e) {
            System.out.println("Ошибка !!!!!!!!!");
        }
        finally{
            br.close();
            out.flush();
            out.close();
        }
    }
}
```

4. Преобразование байтовых потоков в символные

В некоторых ситуациях для решения задачи используются как «байтовые», так и «символьные» классы. Для этого в библиотеке имеются классы-адаптеры: ***InputStreamReader*** – конвертирует ***InputStream*** в ***Reader***, а ***OutputStreamWriter*** – трансформирует ***OutputStream*** в ***Writer***. При этом возможна передача в качестве второго параметра нужной кодовой страницы, позволяющей выводить текст в надлежащем виде.

Конструкторы с кодировкой:

InputStreamReader(<поток для чтения>, "<кодировка>");

OutputStreamWriter(<поток для записи>, "<кодировка>").


Пример 3. Прочитать и вывести на экран информацию из трех источников: файла на диске, интернет-страницы и массива данных типа **byte**. Указать кодировку, поддерживающую кириллицу. (Сравнить с работой программы, приведенной в примере 2 в лабораторной работе №10 (1))

```
public class InConvertInText {  
  
    public static void readAllByByte( Reader in)  throws IOException {  
        while (true) {  
            int oneByte = in.read();    // читает 1 байт  
            if (oneByte != -1) {        // признак конца файла  
                System.out.print((char) oneByte);  
            } else {  
                System.out.print("\n" + " конец ");  
                break;  
            }  
        }  
    }  
}
```

```

public static void main(String[] args) {
    try {
        // С потоком связан файл
        InputStream inFile = new FileInputStream("E:\\MyFile1.txt");    // байтовый
                                                                    // поток
        Reader rFile= new InputStreamReader(inFile,"cp1251");        // символьный
                                                                    // поток
        readAllByByte(rFile);
        System.out.print("\n\n\n");
        inFile.close();
        rFile.close();
        // С потоком связана интернет-страница
        InputStream inUrl = new URL("http://google.com").openStream(); // байтовый
                                                                    // поток
        Reader rUrl=new InputStreamReader(inUrl, "cp1251");        // символьный
                                                                    // поток
        readAllByByte(rUrl);
        System.out.print("\n\n\n");
        inUrl.close();
        rUrl.close();
        // С потоком связан массив тупа byte
        InputStream inArray = new ByteArrayInputStream( new byte[] {5, 8, 3, 9, 11});
        Reader rArray=new InputStreamReader(inArray,"cp1251" ); // символьный
                                                                    // поток
        readAllByByte(rArray);
        System.out.print("\n\n\n");
        inArray.close();
        rArray.close();
    } catch (IOException e) {
        System.out.println("Ошибка: "+ e);
    }
}
}

```

передается «русская»
кодировка
 

Наиболее быстро и корректно работают буферизированные символьные потоки, построенные на байтовых потоках. Конструкция строится с использованием трех классов, как показано в нижеприведенном примере.

Пример 4. Чтение из одного файла и запись в другой файл данных построчно с использованием буферизации символьных потоков основанных на байтовых файловых потоках.

```
public class Buf_WR_IO_4 {
    public static void main(String[] args) throws IOException {
        BufferedReader br = null;
        BufferedWriter bw=null;
        try {
            // Создание потоков для чтения и записи с нужной кодировкой
            br = new BufferedReader(
                new InputStreamReader(
                    new FileInputStream("E:\\MyFile1.txt"),"cp1251"));

            bw = new BufferedWriter(
                new OutputStreamWriter(
                    new FileOutputStream("E:\\MyFile2.txt"),"cp1251"));

            // Переписывание информации из одного файла в другой
            int lineCount = 0; // счетчик строк
            String s;
            while ((s = br.readLine()) != null) {
                lineCount++;
                System.out.println(lineCount + ": " + s);
                bw.write(lineCount + ": " + s); // запись без перевода строки
                bw.newLine(); // принудительный переход на новую строку
            }
        } catch (IOException e) {
            System.out.println("Ошибка !!!!!!!");
        }
        finally{
            br.close();
            bw.flush();
            bw.close();
        }
    }
}
```

5. Использование класса **PrintWriter**

Для записи или вывода на консоль очень удобно использовать класс **PrintWriter**. Он дает возможность указать требуемую кодировку, имеет методы **print()** и **println()** для записи строк без перехода на новую строку или с переходом.

PrintWriter позволяет в качестве параметра принимать выходной поток **System.out** и осуществлять вывод на консоль. При этом он работает намного быстрее, чем **System.out.println()**.

Пример 5. Выполнить чтение из одного файла и запись в другой файл с использованием класса **PrintWriter**.

```
public class Buf_RW_2 {  
    public static void main(String[] args) {  
        BufferedReader br = null;  
        PrintWriter out=null;  
        try {  
            // Создание потоков  
            br = new BufferedReader(  
                new InputStreamReader(  
                    new FileInputStream("E:\\MyFile1.txt"),"cp1251"));  
  
            PrintWriter out = new PrintWriter("E:\\MyFile2.txt","cp1251");  
            // Переписывание информации из одного файла в другой  
            int lineCount = 0;  
            String s;  
            while ((s = br.readLine()) != null) {  
                lineCount++;  
                out.println(lineCount + ": " + s);  
            }  
        } catch (IOException e) {  
            System.out.println("Ошибка !!!!!!!!!");    }  
        finally{  
            br.close();  
            out.flush();  
            out.close();  
        }  
    }  
}
```

Представленный ниже фрагмент кода демонстрирует работу **PrintWriter** с системным выходным потоком:

...

```
PrintWriter out = new PrintWriter(System.out);
```

```
int lineCount = 0;
```

```
String s;
```

```
// Вывод информации из файла на монитор
```

```
while ((s = br.readLine()) != null) {
```

```
    lineCount++;
```

```
    out.println(lineCount + ": " + s);
```

```
}
```

...

6. Файлы с произвольным доступом

В языке Java файл с произвольным доступом к данным реализуется с помощью класса **RandomAccessFile**. Работа возможна только с примитивными типами данных. Чтение и запись обеспечивается в любом месте файла. Файл ведет себя подобно большому массиву байтов, хранящихся в файловой системе. Этот воображаемый массив снабжен своего рода курсором, или индексом, который называют указателем позиции в файле. При вводе данных отсчет байтов производится от текущей позиции указателя, и по завершении операции указатель устанавливается на позицию, следующую за последним из введенных или считанных байтов. Один из базовых методов **seek()** позволяет переместиться к требуемой позиции в файле.

При записи или чтении из файла данных известного числового типа всегда можно рассчитать указатель на конкретный элемент (на номер байта). Например, для файла с данными типа `double` (8 байт):

номер элемента 0 – позиция указателя $0 * 8 = 0$;

номер элемента 1 – позиция указателя $1 * 8 = 8$;

номер элемента 2 – позиция указателя $2 * 8 = 16$ и т.д.

Перевод курсора на 2-й элемент и его считывание осуществляется следующими операциями:

```
file.seek(2*8);      // перемещение  
file.readDouble();  // считывание
```

Перевод курсора в конец файла и запись нового числа:

```
file.seek(file.length()); // length() дает позицию конца файла  
file.writeDouble(x);      // x – число для записи
```

В данном примере `file` – переменная типа **RandomAccessFile**, связанная с конкретным файлом на диске.

При перемещении по элементам данных можно выполнять только те операции, которые разрешены для режима работы заданного при создании экземпляра класса **RandomAccessFile**. Это может быть режим чтения («r») или чтения/записи («rw»). Также есть режим «rws», когда файл открывается для операций чтения-записи и каждое изменение данных файла немедленно записывается на физическое устройство.

При использовании **RandomAccessFile** необходимо четко знать структуру файла.

Класс **RandomAccessFile** наследуется напрямую от **Object**, а не от базовых классов байтовых или символьных потоков системы ввода/вывода Java. Но, тем не менее, работа с **RandomAccessFile** напоминает использование совмещенных в одном классе потоков **DataInputStream** и **DataOutputStream**.

Класс **RandomAccessFile** содержит методы для чтения и записи примитивов и строк UTF-8. Наиболее часто используемые методы приведены в табл. 2.

Таблица 2

Название метода	Тип	Выполняемые действия
read() , read(byte b[]) ,	int	Читает один байт или массив байтов
skipBytes(int n)	int	Пропускает <i>n</i> байт. Т.е. перемещает указатель на <i>n</i> байт вперед
write(int b)	void	Пишет один байт в то место, где стоит указатель
write(byte b[])	void	Пишет массив байтов в то место, где стоит указатель
write(byte b[], int off, int len)	void	Пишет массив байтов в то место, где стоит указатель
getFilePointer()	long	Возвращает номер байта, на который указывает «указатель»
seek(long pos)	void	Перемещает указатель, используемый для чтения/записи, в указанное место pos= номер_элемента * размер_элемента_в_байтах, соответствующее его типу данных
length()	long	Возвращает длину файла в байтах
setLength(long newLength)	void	Устанавливает новую длину файла. Если файл был больше – он обрезается, если меньше – расширяется и новое место заполняется нулями
readBoolean() , readByte() , readChar() , readInt() , readLong() , readFloat() , readDouble() , readLine()	-	Читает число/строку/символ соответствующего типа данных с текущей позиции указателя в файле
close()	void	Закрывает файл
writeByte(int v) , writeInt(int v) , writeLong(long v) , writeBytes(String s) , writeChar(int v) , writeUTF(str)	void	Пишет число/строку/символ соответствующего типа данных с текущей позиции указателя в файле

Пример 6. Работа с числовыми данными в файле с произвольным доступом.

Выполнить следующие подзадачи:

- записать в файл заданное количество целых чисел (1 число=4 байта);
- прочитать данные в прямом и обратном порядке;
- получить информацию о файле и указателе до ввода и после ввода данных;
- отсортировать по возрастанию числа непосредственно в файле.

```
public class RandAccNumb{
public static void main(String[] args) {
    try {
        File folder = new File("E:\\My");
        if (!folder.exists())
            folder.mkdir();
        // Создание папки «My» на диске,
        // если она не существует

        File f1 = new File("E:\\My\\num1Mart.txt");
        f1.createNewFile();
        // Создание файла в папке

        Scanner sc = new Scanner(System.in, "cp1251");

        System.out.print("Сколько чисел надо записать в файл? \n => ");
        int count = sc.nextInt();
        // ввести количество чисел

        // Открыть файл одновременно для чтения и записи "rw"
        RandomAccessFile rf = new RandomAccessFile(f1, "rw");
        System.out.println("Исходный размер файла в байтах =" + rf.length()
            + ", указатель стоит на " + rf.getFilePointer() + "-м байте");

        System.out.println("Введите числа:");
        for (int i = 0; i < count; i++) {
            rf.writeInt(sc.nextInt());
            // Записать числа в файл. На каждое
            // число типа int приходится 4 байта
        }
        System.out.println("Новый размер файла в байтах =" + rf.length()
            + ", указатель стоит на " + rf.getFilePointer() + "-м байте");
        System.out.println("Количество байт на 1 число =" + rf.length() / count);
        rf.close();
    }
}
```

```
// Открыть файл только для чтения "r"
```

```
rf = new RandomAccessFile(f1, "r");
```

```
// Прочитать числа из файла и вывести на экран
```

```
System.out.println("\n Числа в файле:");
```

```
for (int i = 0; i < count; i++) { // i – текущий номер числа  
    rf.seek(i * 4); // перевод указателя на текущее число → i*4 байта  
    // в данной ситуации при последовательном считывании  
    // операцию seek() можно было не использовать  
    System.out.println("Число" + i + ": " + rf.readInt());  
}
```

```
System.out.println("Числа в обратном порядке:");
```

```
for (int i = count - 1; i >= 0; i--) {  
    rf.seek(i * 4); // операцию использовать обязательно!  
    System.out.println("Число" + i + ": " + rf.readInt());  
}
```

```
rf.seek(rf.getFilePointer() - 4); // перевод указателя на последнее число
```

```
System.out.println(" Количество чисел в файле= " + rf.length()/4  
    + ", последнее число= " + rf.readInt());
```

```
// Поиск заданного числа в файле и определение его номера (номеров)
```

```
System.out.print("\nВведите число, которое нужно найти в файле => ");
```

```
int x = sc.nextInt();
```

```
int kol = 0; // количество искомых чисел в файле
```

```
for (int i = 0; i < count; i++) {  
    rf.seek(i * 4);  
    int number = rf.readInt();  
    if (number == x) {  
        kol++;  
        System.out.print("номер " + i + ", ");  
    }  
}
```

```
System.out.println(" количество искомых чисел = " + kol);
```

```
rf.close();
```

// СОРТИРОВКА ЧИСЕЛ В ФАЙЛЕ МЕТОДОМ «ПУЗЫРЬКА»

```
rf = new RandomAccessFile(f1, "rw");    // открыт для чтения и записи
```

```
for (int k = 0; k < count; k++) {        // k – номер просмотра
```

```
    for (int i = 0; i < count - k - 1; i++) {    // i – номер числа
```

```
        rf.seek(i * 4);                        // переход к i-тому числу
```

```
        int number1 = rf.readInt();            чтение i-того и
```

```
        int number2 = rf.readInt();            (i+1)-го чисел в переменные
```

```
        if (number1 > number2) {    // условие для сортировки по возрастанию
```

```
            rf.seek(i * 4);        возврат указателя на i-тое число и
```

```
            rf.writeInt(number2);    перестановка (запись чисел в обратном
```

```
            rf.writeInt(number1);    порядке)
```

```
        } } }
```

```
System.out.println("\n Числа, отсортированные в файле:");
```

```
for (int i = 0; i < count; i++) {
```

```
    rf.seek(i * 4);
```

```
    System.out.print("  " + rf.readInt());
```

```
}
```

```
rf.close();
```

```
} catch (IOException e) {
```

```
    System.out.println("End of file " + e);
```

```
}
```

```
}}
```

Далее приведен пример работы со строками в формате UTF-8.

Пример 7. Выполнить запись строк и чтение их из файла с произвольным доступом.

```
public class RandFtxt{
```

```
    public static void main(String[] args) {
```

```
        try{
```

```
            File folder=new File("E:\\My");
```

```
            if (!folder.exists())
```

```
                folder.mkdir();
```

```
            File f1=new File("E:\\My\\strokiRand.txt");
```

```
            f1.createNewFile();
```



```
Scanner sc = new Scanner(System.in, "cp1251");
System.out.print("Сколько строк надо записать в файл? \n =>");
int count = sc.nextInt();
sc.nextLine();           // очистка буфера после ввода числа
```

```
RandomAccessFile rf = new RandomAccessFile(f1, "rw"); // чтение/запись
rf.setLength(0);
long len=rf.length();
System.out.println("Открыт файл размером "+len+ " байт");
System.out.println("Введите строки:");
int kol=0;               // счетчик букв
```

// Записать строки в файл

```
for (int i = 0; i < count; i++) {
    String s=sc.nextLine();
    rf.writeUTF(s);
    kol+=s.length();
}
len=rf.length();
```

При вводе строк символ абзаца (нажатие Enter) имеет такой же размер в байтах, как и другие символы, и учитывается при вычислении размера файла

```
System.out.println("Размер файла в байтах = "+len);
rf.close();
```

```
//      Открыть файл для чтения "r"
rf = new RandomAccessFile(f1, "r");
```

// Вывод строк из файла на экран

```
System.out.println("Строки из файла:");
rf.seek(0);           // перевести указатель в начало файла (на первое слово)
for (int i = 0; i < count; i++)
    System.out.println("Строка " + i + " начинается с байта "
        + rf.getFilePointer() + " - " + rf.readUTF() + " -
        заканчивается байтом "+ (rf.getFilePointer()-1));
```

```
rf.close();
catch(IOException e){
    System.out.println("End of file " +e);
}
```

```
}}
```

Как указывалось ранее, файлы с прямым доступом позволяют хранить данные примитивных типов и строки в формате UTF-8. Информацию об объектах сложной структуры также можно хранить в таких файлах, если объект не записывать целиком, т.е. не использовать ссылочный тип данных. Информацию об объекте следует записывать (считывать) в виде одинаковой последовательности значений его полей примитивного типа, причем в ряде случаев с выравниванием размера до одинакового количества байтов. Структура такого файла должна быть хорошо продумана. Надо четко знать, как попасть на указатель, соответствующий любому объекту и любому полю. У разных объектов текстовые поля (например, фамилии у сотрудников) будут иметь разную длину и, соответственно, кодироваться разным количеством байтов. Чтобы объекты были одного размера, можно воспользоваться таким приемом, как дописывание после строки UTF-8 недостающего количества байтов при переходе к следующему параметру.

Пример 8. Записать в файл с произвольным доступом данные о заданном количестве сотрудников и считать информацию из файла.

Данные о сотрудниках:

- фамилия, должность – записывается в формате строки UTF-8;
- оклад – число типа `int`.

Один символ в формате UTF – 1 байт, число типа `int` – 4 байта.

Разное количество байтов для фамилии и должности дополним с помощью «дозаписи» любых чисел типа `byte` до общего размера, например, 20. Значение выбрано из расчета того, что фамилия и должность будут иметь длину не более 20 символов.

```
public class RandAccF_record {  
    public static void main(String[] args) {  
        try {  
            File folder = new File("E:\\My");  
            if (!folder.exists())  
                folder.mkdir();  
        }  
    }  
}
```

```

File f1 = new File("E:\\My\\rec_RAF.txt");
if (!f1.exists())
    f1.createNewFile();

RandomAccessFile rf = new RandomAccessFile(f1,"rw"); // чтение и запись
long fSize = rf.length(); // размер файла
Scanner sc = new Scanner(System.in, "cp1251");
System.out.print("Введите количество сотрудников для записи в файл\n"
    + "=> ");
int kol = sc.nextInt();
sc.nextLine(); // очистка буфера после ввода числа

String fam, doljnost;
int oklad;

```

//----ЗАПИСЬ ИНФОРМАЦИИ О СОТРУДНИКАХ В ФАЙЛ----

```

for (int i = 0; i < kol; i++) {
    System.out.print("Введите фамилию " + (i + 1) + " сотрудника => ");
    fam = sc.next();
    rf.seek(rf.length()); // поиск конца файла
    rf.writeUTF(fam); // запись фамилии
    for (int j = 0; j < 20 - fam.length(); j++)
        rf.writeByte(1); // добавление байтов до 20-ти любой цифрой (=1)

    System.out.print("Введите его должность => ");
    doljnost = sc.next();
    rf.writeUTF(doljnost); // запись должности
    for (int j = 0; j < 20 - doljnost.length(); j++)
        rf.writeByte(1); // добавление байтов до кол=20 любым числом

    System.out.print("Введите его оклад => ");
    oklad = sc.nextInt();
    sc.nextLine(); // очистка буфера
    rf.writeInt(oklad); // запись оклада
}
rf.close();

```

//----ЧТЕНИЕ ИНФОРМАЦИИ О СОТРУДНИКАХ ИЗ ФАЙЛА----

```
rf = new RandomAccessFile(f1, "r");
rf.seek(0); // перемещение в начало файла
System.out.println("Информация о сотрудниках");
System.out.println("Фамилия \t\t Должность \t\t Оклад");
for (int i = 0; i < kol; i++) {
    fam = rf.readUTF();
    for (int j = 0; j < 20 - fam.length(); j++)
        rf.readByte();

    doljnost = rf.readUTF();
    for (int j = 0; j < 20 - doljnost.length(); j++)
        rf.readByte();

    oklad = rf.readInt();

    System.out.println(fam + "\t\t" + doljnost + "\t\t" + oklad);
}
rf.close();
} catch (IOException e) {
    System.out.println("End of file " + e);
}}}
```

Работа с числовыми данными известного типа всегда позволяет точно просчитать положение числа (указатель на него) в файле. Подобная работа со строками не всегда является удобной. Часто происходит запись дополнительной символьной информации в файл, и найти адрес нужного объекта простыми расчетами не представляется возможным. Для работы с объектами как со структурами данных рекомендуется воспользоваться другими классами и сериализацией объектов.

7. Понятия сериализации и десериализации

Записать в файл или прочитать из файла информацию об объектах, в том числе объектах собственных классов сложнее, чем данных примитивных и символьных типов. Для этой цели используются классы ***ObjectOutputStream*** (наследник ***OutputStream***), ***ObjectInputStream*** (наследник ***InputStream***) и механизм сериализации.

Сериализация – это процесс сохранения состояния объекта в последовательность байтов; **десериализация** – процесс восстановления объекта из этих байтов.

Для того чтобы иметь возможность сохранить объект в байтовый поток, необходимо чтобы класс, на базе которого объект создан, реализовал стандартный интерфейс *java.io.Serializable*.

Пример записи класса Student, реализующего интерфейс Serializable:

```
class Student implements Serializable {  
    String firstName;           // имя  
    String lastName;           // фамилия  
    float age;                  // возраст  
}
```

Будучи совершенно пустым, интерфейс *Serializable* является лишь *маркерным интерфейсом* – он позволяет среде определить, может ли экземпляр данного класса быть сохранен в байтовый поток и восстановлен из него. Следует отметить, что после сериализации объект может быть не только сохранен в файл на диске, но может, например, передаваться по сети и т.д., а потом быть восстановлен.

8. Пример записи объектов в файл и чтения из файла

Алгоритмы записи и чтения объекта аналогичны алгоритмам работы с примитивными типами данных через байтовые потоки.

Пример 9. Ввести с клавиатуры информацию о стране вида: **название, площадь**. Записать ее в файл, а затем прочитать.

```
package recordCountryFile;  
import java.util.Scanner;  
  
class Strana implements Serializable {  
    String name;                // название страны  
    double square;              // площадь страны  
}
```

```

public class Sereliz_primer{
    public static void main(String[] args) throws IOException, ClassNotFoundException{
        Scanner sc=new Scanner(System.in,"cp1251");
        // создается файл на диске
        File f=new File("E:\\MyFileSer");
        f.createNewFile();

        // -----ЗАПИСЬ ОБЪЕКТА В ФАЙЛ-----
        // Создается поток для записи объекта
        FileOutputStream fos = new FileOutputStream(f);
        ObjectOutputStream oos = new ObjectOutputStream(fos);

        // Вводится информация об объекте (стране)
        Strana strana = new Strana();
        System.out.println("Введите информацию о стране: ");
        System.out.print("Название страны => ");
        strana.name=sc.nextLine();
        System.out.print("Площадь страны => ");
        strana.square=sc.nextDouble();

        // Объект записывается в файл
        oos.writeObject(strana);

        // Дописывается информация и закрывается файловый поток
        oos.flush();
        oos.close();

        // -----ЧТЕНИЕ ОБЪЕКТА ИЗ ФАЙЛА-----
        // Создается поток для чтения объекта из файла
        FileInputStream fis = new FileInputStream(f);
        ObjectInputStream oin = new ObjectInputStream(fis);

        // Объект считывается, и на экран выводится требуемая информация
        strana = (Strana) oin.readObject();
        System.out.println("        Название страны "+
            strana.name); System.out.println(" ее площадь = "+
            strana.square);

        // Поток закрывается
        oos.close();
    }
}

```

9. Задания для самостоятельной работы

Задание 1. В отдельных проектах выполнить примеры 1 – 7 (8 и 9 не надо). Протестировать программы с помощью отладчика. Выявить различие в работе программ в примерах 2 и 3. Заменить тип данных переменных `int` на другие числовые типы по выбору и ознакомиться с методами их чтения/записи и определения положения указателя.

Задание 2. Создать проект, позволяющий из одного текстового файла, содержащего несколько строк (тип `String`) заранее подготовленного текста на русском языке (обратитесь к классике), построчно переписать в другой текстовый файл слова, отвечающие условию.

Условие:

Переписать в результирующий файл слова, которые начинаются с той же буквы, что и первое слово.

Требования:

- слова из предложения выделять методом `split()`;
- в новом файле следует указать номер строки, в которой искомые слова находились в исходном файле;
- для каждой строки указать количество выбранных слов.
-

Задание 3. Используя рассмотренные в данной работе примеры 8 и 9, выполнить задание согласно условию в виде двух проектов:

1-й проект – работа с файлом с произвольным доступом;

2-й проект – работа через сериализацию.

Условие:

Записать в исходный файл информацию о фильмах: Название_фильма, год_выпуска, страна, жанр, стоимость_проката
Количество фильмов задать с клавиатуры.

Создать программным способом другой файл и переписать в него информацию о фильмах, выпущенных в России.

10. Описание результата выполнения лабораторной работы

В отчете по лабораторной работе должны быть представлены все примеры и задания.

Лабораторная работа принимается при наличии отчета всех выполненных заданий.

Структура отчета по лабораторной работе:

1. Титульный лист;
2. Цель работы;
3. Описание задачи;
4. Ход выполнения (содержит код программы);
5. Вывод;

Оформление:

- а) шрифт Times New Roman;
- б) размер шрифта 12 или 14;
- в) межстрочный интервал 1,5.

Отчет выполняется индивидуально и направляется по адресу электронной почты proverkalab@yandex.ru. В теле письма необходимо указать ФИО студента и номер группы.