



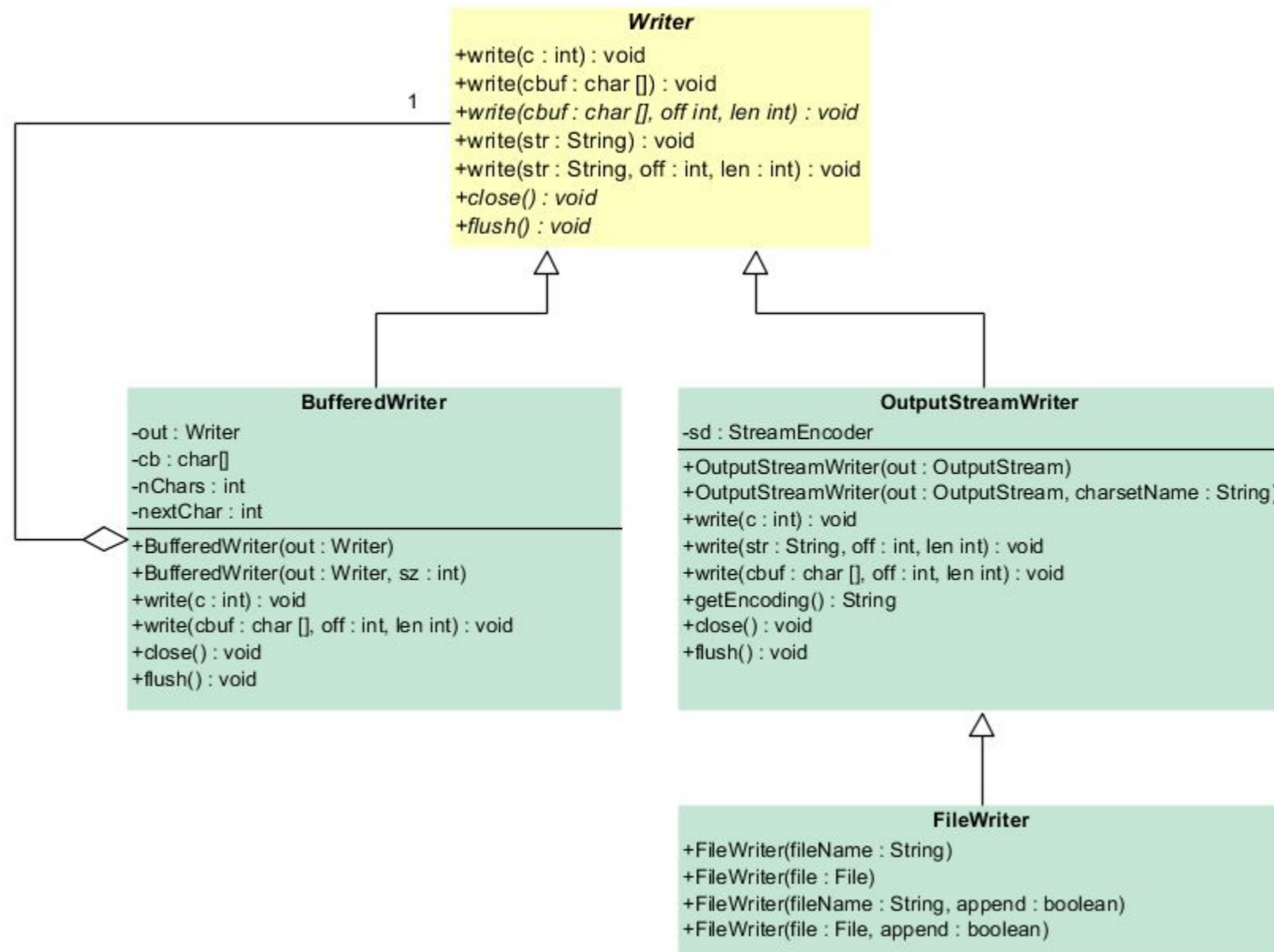
V. Ввод - вывод

3. Символьные потоки



Объект из которого можно прочитывать последовательность char называется символьный поток ввода. Объект в который можно записать последовательность char называется символьный поток вывода. Классы символьных потоков ввода являются подклассами абстрактного класса Reader, а потоков вывода – подклассами абстрактного класса Writer. Классы символьных потоков находятся в пакете java.io.

Потоки вывода



Шаблон Декоратор: Иерархия классов СИМВОЛЬНЫХ ПОТОКОВ ВЫВОДА является примером применения шаблона Декоратор.

```
public abstract class Writer {

    public void write(int c) throws IOException {
        synchronized (lock) {
            if (writeBuffer == null){
                writeBuffer = new char[writeBufferSize];
            }
            writeBuffer[0] = (char) c;
            write(writeBuffer, 0, 1);
        }
    }

    public void write(char cbuf[]) throws IOException
    abstract public void write(char cbuf[], int off, int len) throws IOException;

    public void write(String str) throws IOException
    public void write(String str, int off, int len) throws IOException

    abstract public void flush() throws IOException;
    abstract public void close() throws IOException;
}
```



Абстрактный класс `Writer` – базовый класс для символьных потоков вывода. Для вывода диапазона элементов массива `char` в нём объявлен абстрактный метод `write`. Конкретные классы потомки должны переопределять этот метод. Как правило потомки переопределяют и другие методы `write` более эффективными реализациями. Класс содержит абстрактные методы `close` и `flush`. Метод `close` предназначен для закрытия потока и освобождения неуправляемых ресурсов после окончания записи. Метод `flush` предназначен для опустошения буфера если поток буферизованный. Как правило реализации `close` вызывают `flush`.

```
public class OutputStreamWriter extends Writer {  
  
    private final StreamEncoder se;  
  
    public OutputStreamWriter( OutputStream out )  
    public OutputStreamWriter( OutputStream out , String charsetName )  
  
    public void write(int c) throws IOException  
    public void write(char cbuf[], int off, int len) throws IOException  
    public void write(String str, int off, int len) throws IOException  
  
    public String getEncoding()  
  
    public void flush()  
    public void close()  
}
```



Класс OutputStreamWriter предназначен для превращения символов в байты с помощью кодировки и записи полученных байтов в байтовый поток вывода. Кодировку можно задать в конструкторе. Если кодировка не задаётся используется кодировка по умолчанию.

```
public class FileWriter extends OutputStreamWriter {  
  
    public FileWriter(File file) throws IOException {  
        super(new FileOutputStream(file));  
    }  
  
    public FileWriter(File file, boolean append) throws IOException {  
        super(new FileOutputStream(file, append));  
    }  
}
```



Класс `FileWriter` предназначен для записи последовательности `char` в файл. Объект класса `FileOutputStream` это `OutputStreamWriter` для записи в байтовый файловый поток вывода. При использовании `FileOutputStream` всегда используется кодировка по умолчанию.

Запись в файл по одному символу

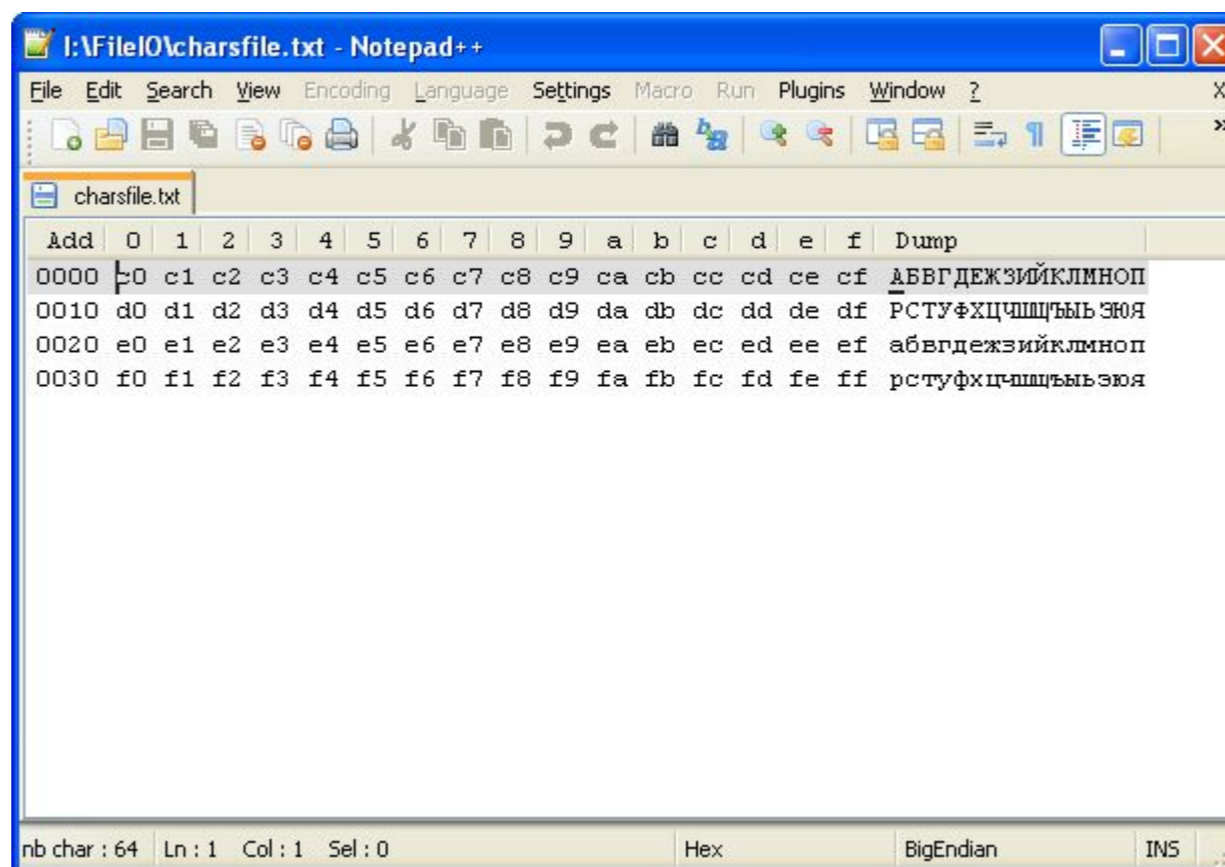
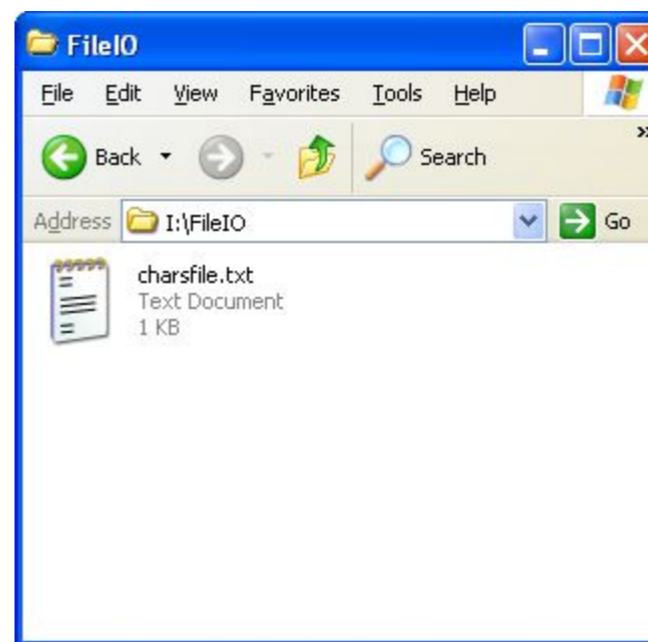
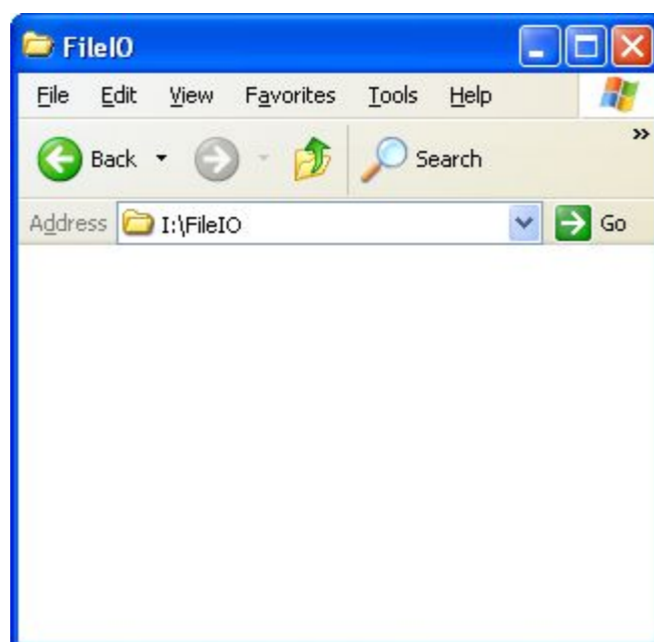
```
public class WriteCharDemo {  
  
    public static void main(String[] args) {  
  
        FileWriter out = null;  
  
        try {  
            out = new FileWriter("I:\\FileIO\\charfile.txt");  
            for (int i = 0x0410; i < 0x0450; i++) {  
                System.out.print((char)i);  
                out.write(i);  
            }  
        } catch (IOException e) {  
            System.out.println("An I/O error occurred");  
        } finally {  
            try {  
                if (out != null)  
                    out.close();  
            } catch (IOException e) {  
                System.out.println("Error closing file");  
            }  
        }  
    }  
}
```

АБВГДЕЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯабвгдежзийклмнопрстуфхцчшщъыьэюя

```
public class WriteCharsDemo {  
  
    public static void main(String[] args) {  
  
        FileWriter out = null;  
        char[] chars = new char[64];  
  
        for (int i = 0; i < 64; i++) {  
            chars[i] = (char) (0x0410 + i);  
        }  
  
        System.out.println(Arrays.toString(chars));  
  
        try {  
            out = new FileWriter("I:\\FileIO\\charsfile.txt");  
            out.write(chars);  
        } catch (IOException e) {  
            System.out.println("An I/O error occurred");  
        } finally {  
            try {  
                if (out != null)  
                    out.close();  
            } catch (IOException e) {  
                System.out.println("Error closing file");  
            }  
        }  
    }  
}
```

[А, Б, В, Г, Д, Е, Ж, З, И, Й, ..., Щ, Ъ, Ы, Ь, Э, Ю, Я, а, б, в, г, д, е, ж, з, и, й, ..., щ, ъ, ы, ь, э, ю, я]

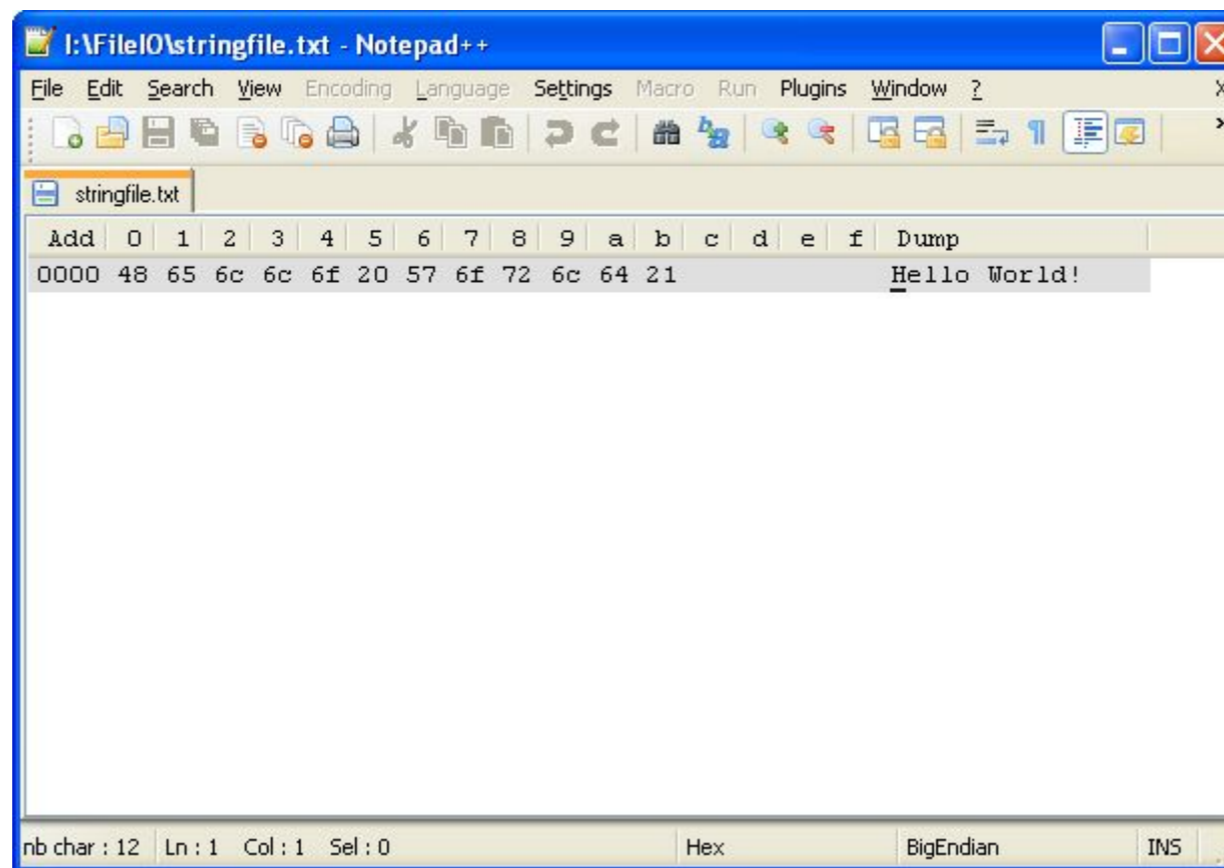
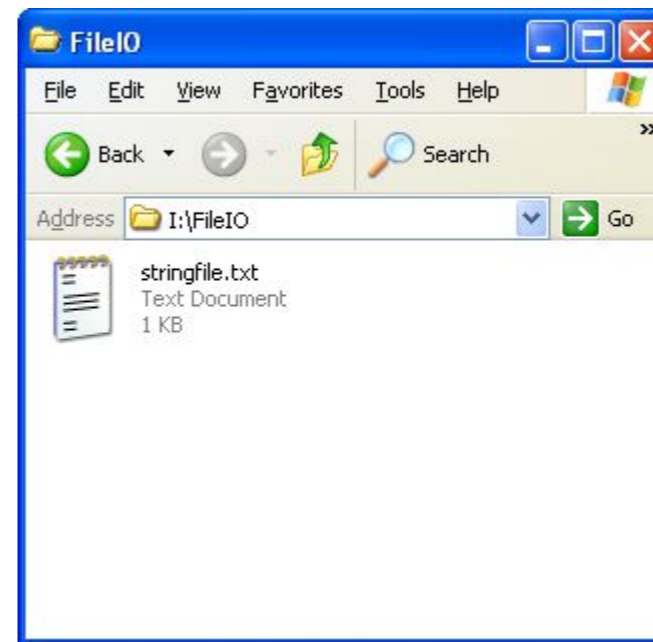
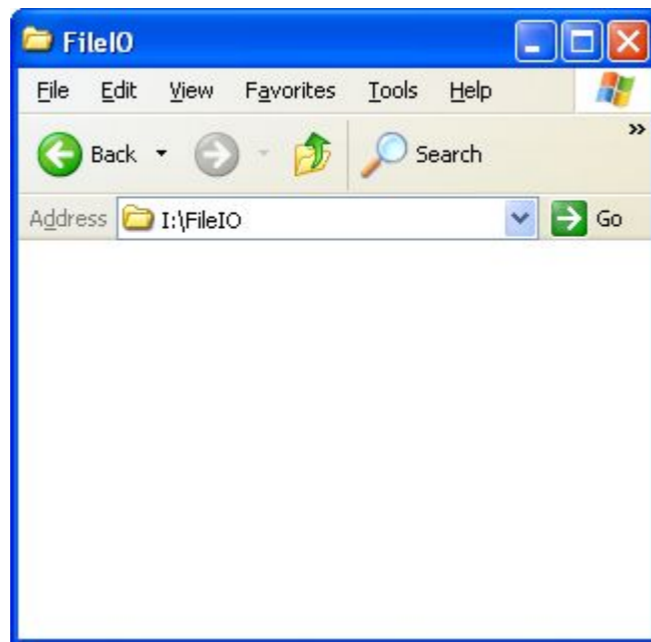
Запись в файл массива СИМВОЛОВ



```
public class WriteStringDemo {  
  
    public static void main(String[] args) throws IOException {  
  
        String source = "Hello World!";  
        FileWriter out = null;  
  
        System.out.println("String to write: " + source );  
        System.out.println("Default charset (encoding): " + Charset.defaultCharset());  
        System.out.println("String bytes using default encoding: " + Arrays.toString(source.getBytes()));  
  
        try {  
            out = new FileWriter("I:\\FileIO\\stringfile.dat");  
            out.write(source);  
        } catch (IOException e) {  
            System.out.println("An I/O error occurred");  
        } finally {  
            try {  
                if (out != null)  
                    out.close();  
            } catch (IOException e) {  
                System.out.println("Error closing file");  
            }  
        }  
    }  
}
```

```
String to write: Hello World!  
Default charset (encoding): windows-1251  
String bytes using default encoding: [72, 101, 108, 108, 111, 32, 87, 111, 114, 108, 100, 33]
```

Запись строки в файл



Запись строки в файл в кодировке UTF-16

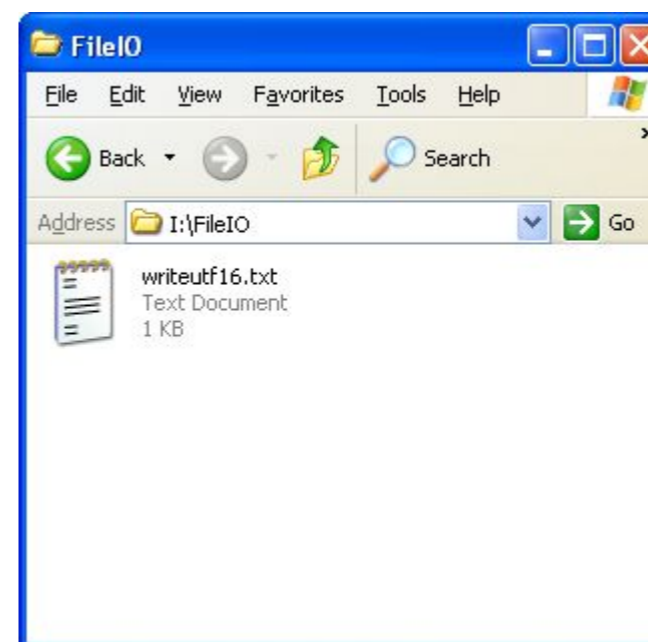
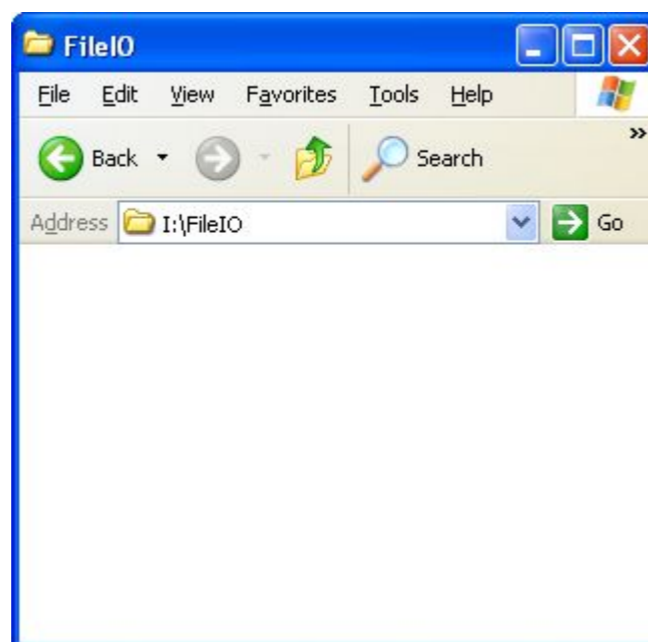
```
public class WriteEncodingDemo {  
  
    public static void main(String[] args) throws UnsupportedEncodingException{  
  
        String source = "Hello World!";  
        Writer out = null;  
  
        System.out.println("String to write: " + source );  
        System.out.println("String bytes using UTF-16: " + Arrays.toString(source.getBytes("UTF-16")));  
  
        try {  
  
            FileOutputStream os = new FileOutputStream("I:\\FileIO\\WriteString.txt");  
            out = new OutputStreamWriter(os, "UTF-16");  
  
            out.write(source);  
        } catch (IOException e) {  
            System.out.println("An I/O error occurred");  
        } finally {  
            try {  
                if (out != null)  
                    out.close();  
            } catch (IOException e) {  
                System.out.println("Error closing file");  
            }  
        }  
    }  
}
```


String to write: Hello World!

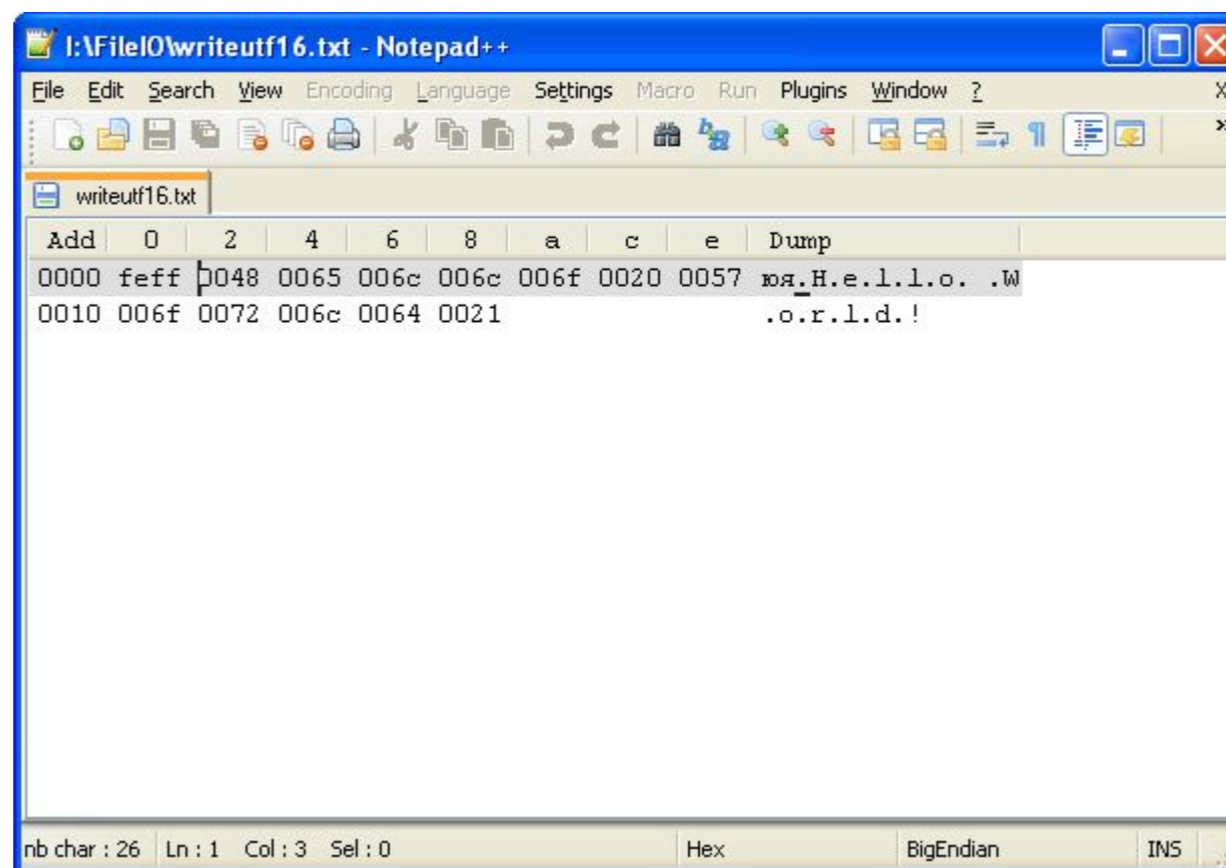
String bytes using UTF-16:

[-2, -1, 0, 72, 0, 101, 0, 108, 0, 108, 0, 111, 0, 32, 0, 87, 0, 111, 0, 114, 0, 108, 0, 100, 0, 33]

Запись строки в файл в кодировке UTF-16



 При кодировании строки в UTF-16 добавляется BOM.



Буферизованный вывод


```
public class BufferedWriter extends Writer {  
  
    private Writer out;  
  
    private char cb[];  
    private int nChars, nextChar;  
    private static int defaultCharBufferSize = 8192;  
  
    public BufferedWriter(Writer out)  
    public BufferedWriter(Writer out, int sz)  
  
    public void write(int c) throws IOException  
    public void write(char cbuf[], int off, int len) throws IOException  
    public void write(String s, int off, int len) throws IOException  
  
    public void flush() throws IOException {  
        ...  
        out.flush();  
        ...  
    }  
  
    public void close() throws IOException {  
        ...  
        out.close();  
        ...  
    }  
}
```

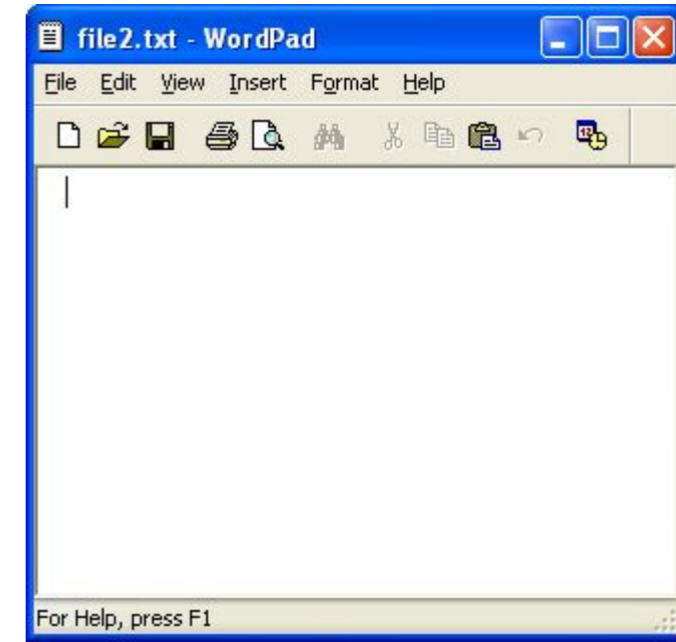
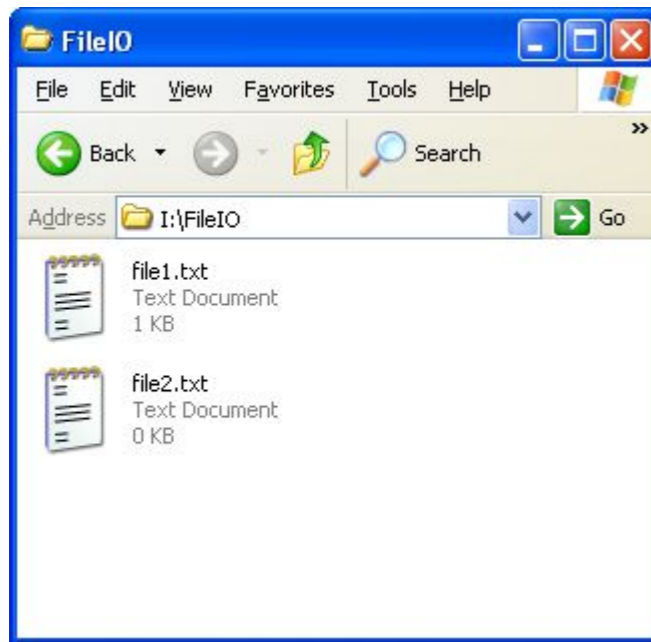


При использовании буферизованного потока вывода данные могут накапливаться в буфере и выводиться при наполнении буфера. Таким образом можно снизить количество операций записи в обрабатываемый поток и повысить эффективность.

```
public class WriteFlushDemo {  
  
    public static void main(String[] args) throws IOException {  
  
        String source = "Hello World!";  
  
        BufferedWriter out1 = null;  
        BufferedWriter out2 = null;  
  
        System.out.println("String to write: " + source );  
  
        try {  
            out1 = new BufferedWriter(new FileWriter("I:\\FileIO\\file1.txt"));  
            out2 = new BufferedWriter(new FileWriter("I:\\FileIO\\file2.txt"));  
  
            out1.write(source);  
            out2.write(source);  
  
        } catch (IOException e) {  
            System.out.println("An I/O error occurred");  
        } finally {  
            try {  
                if (out1 != null)  
                    out1.close();  
            } catch (IOException e) {  
                System.out.println("Error closing file");  
            }  
        }  
    }  
}
```

String to write: Hello World!

Опустошение буфера



```
public class WriteBufPerform {

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

        BufferedWriter outbuf = null;
        FileWriter out = null;

        long time = System.currentTimeMillis();
        try {
            outbuf = new BufferedWriter(new FileWriter(
                "I:\\FileIO\\outbuf.txt"));

            for (int i = 0; i < 10000000; i++) {
                outbuf.write(65);
            }
        } catch (IOException e) {
            System.out.println("An I/O error occurred");
        } finally {
            try {
                if (outbuf != null)
                    outbuf.close();
            } catch (Exception e) {
                System.out.println("Error closing file");
            }
        }
        time = System.currentTimeMillis() - time;
        System.out.println("Buffered output time: " + time);

        ...
    }
}
```

```
public class WriteBufPerform {  
  
    public static void main(String[] args) throws IOException {  
  
        ...  
  
        time = System.currentTimeMillis();  
        try {  
            out = new FileWriter("I:\\FileIO\\outnobuf.txt");  
  
            for (int i = 0; i < 10000000; i++) {  
                out.write(65);  
            }  
        } catch (IOException e) {  
            System.out.println("An I/O error occurred");  
        } finally {  
            try {  
                if (out != null)  
                    out.close();  
            } catch (IOException e) {  
                System.out.println("Error closing file");  
            }  
        }  
        time = System.currentTimeMillis() - time;  
        System.out.println("Non-buffered output time: " + time);  
    }  
}
```


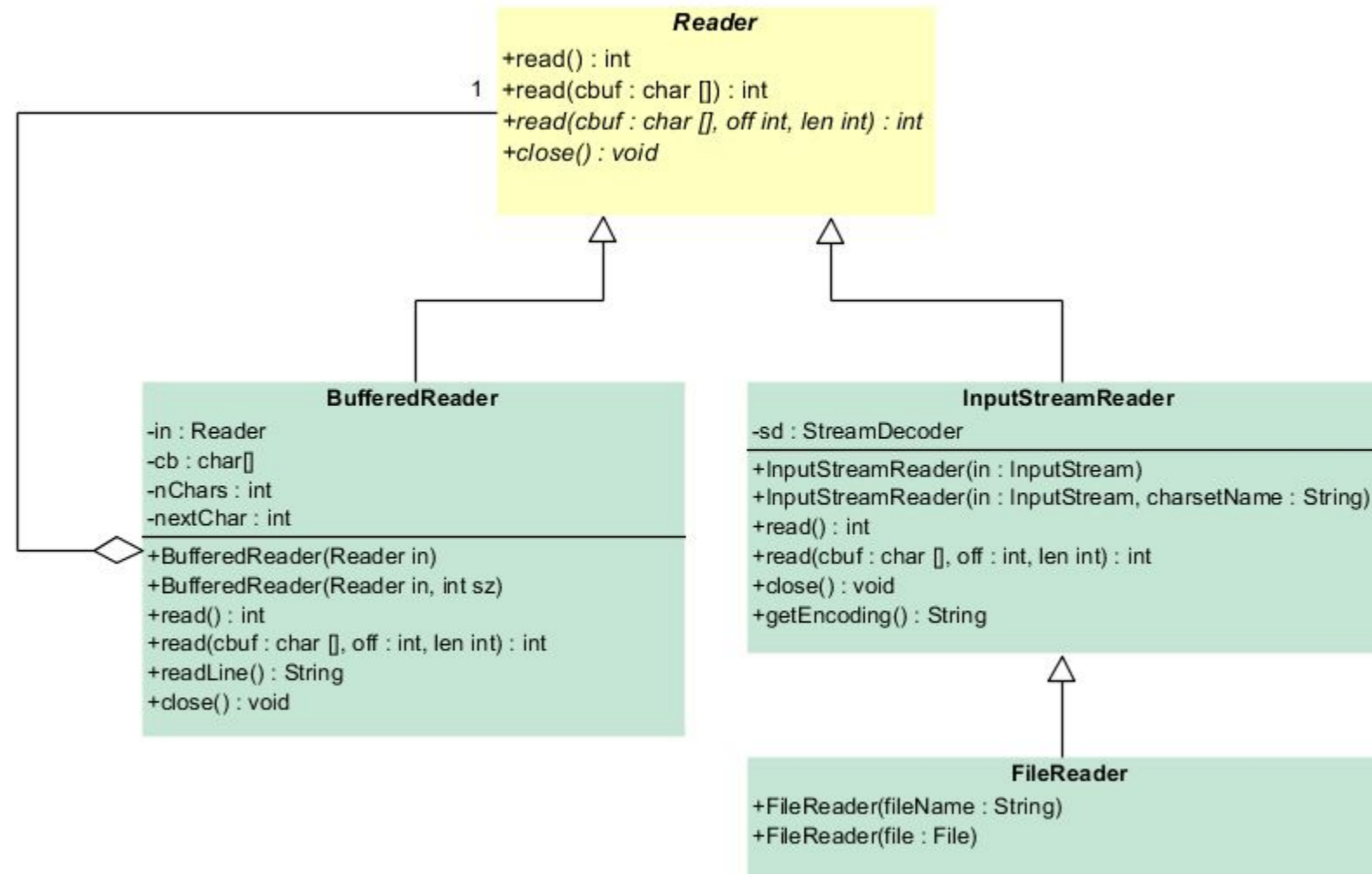
```
Buffered output time: 516  
Non-buffered output time: 1469
```

Почему такой маленький выигрыш?



Для байтовых потоков благодаря использованию буфера был получен выигрыш в 100 раз. При использовании аналогичного буфера для символьных потоков выигрыш был в 3 раза. Дело в том, что символьные потоки уже буферизованы. В классе StreamEncoder есть буфер.

Потоки ввода



Шаблон Декоратор: Иерархия классов СИМВОЛЬНЫХ ПОТОКОВ ВВОДА является примером применения шаблона Декоратор.


```
public abstract class Reader {  
  
    public int read() throws IOException {  
  
        char cb[] = new char[1];  
        if (read(cb, 0, 1) == -1)  
            return -1;  
        else  
            return cb[0];  
    }  
  
    public int read(char cbuf[]) throws IOException {  
        return read(cbuf, 0, cbuf.length);  
    }  
  
    public abstract int read(char[] cbuf, int off, int len) throws IOException;  
  
    public abstract void close() throws IOException;  
}
```



Абстрактный класс Reader – базовый класс для символьных потоков ввода. Для чтения в диапазон элементов массива char в нём объявлен абстрактный метод read. Определения этого метода в потомках должны возвращать число char которое удалось считать или -1 если ни одного char считать нельзя из-за того что достигнут конец потока. Для чтения одного char в нём объявлен метод read этот метод считывает один char и возвращает его значение или -1 если он встретил конец потока. По этой причине тип возвращаемого значения int а не char. Конкретные классы потомки должны переопределять абстрактный метод read. Как правило потомки переопределяют и другие методы read более эффективными реализациями. Класс содержит абстрактный методы close. Метод close предназначен для закрытия потока и освобождения неуправляемых ресурсов после окончания чтения.

```
public class InputStreamReader extends Reader {  
  
    private final StreamDecoder sd;  
  
    InputStreamReader( InputStream in )  
    InputStreamReader( InputStream in , String charsetName )  
  
    public int read() throws IOException  
    public int read(char[] cbuf, int off, int len) throws IOException  
  
    String getEncoding()  
    void close()  
}
```



Класс InputStreamReader предназначен для чтения байтов из байтового потока ввода и превращения их в символы с помощью кодировки. Кодировку можно задать в конструкторе. Если кодировка не задаётся используется кодировка по умолчанию.

```
public class FileReader extends InputStreamReader {  
  
    public FileReader(String fileName) throws FileNotFoundException {  
        super(new FileInputStream(fileName));  
    }  
  
}
```



Класс `FileReader` предназначен для чтения последовательности `char` из файла. Объект класса `FileReader` это `InputStreamReader` для чтения из байтового файлового потока ввода. При использовании `FileReader` всегда используется кодировка по умолчанию.

```
public class ReadCharDemo {  
  
    public static void main(String[] args) throws IOException {  
  
        FileReader in = null;  
        int temp;  
  
        try {  
            in = new FileReader("I:\\FileIO\\charfile.txt");  
            for (int i = 0; i < 64; i++) {  
                temp = in.read();  
                if (temp == -1)  
                    break;  
  
                System.out.print((char) temp);  
            }  
        } catch (IOException e) {  
            System.out.println("An I/O error occurred");  
        } finally {  
            try {  
                if (in != null)  
                    in.close();  
            } catch (IOException e) {  
                System.out.println("Error closing file");  
            }  
        }  
    }  
}
```

АБВГДЕЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯабвгдежзийклмнопрстуфхцчщъыьэюя

```
public class ReadCharsDemo {  
  
    public static void main(String[] args) throws IOException {  
  
        FileReader in = null;  
  
        try {  
            in = new FileReader("I:\\FileIO\\charfile.txt");  
            char[] chars = new char[64];  
            int nread = in.read(chars);  
            if (nread > 0) {  
                System.out.println("Read " + nread + " characters");  
                System.out.println(Arrays.toString(chars));  
            }  
        } catch (IOException e) {  
            System.out.println("An I/O error occurred");  
        } finally {  
            try {  
                if (in != null)  
                    in.close();  
            } catch (IOException e) {  
                System.out.println("Error closing file");  
            }  
        }  
    }  
}
```

Read 64 characters

[А, Б, В, Г, Д, Е, Ж, З, И, Й, ..., Щ, Ъ, Ы, Ь, Э, Ю, Я, а, б, в, г, д, е, ж, з, и, й, ..., щ, ъ, ы, ь, э, ю, я]

Чтение символов из файла в кодировке UTF-16

```
public class ReadEncodingDemo {  
  
    public static void main(String[] args) throws Exception {  
  
        FileInputStream is = new FileInputStream("I:\\FileIO\\writeutf16.txt");  
        Reader in = new InputStreamReader(is, "UTF-16");  
  
        int data = in.read();  
        while (data != -1) {  
            char dataChar = (char) data;  
            data = in.read();  
            System.out.print(dataChar);  
        }  
        in.close();  
    }  
}
```

Hello World!

Буферизованный ввод

```
public class BufferedReader extends Reader {

    private Reader in;
    private char cb[];
    private int nChars, nextChar;
    private static int defaultCharBufferSize = 8192;

    public BufferedReader(Reader in, int sz) {
        super(in);
        if (sz <= 0)
            throw new IllegalArgumentException("Buffer size <= 0");
        this.in = in;
        cb = new char[sz];
        nextChar = nChars = 0;
    }

    public BufferedReader(Reader in) {
        this(in, defaultCharBufferSize);
    }

    int read()
    int read(char[] cbuf, int off, int len)
    String readLine()

    public void close() throws IOException {
        ...
        out.close();
        ...
    }
}
```



При использовании буферизованного потока ввода данные читаются в буфер из оборачиваемого потока большими порциями а затем по мере необходимости читаются уже из буфера. Таким образом можно снизить количество операций чтения из оборачиваемого потока и повысить эффективность.


```
public class ReadBufPerform {  
  
    public static void main(String[] args) throws IOException {  
  
        BufferedReader inbuf = null;  
        FileReader in = null;  
  
        int temp;  
  
        long time = System.currentTimeMillis();  
        try {  
            inbuf = new BufferedReader(new FileReader("I:\\FileIO\\outbuf.txt"));  
  
            for (int i = 0; i < 10000000; i++) {  
                temp = inbuf.read();  
            }  
        } catch (IOException e) {  
            System.out.println("An I/O error occurred");  
        } finally {  
            try {  
                if (inbuf != null)  
                    inbuf.close();  
            } catch (IOException e) {  
                System.out.println("Error closing file");  
            }  
        }  
        ...  
    }  
}
```

```
public class ReadBufPerform {  
  
    public static void main(String[] args) throws IOException {  
  
        ...  
        time = System.currentTimeMillis();  
        try {  
            in = new FileReader("I:\\FileIO\\outnobuf.txt");  
  
            for (int i = 0; i < 10000000; i++) {  
                temp = in.read();  
            }  
        } catch (IOException e) {  
            System.out.println("An I/O error occurred");  
        } finally {  
            try {  
                if (in != null)  
                    in.close();  
            } catch (IOException e) {  
                System.out.println("Error closing file");  
            }  
        }  
        time = System.currentTimeMillis() - time;  
        System.out.println("Non-buffered input time: " + time);  
    }  
}
```

```
Buffered input time: 437  
Non-buffered input time: 891
```

Почему такой маленький выигрыш?



Для байтовых потоков благодаря использованию буфера был получен выигрыш в 30 раз. При использовании аналогичного буфера для символьных потоков выигрыш был в 2 раза. Дело в том что символьные потоки уже буферизованы. В классе StreamDecoder есть буфер.

Спасибо

**Россия, 127018,
Москва, ул. Полковая 3, стр. 14
Тел.: +7(495) 780 7575, 789 9339
Факс: +7(495) 780 7576, 789 9338
info@diasoft.ru, www.diasoft.ru**