
DIASOFT
всё по-настоящему

DIASOFT
всё по-настоящему

# V. Ввод - вывод

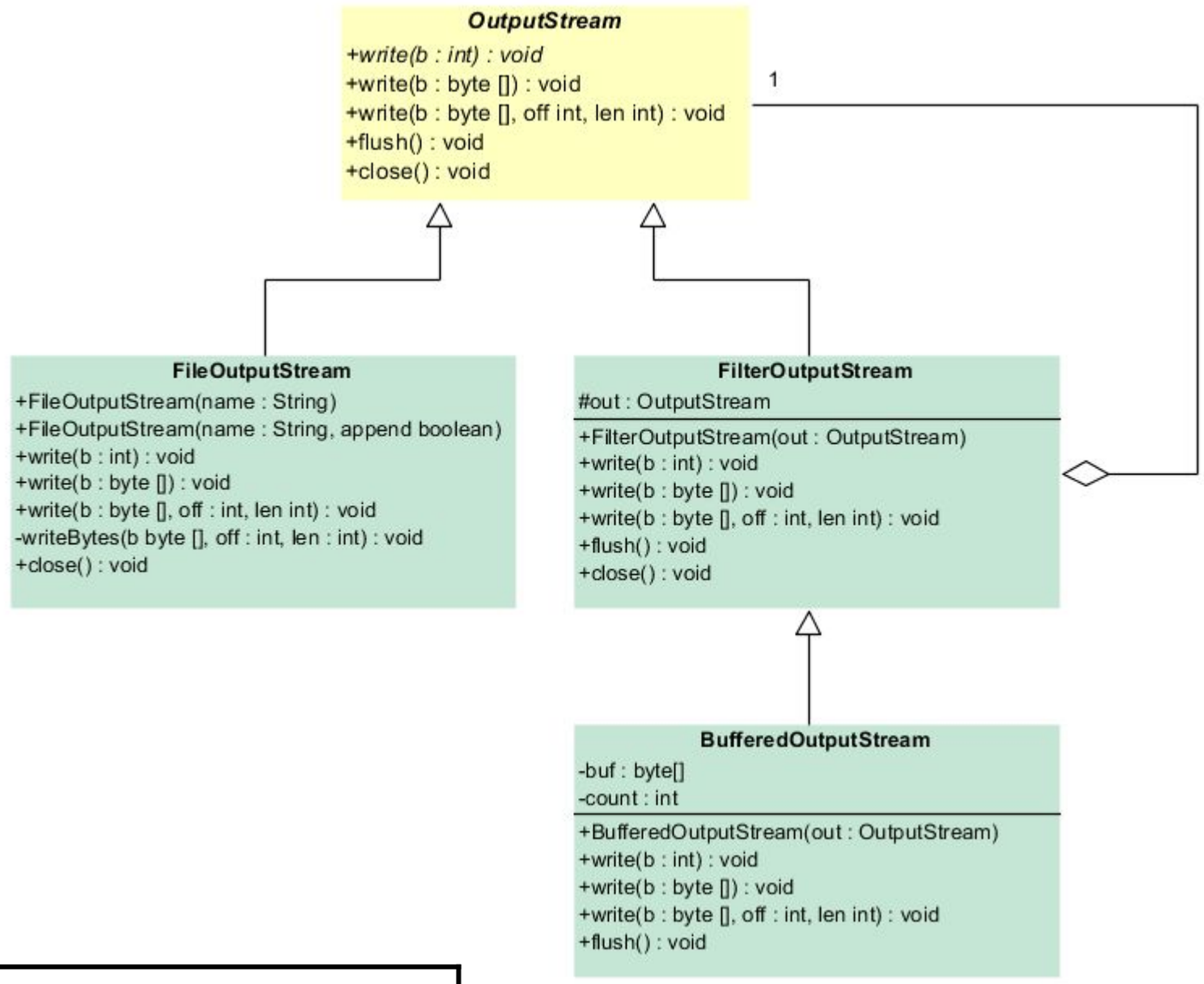
## 2. Байтовые потоки

# Потоки

Объект из которого можно прочитать последовательность байтов называется байтовый поток ввода. Объект в который можно записать последовательность байтов называется байтовый поток вывода. Классы байтовых потоков ввода являются подклассами абстрактного класса InputStream, потоков вывода – подклассами абстрактного класса OutputStream. Классы байтовых потоков находятся в пакете java.io.

# Потоки вывода

# Иерархия классов байтовых потоков вывода




**Шаблон Декоратор:** Иерархия классов байтовых потоков вывода является примером применения шаблона Декоратор.

# Класс OutputStream

```
public abstract class OutputStream implements Closeable, Flushable
{
    public abstract void write(int b)

    public void write(byte b[]) throws IOException {
        write(b, 0, b.length);
    }

    public void write(byte b[], int off, int len) throws IOException {
        if (b == null) {
            throw new NullPointerException();
        } else if ((off < 0) || (off > b.length) || (len < 0) ||
            ((off + len) > b.length) || ((off + len) < 0)) {
            throw new IndexOutOfBoundsException();
        } else if (len == 0) {
            return;
        }
        for (int i = 0 ; i < len ; i++) {
            write(b[off + i]);
        }
    }

    public void flush() throws IOException {
    }

    public void close() throws IOException {
    }
}
```

Абстрактный класс OutputStream – базовый класс для потоков вывода. Для вывода одного байта 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 пустые реализации методов close и flush. Метод close предназначен для закрытия потока после окончания записи. Закрытие потока освобождает ограниченные системные ресурсы, а также освобождает буфер если он используется. Метод flush предназначен для опустошения буфера если поток буферизованный.

# Класс FileOutputStream

```
public class FileOutputStream extends OutputStream
{
    public FileOutputStream(String name) throws FileNotFoundException
    public FileOutputStream(String name, append boolean) throws FileNotFoundException

    public native void write(int b) throws IOException;
    private native void writeBytes(byte b[], int off, int len) throws IOException;

    public void write(byte b[]) throws IOException {
        writeBytes(b, 0, b.length);
    }

    public void write(byte b[], int off, int len) throws IOException {
        writeBytes(b, off, len);
    }

    public void close() throws IOException
}
```

| | |
|---|---|
|  | Класс FileOutputStream предназначен для записи последовательности байтов в файлы. Он переопределяет методы write и метод close из класса OutputStream. Метод close закрывает поток и освобождает файловый дескриптор. Конструктор позволяет задать имя файла для записи. Также можно указать следует ли перезаписывать существующий файл или дописывать в него. |

# Запись в файл по одному байту

DIASOFT
всё по-настоящему

```
public class WriteByteDemo {

   public static void main(String[] args) {

      FileOutputStream out = null;
      int[] ints = new int[256];

      try {
         out = new FileOutputStream("I:\\FileIO\\ bytesfile.dat");
         for (int i = 0; i < 256; i++) {
            ints[i] = i;
            out.write(i);
         }
      } catch (IOException e) {
         System.out.println("An I/O error occured");
      } finally {
         try {
            if (out != null)
               out.close();
         } catch (IOException e) {
            System.out.println("Error closing file");
         }
      }
      System.out.println(Arrays.toString(ints));
   }
}
```

```
[0, 1, 2, 3, 4, 5, ... ,125, 126, 127, 128, 129, 130, 131, ... ,250, 251, 252, 253, 254, 255]
```

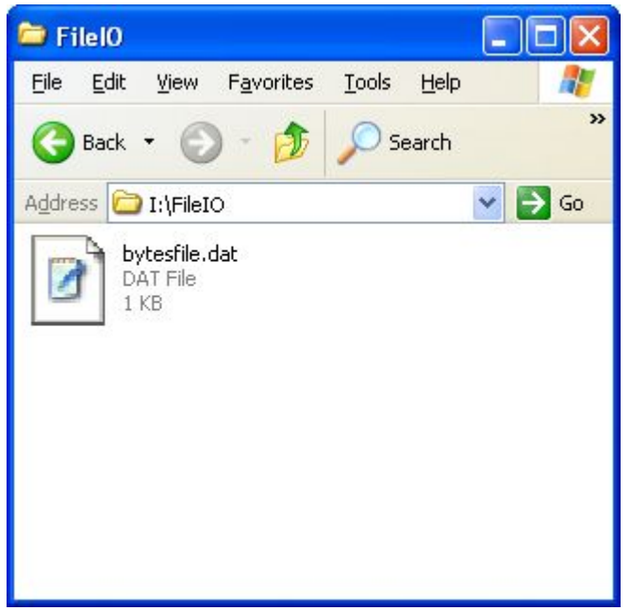
# Запись в файл массива байтов

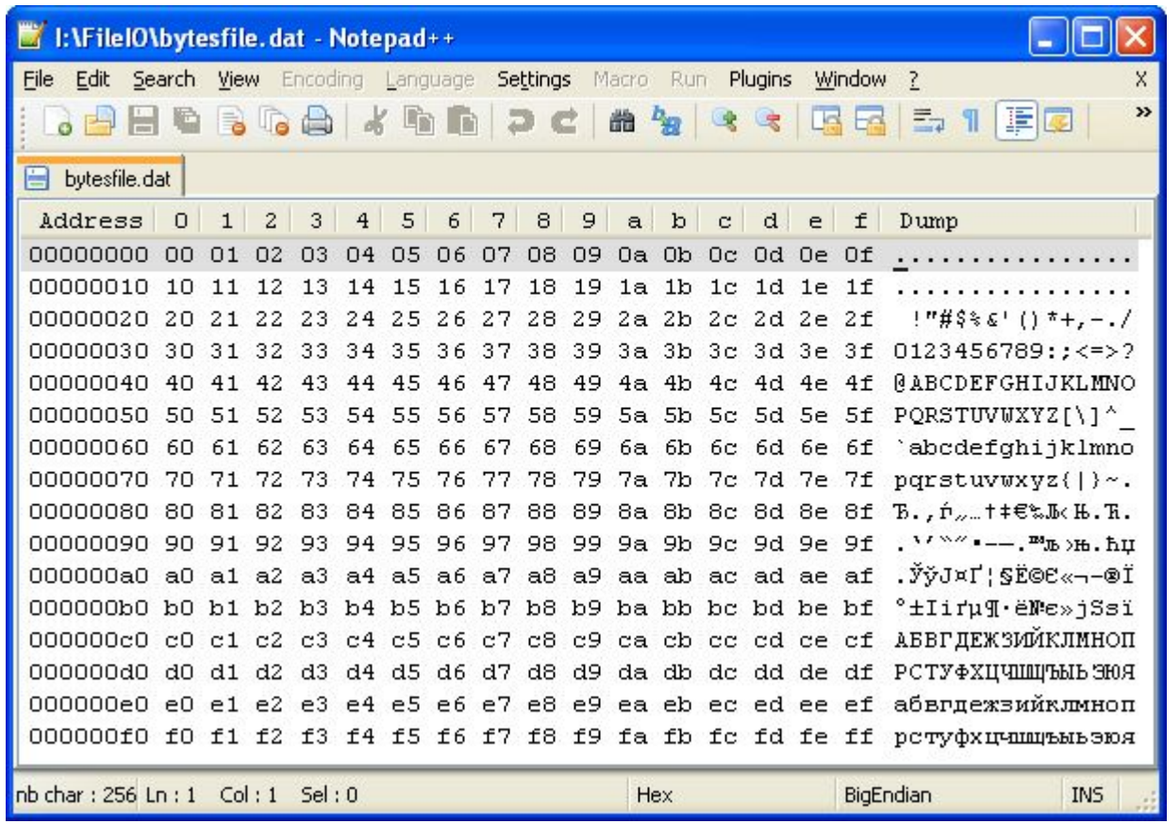
DIASOFT
всё по-настоящему

```
public class WriteBytesDemo {

   public static void main(String[] args) {

      FileOutputStream out = null;
      byte[] bytes = new byte[256];

      for (int i = 0; i < 256; i++) {
         bytes[i] = (byte) i;
      }

      try {
         out = new FileOutputStream("I:\\FileIO\\ bytesfile.dat");
         out.write(bytes);
      } catch (IOException e) {
         System.out.println("An I/O error occured");
      } finally {
         try {
            if (out != null)
               out.close();
         } catch (IOException e) {
            System.out.println("Error closing file");
         }
      }
      System.out.println(Arrays.toString(bytes));
   }
}
```

```
[0, 1, 2, 3, 4, 5, ... ,125, 126, 127, -128, -127, -126, -125, ... , -5, -4, -3, -2, -1]
```

# Запись в файл

# Запись строки в файл

DIASOFT
всё по-настоящему

```
public class WriteStringDemo {

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

      String source = "Hello World!";
      byte[] bytes = source.getBytes();

      FileOutputStream out = null;

      try {
         out = new FileOutputStream("I:\\FileIO\\stringfile.dat");
         out.write(bytes);
      } catch (IOException e) {
         System.out.println("An I/O error occured");
      } finally {
         try {
            if (out != null)
               out.close();
         } catch (IOException e) {
            System.out.println("Error closing file");
         }
      }
      System.out.println(Arrays.toString(bytes));
   }
}
```

```
[72, 101, 108, 108, 111, 32, 87, 111, 114, 108, 100, 33]
Hello World!
```

# Буферизованный вывод

# Класс FilterOutputStream

```
public class FilterOutputStream extends OutputStream {

    protected OutputStream out;

    public FilterOutputStream(OutputStream out) {
        this.out = out;
    }

    public void write(int b) throws IOException {
        out.write(b);
    }

    public void write(byte b[]) throws IOException {
        write(b, 0, b.length);
    }

    public void write(byte b[], int off, int len) throws IOException {
        if ((off | len | (b.length - (len + off)) | (off + len)) < 0)
            throw new IndexOutOfBoundsException();
        for (int i = 0 ; i < len ; i++) {
           write(b[off + i]);
        }
    }

    public void flush() throws IOException {
        out.flush();
    }

    public void close() throws IOException {
        try {
            flush();
        } catch (IOException ignored) { }
        out.close();
    }
}
```

| | |
|---|---|
| C | Класс FilterOutputStream – базовый класс для всех фильтрующих потоков вывода. Эти классы оборачивают существующий поток вывода который используется как сток данных возможно преобразуя данные или предоставляя дополнительную функциональность. Класс FilterOutputStream просто переопределяет все методы класса OutputStream версиями которые перенаправляют все запросы оборачиваемому потоку. Классы потомки FilterOutputStream могут переопределять эти методы а также предоставлять дополнительные методы и поля. |

# Класс BufferedOutputStream

```
public class BufferedOutputStream extends FilterOutputStream {

    protected byte buf[];
    protected int count;

    public BufferedOutputStream(OutputStream out) {
        this(out, 8192);
    }

    public BufferedOutputStream(OutputStream out, int size) {
        super(out);
        if (size <= 0) {
            throw new IllegalArgumentException("Buffer size <= 0");
        }
        buf = new byte[size];
    }

    private void flushBuffer() throws IOException {
        if (count > 0) {
        out.write(buf, 0, count);
        count = 0;
        }
    }

    public synchronized void flush() throws IOException {
        flushBuffer();
        out.flush();
    }
    ...
}
```

Класс BufferedOutputStream - реализует буферизованный поток вывода для повышения эффективности вывода данных. В конструкторе необходимо задать оборачиваемый поток. Для хранения данных используется массив byte размер которого можно задать в конструкторе. Метод flush переопределён он выполняет опустошение буфера и вызов метода flush у оборачиваемого потока.

# Класс BufferedOutputStream

```
public class BufferedOutputStream extends FilterOutputStream {

    ...

    public synchronized void write(int b) throws IOException {
        if (count >= buf.length) {
            flushBuffer();
        }
        buf[count++] = (byte)b;
    }

    public synchronized void write(byte b[], int off, int len) throws IOException {
        if (len >= buf.length) {
            flushBuffer();
            out.write(b, off, len);
            return;
        }
        if (len > buf.length - count) {
            flushBuffer();
        }
        System.arraycopy(b, off, buf, count, len);
        count += len;
    }
}
```

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

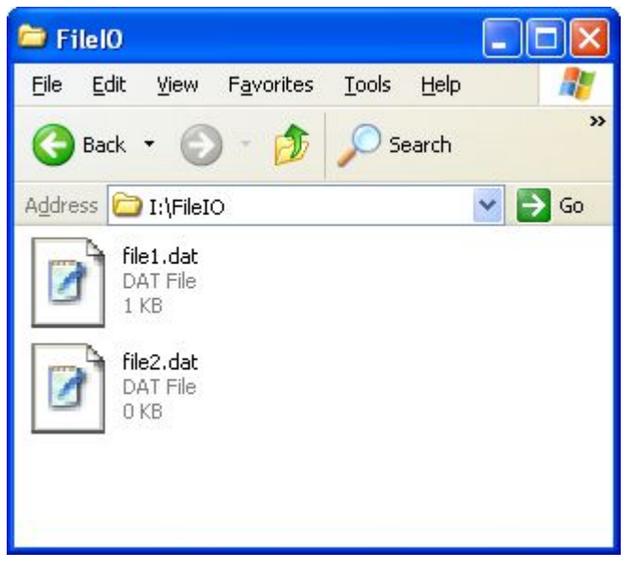
# Опустошение буфера

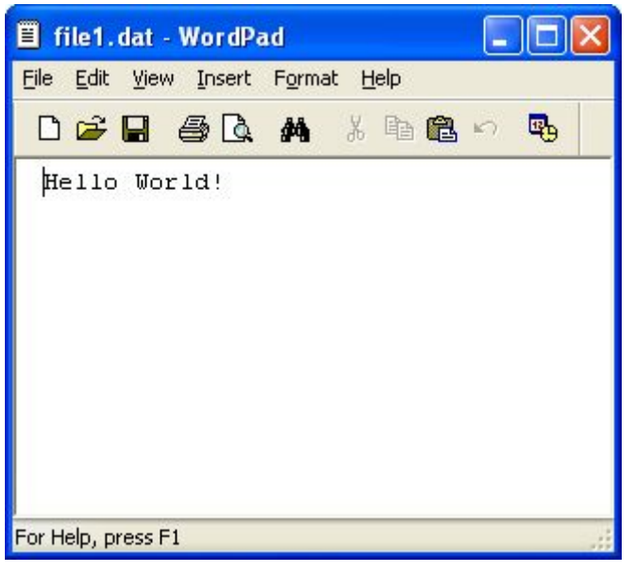
DIASOFT
всё по-настоящему

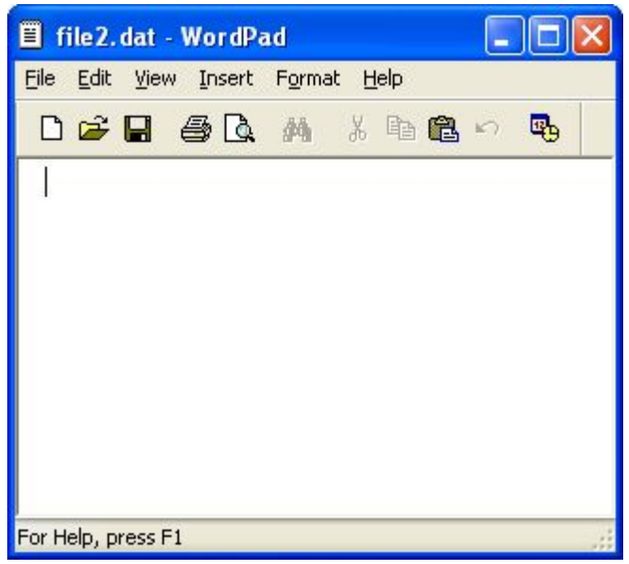
```
public class WriteFlushDemo {

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

      String source = "Hello World!";
      byte[] bytes = source.getBytes();

      BufferedOutputStream out1 = null;
      BufferedOutputStream out2 = null;

      try {
         out1 = new BufferedOutputStream(new FileOutputStream("I:\\FileIO\\file1.dat"));
         out2 = new BufferedOutputStream(new FileOutputStream("I:\\FileIO\\file2.dat"));

         out1.write(bytes);
         out2.write(bytes);

      } catch (IOException e) {
         System.out.println("An I/O error occured");
      } finally {
         try {
            if (out1 != null)
               out1.close();
         } catch (IOException e) {
            System.out.println("Error closing file");
         }
      }
      System.out.println(Arrays.toString(bytes));
   }
}
```

```
[72, 101, 108, 108, 111, 32, 87, 111, 114, 108, 100, 33]
```

# Опустошение буфера



# Производительность буферизованного вывода

DIASOFT
всё по-настоящему

```
public class WriteBufPerform {

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

      BufferedOutputStream outbuf = null;
      FileOutputStream out = null;

      long time = System.currentTimeMillis();
      try {
         outbuf = new BufferedOutputStream(new FileOutputStream("I:\\FileIO\\outbuf.dat"));

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

# Производительность небуферизованного вывода

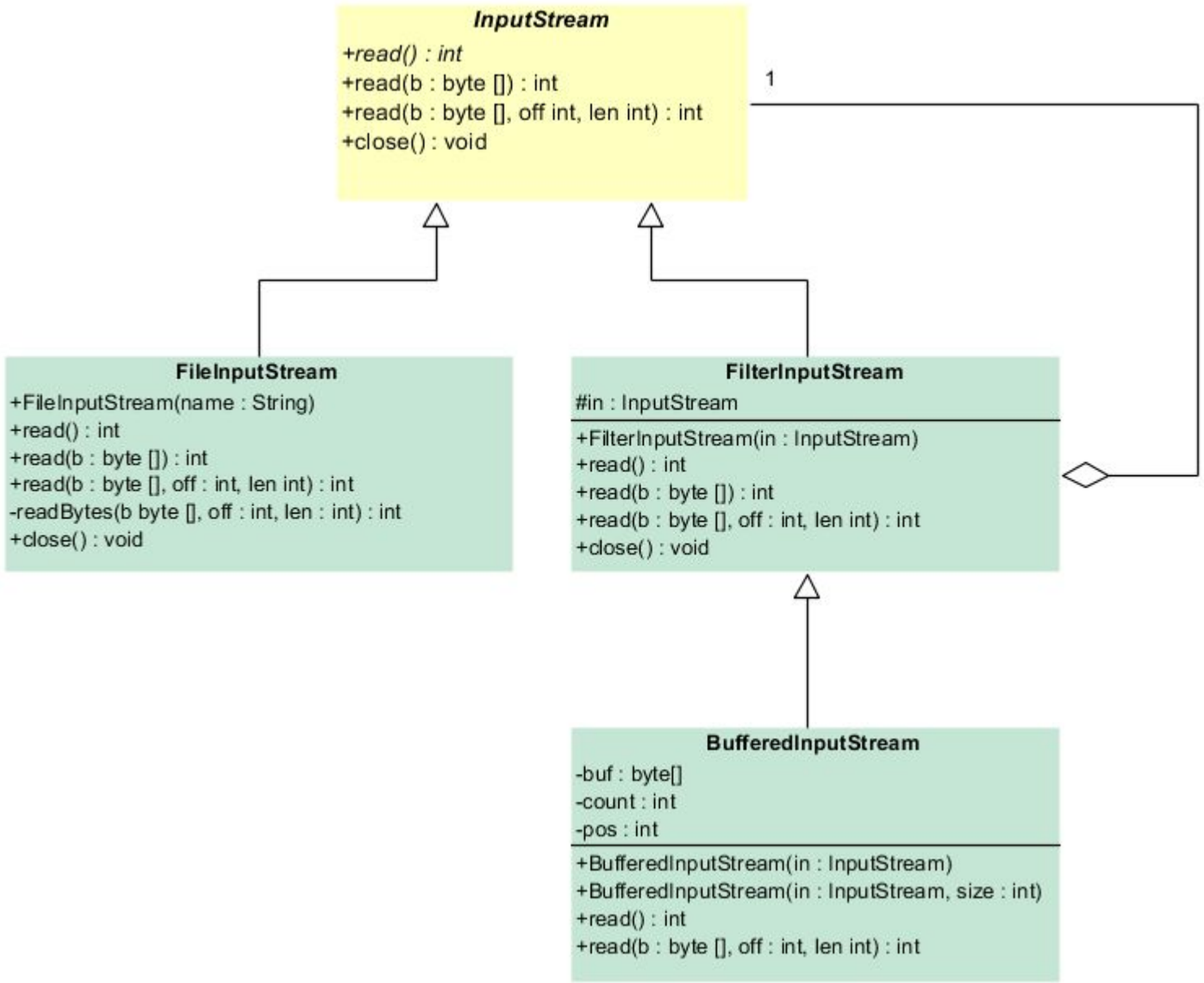
DIASOFT
всё по-настоящему

```
public class WriteBufPerform {

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

      ...

      time = System.currentTimeMillis();
      try {
         out = new FileOutputStream("I:\\FileIO\\outnobuf.dat");

         for (int i = 0; i < 10000000; i++) {
            out.write(65);
         }
      } catch (IOException e) {
         System.out.println("An I/O error occured");
      } 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: 344
Non-buffered output time: 35766
```

# Потоки ввода

# Иерархия классов байтовых потоков ввода




**Шаблон Декоратор:** Иерархия классов байтовых потоков вввода является примером применения шаблона Декоратор.

# Класс InputStream

```
public abstract class InputStream implements Closeable {

    public abstract int read() throws IOException;

    public int read(byte b[]) throws IOException {
        return read(b, 0, b.length);
    }

    public int read(byte b[], int off, int len) throws IOException

    public void close() throws IOException {}
}
```

Абстрактный класс InputStream – базовый класс для потоков ввода. Для чтения одного байта в нём объявлен абстрактный метод read. Этот метод считывает один байт и возвращает его значение или -1 если он сразу же встретил конец потока. По этой причине тип возвращаемого значения int, а не byte. Конкретные классы потомки  должны переопределять этот метод. Как правило потомки переопределяют и другие методы read более эффективными реализациями. Другие методы read пытаются считать заданное количество байтов в массив. Если предпринимается попытка считать 0 байтов методы вернут 0. В противном случае возвращается количество считанных байтов или -1 если нельзя считать ни одного байта из-за того что сразу был встречен конец потока.

# Класс FileInputStream

```
public class FileInputStream extends InputStream
{
    public FileInputStream(String name) throws FileNotFoundException {
        this(name != null ? new File(name) : null);
    }
    public FileInputStream(File file)

    public native int read() throws IOException;

    public int read(byte b[]) throws IOException {
        return readBytes(b, 0, b.length);
    }

    public int read(byte b[], int off, int len) throws IOException {
        return readBytes(b, off, len);
    }

    private native int readBytes(byte b[], int off, int len) throws IOException;

    public void close() throws IOException
}
```

Класс FileInputStream предназначен для чтения последовательности байтов из файла. Он переопределяет методы read и метод close из класса InputStream. Конструктор позволяет задать имя файла для чтения. Метод close закрывает поток и освобождает файловый дескриптор.

# Чтение из файла по одному байту

DIASOFT
всё по-настоящему

```
public class ReadByteDemo {

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

      FileInputStream in = null;
      int[] ints = new int[256];
      int temp;

      try {
         in = new FileInputStream("I:\\FileIO\\bytesfile.dat");
         for (int i = 0; i < 256; i++) {
            temp = in.read();
            if (temp == -1)
               break;

            ints[i] = temp;
         }
      } catch (IOException e) {
         System.out.println("An I/O error occured");
      } finally {
         try {
            if (in != null)
               in.close();
         } catch (IOException e) {
            System.out.println("Error closing file");
         }
      }
      System.out.println(Arrays.toString(ints));
   }
}
```

```
[0, 1, 2, 3, 4, 5, ... ,125, 126, 127, 128, 129, 130, 131, ... ,250, 251, 252, 253, 254, 255]
```

# Чтение в массив байтов

DIASOFT
всё по-настоящему

```
public class ReadBytesDemo {

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

      FileInputStream in = null;
      byte[] bytes = new byte[256];

      try {
         in = new FileInputStream("I:\\FileIO\\bytesfile.dat");
         in.read(bytes);
      } catch (IOException e) {
         System.out.println("An I/O error occured");
      } finally {
         try {
            if (in != null)
               in.close();
         } catch (IOException e) {
            System.out.println("Error closing file");
         }
      }
      System.out.println(Arrays.toString(bytes));
   }
}
```

```
[0, 1, 2, 3, 4, 5, ... ,125, 126, 127, -128, -127, -126, -125, ... , -5, -4, -3, -2, -1]
```

# Чтение строки из файла

DIASOFT
всё по-настоящему

```
public class ReadStringDemo {

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

      FileInputStream in = null;
      byte[] bytes = new byte[256];
      int nbytes = 0;

      try {
         in = new FileInputStream("I:\\FileIO\\stringfile.dat");
         nbytes = in.read(bytes);
      } catch (IOException e) {
         System.out.println("An I/O error occured");
      } finally {
         try {
            if (in != null)
               in.close();
         } catch (IOException e) {
            System.out.println("Error closing file");
         }
      }
      String hello = new String(bytes, 0, nbytes);
      System.out.println(Arrays.toString(bytes));
      System.out.println(hello);
   }
}
```

```
[72, 101, 108, 108, 111, 32, 87, 111, 114, 108, 100, 33, 0, 0, ... , 0, 0, 0, 0, 0]
Hello World!
```

# Буферизованный ввод

# Класс FilterInputStream

```
public class FilterInputStream extends InputStream
{
    protected InputStream in;

    protected FilterInputStream(InputStream in) {
        this.in = in;
    }

    public int read() throws IOException {
        return in.read();
    }

    public int read(byte b[]) throws IOException {
        return read(b, 0, b.length);
    }

    public int read(byte b[], int off, int len) throws IOException {
        return in.read(b, off, len);
    }

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

| | |
|---|---|
| C | Класс FilterInputStream – базовый класс для всех фильтрующих потоков ввода. Эти классы оборачивают существующий поток ввода который используется как источник данных возможно преобразуя данные или предоставляя дополнительную функциональность. Класс FilterInputStream просто переопределяет все методы класса InputStream версиями которые перенаправляют все запросы оборачиваемому потоку. Классы потомки FilterInputStream могут переопределять эти методы, а также предоставлять дополнительные методы и поля. |

# Класс BufferedInputStream

```
public class BufferedInputStream extends FilterInputStream
{
   protected byte[] buf
   protected int count;
   protected int pos;

   public BufferedInputStream(InputStream in)
   public BufferedInputStream(InputStream in, int size)

   public int read()
   public int read(byte[] b, int off, int len)
}
```

Класс BufferedInputStream - реализует буферизованный поток ввода для повышения эффективности чтения данных. В конструкторе необходимо задать оборачиваемый поток. Для хранения данных используется массив byte размер которого можно задать в конструкторе.

# Производительность буферизованного ввода

DIASOFT
всё по-настоящему

```
public class ReadBufPerform {

   public static void main(String[] args) throws IOException {
      BufferedInputStream inbuf = null;
      FileInputStream in = null;

      int temp;

      long time = System.currentTimeMillis();
      try {
         inbuf = new BufferedInputStream(new FileInputStream(
               "I:\\FileIO\\outbuf.dat"));

         for (int i = 0; i < 10000000; i++) {
            temp = inbuf.read();
         }
      } catch (IOException e) {
         System.out.println("An I/O error occured");
      } finally {
         try {
            if (inbuf != null)
               inbuf.close();
         } catch (IOException e) {
            System.out.println("Error closing file");
         }
      }
      time = System.currentTimeMillis() - time;
      System.out.println("Buffered input time: " + time);
      ...
   }
}
```

# Производительность небуферизованного ввода

DIASOFT
всё по-настоящему

```
public class ReadBufPerform {

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

      ...

      time = System.currentTimeMillis();
      try {
         in = new FileInputStream("I:\\FileIO\\outnobuf.dat");

         for (int i = 0; i < 10000000; i++) {
            temp = in.read();
         }
      } catch (IOException e) {
         System.out.println("An I/O error occured");
      } 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: 375
Non-buffered input time: 10031
```

# 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**