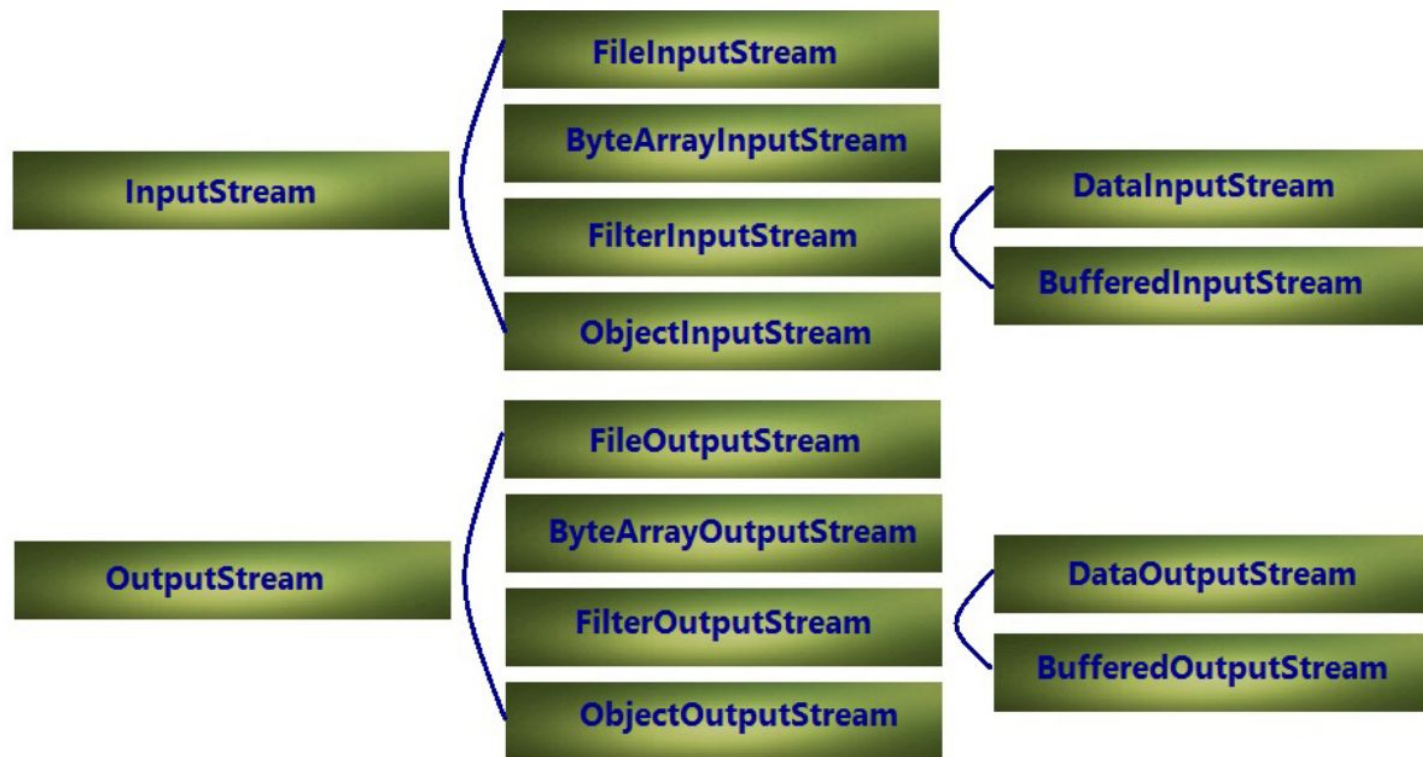


# Работа с файлами

# Байтовые потоки ввода-вывода



# Символьные потоки ввода-вывода



# File

File, определенный в пакете java.io, не работает напрямую с потоками. Его задачей является **управление информацией о файлах и каталогах**. Хотя на уровне операционной системы файлы и каталоги отличаются, но в Java они описываются одним классом File.

```
File(String путь_к_каталогу)
File(String путь_к_каталогу, String имя_файла)
File(File каталог, String имя_файла)
```

Например:

```
// создаем объект File для каталога
File dir1 = new File("C://SomeDir");
// создаем объекты для файлов, которые находятся в каталоге
File file1 = new File("C://SomeDir", "Hello.txt");
File file2 = new File(dir1, "Hello2.txt");
```

# File - методы

**boolean createNewFile():** создает новый файл по пути, который передан в конструктор. В случае удачного создания возвращает true, иначе false

**boolean delete():** удаляет каталог или файл по пути, который передан в конструктор. При удачном удалении возвращает true.

**boolean exists():** проверяет, существует ли по указанному в конструкторе пути файл или каталог. И если файл или каталог существует, то возвращает true, иначе возвращает false

**String getAbsolutePath():** возвращает абсолютный путь для пути, переданного в конструктор объекта

**String getName():** возвращает краткое имя файла или каталога

**String getParent():** возвращает имя родительского каталога

**boolean isDirectory():** возвращает значение true, если по указанному пути располагается каталог

**boolean isFile():** возвращает значение true, если по указанному пути находится файл

**boolean isHidden():** возвращает значение true, если каталог или файл являются скрытыми

**long length():** возвращает размер файла в байтах

**long lastModified():** возвращает время последнего изменения файла или каталога. Значение представляет количество миллисекунд, прошедших с начала эпохи Unix

# File - методы

**String[] list():** возвращает массив названий файлов и подкаталогов, которые находятся в определенном каталоге

**File[] listFiles():** возвращает массив файлов и подкаталогов, которые находятся в определенном каталоге

**boolean mkdir():** создает новый каталог и при удачном создании возвращает значение true

**boolean mkdirs():** создает много папок сразу

**boolean renameTo(File dest):** переименовывает файл или каталог

**deleteOnExit()** - если вызвать этот метод для объекта класса File, то файл на диске, ассоциированный с этим объектом, будет удален автоматически, при завершении работы JVM;

**getFreeSpace()** - возвращает кол-во свободных байт на том диске, где расположен текущий объект;

**getTotalSpace()** - возвращает общий размер диска, на котором расположен текущий объект;

**setReadOnly()** - делает файл доступным только для чтения.

**есть еще методы**

# File - пример работы с каталогом

```
public class Program {  
  
    public static void main(String[] args) {  
  
        // определяем объект для каталога  
        File dir = new File("C://SomeDir//NewDir");  
        boolean created = dir.mkdir();  
        if(created)  
            System.out.println("Folder has been created");  
        // переименуем каталог  
        File newDir = new File("C://SomeDir//NewDirRenamed");  
        dir.renameTo(newDir);  
        // удалим каталог  
        boolean deleted = newDir.delete();  
        if(deleted)  
            System.out.println("Folder has been deleted");  
    }  
}
```

## File - пример 1 работы с файлами

```
File myFile = new File("C://SomeDir//notes.txt");
System.out.println("File name: " + myFile.getName());
System.out.println("Parent folder: " + myFile.getParent());
if(myFile.exists())
    System.out.println("File exists");
else
    System.out.println("File not found");

System.out.println("File size: " + myFile.length());
if(myFile.canRead())
    System.out.println("File can be read");
else
    System.out.println("File can not be read");
```



## File - пример 2 работы с файлами

```
// создадим новый файл
File newFile = new File("C://SomeDir//MyFile");
try
{
    boolean created = newFile.createNewFile();
    if(created)
        System.out.println("File has been created");
}
catch(IOException ex){

    System.out.println(ex.getMessage());
}
```

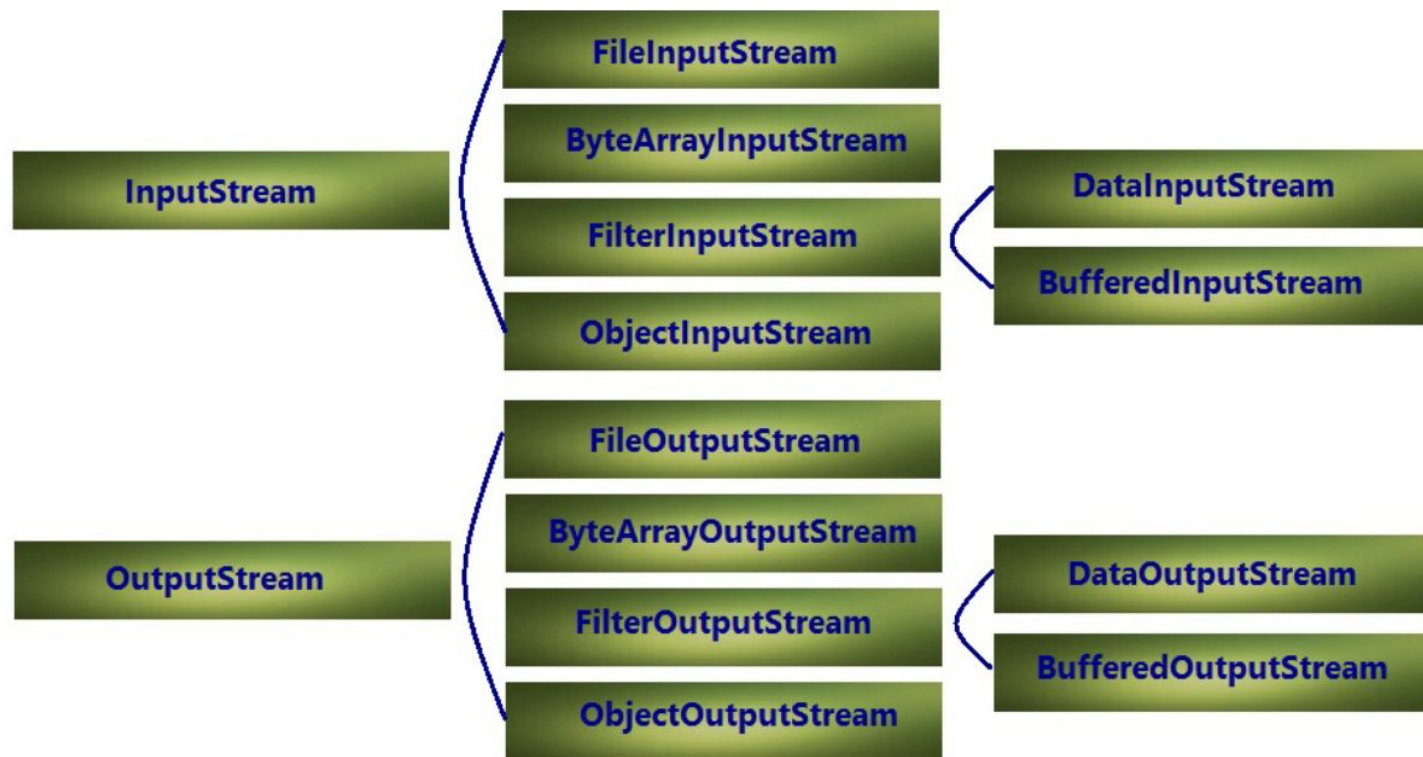
## Полезно знать

[System.getProperty\("user.dir"\)](#) - получить путь к корневой папке проекта

[File.separator](#) - платформозависимый символ, который используется для разделения каталогов на пути к файлу. Например, для Windows это '\\', а для UNIX это '/'.

**InputStream**  
**OutputStream**

# Байтовые потоки ввода-вывода



# InputStream

**read()** Возвращает текущий доступный символ из входного потока в виде целого значения.

**read(byte b[])** Читает b.length байт из входного потока в массив b. Возвращает количество прочитанных из потока байт.

**read(byte b[], int start, int len)** Читает len байт из входного потока в массив b, но располагает их в массиве, начиная с элемента start. Возвращает количество прочитанных байт.

**skip(long n)** Пропускает во входном потоке n байт. Возвращает количество пропущенных байт.

**available()** Возвращает количество байт в потоке, доступных для чтения.

**mark(int readlimit)** Ставит метку в текущей позиции входного потока, которую можно будет использовать до тех пор, пока из потока не будет прочитано readlimit байтов.

**reset()** Возвращает указатель потока на установленную ранее метку.

**markSupported()** Возвращает true, если данный поток поддерживает операции mark и reset.

**close()** Закрывает входной поток.

# FileInputStream

**FileInputStream** - это подкласс `InputStream`, который используется для чтения двоичных файлов, таких как фотографии, музыка, видео. Полученные данные являются необработанными bytes (raw bytes). Для обычных текстовых файлов вместо этого следует использовать `FileReader`

Наиболее часто употребляемые конструкторы (Каждый из них может сгенерировать исключение типа `FileNotFoundException`):

`FileInputStream(String путь_к_файлу)`

`FileInputStream(File объект_файла)`

Например:

```
FileInputStream f1 = new FileInputStream("/home/linuxuser/scriptstep.sh");
```

```
File f2 = new File("/home/linuxuser/scriptstep.sh");
```

```
FileInputStream f3 = new FileInputStream(f2);
```

[Большой пример](#)

# OutputStream

**int close()** - закрывает выходной поток. Следующие попытки записи передадут исключение `IOException`

**void flush()** - финализирует выходное состояние, очищая все буферы вывода

**abstract void write (int oneByte)** - записывает единственный байт в выходной поток

**void write (byte[] buffer)** - записывает полный массив байтов в выходной поток

**void write (byte[] buffer, int offset, int count)** - записывает диапазон из count байт из массива, начиная с смещения offset

# FileOutputStream

Класс FileOutputStream предназначен для записи байтов в файл.

Наиболее часто употребляемые конструкторы (**Каждый из них может сгенерировать исключение типа FileNotFoundException**):

`FileOutputStream(String filePath)`

`FileOutputStream(File fileObj)`

`FileOutputStream(String filePath, boolean append)`

`FileOutputStream(File fileObj, boolean append)`

**append** - если он равен true, то данные дозаписываются в конец файла, а при false - файл полностью перезаписывается

Например:

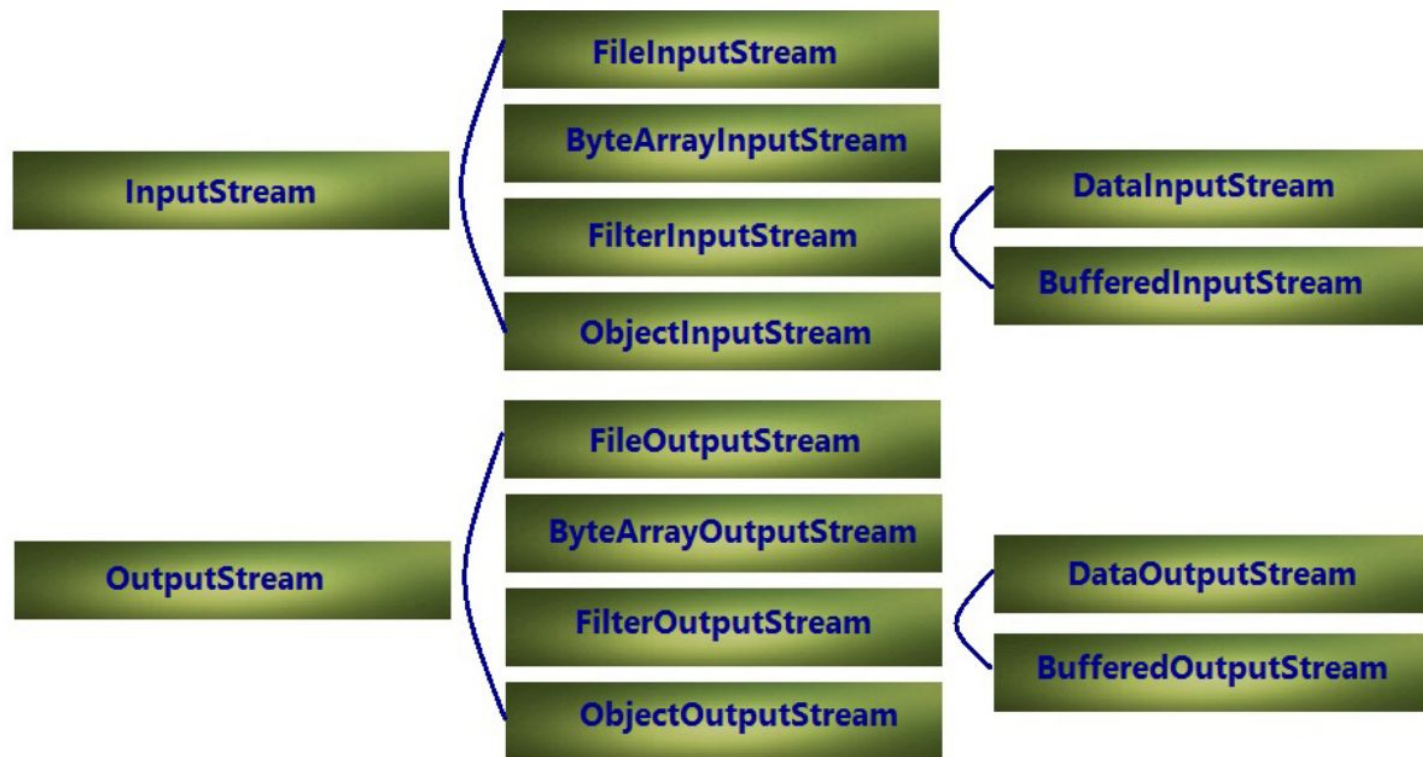
```
FileOutputStream fos=new FileOutputStream("C://SomeDir//notes.txt")
```

[пример](#)



**ByteArrayInputStream**  
**ByteArrayOutputStream**

# Байтовые потоки ввода-вывода



# ByteArrayInputStream

**ByteArrayInputStream** - это подкласс `InputStream`. Верный имени, `ByteArrayInputStream` используется для чтения массива `byte` посредством `InputStream`. Все методы `ByteArrayInputStream` наследуются от `InputStream`. Он является частью пакета `java.io` и в основном используется для преобразования `ByteArray` в `InputStream`.

Конструкторы:

`ByteArrayInputStream(byte[] buf)`

`ByteArrayInputStream(byte[] buf, int offset, int length)` - где `off` – первый байт для чтения, а `len` – количество байтов, которые нужно считать.

Пример:

```
public static void main(String[] args) throws IOException {  
  
    byte[] byteArray = new byte[] {84, 104, 105, 115, 32, 105, 115, 32, 116, 101, 120, 116};  
  
    ByteArrayInputStream is = new ByteArrayInputStream(byteArray);  
  
    int b;  
    while((b = is.read()) != -1) {  
        // Convert byte to character.  
        char ch = (char) b;  
        System.out.println(b + " --> " + ch);  
    }  
}
```

[пример](#)

# ByteArrayInputStream

Класс **java.io.ByteArrayInputStream** содержит внутренний буфер, который содержит байты, которые могут быть прочитаны из потока. Внутренний счетчик отслеживает следующий байт, который должен быть предоставлен методом `read`.

- Важные моменты, касающиеся `ByteArrayInputStream`:  
Заккрытие `ByteArrayInputStream` не имеет никакого эффекта.
- Методы в этом классе могут вызываться после закрытия потока без генерации исключения `IOException`.

```
public static void main(String[] args) throws IOException {  
    byte[] b = {20,30,40,50,60,70,80,90,100};  
    ByteArrayInputStream by = new ByteArrayInputStream(b);  
  
    byte[] buf = new byte[8];  
    int n = by.read(buf,3,4);  
  
    System.out.println("Number of bytes read: " + n);  
  
    for(byte c : buf) {  
        if(c==0)  
            System.out.print("*, ");  
        else  
            System.out.print(c + ",");  
    }  
}
```

[больше примеров](#)

# ByteArrayOutputStream

**ByteArrayOutputStream** - это подкласс `OutputStream`. Поток класса `ByteArrayOutputStream` создает буфер в памяти, и все данные, отправленные в поток, хранятся в буфере. Он является частью пакета `java.io` и в основном используется для преобразования `ByteArray` в `OutputStream`.

Конструкторы:

`ByteArrayOutputStream()` - Конструктор создает `ByteArrayOutputStream` с буфером в 32 байт

`ByteArrayOutputStream(int a)` - Конструктор создает `ByteArrayOutputStream` с буфером заданного размера

Пример:

```
ByteArrayOutputStream outputByte = new ByteArrayOutputStream(12);

while(outputByte.size() != 5) {
    outputByte.write("hello".getBytes());
}
```

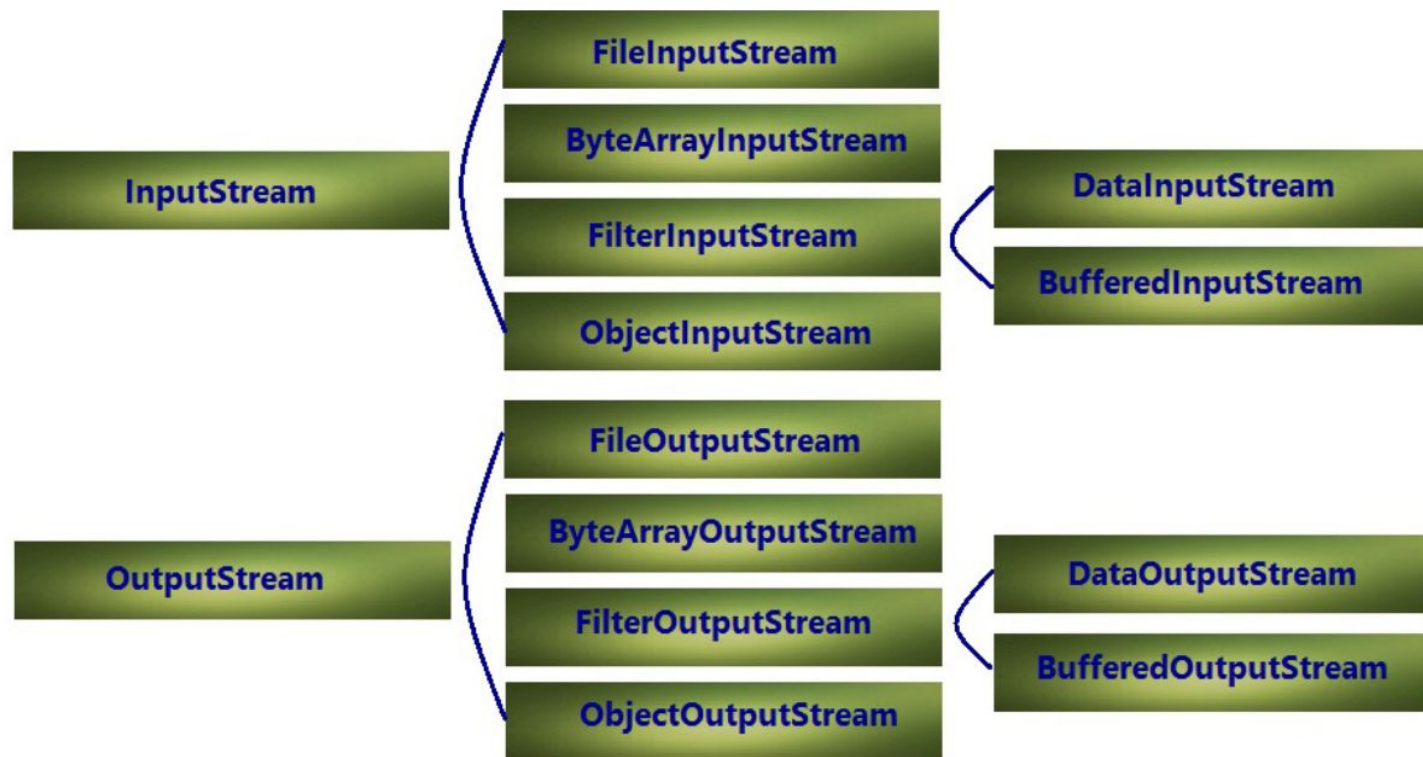
[пример](#)

# Методы ByteArrayOutputStream

| № | Метод и описание   |
|---|--|
| 1 | <b>public void reset()</b><br>Метод сбрасывает количество действительных байт в выходном потоке байтового массива до нуля, поэтому все накопленное на выходе будет сброшено.   |
| 2 | <b>public byte[] toByteArray()</b><br>Метод создает недавно выделенный массив байтов. Его размер будет текущим размером выходного потока, и содержимое буфера будет скопировано в него. Возвращает текущее содержимое выходного потока в виде байтового массива. |
| 3 | <b>public String toString()</b><br>Преобразует содержимое буфера в строку. Перевод будет выполняться в соответствии с кодировкой установленной по умолчанию. Возвращает строку, переведенную из содержимого буфера.  |
| 4 | <b>public void write(int w)</b><br>Запись указанного массива в выходной поток.   |
| 5 | <b>public void write(byte []b, int of, int len)</b><br>Запись len количества байтов, начиная смещение с of.  |
| 6 | <b>public void writeTo(OutputStream outSt)</b><br>Запись всего содержимого потока в указанный аргумент потока.   |

# **ObjectInputStream и ObjectOutputStream**

# Байтовые потоки ввода-вывода





# ObjectOutputStream

**ObjectOutputStream** - для записи объектов в файл

Конструктор:

`ObjectOutputStream(OutputStream out)`

Методы:

- **void close():** закрывает поток
- **void flush():** очищает буфер и сбрасывает его содержимое в выходной поток
- **void write(byte[] buf):** записывает в поток массив байтов
- **void write(int val):** записывает в поток один младший байт из val
- **void writeBoolean(boolean val):** записывает в поток значение boolean
- **void writeByte(int val):** записывает в поток один младший байт из val
- **void writeChar(int val):** записывает в поток значение типа char, представленное целочисленным значением
- **void writeDouble(double val):** записывает в поток значение типа double
- **void writeFloat(float val):** записывает в поток значение типа float
- **void writeInt(int val):** записывает целочисленное значение int
- **void writeLong(long val):** записывает значение типа long
- **void writeShort(int val):** записывает значение типа short
- **void writeUTF(String str):** записывает в поток строку в кодировке UTF-8
- **void writeObject(Object obj):** записывает в поток отдельный объект

# ObjectOutputStream

```
FileOutputStream fout=null;
ObjectOutputStream oout=null;
try {
    fout = new FileOutputStream("fish.txt");
    Fish f = new Fish("salmon",2.5,180);
    oout = new ObjectOutputStream(fout);
    oout.writeObject(f);
} catch (FileNotFoundException ex) {
    Logger.getLogger(Filetest.class.getName()).
        log(Level.SEVERE, null, ex);
} catch (IOException ex) {
    Logger.getLogger(Filetest.class.getName()).
        log(Level.SEVERE, null, ex);
}
finally
{
    try {
        oout.close();
    } catch (IOException ex) {
        Logger.getLogger(Filetest.class.getName()).
            log(Level.SEVERE, null, ex);
    }
}
```

# ObjectInputStream

**ObjectInputStream** - для десериализации объектов в поток

```
FileInputStream fin = null;
try {
    fin = new FileInputStream(new File("fish.txt"));
    ObjectInputStream oin = new ObjectInputStream(fin);
    Fish f = (Fish) oin.readObject();
    System.out.println(f);
} catch (FileNotFoundException ex) {
    Logger.getLogger(Filetest.class.getName()).
        log(Level.SEVERE, null, ex);
} catch (IOException ex) {
    Logger.getLogger(Filetest.class.getName()).
        log(Level.SEVERE, null, ex);
} catch (ClassNotFoundException ex) {
    Logger.getLogger(Filetest.class.getName()).
        log(Level.SEVERE, null, ex);
} finally {
    try {
        fin.close();
    } catch (IOException ex) {
        Logger.getLogger(Filetest.class.getName()).
            log(Level.SEVERE, null, ex);
    }
}
```

# BufferedInputStream и BufferedOutputStream

# BufferedOutputStream

- Конструктор **BufferedOutputStream(OutputStream,int)** создает объект **BufferedOutputStream** с массивом **buffer** заданного размера.
- Конструктор **BufferedOutputStream(OutputStream)** создает объект **BufferedOutputStream** с массивом **buffer** размера по умолчанию (size = 8192).

```
String text = "This lines of text should be written  
              in file\r\n"  
    + "using buffered stream.\r\n"  
    + "Just one more line.\r\n";  
try(FileOutputStream out=new FileOutputStream("notes.txt");  
    BufferedOutputStream bos =  
        new BufferedOutputStream(out))  
{  
    byte[] buffer = text.getBytes();  
  
    bos.write(buffer, 0, buffer.length);  
}  
catch(IOException ex){  
    System.out.println(ex.getMessage());  
}
```

# BufferedInputStream

Конструктор **BufferedInputStream(InputStream,int)** создает объект **BufferedInputStream** с массивом **buffer** указанного размера.

Конструктор **BufferedInputStream(InputStream)** создает объект **BufferedInputStream** с массивом **buffer** размера по умолчанию (size = 8192).

```
try(FileInputStream fin =
    new FileInputStream(new File("notes.txt"));
    BufferedInputStream bis =
        new BufferedInputStream(fin)) {
    int c;
    while((c=bis.read())!=-1){
        System.out.print((char)c);
    }
}
catch(Exception e){
    System.out.println(e.getMessage());
}
```

# Символьные потоки ввода-вывода



# Reader

абстрактный класс, определяющий символьный потоковый ввод. В случае ошибок все методы класса передают исключение `IOException`.

Существует 3 основных *read()* метода:

- ***int read()*** - возвращает представление очередного доступного символа во входном потоке в виде целого числа.
- ***int read(char[] buffer)*** - пытается прочесть максимум *buffer.length* символов из входного потока в массив *buffer*. Возвращает количество символов, в действительности прочитанных из потока.
- ***int read(char[] buffer, int offset, int length)*** - пытается прочесть максимум *length* символов, расположив их в массиве *buffer*, начиная с элемента *offset*. Возвращает количество реально прочитанных символов.
- ***close()*** – метод закрывает поток.



# Writer

абстрактный класс, определяющий символьный потоковый вывод. В случае ошибок все методы класса передают исключение `IOException`.

Существуют 5 основных `write()`-метода:

- **`void write(int c)`** – записывает один символ в поток.
- **`void write(char[] buffer)`** – записывает массив символов в поток.
- **`void write(char[] buffer, int offset, int length)`** – записывает в поток подмассив символов длиной `length`, начиная с позиции `offset`.
- **`void write(String aString)`** – записывает строку в поток.
- **`void write(String aString, int offset, int length)`** – записывает в поток подстроку символов длиной `length`, начиная с позиции `offset`.

# FileReader

Класс **FileReader**, производный от класса *Reader*, можно использовать для чтения содержимого файла. В конструкторе класса нужно указать либо путь к файлу, либо объект типа *File*.

| № | Конструктор и описание  |
|---|---|
| 1 | <b>FileReader(File file)</b><br>Конструктор создает новый FileReader с учетом файла, который требуется прочитать. |
| 2 | <b>FileReader(FileDescriptor fd)</b><br>Конструктор создает новый FileReader, с учетом FileDescriptor для чтения. |
| 3 | <b>FileReader(String fileName)</b><br>Конструктор создает новый FileReader, учитывая имя файла для чтения.        |

```
// Создание объекта FileReader
FileReader fr = new FileReader(file);
char [] a = new char[200]; // Количество символов, которое будем считывать
fr.read(a); // Чтение содержимого в массив

for(char c : a)
    System.out.print(c); // Вывод символов один за другим
fr.close();
```

[FileDescriptor](#)

# FileWriter

Класс **FileWriter** создаёт объект класса, производного от класса *Writer*, который вы можете применять для записи файла. Создание объекта не зависит от наличия файла, он будет создан в случае необходимости. Если файл существует и он доступен только для чтения, то передаётся исключение *IOException*.

| № | Конструктор и описание  |
|---|---|
| 1 | <b>FileWriter(File file)</b><br>Конструктор создает объект <code>FileWriter</code> для объекта <code>File</code> .  |
| 2 | <b>FileWriter(File file, boolean append)</b><br>Конструктор создает объект <code>FileWriter</code> , заданный объектом <code>File</code> с логическим значением, указывающим, следует ли добавлять записанные данные. |
| 3 | <b>FileWriter(FileDescriptor fd)</b><br>Конструктор создает объект <code>FileWriter</code> , связанный с данным файловым дескриптором.  |
| 4 | <b>FileWriter(String fileName)</b><br>Конструктор создает объект <code>FileWriter</code> , учитывая имя файла.  |
| 5 | <b>FileWriter(String fileName, boolean append)</b><br>Конструктор создает объект <code>FileWriter</code> с именем файла с логическим значением, указывающим, следует ли добавлять записанные данные.                  |

```
// Создание объекта FileWriter
FileWriter writer = new FileWriter(file);

// Запись содержимого в файл
writer.write("Это простой пример,\n в котором мы осуществляем запись");
writer.flush();
writer.close();
```

# FileWriter

Класс **FileWriter** создаёт объект класса, производного от класса *Writer*, который вы можете применять для записи файла. Создание объекта не зависит от наличия файла, он будет создан в случае необходимости. Если файл существует и он доступен только для чтения, то передаётся исключение *IOException*.

| № | Конструктор и описание  |
|---|---|
| 1 | <b>FileWriter(File file)</b><br>Конструктор создает объект <code>FileWriter</code> для объекта <code>File</code> .  |
| 2 | <b>FileWriter(File file, boolean append)</b><br>Конструктор создает объект <code>FileWriter</code> , заданный объектом <code>File</code> с логическим значением, указывающим, следует ли добавлять записанные данные. |
| 3 | <b>FileWriter(FileDescriptor fd)</b><br>Конструктор создает объект <code>FileWriter</code> , связанный с данным файловым дескриптором.  |
| 4 | <b>FileWriter(String fileName)</b><br>Конструктор создает объект <code>FileWriter</code> , учитывая имя файла.  |
| 5 | <b>FileWriter(String fileName, boolean append)</b><br>Конструктор создает объект <code>FileWriter</code> с именем файла с логическим значением, указывающим, следует ли добавлять записанные данные.                  |

```
// Создание объекта FileWriter
FileWriter writer = new FileWriter(file);

// Запись содержимого в файл
writer.write("Это простой пример,\n в котором мы осуществляем запись");
writer.flush();
writer.close();
```

# FileWriter

Класс **FileWriter** создаёт объект класса, производного от класса *Writer*, который вы можете применять для записи файла. Создание объекта не зависит от наличия файла, он будет создан в случае необходимости. Если файл существует и он доступен только для чтения, то передаётся исключение *IOException*.

| № | Конструктор и описание  |
|---|---|
| 1 | <b>FileWriter(File file)</b><br>Конструктор создает объект <code>FileWriter</code> для объекта <code>File</code> .  |
| 2 | <b>FileWriter(File file, boolean append)</b><br>Конструктор создает объект <code>FileWriter</code> , заданный объектом <code>File</code> с логическим значением, указывающим, следует ли добавлять записанные данные. |
| 3 | <b>FileWriter(FileDescriptor fd)</b><br>Конструктор создает объект <code>FileWriter</code> , связанный с данным файловым дескриптором.  |
| 4 | <b>FileWriter(String fileName)</b><br>Конструктор создает объект <code>FileWriter</code> , учитывая имя файла.  |
| 5 | <b>FileWriter(String fileName, boolean append)</b><br>Конструктор создает объект <code>FileWriter</code> с именем файла с логическим значением, указывающим, следует ли добавлять записанные данные.                  |

```
// Создание объекта FileWriter
FileWriter writer = new FileWriter(file);

// Запись содержимого в файл
writer.write("Это простой пример,\n в котором мы осуществляем запись");
writer.flush();
writer.close();
```

# InputStreamReader

Класс `Java.io.InputStreamReader` представляет собой мост между байтовыми потоками и символьными потоками. Он считывает байты и декодирует их в символы с использованием указанной кодировки.

## Конструкторы

**`InputStreamReader (InputStream in)`** - Это создает `InputStreamReader`, который использует набор символов по умолчанию.

**`InputStreamReader (InputStream in, Charset cs)`**

Это создает `InputStreamReader`, который использует данный набор символов.

**`InputStreamReader (InputStream in, CharsetDecoder dec)`**

Это создает `InputStreamReader`, который использует данный декодер `charset`.

**`InputStreamReader (InputStream in, String charsetName)`**

Это создает `InputStreamReader`, который использует именованную кодировку.

**`void close()`**

Этот метод закрывает поток и освобождает любые системные ресурсы, связанные с ним.

**`String getEncoding()`**

Этот метод возвращает имя кодировки символов, используемой этим потоком.

**`int read()`**

Этот метод читает один символ.

**`int read(char [] cbuf, int offset, int length)`**

Этот метод читает символы в часть массива

# InputStreamReader

Класс `Java.io.InputStreamReader` представляет собой мост между байтовыми потоками и символьными потоками. Он считывает байты и декодирует их в символы с использованием указанной кодировки.

## Конструкторы

**`InputStreamReader (InputStream in)`** - Это создает `InputStreamReader`, который использует набор символов по умолчанию.

**`InputStreamReader (InputStream in, Charset cs)`**

Это создает `InputStreamReader`, который использует данный набор символов.

**`InputStreamReader (InputStream in, CharsetDecoder dec)`**

Это создает `InputStreamReader`, который использует данный декодер `charset`.

**`InputStreamReader (InputStream in, String charsetName)`**

Это создает `InputStreamReader`, который использует именованную кодировку.

**`void close()`**

Этот метод закрывает поток и освобождает любые системные ресурсы, связанные с ним.

**`String getEncoding()`**

Этот метод возвращает имя кодировки символов, используемой этим потоком.

**`int read()`**

Этот метод читает один символ.

**`int read(char [] cbuf, int offset, int length)`**

Этот метод читает символы в часть массива