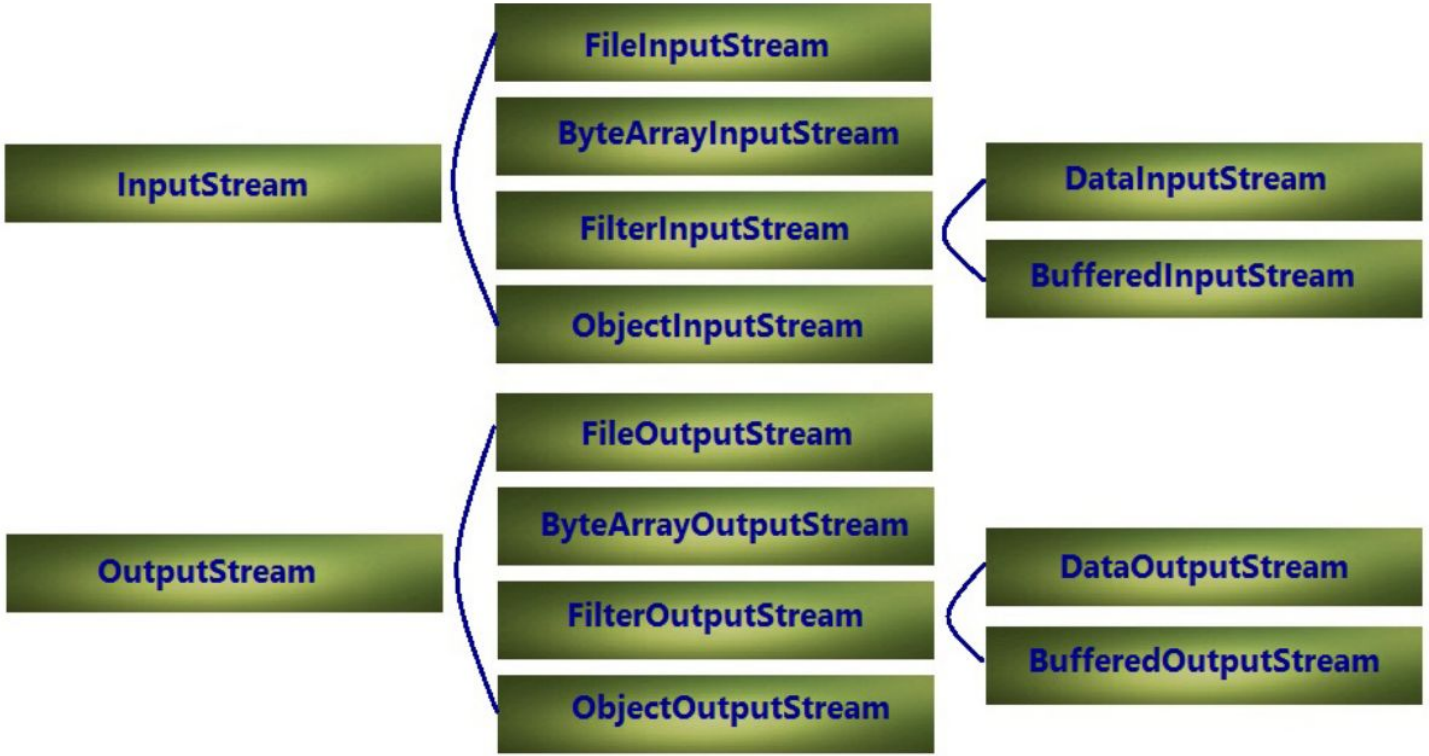
Работа с файлами

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
l**ong length():** возвращает размер файла в байтах
l**ong 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 дален 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());
}
```

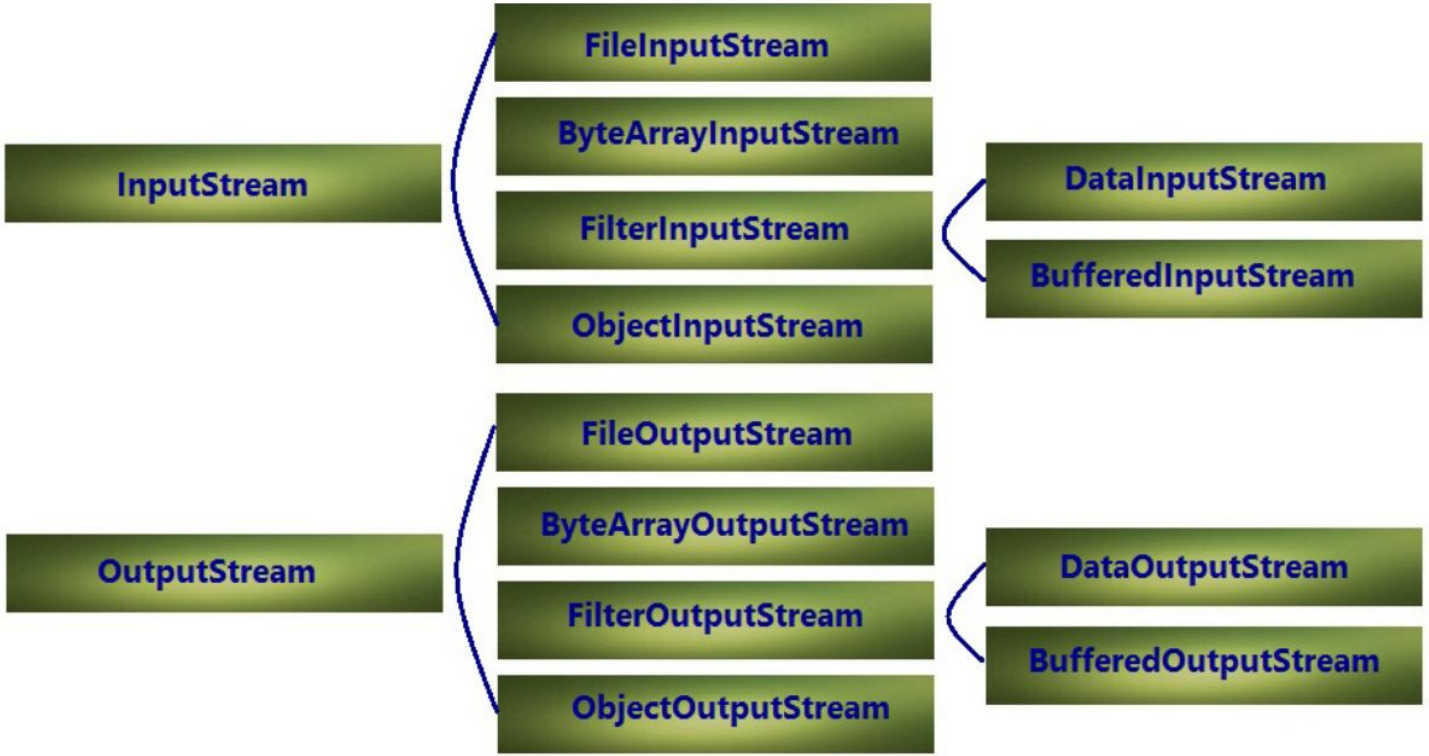
Полезно знать

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.

r**ead(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. озвращает 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)

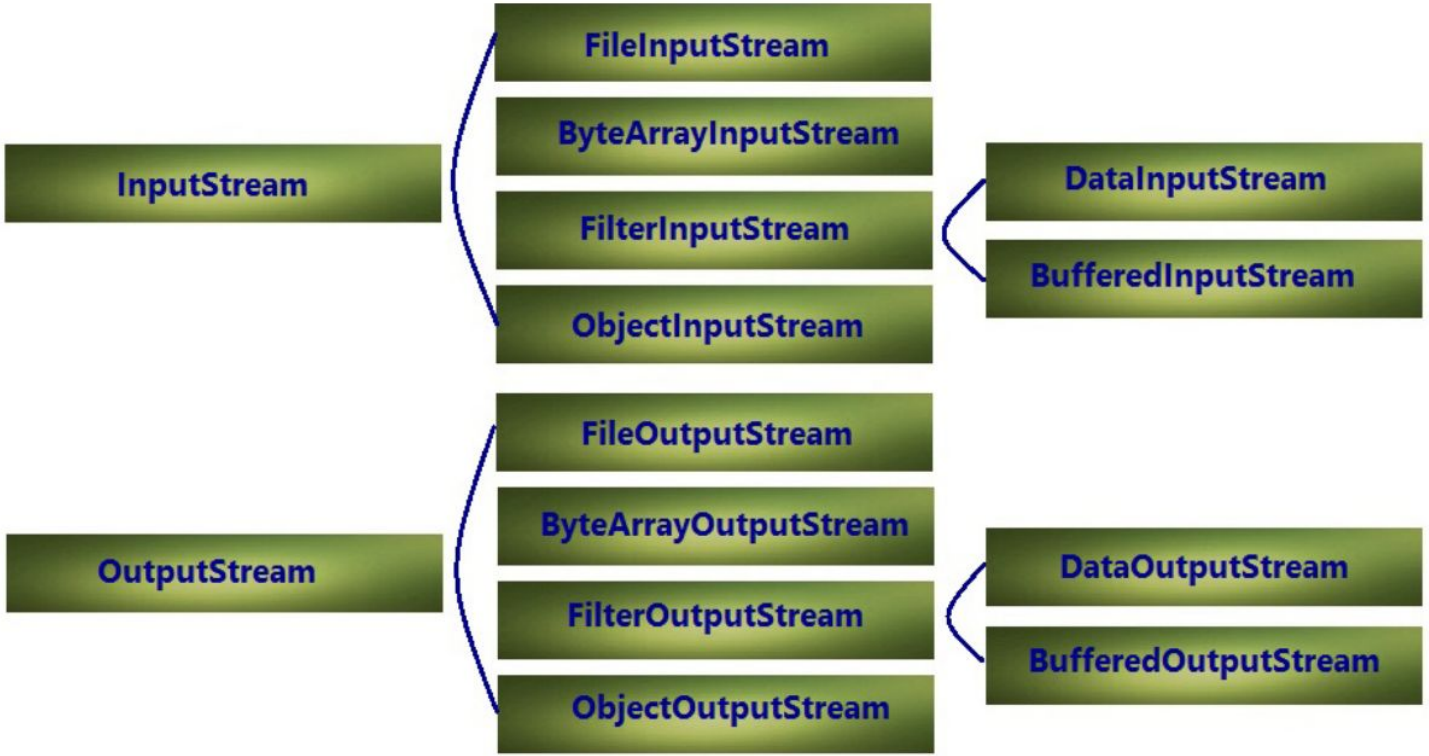
append - если он равен true, то данные дозаписываются в конец файла, а при false - файл полностью перезаписывается
Например:

```
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:
  Закрытие 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());
}
```

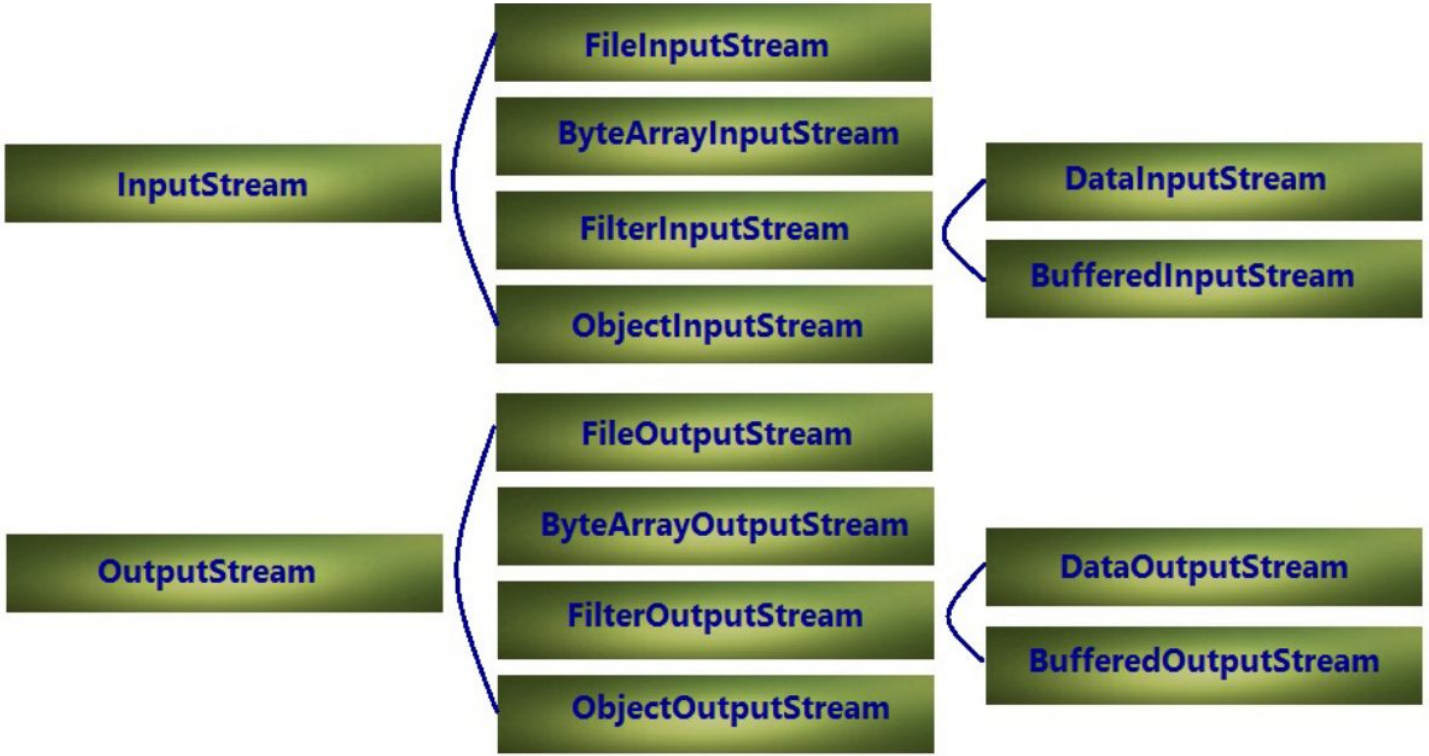
пример

Методы ByteArrayOutputStream

№	Метод и описание
1	**public void reset()** Метод сбрасывает количество действительных байт в выходном потоке байтового массива до нуля, поэтому все накопленное на выходе будет сброшено.
2	**public byte[] toByteArray()** 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	**public String toString()** Преобразует содержимое буфера в строку. Перевод будет выполняться в соответствии с кодировкой установленной по умолчанию. Возвращает строку, переведенную из содержимого буфера.
4	**public void write(int w)** Запись указанного массива в выходной поток.
5	**public void write(byte []b, int of, int len)** Запись len количества байтов, начиная смещение с of.
6	**public void writeTo(OutputStream outSt)** 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);
    }
}
```