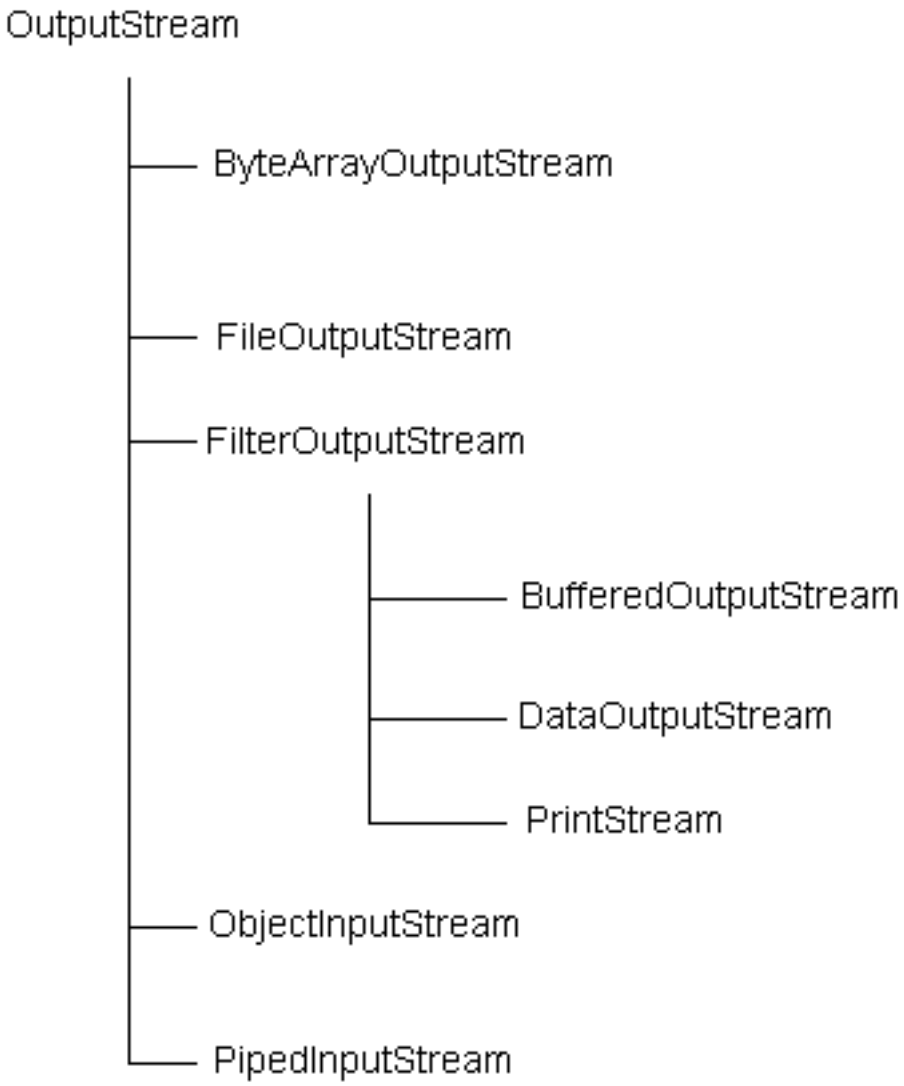
Конспект лекций по Java. Занятие 15

В.Фесюнов

1. Ввод/вывод (I/O) в Java (продолжение)

1.1. Иерархия OutputStream



Принципы построения иерархии **OutputStream** те же, что и в **InputStream** . Т.е. вся основная функциональность определяется абстрактным базовым классом `OutputStream` . Рассмотрим основные методы этой иерархии, определенные в классе `OutputStream` .

```
public abstract void write(int b) throws IOException
```

Записывает в поток один байт — младший байт числа, заданного параметром b.

```
public void write(byte[] b) throws IOException
```

Записывает в поток массив байт.

```
public void write(byte[] b, int off, int len)
                  throws IOException
```

Модификация предыдущего метода, позволяющая задать смещение в массиве и длину записываемой порции.

```
public void flush() throws IOException
```

Выталкивает из буфера в поток все, что в нем накоплено.

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

Закрывает поток.

Класс **ByteArrayOutputStream** .

Позволяет заполнять массив байт операциями вывода в поток.

Класс **FileOutputStream** .

Это основной класс для работы с файлами. Имеет такие основные конструкторы:

```
public FileOutputStream(File file) throws FileNotFoundException
```

```
public FileOutputStream(String name) throws FileNotFoundException
```

```
public FileOutputStream(String name, boolean append)
       throws FileNotFoundException
```

Смысл конструкторов, кроме, возможно, последнего понятен из их описания. Но имеется несколько нюансов.

- При открытии файла на запись, если файл не существует, то он создается.
- Если файл существует, то он будет полностью обновлен. Т.е. если открыть и сразу

закрыть файл, то реальный файл на диске будет нулевой длины.
- Исключением из предыдущего правила является последний из конструкторов. Если в нем в качестве третьего параметра (*append*) указать **true** , то файл будет дописываться.

Какой-либо дополнительной функциональности по сравнению с базовым классом класс `FileOutputStream` не добавляет.

Класс **BufferedOutputStream** .

Служит для организации более эффективного буферизованного вывода в поток.

Класс **DataOutputStream** .

Содержит много хороших методов, но на практике, чаще всего, бесполезен ввиду того, что работает с кодировкой Unicode.

Класс **PrintStream** .

Содержит много полезных дополнительных методов. Это методы print и println для всех базовых типов. Наиболее часто используются следующие из этих методов

```
public void print(String x)

public void println(String x)
```

и другие, в принципе, не нужны, т.к. оператор вида

```
println("строка"+число);
```

позволяет вывести и числа. А

```
println("строка"+объект);
```

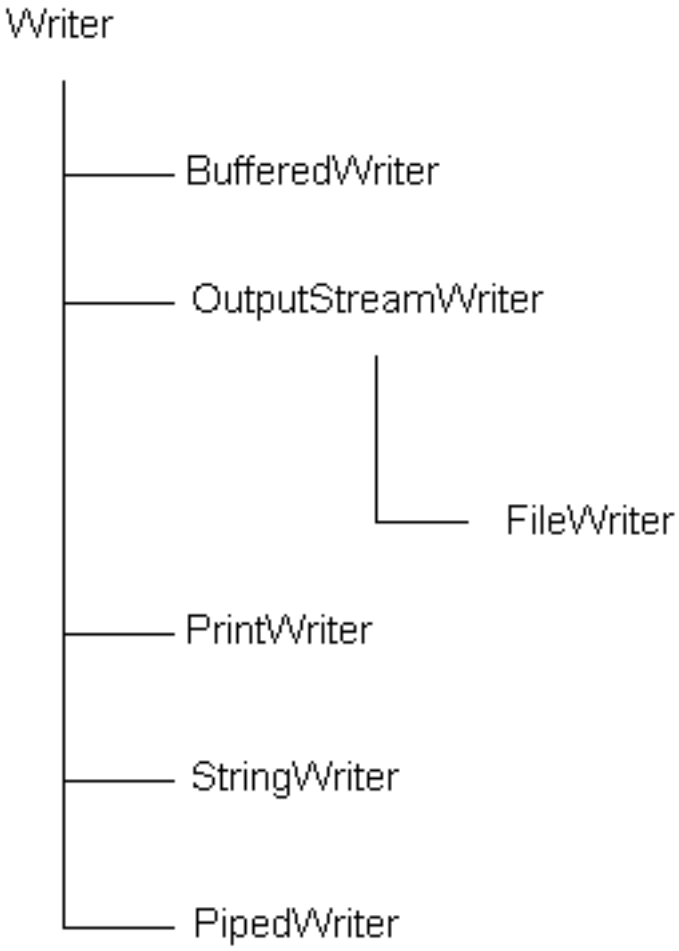
формирует строку из объекта при помощи метода `toString` .

- Существуют два стандартных (предопределенных) потока вывода типа `PrintStream: System.out` и `System.err` . Оба они по умолчанию направляют информацию на консоль, но их можно перенаправить в файлы.

Класс **ObjectOutputStream** мы будем рассматривать позднее.

Класс **PipedOutputStream** предназначен для передачи информации между программами через *каналы* (*pipes*).

1.2. Иерархия Writer



Как и в случае с иерархией **Reader** иерархия **Writer** сама по себе ничего не добавляет к функциональности иерархии **OutputStream** и отличается от нее всего несколькими нюансами.

Так, вместо `ByteArrayOutputStream` в данной иерархии присутствует более

полезный класс `StringWriter` , позволяющий выводить информацию в строку.

Практическое использование иерархии Writer

Основным способом создания потока вывода в файл является

```
PrintWriter fout = new PrintWriter(
     new BufferedWriter(new FileWriter(имя_файла)));
```

После создания такого потока в него можно выводить информацию при помощи методов `print` и `println` .

2. Развернутый пример

Рассмотрим содержательный пример, в котором используются те средства, что мы рассмотрели на данном и предыдущем занятии.

Программа *WordConverter.java* предназначена для проведения поиска и замены слов в тексте. Она принимает три параметра вызова: имя файла, строку для поиска и строку для замены. В результате ее работы создается файл с именем **res.txt** , содержащий результаты замены и печатается краткий протокол.

Файл *WordConverter.java* :

```
import java.util.*;
import java.io.*;

public class WordConverter {

  public static void main(String args[]) {

    if( args.length < 2 ) {
      System.out.println("Формат вызова:");
      System.out.println("java WordConverter <имя файла>
      <строка для поиска> <строка для замены>");
      return;
    }
    String thisLine;
    BufferedReader fin = null;
```

```
    PrintWriter fout = null;

    try {
       fin = new BufferedReader(new FileReader(args[0]));
       fout = new PrintWriter(new BufferedWriter(new FileWriter("res.txt")));
       int lineCnt = 0, replCnt = 0;
       for (; (thisLine = fin.readLine()) != null; lineCnt++) {
           StringTokenizer st = new StringTokenizer(thisLine, ,
            " \t,;+-[]().*&=~!|:?<>{}"true);
           while (st.hasMoreTokens()) {
               String token = st.nextToken();
               if ( token.equals(args[1]) ) {
                   fout.print( args[2] );
                   replCnt++;
               } else
                   fout.print( token );
           }
           fout.println();
       }
       System.out.println();
       System.out.println("=== Результаты: ===");
       System.out.println("    Строк в файле    : "+lineCnt);
       System.out.println("    Произведено замен: "+replCnt);
    } catch (FileNotFoundException e) {
        System.out.println("Файл " + args[0] + " не найден");
    } catch (Exception e) {
        e.printStackTrace();
    }  finally {
        if ( fin != null ) {
            try {
                fin.close();         // !!! Закрыть файл
            } catch ( IOException ex ) {
                System.out.println("Ошибка закрытия файла " + args[0]);
                System.out.println("Error: " + ex);
            }
            fin = null;
        }
        if ( fout != null ) {
            fout.close();         // !!! Закрыть файл
            fout = null;
```

```
        }
    }
```

Оттранслируем и запустим на выполнение данную программу. Для примера можно взять в качестве исходного файла текст самой программы WordConverter.java, в качестве строки поиска "thisLine", в качестве строки для замены "theLine".

Здесь использованы те средства и приемы, которые мы рассмотрели на текущем и прошлом занятии. В частности, класс `StringTokenizer` , рассмотренный в 14-м занятии.

Продолжим изучение библиотеки ввода/вывода Java.

3. Класс RandomAccessFile

Буквально: файл прямого доступа. Данный класс обеспечивает как операции чтения файла, так и операции записи в файл. Т.е. он объединяет возможности, заложенные в иерархиях Reader и Writer.

Кроме того, он имеет методы, позволяющие выполнять позиционирование на заданный байт файла.

Рассмотрим конструкторы класса **RandomAccessFile** :

```
public RandomAccessFile(File file, String mode)
   throws FileNotFoundException

public RandomAccessFile(String name, String mode)
   throws FileNotFoundException
```

Они отличаются от аналогичных конструкторов классов `FileReader` и `FileWriter` наличием параметра mode. Этот параметр может быть равен либо "r" (только чтение), либо "rw" (чтение и запись).

Класс **RandomAccessFile** имеет большое количество методов как для чтения, так и для записи данных. В частности, он имеет те же методы чтения, что и в `InputStream` или `Reader` .

Кроме того, в нем есть метод `readLine` , аналогичный одноименному методу из

`BufferedReader` , а также методы из `DataInputStream` : по одному методу вида `readXXX` для каждого базового типа, например,

```
public final float readFloat() throws IOException
```

Этот метод позволяет считать число с плавающей точкой.

Методы записи такие же, как `OutputStream` или `Writer` , плюс методы из `DataOutputStream` .

К сожалению эксперименты с методам `readLine` и методами из `DataInputStream` и `DataOutputStream` приводят к выводу о наличии проблем при их практическом использовании. Метод `readLine` имеет проблемы с национальными кодировками, а методы из `DataInputStream` и `DataOutputStream` работают с кодировкой *Unicode* .

Кроме этих методов `RandomAccessFile` имеет ряд методов для обеспечения произвольного доступа. Это самая важная часть функциональности, которую обеспечивает класс `RandomAccessFile` .

```
public long getFilePointer() throws IOException
```

Возвращает текущую позицию в файле.

```
public long length() throws IOException
```

Возвращает длину файла.

```
public void seek(long pos) throws IOException
```

Устанавливает новую текущую позицию в файле.

```
public void setLength(long newLength)
       throws IOException
```

Устанавливает новую длину файла. При этом происходит либо усечение, либо расширение файла.

```
public int skipBytes(int n) throws IOException
```

Пропускает n байт файла. Возвращает реально пропущенное число байт (оно будет меньше n, если файл недостаточно длинный для пропуска n байт).

4. Класс File

Это очень важный класс. Без него нельзя было бы удалить существующий файл, создать каталог (папку), узнать какие файлы есть в данном каталоге.

Класс `File` имеет следующие конструкторы:

```
public File(String pathname)

public File(String parent, String child)

public File(File parent, String child)
```

- Конструкторы класса `File` не порождают `IOException` . При создании экземпляра класса `File` реальный файл с указанным именем может не существовать. При этом он не создается при создании объекта `File` . Реальная работа с файлом выполняется методами данного класса.

Рассмотрим наиболее важные методы данного класса.

```
public boolean exists()
```

Проверяет, существует ли данный файл.

```
public boolean canRead()
```

Проверяет, можно ли читать данный файл. Современные ОС позволяют устанавливать различные полномочия доступа к файлу для разных пользователей. В частности, можно запретить чтение файла тем или иным пользователям. Данный метод возвращает false также в том случае, если файл не существует.

```
public boolean canWrite()
```

Проверяет, можно ли писать в файл.

```
public boolean isDirectory()
```

Является ли данный файл каталогом (папкой).

```
  public boolean isFile()
```

Является ли данный файл обычным файлом (не каталогом).

```
  public boolean delete()
```

Удаляет файл, если он существует. Файл может быть и каталогом. Тогда он удаляется только в том случае, если он пуст.

```
  public File[] listFiles()
```

Возвращает массив файлов в каталоге (массив объектов класса File).

```
  public boolean mkdir()
```

Создает пустой каталог.

Есть еще целый ряд полезных методов в данном классе (см. документацию).

5. Практическая работа

1. Рассмотрим программу для удаления всех файлов в указанном каталоге (каталог задается параметром вызова программы).

```
// DirClean.java: удаление всех файлов в указанном каталоге

import java.io.*;

public class DirClean {

    public static void main(String args[]) {
//--- 1. Проверим параметры вызова программы
        if ( args.length == 0 ) {
            System.out.println(" Формат вызова: java DirClean имя_каталога");
            return;
        }
//--- 2. Создадим объект File для данного каталога и проверим, что
//       — он существует;
```

```
//       — что это действительно каталог.
        File dir = new File(args[0]);
        if ( !dir.isDirectory() ) {
            System.out.println("Каталог "+args[0]+" не существует или
"+
                               "не является каталогом");
            return;
        }
//--- 3. Получим массив всех файлов в данном подкаталоге
        File[] fList = dir.listFiles();
        int ftotal = fList.length;
        if ( ftotal == 0 ) { // нечего удалять
            System.out.println("Каталог "+args[0]+" уже пуст");
            return;
        }
//--- 4. Запрос к пользователю на подтверждение операции удаления
        System.out.println(" Подтвердите удаление "+ftotal+
                           " файлов в каталоге "+args[0]+" Y/N");
        try {
            byte r[] = new byte[1];
            r[0] = (byte)System.in.read();
            char resp = (new String(r)).charAt(0);
            if ( resp == 'y' || resp == 'Y' ) {
                for ( int i = 0; i < ftotal; i++ )
                    fList[i].delete();
                System.out.println(" Удалено " + ftotal + " файлов");
            }
        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }

}
```

В данной программе используются следующие методы класса File: `isDirectory, listFiles, delete` . Нужно разобраться в этой программе и написать ее вариант, который удаляет только те файлы, которые доступны и на чтение и на запись.

2. (Самостоятельная работа).

Изменим программу *WordConverter.java* так, чтобы она в результате своей работы порождала бы не новый файл *res.txt* , а модифицировала существующий. При этом исходный файл она должна сохранить с расширением *bak* .

Изучим проблему. Во-первых, мы не можем сразу писать в тот же файл, из которого мы читаем. Поэтому для вывода нам понадобиться какой-то файл. После окончания обработки исходного файла мы должны его переименовать в файл с расширением *bak* (нужно не забыть закрыть его перед переименованием). Когда мы переименуем исходный файл, мы можем переименовать выходной файл.

Таким образом, нам потребуется.

a. Возможность создания временного файла.

b. Возможность переименования файлов.

Все это есть в классе `File` . Во-первых, есть два метода `createTempFile` . Лучше использовать второй из них.

```
public static File createTempFile(String prefix,
       String suffix, File directory)
       throws IOException
```

Иначе могут возникнуть проблемы с переименованием файла, если временный файл будет создан физически не на том же диске, что и исходный. Но в этом случае нам нужно иметь параметр *directory* — каталог, в котором находится исходный файл. Для получения каталога можно использовать метод

```
public File getParentFile()
```

Возможность переименования файлов тоже имеется в классе File — это метод

```
public boolean renameTo(File dest)
```

Теперь осталось только все это собрать воедино и получить работающую программу.