

# Лекция 7. Java I/O.

Лекции по Java SE

Колесников Сергей, инженер-разработчик

NetCracker



**NetCracker**<sup>®</sup>

# План лекции

- Java IO
  - Что это такое
  - Работа с потоками ввода\вывода
    - Символьные и байтовые потоки
    - Упаковка потоков
    - Предопределенные потоки
  - Java NIO
  - Работа с файловой системой
    - `java.io.File`
    - `java.nio.file.Path`
- Разбор примеров в IDE

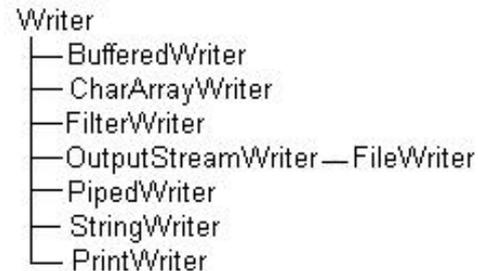
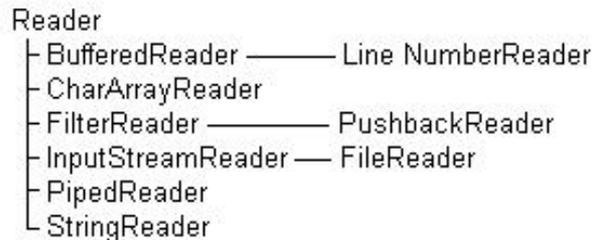
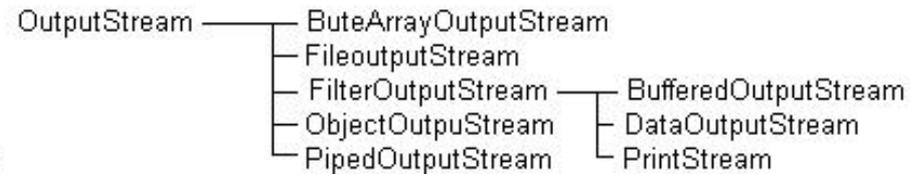
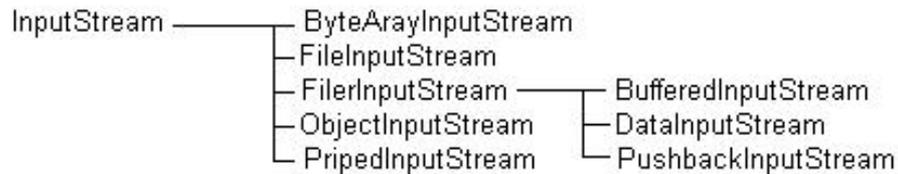
# 1. Обмен данными

- Разнообразие задач
  - Сетевые приложения
  - Обработка ввода с клавиатуры
  - Запись и считывание данных из файлов
  - И еще множество других
- Выделение системы ввода/вывода
  - Широкий спектр источников и получателей
  - Различные форматы передачи данных
  - Ввод/вывод не должен зависеть от платформы

## 2. Потoki данных

- Основа – **поток**. Поток – абстракция, производящая или потребляющая информацию
- Java I/O System связывает поток с физическим устройством
- Потoki делятся на **входные** и **выходные** – Input\* и Output\*
- Потoki бывают **байтовые** и **символьные**
  - Байтовые потоки для данных в двоичном коде
  - Символьные потоки для символов в Unicode

# 3. Иерархия потоков и пример



```
try {
    FileInputStream fis = new FileInputStream("file.bin");
    while (fis.available() > 0)
        System.out.println(fis.read());
} catch (FileNotFoundException e) {
    System.out.println("File not found");
} catch (IOException e) {
    System.out.println("IO Exception");
}
```

## 4. Байтовые потоки

- Базовые классы: `java.io.InputStream` и `java.io.OutputStream`
- Наиболее важные методы: `int read()`, `int read(byte[])` и `void write(byte[])`

<code>FileInputStream</code>	Читает из файла
<code>ByteArrayInputStream</code>	Читает из массива байтов
<code>DataInputStream</code>	Содержит методы чтения для стандартных данных java ( <code>readBoolean()</code> , <code>readInt()</code> , etc.)
<code>BufferedInputStream</code>	Буферизированный поток ввода
<code>PrintStream</code>	Поддерживает <code>print()</code> , <code>println()</code>
<code>PipedInputStream</code>	Канал ввода
<code>PushBackInputStream</code>	Возможна операция «unread», возвращающая 1 байт в поток
<code>RandomAccessFile</code>	i/o произвольного доступа
<code>SequenceInputStream</code>	Поток ввода - комбинация нескольких других <code>InputStream</code> , которые будут читаться последовательно, один за другим

\*Аналогично для  
`OutputStream`

# 5. СИМВОЛЬНЫЕ ПОТОКИ

- Базовые классы: `java.io.Reader` и `java.io.Writer`
- Наиболее важные методы: `int read(char[])` и `void write(char[])`

<code>BufferedReader</code>	Буферизированный поток ввода
<code>CharArrayReader</code>	Читает из массива символов
<code>FileReader</code>	Читает из файла
<code>LineNumberReader</code>	Считает строки
<code>PipedReader</code>	Канал ввода
<code>PrintWriter</code>	Поддерживает <code>print()</code> , <code>println()</code>
<code>PushBackReader</code>	Возвращает символ в поток ввода
<code>StringReader</code>	Читает из строки

\*Аналогично для  
Writer

## 6. Упаковка потоков (wrapping)

- Позволяет конвертировать байтовый поток в символьный

```
InputStream inStream = System.in;  
InputStreamReader inReader = new InputStreamReader(inStream);
```

- Позволяет изменить функциональность работы с потоком

```
FileReader fileReader = new FileReader("file.txt");  
BufferedReader bufReader = new BufferedReader(fileReader);  
String line = bufReader.readLine();//читает до разделителя строк ('\r' или  
'\n')
```

# 7. Предопределенные потоки

- Встроены в `java.lang.System`
- Байтовый поток ввода `System.in`
- Байтовый поток вывода `System.out`
- Байтовый поток вывода сообщений об ошибках `System.err`
- Класс `java.io.Console`
  - Доступ через `System`
  - Только для Java 6

```
public class ConsoleTest2 {  
    public static void main(String[] args) throws IOException {  
        Console con = System.console();  
        String login = con.readLine("login: ");  
        char[] password = con.readPassword("password: ");  
        con.printf("Hello %s!", login);  
    }  
}
```

## 8. Java.nio

- Java 1.4 and higher!
- Лучше производительность
- Buffers + Channels
- Неблокирующий ввод\вывод
- Selectors

## 9. Java.nio пример

- Чтение и запись

```
FileInputStream fin = new FileInputStream("readandshow.txt");  
FileChannel fc = fin.getChannel();
```

```
ByteBuffer buffer = ByteBuffer.allocate(1024);  
fc.read(buffer);
```

```
FileOutputStream fout = new FileOutputStream("writesomebytes.txt");  
FileChannel fc = fout.getChannel();
```

```
ByteBuffer buffer = ByteBuffer.allocate(1024);
```

```
for (int i=0; i<message.length; ++i) {  
    buffer.put( message[i] );
```

```
}
```

```
buffer.flip();
```

```
fc.write( buffer );
```

# 10. Java.nio блокировка файлов

- Пример блокировки

```
FileOutputStream fos = new FileOutputStream("file.txt");
FileLock fl = fos.getChannel().tryLock();

if (fl != null) {
    System.out.println("File is locked");
    fl.release();
    System.out.println("Lock is released");
}

fos.close
```

- Блокировка части файла

```
tryLock(long position, long size, boolean shared);
```

# 11. Работа с файловой системой

- За работу с файловой системой отвечает `java.io.File`
- `File file = new File("file.txt")` – создание инструмента для работы с файлом и директорией
- Инкапсулирует **платформенно-независимые** методы работы с файлами и директориями:
  - создание, переименование, удаление
  - проверка типа пути: файл или каталог
  - проверка атрибутов файлов и каталогов
  - проверка существования файлов и каталогов
- Определяет доступ лишь к метаданным файловой системы  
(для чтения и записи используем потоки!)

## 12. Java.nio.file.Path

- Еще один способ работы с файловой системой.
- Java 7 and higher!

```
Path path = Paths.get("index.html");
```

- Throws Exceptions

```
if (!file.delete()){  
    //What happens ?  
}
```

```
try {  
    path.delete();  
} catch (IOException e) {  
    //  
}
```

- Атрибуты специфичные для ОС

```
DosFileAttributeView dosView =  
path.getFileAttributeView(DosFileAttributeView.class,  
    LinkOption.NOFOLLOW_LINKS);  
if (dosView != null)  
    dosFileAttributes dos = dosView.readAttributes();
```

# 13. Java.nio.file.Path (продолжение)

- Checking directories for modifications

```
WatchService watcher = path.getFileSystem().newWatchService();

path.register(watcher,
    StandardWatchEventKind.ENTRY_CREATE,
    StandardWatchEventKind.ENTRY_MODIFY,
    StandardWatchEventKind.ENTRY_DELETE);

while (true) {
    WatchKey watchKey = watcher.take();

    for (WatchEvent event : watchKey.pollEvents()) {
        System.out.println(event.kind() + " : " + event.context());
    }

    watchKey.reset();
}
```