

Java Advanced

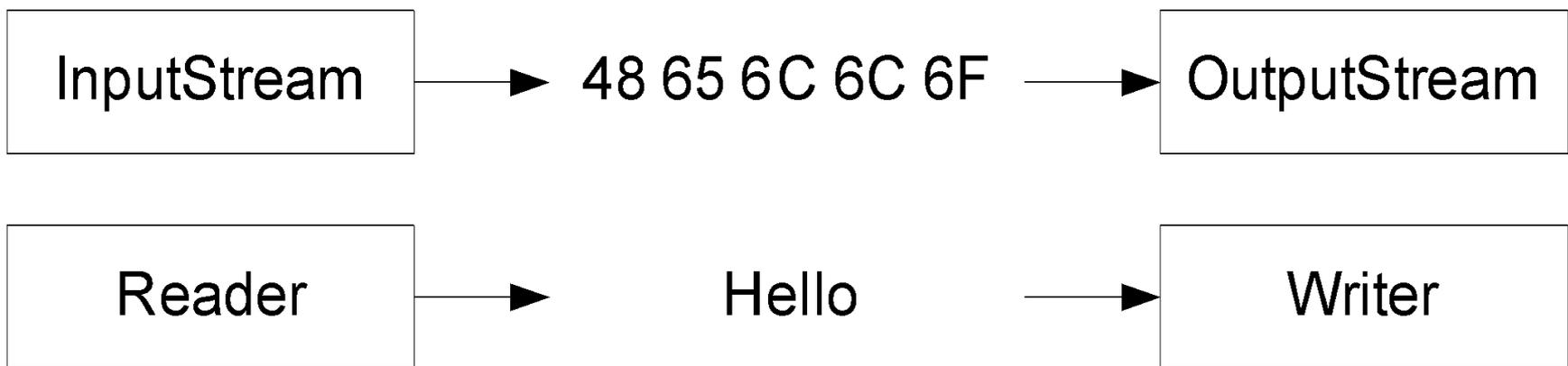
ВВОД-ВЫВОД

Содержание

1. Потоки ввода-вывода
2. Файловый ввод-вывод и конвертация потоков
3. Фильтрующие потоки
4. Дополнительные возможности потоков
5. Расширенный ввод-вывод
6. Дескрипторы файлов
7. Текстовый ввод
8. Ввод-вывод и исключения
9. Заключение

Ввод-вывод в Java

- Потоки ввода-вывода
- Пакет `java.io`

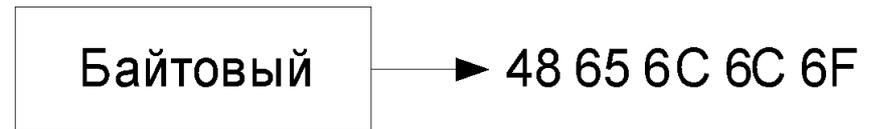


Часть 1

ПОТОКИ ВВОДА-ВЫВОДА

Виды потоков

- Направление
 - Ввод
 - Вывод
- Содержимое
 - Байтовые
 - Символьные



Классы потоков

	Байтовый	Символьный
Ввод	InputStream	Reader
Вывод	OutputStream	Writer

Исключительные ситуации

- Класс `IOException`
 - Корень иерархии исключений ввода-вывода
 - Бросается всеми операциями ввода/вывода
- Класс `EOFException`
 - Достигнут конец потока
- Класс `FileNotFoundException`
 - Файл не найден
- Класс `UnsupportedEncodingException`
 - Неизвестная кодировка

Потоки ввода

- Основные операции
 - `int read()` — чтение элемента
 - `read(T[] v)`, `read(T[] v, off, len)` — чтение элементов в массив
- Дополнительные операции
 - `skip(n)` — пропуск `n` элементов
 - `close()` — закрытие потока
- Пометки и возвраты
 - `mark(limit)` — пометка текущей позиции
 - `markSupported()` – поддерживается ли `mark()`
 - `reset()` — возврат к помеченной позиции

Потоки вывода

- Основные операции
 - `write(int v)` — запись элемента
 - `write(T[] v)` — запись массива элементов
 - `write(T[] v, off, len)` — запись части массива
- Дополнительные операции
 - `flush()` — запись буфера
 - `close()` — закрытие потока

Пример: ?

- ?

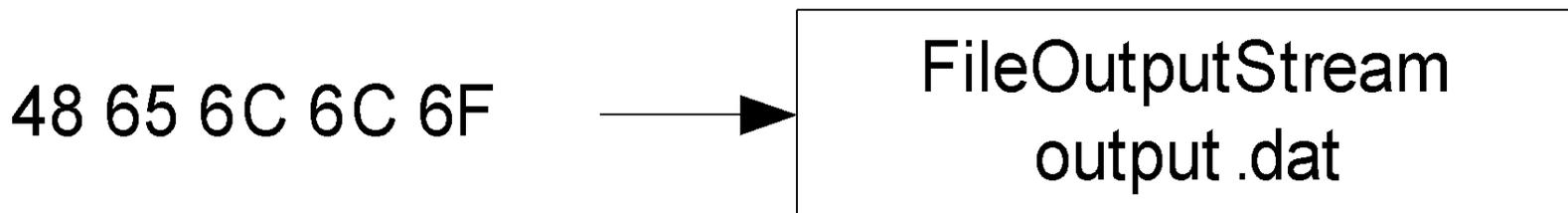
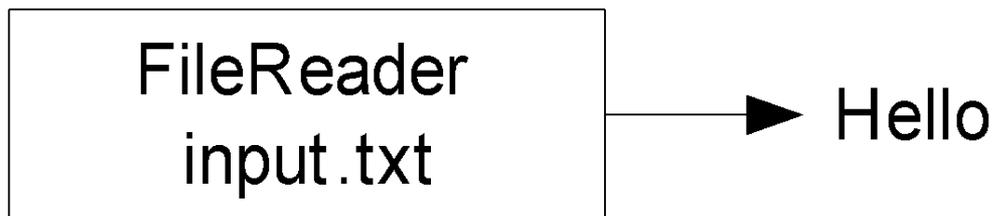
```
void copy(InputStream is, OutputStream os)
    throws IOException
{
    byte[] b = new byte[1024];
    int c = 0;
    while ((c = is.read(b)) >= 0) {
        os.write(b, 0, c);
    }
}
```

Часть 2

Файловый ввод-вывод и конвертация потоков

Классы файлового ввода-вывода

- Классы File*
 - FileInputStream
 - FileOutputStream
 - FileReader
 - FileWriter



Создание файловых потоков

- Для символьных потоков используется кодировка по умолчанию
- Для ввода/вывода
 - `File*(File file)` — по дескриптору
 - `File*(String file)` — по имени
- Для дописывания
 - `File*(File file, boolean append)` — по дескриптору
 - `File*(String file, boolean append)` — по имени

Пример: ?

- ?

```
Reader reader = new FileReader("input.txt");
Writer writer = new FileWriter("output.txt");
int c = 0;
while ((c = reader.read()) >= 0) {
    writer.write(Character.toUpperCase((char) c));
}
reader.close();
writer.close();
```

Байтовый поток □ СИМВОЛЬНЫЙ

- При чтении возможно преобразование байтового потока в СИМВОЛЬНЫЙ, с указанием кодировки
- Класс `InputStreamReader`
 - `InputStreamReader(InputStream, encoding?)`

Символьный поток □ байтовый

- При записи возможно преобразование символьного потока в байтовый, с указанием кодировки
- Класс `OutputStreamWriter`
 - `OutputStreamWriter(OutputStream, encoding?)`

Пример: ?

- ?

```
Reader reader = new InputStreamReader(  
    new FileInputStream("input.txt"), "Cp1251");  
Writer writer = new OutputStreamWriter(  
    new FileOutputStream("output.txt"), "Cp866");  
int c = 0;  
while ((c = reader.read()) >= 0) {  
    writer.write(c);  
}  
reader.close();  
writer.close();
```

Часть 4

Дополнительные ВОЗМОЖНОСТИ ПОТОКОВ

Эмуляция чтения

- Чтение производится из буфера в памяти, передаваемого конструктору
- Классы
 - `ByteArrayInputStream` – чтение из массива байт
 - `CharArrayReader` – чтение из массива СИМВОЛОВ
 - `StringReader` – чтение из строки

Эмуляция записи

- Запись производится в буфер в памяти, который доступен в любое время
- Классы
 - `ByteArrayOutputStream` – запись в массив байт (`toArray()`)
 - `CharArrayWriter` – запись в массив СИМВОЛОВ (`toString()`, `toCharArray()`)
 - `StringWriter` – запись в `StringBuffer` (`toString()`, `toStringBuffer()`)

Конкатенация потоков

- Несколько байтовых потоков можно конкатенировать
- Если первый из потоков закончился, производится чтение из второго и т.д.
- Класс `SequenceInputStream`
 - `SequenceInputStream(InputStream, InputStream)` – конкатенация двух потоков
 - `SequenceInputStream(Enumeration)` – конкатенация нескольких потоков

Вывод с подавлением ошибок

- Вывод осуществляется построчно, с подавлением ошибок
- Класс `PrintStream`, `PrintWriter`
 - `checkError()` – проверить, была ли ошибка
 - `print(...)` – запись без перевода строки
 - `println(...)` – запись с переводом строки
 - `System.out` – это `PrintStream`

Часть 5

Расширенный ввод- вывод

Расширенная запись данных

- Платформонезависимая запись примитивных типов и строк
- Интерфейс `DataOutput`
 - `writeT(T)` – запись примитивных типов
 - `writeUTF()` – запись строки в кодировке UTF-8
 - Кодировка модифицирована!
- Реализация
 - `DataOutputStream`

Расширенное чтение данных

- Платформонезависимое чтение примитивных типов и строк
- Интерфейс `DataInput`
 - `T readT()` – чтение примитивных типов
 - `readUnsignedByte()` / `readUnsignedShort()` – чтение беззнаковых целых
 - `readUTF()` – чтение строки в кодировке UTF-8
- Реализация
 - `DataInputStream`

Файлы с произвольным доступом

- Класс `RandomAccessFile`
 - Реализует `DataInput`, `DataOutput`
- Конструктор
 - `RandomAccessFile(file, mode)` – открыть файл в заданном режиме

Строка	Режим
<code>r</code>	Чтение
<code>w</code>	Запись
<code>rw</code>	Чтение и запись
<code>rws</code>	Синхронное чтение и запись

Дополнительные операции

- Методы
 - `length()` – получить размер файла
 - `setLength()` – установить размер файла
 - `getFilePointer()` – получить положение указателя
 - `seek(long)` – установить положения указателя

Часть 5

ТЕКСТОВЫЙ ВВОД

Scanner

- **Scanner** позволяет читать значения примитивных типов и строки из текстовых источников
- Входные данные – интерпретируются как текст
 - **Readable**
 - **InputStream [charset]**
 - **File [charset]**
 - **String**

Scanner – разделители

- Входные данные текст:
 - «123 abc -22»
– число, пробел, строка, пробел, число
- Текстовые данные разделяются с помощью *разделителя (delimiter)*
 - `delimiter()`
 - `useDelimiter()`
 - по умолчанию – «*whitespace*»
 - пробелы
 - табуляции
 - переводы строк

Scanner – чтение строк

- Методы чтения строк
 - `next()` – следующая строка, отделенная разделителем
 - `hasNext()`
 - `NoSuchElementException`
 - Реализует `Iterator<String>`
 - `nextLine()` – считать символы до конца строки
 - `hasNextLine()`

Scanner – чтение примитивов

- `T nextT()` – чтение значений примитивного типа
 - сначала считывается текст до разделителя
 - затем этот текст преобразуется в значение примитивного типа
 - `InputMismatchException`
- `nextT(int radix)` – чтение данных в системе счисления с заданным основанием
 - `nextByte()`, `nextShort()`, `nextInt()`, `nextLong()`, `nextBigInteger()`

Scanner – ?

```
Scanner scanner =  
    new Scanner("123 a -5\n abc 12a 546 df");  
while (scanner.hasNext()) {  
    if (!scanner.hasNextInt()) {  
        scanner.next();  
    } else {  
        System.out.println(scanner.nextInt());  
    }  
}
```

Scanner – другие методы

- Кодировки
 - `locale()`
 - `useLocale()`
- Система счисления
 - `radix()`
 - `useRadix()`
- Исключения
 - `IOException` подавляется
 - `ioException()` – последнее исключение ввода-вывода, `null`, если исключения не было

Часть 6

Дескрипторы файлов

Дескрипторы файлов

- Позволяют осуществлять манипуляции с файлами
- Класс `File`
- Создание дескриптора по имени
 - `File(pathname)` – абсолютный или относительный путь
- В дескриптора по имени и директории
 - `File(File dir, name)`
 - `File(String dir, name)`

Разделители

- `separator` / `separatorChar` – платформозависимый разделитель директорий
- `pathSeparator` / `pathSeparatorChar` – платформозависимый разделитель в файлах и директорий в путях

Операции с дескрипторами

- Получение информации
 - `getName()` – имя
 - `getPath()` – путь
 - `getAbsolutePath()` – абсолютный путь
 - `getAbsoluteFile()` – абсолютный дескриптор
 - `getCanonicalPath()` – канонический путь
 - `getCanonicalFile()` – канонический дескриптор
- Определение родителя
 - `String getParent()` – как строки
 - `File getParentFile()` – как дескриптора

Операции с файлами (1)

- Проверка типа
 - `isFile()` – является ли файлом
 - `isDirectory()` – является ли директорией
 - `isHidden()` – является ли скрытым
- Получение информации о файле
 - `exist()` – проверка существования
 - `length()` – длина файла
 - `lastModified()` – время последней модификации

Операции с файлами (2)

- Создание
 - `mkdir()` – создать одну директорию
 - `makedirs()` – создать все директории
 - `createNewFile()` – создать пустой файл
- Удаление
 - `delete()` – удалить немедленно
 - `deleteOnExit()` – удалить после завершения
- Переименование / перенос
 - `renameTo(file)` – переименовать / перенести в заданное место

Листинг директории

- Листинг всех файлов
 - `String[] list()` – получить имена файлов
 - `File[] listFiles()` – получить дескрипторы файлов
- Листинг по критерию
 - `String[] list(FileNameFilter)` – получить имена файлов
 - `File[] listFiles(FileFilter)` – получить дескрипторы файлов

Часть 7

Ввод-вывод и исключения

Обычная обработка исключений

```
Reader reader = new FileReader("input.txt");
try {
    // Операции с файлом
} finally {
    if (reader != null) {
        reader.close();
    }
}
```

Надежная обработка исключений

```
Reader reader = new FileReader("input.txt");
try {
    // Операции с файлом
    reader.close();
} catch (IOException e) {
    try {
        reader.close();
    } catch (IOException e) { /* Ignoring */ }
    throw e;
}
```

Альтернативный метод

```
Reader reader = null;
try {
    reader = new FileReader("input.txt");
    ...
} finally {
    if (reader != null) {
        reader.close();
    }
}
```

Случай нескольких потоков

```
Reader reader = new FileReader("input.txt");
try {
    Writer writer = new FileWriter("output.txt");
    try {
        // Операции ввода-вывода
        ...
    } finally {
        writer.close();
    }
} finally {
    reader.close();
}
```

Часть 8

Заключение

Ссылки

- I/O tutorial // <http://java.sun.com/docs/books/tutorial/essential/io/index.html>
- I/O in Java 2 Platform // <http://java.sun.com/j2se/1.5.0/docs/guide/io/>

Вопросы