Мультимедийный курс

Программирование на Java

Лекция 08

Система ввода-вывода Java

Автор:

- Борисенко В. П.

Лаборатория Интернет-технологий

Понятие потоков ввода/вывода

- Потоком ввода/вывода (I/O Stream) называется произвольный источник или приемник, который способен генерировать либо получать некоторые данные
- Все потоки ведут себя одинаковым образом, хотя физические устройства, с которыми они связаны, могут сильно различаться
- Реализация конкретным потоком низкоуровневого способа приема/передачи информации скрыта от программиста

Системы ввода/вывода Java

- Основная система ввода/вывода Java представлена пакетом **java.io**
- Пакет **java.nio** содержит API для работы с новой системой ввода/вывода
- Потоки для работы с архивами содержаться в пакете java.util

Лаборатория Интернет-технологий

Виды потоков ввода/вывода

Всего существует 2 вида потоков ввода/вывода:
- **байтовые**
- **символьные**

Байтовые потоки - последовательность байт (byte)

Символьные - последовательность двухбайтовых символов Unicode (char).

Лаборатория Интернет-технологий

Суперклассы java.io API

Все потоки ядра Java (стандартного API) – это потомки 4-х суперклассов, которые являются абстрактными и напрямую наследуются от класса Object.

Суперкласс иерархии	Потоки
java.io.InputStream	входные байтовые потоки
java.io.OutputStream	выходные байтовые потоки
java.io.Reader	входные символьные потоки
java.io.Writer	выходные символьные потоки

Лаборатория Интернет-технологий

Парные потоки

Предназначение каждого класса-потока заключается в том, чтобы передать или принять последовательность символов или байт.

API Java содержит более 60 потоков, каждый из которых содержит свой собственный набор методов для управлением процессом приема/передачи информации.

Для некоторых потоков существуют парные им в том смысле, что парный поток содержит зеркальное отображение функциональности исходного потока относительно направления передачи информации.

Лаборатория Интернет-технологий

Парные классы в иерархиях байтовых потоков

*Input*Stream	*Output*Stream
ByteArray*Input*Stream	ByteArray*Output*Stream
File*Input*Stream	File*Output*Stream
Stringbuffer*Input*Stream	-
Object*Input*Stream	Object*Output*Stream
Filter*Input*Stream	Filter*Output*Stream
Buffered*Input*Stream	Buffered*Output*Stream
-	PrintStream
Zip*Input*Stream	Zip*Output*Stream
Pushback*Input*Stream	-
Data*Input*Stream	Data*Output*Stream

Лаборатория Интернет-технологий

Парные классы в иерархиях символьных потоков

Reader	*Writer*
Buffered*Reader*	Buffered*Writer*
-	Print*Writer*
String*Reader*	String*Writer*
Filter*Reader*	Filter*Writer*
Pushback*Reader*	-
InputStream*Reader*	OutputStream*Writer*
File*Reader*	File*Writer*

Лаборатория Интернет-технологий

Класс InputStream

- Абстрактный класс **InputStream** предоставляет минимальный набор методов для работы с входным потоком **байтов**:
  - **int available()** - возвращает количество еще доступных байт потока
  - **int read()** - возвращает очередной байт. Значения от 0 до 255. Если чтение невозможно, возвращает -1
  - **int read(byte[] buf, int offset, int count)** - вводит байты в массив. Возвращает количество реально введенных байтов
  - **long skip(long n)** - пропускает n байтов потока
  - **void close()** - закрывает поток и освобождает занятые системные ресурсы

Лаборатория Интернет-технологий

Потомки класса InputStream

- **ObjectInputStream** - поток объектов. Создается при сохранении объектов системными средствами
- **SequenceInputStream** - последовательное соединение нескольких входных потоков
- **ByteArrayInputStream** - использует массив байтов как источник данных
- **PipedInputStream** - совместно с PipedOutputStream обеспечивает обмен данными между двумя потоками выполнения
- **FileInputStream** - обеспечивает чтение из файла
- **StringBufferInputStream** - использует изменяемую строку StringBuffer как источник данных
- **FilterInputStream** - абстрактный класс надстройки над классом InputStream

Лаборатория Интернет-технологий

Классы надстройки

Классы

FilterInputStream, FilterOutputStream;
FilterReader, FilterWriter

являются, соответственно, классами надстройками над классами

InputStream, OutputStream;
Reader и Writer

Суперклассы надстроек являются абстрактными классами.

API Java содержит набор неабстрактных классов-надстроек, которые являются потомками базовых надстроек.

Лаборатория Интернет-технологий

Классы надстройки

Основное предназначение надстроек - наделение существующего потока новыми свойствами.

Комбинируя исходный поток и классы надстройки, можно создать новый поток с заданным набором свойств.

Если нужно наделить существующий поток некоторым свойством, достаточно надстроить его соответствующим классом надстройкой и работать с объектом последнего.

Надстраивание (декорация)

В отличие от наследования надстраивание не ведет к появлению большого числа библиотечных классов. Так если мы имеем классы A1, A2, …, An и хотим комбинировать их свойства путем наследования, мы вынуждены создать порядка n * n новых классов. Если делать то же путем надстраивания, понадобится всего n новых классов

В java.io имеется несколько потомков FilterInputStream:

- **DataInputStream**
- **BufferedInputStream**
- **PushBackInputStream**

Лаборатория Интернет-технологий

Класс DataInputStream

Класс DataInputStream наследует класс надстройку FilterInputStream и позволяет читать данные из входного байтового потока в формате примитивных типов данных: double, boolean и т.д.

Парный класс DataOutputStream наследует класс FilterOutputStream и позволяет записывать значения примитивных типов в выходной байтовый поток, который затем можно будет прочесть используя класс DataInputStream.

Замечание. Экземпляры классов DataInputStream и DataOutputStream надстраивают, соответственно, входной и выходной потоки, которые передаются им как параметры конструкторов при их создании.

Буферизация

Для ускорения файловых операций чтения/записи следует использовать буферизированные классы: BufferedInputStream и BufferedReader.

```
BufferedReader in1 = new BufferedReader(new
    InputStreamReader(new FileInputStream("file.txt")));

BufferedReader in2 = new BufferedReader(new
    FileReader("file.txt"));

BufferedInputStream in3 = new BufferedInputStream(new
    FileInputStream("file.txt"));
```

Класс BufferedInputStream

Класс BufferedInputStream наследует класс надстройку FiltertInputStream.

Объект этого класса надстраивает входной байтовый поток и поддерживает буфер определенного размера.

Входной поток и размер буфера передаются объекту BufferedInputStream при его создании с помощью конструктора в качестве параметров (размер буфера по умолчанию как правило достаточен для решения большинства возникающих задач).

Парный класс BufferedOutputStream наследует надстройку FilterOutputStream и надстраивает выходной поток, добавляя возможность использовать буфер.

Лаборатория Интернет-технологий

Класс PushbackInputStream

Класс PushbackInputStream надстраивает входной байтовый поток и позволяет кроме чтения осуществлять запись прочтенных байт обратно во входной поток.

Замечание. Класс PushbackInputStream не имеет парный класс.

Замечание. Существует аналогичный класс для входных символьных потоков.

Поле in класса System

Статическое поле in класса System имеет тип InputStream и связано по умолчанию с консольным вводом (клавиатурой). Как правило, приходится надстраивать этот входной поток.

```
BufferedReader in = new BufferedReader(
    new InputStreamReader(System.in));

String s = null;
while (!(s=in.readLine()).equals(""))
    System.out.println(s);
```

Класс OutputStream

- Абстрактный класс OutputStream предоставляет минимальный набор методов для работы с выходным потоком **байтов**

  - **void write(int b)** - Абстрактный метод записи в поток одного байта
  - **void write(byte[] buf, int offset, int count)** - Запись в поток массива байтов или его части
  - **void flush()** - Форсированная выгрузка буфера для буферизированных потоков. Если получателем служит другой поток, его буфер тоже сбрасывается
  - **void close()** - Закрытие потока и высвобождение системных ресурсов

Лаборатория Интернет-технологий

Потомки класса OutputStream

- **ObjectOutputStream** - поток двоичных представлений объектов. Создается при сериализации
- **ByteArrayOutputStream** - использует массив байтов как приемник данных
- **PipedOutputStream** - вместе с PipedInputStream составляет пару потоков для обмена данными между потоками выполнения (threads)
- **FileOutputStream** - поток для записи в файл
- **FilterOutputStream** - абстрактный класс надстройки

Лаборатория Интернет-технологий

Надстройки для OutputStream

- Надстройками для OuptupStream являются наследники абстрактного класса **FilterOutputStream**

  - **PrintStream** – добавляет возможность преобразования простых типов данных в последовательность байтов. Делает это при помощи перегруженного метода print(), который преобразует и помещает их в выходной поток

  - **BufferedOutputStream** – буферизированный выходной поток. Ускоряет вывод.

  - **DataOutputStream** - поток для вывода значений простых типов. Имеет такие методы как writeBoolean(), writeInt(), writeLong(), writeFloat() и т.п.

Буферизированный ввод/вывод

```
public class FileCopy {
   public static void main(String[] args) {
      try {
        BufferedInputStream bis = new BufferedInputStream(new FileInputStream("erste.jpg"));
        BufferedOutputStream bos = new BufferedOutputStream(new FileOutputStream("zweite.jpg"));

        int c = 0;
        while (true) {
       c = bis.read();
       if (c != -1)
              bos.write(c);
       else
              break;
       }
       bis.close();
       bos.flush(); //освобождаем буфер (принудительно записываем содержимое буфера в файл)
       bos.close(); //закрываем поток записи (обязательно!)
      }
      catch (java.io.IOException e) {
       System.out.println(e.toString());
      }
    }
}
```

Лаборатория Интернет-технологий

Символьные потоки

- Для работы с символьными потоками в Java существуют два базовых класса – Reader и Writer
- **Reader** содержит абстрактные методы read(…) и close(). Дополнительные методы объявлены в потомках этого класса
- **Writer** содержит абстрактные методы write(…), flush() и close()

Лаборатория Интернет-технологий

Некоторые потомки класса Writer

- BufferedWriter - буферизированный выводной поток. Размер буфера можно менять, хотя размер, принятый по умолчанию, пригоден для большинства задач
- CharArrayWriter - позволяет выводить символы в массив как в поток
- StringWriter - позволяет выводить символы в изменяемую строку как в поток
- PrintWriter - поток, снабженный операторами print() и println()
- PipedWriter - средство межпоточного общения
- OutputStreamWriter – мост между классом OutputStream и классом Writer. Символы, записанные в этот поток, превращаются в байты. При этом можно выбирать способ кодирования символов
- FileWriter - поток для записи символов в файл
- FilterWriter – служит для быстрого создания пользовательских надстроек

Лаборатория Интернет-технологий

Потомки класса Reader

- BufferedReader - буферизированный вводной поток символов
- CharArrayReader - позволяет читать символы из массива как из потока
- StringReader - то же из строки
- PipedReader - парный поток к PipedWriter
- InputStreamReader – при помощи методов класса Reader читает байты из потока InputStream и превращает их в символы. В процессе превращения использует разные системы кодирования
- FileReader - поток для чтения символов из файла
- FilterReader – служит для создания надстроек

Лаборатория Интернет-технологий

Пример программы

- Вводить строки с клавиатуры и записывать их в файл на диске.

```
try {
    // Создаем буферизованный  символьный входной поток
    BufferedReader in = new BufferedReader(
      new InputStreamReader(System.in));
    // Используем класс PrintWriter для вывода
    PrintWriter out = new PrintWriter (new FileWriter("data.txt"));
    // Записываем строки, пока не введем строку "stop"
    while (true) {
      String s = in.readLine();
      if (s.equals("stop"))
            break;
      out.println(s);
    }
    out.close();
} catch (IOException ex) {
   // Обработать исключение
}
```

Лаборатория Интернет-технологий

Класс OutputStreamWriter

Класс OutputStreamWriter наследуется от класса Writer, и преобразует выходной символьный поток в выходной байтовый поток. Класс имеет несколько конструкторов, каждый из которых принимает в качестве одного из своих параметров выходной символьный поток.

OutputStreamWriter(OutputStream out)
OutputStreamWriter(OutputStream out, String charsetName)

Второй параметр указывает на кодировку, при этом каждому символу ставится в соответствие совокупность байт, которая является числовым кодом символа в этой кодировке.

Замечание. Если при создании объекта класса OutputStreamWriter используется конструктор без указания кодировки, то конвертирование осуществляется с использованием кодировки по умолчанию.

Лаборатория Интернет-технологий

Кодировка по умолчанию

При запуске программы кодировку по умолчанию устанавливает JVM в зависимости от операционной системы в которой выполняется программа и ее настроек.

ОС Windows использует в качестве кодировки по умолчанию Windows-1251 (Cp1251), для вывода в консоль используется DOS-кодировка Cp866 (Win OS русской локализации).

Указание кодировки при компиляции

Для правильного отображения строковых литералов, записанных в программе, следует обеспечить правильное конвертирование этих символов в Unicode при компиляции с помощью javac, указав это при помощи ключа -encoding.

Например, если код программы записан в DOS кодировке Cp866, то компилировать необходимо так:

```
javac –encoding Cp866 NameOfJavaFile
```

Перекодировка вывода

Все строковые литералы в байт коде классов содержаться в формате Unicode.

При выводе таких строк на экран, в файл и т.д. осуществляется их перекодировка с использованием кодировки по умолчанию.

Например, в ОС Windows кодировкой по умолчанию является Cp1251, поэтому произойдет конвертирование Unicode->Cp1251.

Если вывод осуществляется в консольное окно (с помощью метода System.out.println), то такие строки в общем случае будут отображены неправильно, т.к. Windows для отображения символов в консольном окне использует кодировку Cp866.

Чтобы избежать этого, необходимо явно указать в какой кодировке должны выводится символы.

Достигается это с помощью надстройки стандартного потока вывода.

Лаборатория Интернет-технологий

Перекодировка вывода

```
PrintWriter out = new PrintWriter(new
      OutputStreamWriter(System.out, "Cp866"), true);
out.println(s); // вывод на экран строки s в кодировке Cp866
```

Второй параметр конструктора PrintWriter указывает на то, что каждый вызов метода println будет принудительно сбрасывать буфер, т.е., после каждого вызова println будет происходить вывод на экран строкового значения параметра этого метода. В противном случае вывод на экран произойдет только тогда, когда буфер принудительно будет сброшен с помощью вызова метода *flush*.

Аналогично можно надстроить по сути любой поток, таким образом достигается возможность осуществлять перекодирование символов между любыми двумя допустимыми кодировками.

Поле out класса System

Статическое поле *out* класса System имеет тип java.io.PrintStream, который представляет собой надстройку над *байтовым* выходным потоком OutputStream и по умолчанию связан с консольным выводом (дисплеем).

Это, так называемый, поток стандартного вывода.

Программно он может быть надстроен для того, чтобы осуществлять перекодировку символов выводимых данных.

Класс RandomAccessFile

- RandomAccessFile применяется для работы с файлами произвольного доступа
- Для перемещения по файлу в RandomAccessFile применяется метод seek()
- RandomAccessFile не участвует в рассмотренной выше иерархии, но реализует интерфейсы DataInput и DataOutput (те же, что реализованы классами DataInputStream и DataOutputStream)

Лаборатория Интернет-технологий

Пример работы с RandomAccessFile

- Создать файл прямого доступа, выполнить запись в файл и чтение из файла

```
RandomAccessFile rf = new RandomAccessFile("rtest.dat", "rw");
// Записать в файл 10 чисел и закрыть файл
for(int i = 0; i < 10; i++)
    rf.writeDouble(i * 1.414);
rf.close();
// Открыть файл, записать в него еще одно число и снова закрыть
rf = new RandomAccessFile("rtest.dat", "rw");
rf.seek(5 * 8);
rf.writeDouble(47.0001);
rf.close();
// Открыть файл с возможностью только чтения "r"
rf = new RandomAccessFile("rtest.dat", "r");
// Прочитать 10 чисел и показать их на экране
for(int i = 0; i < 10; i++)
  System.out.println("Value " + i + ": " + rf.readDouble());
rf.close();
```

Лаборатория Интернет-технологий

Класс File

- Класс File предназначен для работы с элементами файловой системы – каталогами и файлами
- Каждый объект File представляет абстрактный файл или каталог, возможно и не существующий
- Абстрактный путь, который заключает в себе объект File, состоит из не обязательного системно-зависимого префикса и последовательности имен
  - Префикс выглядит по-разному в различных операционных системах: символ устройства "C:", "D:" в системе Windows, символ корневого каталога "/" в системе UNIX, символы "\\" в UNC и т.д.
  - Каждое имя последовательности является именем каталога, а последнее имя может быть именем каталога или файла
- Путь может быть абсолютным или относительным

Лаборатория Интернет-технологий

Конструкторы класса File

- **File(String filePath),** где filePath – имя файла на диске
- **File(String dirPath, String filePath)**, здесь параметры dirPath и filePath вместе задают то же, что один параметр в предыдущем конструкторе
- **File(File dirObj, String fileName)**, вместо имени каталога выступает другой объект File
- Объект File является неизменяемым объектом !

Лаборатория Интернет-технологий

Каталоги

- Каталог – это особый файл, который содержит в себе список других файлов и каталогов
- Для каталога метод isDirectory() возвращает true
- Метод File[] listFiles() возвращает список подкаталогов и файлов данного каталога
- Пример: получить массив файлов и каталогов, которые находятся в рабочем (или текущем) каталоге

```
File path = new File(".");
File[] list = path.listFiles();
for(int i = 0; i < list.length; i++)
    System.out.println(list[i].getName());
```

Лаборатория Интернет-технологий

Фильтры (интерфейс FileFilter)

- Интерфейс FileFilter применяется для проверки, подпадает ли объект File под некоторое условие
- Метод boolean accept(File file) возвращает истину, если аргумент удовлетворяет условию
- Метода listFiles(FileFilter filter) класса File принимает в качестве аргумента объект FileFilter и возвращает уже профильтрованный массив из объектов

Пример работы с фильтрами

- Выбрать из текущего каталога лишь те файлы, которые содержат в своем последнем имени буквосочетание, заданное в командной строке

```
public static void main(final String[] args) {
    File path = new File(".");

    // Получить массив объектов
    File[] list = path.listFiles(new FileFilter() {
      public boolean accept(File file) {
            String f = file.getName();
            return !file.isDirectory() && f.indexOf(args[0]) != -1;
      }
    });
    // Напечатать имена файлов
    for(int i = 0; i < list.length; i++) {
      System.out.println(list[i].getName());
    }
}
```

Лаборатория Интернет-технологий

Новый ввод/вывод

- Ее цель – увеличение производительности и обеспечения безопасности при одновременном конкурентном доступе к данным из нескольких потоков.

- Основными понятиями нового ввода/вывода являются
  - Канал (Channel)
  - Буфер (Buffer)

- При работе с каналом прямого взаимодействия с ним нет. Приложение "посылает" буфер в канал, который затем либо извлекает данные из буфера, либо помещает их в него

Буфер

- Буфер представляет собой контейнер для данных простых типов, таких как byte, int, float и др. кроме boolean
- Кроме собственно данных, буфер имеет
  - текущую позицию
  - лимит
  - емкость
- Операции над буфером можно поделить на
  - абсолютные - считывают или записывают один или несколько элементов начиная с текущей позиции и увеличивают или уменьшают текущую позицию на количество прочитанных элементов
  - относительные - производятся начиная с указанного индекса и не изменяют текущей позиции

Лаборатория Интернет-технологий

Методы класса Buffer

- clear() – подготавливает буфер для операции записи в него данных
  - Он устанавливает лимит равным емкости и позицию равной нулю.
  - Таким образом, при чтении данных из канала и записи их в буфер, они будут туда помещаться с начальной позиции до тех пор, пока буфер не будет полностью заполнен
- flip() – подготавливает буфер для чтения из него данных.
  - Он устанавливает лимит равным текущей позиции и после этого устанавливает позицию равной нулю.
  - Таким образом, при записи данных в канал они будут считываться из буфера начиная с начала до того места, до которого он был заполнен
- rewind() – подготавливает буфер для повторного прочтения данных.
  - Он не изменяет лимит и устанавливает позицию равной нулю

Лаборатория Интернет-технологий

Байтовый буфер (ByteBuffer)

- Байтовый буфер предназначен для работы с байтовыми данными

- Создать буфер тремя способами:
  - На основе готового массива байт с помощью статического метода wrap(byte[])

```
ByteBuffer bb = ByteBuffer.wrap(new byte[]{12,12});
```

  - Пустой буфер заданного размера с помощью метода allocate(int)

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

  - Прямой буфер с помощью метода allocateDirect(int).

Лаборатория Интернет-технологий

Прямые и непрямые буферы

- ByteBuffer может быть прямым и непрямым
- При работе с прямым (direct) буфером виртуальная машина использует напрямую системные операции ввода/вывода. При этом
  - операции чтения-записи в случае использования прямого буфера проходят быстрее
  - на создание такого буфера требуется, как правило, большее количество ресурсов
  - содержимое буфера не контролируется сборщиком мусора
- Использовать прямые буферы целесообразно лишь для больших объемов данных, к которым обращаются в течение продолжительного времени

Чтение-запись данных в буфер

- Для относительного получения байта данных из буфера используется метод get(), для абсолютного - get(int position)
- Для записи байта данных в буфер используется методы put() и put(int)
- Также существуют методы для чтения/записи массивов байтов - get(byte[] dst), и др.
- При необходимости чтения/записи данных простых типов используются методы getXXX()/getXXX(int) и putXXX()/ putXXX(int)
- Все эти методы возвращают тот же самый объект ByteBuffer, поэтому допустима следующая запись:

```
bb.putInt(0xCAFEBABE).putShort(3).put(255).putFloat(4.5);
```

Буферы-представления

- При необходимости работать с однотипными данными лучше использовать классы-представления
  - CharBuffer
  - DoubleBuffer
  - FloatBuffer
  - IntBuffer
  - LongBuffer
  - ShortBuffer

- Для создания этих буферов используются те же методы allocate и wrap, однако размер буфера в данном случае устанавливается в его единицах данных.

- Также можно создать представление ByteBuffer в виде, например, CharBuffer:

```
ByteBuffer bb = ByteBuffer.allocate(BSIZE);
bb.asCharBuffer().put("Привет!");
```

Пример работы с буфером-представлением

```
public class IntBufferDemo {

  private static final int BSIZE = 1024;

  public static void main(String[] args) {

    ByteBuffer bb = ByteBuffer.allocate(BSIZE);
    IntBuffer ib =    bb.asIntBuffer();

   // Сохранение массива целых чисел
   ib.put(new int[] { 11, 42, 47, 99, 143, 811, 1016 });

   // Чтение и запись в абсолютных позициях:
   System.out.println(ib.get(3));
   ib.put(3, 1811);
   ib.rewind();

   while(ib.hasRemaining()) {
     int i = ib.get();
     if(i == 0) break; // Иначе получим буфер целиком
     System.out.println(i);
     }
   }
}
```

Лаборатория Интернет-технологий

Файловый канал

- Канал представляет собой открытое соединение к некоторой сущности, такой как, например, аппаратное устройство, файл, сетевой сокет или программный компонент, которая может производить операции ввода/вывода
- Класс FileChannel позволяет организовать канал доступа к файлу
- Для получения файлового канала служат метод getChannel() классов FileInputStream, FileOutputStream и RandomAccessFile

Лаборатория Интернет-технологий

Работа с FileChannel

- Файловый канал имеет свою позицию, которая устанавливается методом position(long)
- Методы read(ByteBuffer) и read(ByteBuffer, int) служат для чтения данных из канала в переданный буфер с текущей позиции (относительно) или с указанной позиции (абсолютно) соответственно
- Аналогично используются методы write(...)
- Для блокировки файла или его части используются методы lock(...). Их использование гарантирует то, что файл, к которому осуществляется доступ, будет блокирован для других процессов

Лаборатория Интернет-технологий

Пример работы с FileChannel

```
public class GetChannel {
  private static final int BSIZE = 1024;

  public static void main(String[] args) throws Exception {
    // Запись в файл:
    FileChannel fc = new FileOutputStream("data.txt").getChannel();
    fc.write(ByteBuffer.wrap("Немного текста ".getBytes()));
    fc.close();
    // Добавление в конец файла:
    fc = new RandomAccessFile("data.txt", "rw").getChannel();
    fc.position(fc.size()); // Переходим в конец
    fc.write(ByteBuffer.wrap("Еще немного".getBytes()));
    fc.close();
    // Чтение файла:
    fc = new FileInputStream("data.txt").getChannel();
    ByteBuffer buff = ByteBuffer.allocate(BSIZE);
    fc.read(buff);
    buff.flip();
    while(buff.hasRemaining())
      System.out.print((char)buff.get());
  }
}
```

Лаборатория Интернет-технологий

Копирование файлов с использованием FileChannel

```
public class ChannelCopy {
   private static final int BSIZE = 1024;
   public static void main(String[] args) throws Exception {
   if(args.length != 2) {
     System.out.println("параметры: ФайлИсточник ФайлПолучатель");
     System.exit(1);
   }
   FileChannel in = new FileInputStream(args[0]).getChannel(),
        out = new FileOutputStream(args[1]).getChannel();
   ByteBuffer buffer = ByteBuffer.allocate(BSIZE);
   while(in.read(buffer) != -1) {
      buffer.flip(); // Подготовим для записи
      out.write(buffer);
      buffer.clear(); // Подготовим для чтения
   }
 }
}
```

Лаборатория Интернет-технологий

Более эффективный способ копирования файлов

```
public class TransferTo {
   public static void main(String[] args) throws Exception {
   if(args.length != 2) {
      System.out.println("параметры: ФайлИсточник
     ФайлПолучатель");
      System.exit(1);
   }
   FileChannel in = new FileInputStream(args[0]).getChannel();
   FileChannel out = new FileOutputStream(args[1]).getChannel();
   in.transferTo(0, in.size(), out);
   // Или так:
   // out.transferFrom(in, 0, in.size());
 }
}
```

Лаборатория Интернет-технологий

Блокировка файлов

- Блокировка файлов осуществляется с помощью методов
  - FileLock lock(…)
  - FileLock tryLock(…)
- Метод tryLock() не приостанавливает программу. Он пытается овладеть объектом блокировки, но если ему это не удается (если другой процесс уже владеет этим объектом или файл не является разделяемым), то он просто возвращает null
- Метод lock() ждет до тех пор, пока
  - не удастся получить объект блокировки
  - поток, в котором этот метод был вызван, не будет прерван
  - пока не будет закрыт канал, для которого был вызван метод lock()
- Блокировка снимается методом release()

Пример блокировки файла

- Механизм блокировки Java напрямую связан со средствами операционной системы

```
public class FileLocking {
  public static void main(String[] args) throws Exception {
    FileOutputStream fos= new FileOutputStream("file.txt");
    FileLock fl = fos.getChannel().tryLock();
    if(fl != null) {
      System.out.println("Locked File");
      Thread.sleep(1000);
      fl.release();
      System.out.println("Released Lock");
    }
    fos.close();
  }
}
```

Лаборатория Интернет-технологий

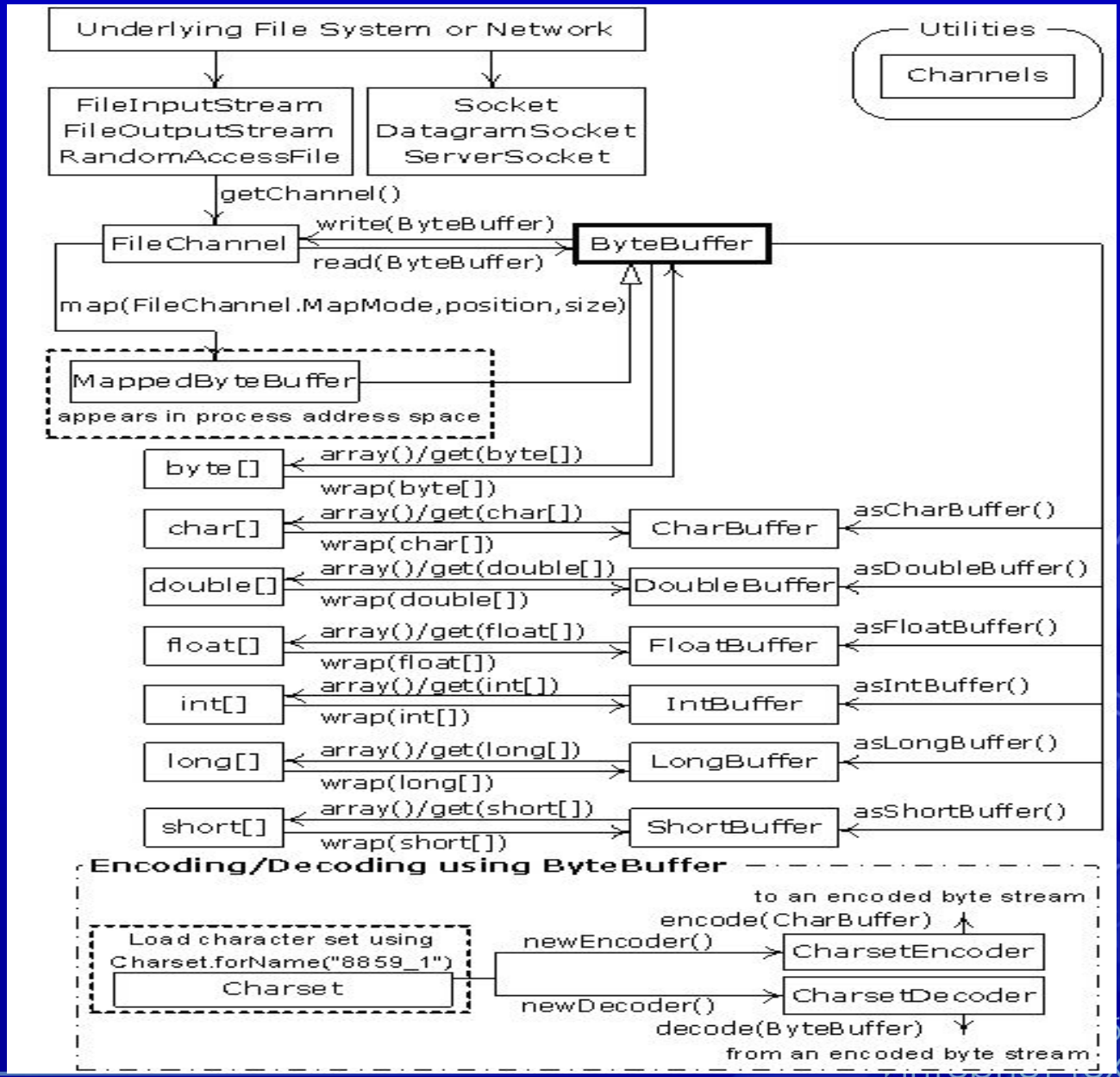
Файлы, отображаемые в памяти

- Механизм отображения файлов в память позволяет вам создавать и изменять файлы, размер которых слишком велик для прямого размещения в памяти.
- В таком случае считается, что файл целиком находится в памяти, и работают с ним как с очень большим массивом
  - Такой подход значительно упрощает код, который вы пишете для изменения файла

```
public class LargeMappedFiles {
  static int length = 0x8FFFFFF; // 128 Mb
  public static void main(String[] args) throws Exception {
    MappedByteBuffer out =
      new RandomAccessFile("test.dat", "rw").getChannel()
                              .map(FileChannel.MapMode.READ_WRITE, 0, length);
    for(int i = 0; i < length; i++)
      out.put((byte)'x');
    System.out.println("Finished writing");
    for(int i = length/2; i < length/2 + 6; i++)
      System.out.print((char)out.get(i));
  }
}
```

Лаборатория Интернет-технологий

Диаграмма отношений пакета nio



Сериализация

- Сериализация позволяет превратить объект в поток байтов, чтобы, когда понадобится, полностью восстановить объект из потока
- Сериализация необходима для
  - сохранения объектов в постоянной памяти
  - транспортировки параметров при удаленном вызове методов (RMI - Remote Methods Invocation)
  - сохранения на диске компонентов JavaBeans
  - И т.д.

Интерфейс Serializable

- Чтобы обладать способностью к сериализации, класс должен:
- Реализовать интерфейс-метку Serializable
  - Интерфейс Serializable не содержит никаких методов. Он просто служит индикатором того, что класс может быть сериализован

```
public class MyClass implements Serializable{
      …
}
```

- Все атрибуты класса должны быть сериализуемы
  - Атрибуты простых типов являются сериализуемыми по умолчанию
  - Если атрибут не должен быть сохранен в процессе сериализации, для него необходимо задать модификатор **transient**
    - При сериализации он будет проигнорирован
    - При десериализации значение этого атрибута будет пустым
- Все подтипы сериализуемого класса являются сериализуемыми

Запись-чтение объектов

- Сериализованные объекты можно записывать и считывать при помощи классов ObjectOutputStream и ObjectInputStream.
- Они таже реализуют интерфейсы DataInput / DataOutput, что дает возможность записывать в поток не только объекты, но и простые типы данных.
- writeObject(Object obj) – запись объекта (класс ObjectOutputStream)
- Object readObject() – чтение объекта (класс ObjectInputStream). Метод readObject может также генерировать java.lang.ClassNotFoundException
- При десериализации объекта, он возвращается в виде объекта класса Object - верхнего класса всей иерархии классов Java. Для того, чтобы использовать десериализованный класс, необходимо произвести явное преобразование его к необходимому типу

Пример сериализации объектов

```
public class Point implements java.io.Serializable {
   private int x=0, y = 0;
   public Point() {}
   public Point(int x, int y) {
     this.x = x; this.y = y;
   }
   public String toString() { return "("+x+","+y+")"; }
}

// Сериализация
java.io.ObjectOutputStream ois = new java.io.ObjectOutputStream(new
      java.io.FileOutputStream("state.bin"));
ois.writeDouble(3.14159265D);
ois.writeObject("The value of PI");
ois.writeObject(new Point(10,253)); //запись объекта класса Point
ois.flush();
ois.close();

// Десериализация
java.io.ObjectInputStream ois = new java.io.ObjectInputStream(new
      java.io.FileInputStream("state.bin"));
System.out.println("Double: " + ois.readDouble());
System.out.println("String: " + ois.readObject().toString());
System.out.println("Point: " + (Point) ois.readObject());
ois.close();
```

Лаборатория Интернет-технологий

Сериализация наследников несериализуемого класса

- Если необходимо, чтобы подкласс несериализумого класса мог быть сериализуем, то:
  - Сохранение и восстановление public, protected и доступных в рамках пакета полей суперкласса осуществляется самим подклассом
  - Суперкласс должен содержать доступный (public или protected) конструктор без параметров для инициализации полей
    - Ошибка (отсутствие конструктора у суперкласса) в таком случае будет обнаружена во время выполнения
  - При десериализации поля несериализумого класса будут инициализированы с помощью конструктора без параметров
  - Поля сериализуемых классов будут восстановлены из потока

Пример

- Если суперкласс сериализуем:

```
public class Point implements Serializable{
  public int x = 0;   public int y = 0;
   // без пустого конструктора можно обойтись
  public Point(int x, int y) {this.x = x;   this.y = y; }
}
public class PointXYZ extends Point {
        // нет необходимости указывать  implements Serializable
  private int z = 0;
  public PointXYZ(int x, int y, int z) {   super(x,y);   this.z = z; }
  public String toString() { return "x = "+ x +"; y = " + y + "; z = " + z ; }
}
```

- Сериализация:

```
oos.writeObject(new PointXYZ(10,20,30));
```

- В результате десериализации объект типа PointXYZ будет восстановлен:

```
x = 10; y = 20; z = 30
```

Лаборатория Интернет-технологий

Пример 2

- Если суперкласс несериализуем:

```
public class Point {
  public int x = 0;   public int y = 0;
  public Point() {   } // без этого конструктора возникнет ошибка
  public Point(int x, int y) {this.x = x;   this.y = y; }
}
public class PointXYZ extends Point implements Serializable{
  private int z = 0;
  public PointXYZ(int x, int y, int z) {   super(z,y);   this.z = z; }
  public String toString() { return "x = "+ x +"; y = " + y + "; z = " + z ; }
}
```

- Сериализация:

```
oos.writeObject(new PointXYZ(10,20,30));
```

- В результате десериализации объект типа PointXYZ будет восстановлен следующим образом:

```
x = 0; y = 0; z = 30
```

Лаборатория Интернет-технологий

Управление процессом сериализации

- Для выполнения специальной обработки при сериализации и десериализации класс должен реализовать следующие методы:

- private void **writeObject**(java.io.ObjectOutputStream out) throws IOException
  - Метод предназначен для записи состояния объекта данного класса так, чтобы соответствующий метод readObject мог их восстановить.
  - Для сохранения полей объекта может быть использован встроенный механизм , вызываемый out.defaultWriteObject.

- private void **readObject**(java.io.ObjectInputStream in) throws IOException, ClassNotFoundException;
  - Метод предназначен для чтения из потока и восстановления полей класса. Для восстановления нестатических и не-transient полей может быть использован встроенный механизм, вызываемый in.defaultReadObject.
  - Метод defaultReadObject использует данные из потока для присвоения значений полей сохраненного объекта соответствующим (по имени) полям текущего объекта

Пример

```
public class PointXYZ extends Point implements Serializable{

    private int z = 0;
    …
    private void writeObject(java.io.ObjectOutputStream out)throws IOException {
        out.writeInt(x);
        out.writeInt(y);
        out.defaultWriteObject(); // сохраняет поле z
      }

    private void  readObject(java.io.ObjectInputStream in) throws IOException,
    ClassNotFoundException{
        x = in.readInt();
        y = in.readInt();
        in.defaultReadObject();
    }
}
```

- Десериализация:

- x = 10; y = 20; z = 30

Лаборатория Интернет-технологий

Архивирование

- Библиотека ввода/вывода Java содержит классы, поддерживающие чтение и запись потоков в компрессированном формате
- Эти классы являются оберткой для существующих классов ввода/вывода для обеспечения возможности компрессирования
- Они являются частью иерархии InputStream и OutputStream

Лаборатория Интернет-технологий

Классы для работы с архивами

- DeflaterOutputStream – базовый класс для классов компрессии
- InflaterInputStream – базовый класс для классов декомпрессии.
- ZipOutputStream - DeflaterOutputStream, который компрессирует данные в файл формата Zip.
- ZipInputStream - InflaterInputStream, который декомпрессирует данные, хранящиеся в файле формата Zip.
- GZIPOutputStream – DeflaterOutputStream, который компрессирует данные в файл формата GZIP.
- GZIPInputStream – InflaterInputStream, который декомпрессирует данные, хранящиеся в файле формата GZIP

Лаборатория Интернет-технологий

Работа с ZipOutputSream

```
ZipOutputStream out = new ZipOutputStream(new
FileOutputStream(“archive.zip”));
pack("111.txt", out);
pack(“222.txt", out);
out.close();

// Упаковывает файл по имени fin
static void pack(String fin, ZipOutputStream out) throws IOException {
    // Открыть вводной файл
    FileInputStream in = new FileInputStream(fin);
    // Создать вход
    out.putNextEntry(new ZipEntry(fin));
    // Выполнить сжатие
    int c;
    while((c = in.read()) != -1)
      out.write(c);
    in.close();
}
```

Лаборатория Интернет-технологий

Работа с ZipInputStream

```
ZipInputStream in = new ZipInputStream(new BufferedInputStream(
                    new FileInputStream("111.zip")));

    ZipEntry entry;
    while ((entry = in.getNextEntry()) != null) {
      unpack(in, entry.getName());
    }

static void unpack(ZipInputStream in, String fout) throws IOException {
    // Создать выходной поток
    BufferedOutputStream out = new BufferedOutputStream(
                        new FileOutputStream(fout));
    int c;
    while((c = in.read()) != -1) {
      out.write(c);
    }
    out.close();
}
```

Лаборатория Интернет-технологий

Логирование

- Логирование — это механизм протоколирования различной информации о событиях, происходящих в процессе выполнения программы.
- Логирование является прикладной задачей и как правило используется для задач поиска неисправностей, задач учета, задач обеспечения качества.
- Основные понятия логирования:
  - Приемник информации
  - Уровень
  - Логгер
  - Форматтер

Лаборатория Интернет-технологий

Пакет java.util.logging

- Пакет java.util.logging предоставляет классы и интерфейсы Java™ 2 для реализации логирования.

- Ключевые элементы этого пакета:
  - Logger: главная сущность, с помощью которой осуществляется логирование.
  - LogRecord: используется для передачи запросов логирования между подсистемой логирования и отдельными обработчиками логов.
  - Handler: Экспортирует объекты LogRecord в различные приемники информации, такие как, память, выходные потоки, консоли и сокеты.
  - Level: Определяет набор стандартных уровней логирования, которые могут быть использованы для контроля выходной информации.
  - Filter: Обеспечивает возможность детального контроля выходной информации логирования.
  - Formatter: Обеспечивает возможность форматирования выходной

Лаборатория Интернет-технологий

Библиотека log4j

Библиотека логирования log4j — это проект корпорации Apache Software Foundation.

Основные компоненты библиотеки:

- Logger — главная сущность логирования.
- Appender — приемник информации.
- Layout — формат выходной информации.

Лаборатория Интернет-технологий

Задание к лекции

- Создать класс, который производит последовательно сериализацию и архивирование объектов.
  - Один метод должен получать объект в качестве параметра и возвращать массив байт, представляющих собой заархивированный объект.
  - Второй метод должен выполнять обратную операцию.
  - Ход выполнения программы должен логироваться в файл и в консоль. Для логирования используйте на выбор либо пакет java.util.logging, либо библиотеку log4j. Логирование должно быть настроено с помощью соответствующего файла дескриптора

Лаборатория Интернет-технологий