Ввод-вывод в Java

Пакет java.io

- Концепция работы в пакете java.io включает две составляющие:
  - Работа с файлами и каталогами с помощью объектов File.
  - Работа с потоками ввода-вывода.

File

File(**File** parent, **String** child)

File(**String** pathname)

File(**String** parent, **String** child)

File(**URI** uri)

Для платформы Windows® вместо символа ”\” в строках, соответствующих путям, должна использоваться последовательность ”\\”.

File

- абстрактное представление имени файла или пути к имени каталога
- метод list() – возвращает массив строк с именами файлов в каталоге

```
File f1=new File("..");                    // "." , "/" , "C:/../"
System.out.println
        ("AbsolutePath"+f1.getAbsolutePath());
System.out.println("exists(): "+f1.exists());
System.out.println("canRead(): "+f1.canRead());
System.out.println("canWrite(): "+f1.canWrite());
```

Список файлов в каталоге

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

public class DirList {
    public static void main(String[ ] args) {
        File path = new File(".");
        String[ ] list;
        if(args.length == 0) list = path.list( );
        else
            list = path.list(new DirFilter(args[0]));
        for(int i = 0; i < list.length; i++)
            System.out.println(list[i]);
} }
```

Класс DirFilter

```
class DirFilter implements FilenameFilter {
    private Pattern ptr;
    DirFilter(String afn) {
                ptr = Pattern.compile( afn);
        }
    public boolean accept(File dir, String name) {
        return ptr.matcher(name).matches();
        }
}
```

Паттерн «Стратегия»

- list() – базовая функциональность
- FilenameFilter – предоставляет алгоритм, необходимый для работы list()

Ввод-вывод (stream)

- Stream – источник или приемник данных
- В качестве источника (приемника) данных потока может быть использован как стационарный источник данных (файл, массив, строка), так и динамический – другой поток. При этом в ряде случаев выход одного потока может служить входом другого

Буферизация

- Для повышения производительности операций ввода-вывода
- Для осуществления дополнительных операции – устанавливать метки (маркеры) для какого-либо элемента, находящегося в буфере потока и даже делать возврат считанных элементов в поток (в пределах буфера).

Фильтрация

- Для реализации различных надстроек над простейшими потоками («Декорации») при работе с различными данными

Ввод

- InputStream – для байтовых потоков
- Reader – для символьных потоков
- read() – основной метод

Абстрактный класс InputStream

Все методы класса InputStream, кроме markSupported и mark, возбуждают исключение IOException.

int available()	
int read() int read(byte[] b) int read(byte[] b, int offset, int count)	При достижении конца потока возвращает -1
long skip(long count)	
boolean markSupported() void mark(int limit) void reset()	
void close()	

Разновидности входных потоков InputStream

- ByteArrayInputStream поток - массив байтов
- StringBufferInputStream поток - строка
- FileInputStream поток - файл
- PipedInputStream поток - PipedOutputStream
- SequenceInputStream объединение потоков
- FilterInputStream абстрактный класс для реализации надстроек

Надстройки над FilteredInputStream

- BufferedIputSream
- DataInputStream
- LineNumberInputStream
- PushBackInputStream

Ввод из файла

- Побайтовый

```
InputStream myFile;
try {  myFile = new FileInputStream(“file.dat”);
  while (myFile.available()>0) {
    System.out.print((char)myFile.read());  }
 myFile.close();
} catch (IOException e) {
  System.err.println(e.getMessage());
 }
```

Ввод из файла

- С буферизацией

```
BufferedInputStream  br =
                new BufferedInputStream (
                        new FileInputStream(filename));
```

Ввод из файла

- Типизированные данные

```
DataInputStream in = new DataInputStream (
                new BufferedInputSteam(
                        new FileInputStream(“Data.txt”)));

double d= in.readDouble();
String s = in.readUTF();
int n= in.readInt();
```

Символьный ввод

- классы на основе Reader
  - InputStreamReader
  - FileReader
  - StringReader
  - CharArrayReader
  - PipedReader
  - FilterReader
  - BufferedReader
  - LineNumberReader
  - PushBackReader

Текстовый ввод

```
BufferedReader in = new BufferedReader (
        new FileReader (filename));
String s;
 while ( (s = in.readLine()) != null)
   System.out.println(s);
```

Вывод

- OutputStream - побайтовый вывод
- Writer - символьный

Абстрактный класс OutputStream

Все методы этого класса возбуждают IOException в случае ошибки записи.

void write(int b) void write(byte[] b) void write(byte[] b, int offset, int count)	
void flush()	Форсирует вывод данных из выходного буфера и его очистку.
void close()	Последующие попытки записи в этот поток приводят к возбуждению исключения IOException.

Разновидности выходных потоков OutputStream

- ByteArrayOutputStream
- FileOutputStream
- PipedOutputStream поток - PipedInputStream
- FilterOutputStream

Надстройки

- DataOutputStream
- BufferedOutputStream
- PrintStream

Запись данных

```
DataOutputStream out = new DataOutputStream (
    new BufferedOutputStream(
        new FileOutputStream(“data.inf”)));
out.writeDouble(3.14159);
out.writeInt(123);
out.writeUTF(“text string”);
```

Текстовый вывод

- Классы Writer
  - OutputStreamWriter
  - FileWriter
  - StringWriter
  - CharArrayWriter
  - PipedWriter
  - FilterWriter
  - BufferedWriter
  - PrintWriter

Текстовый вывод

```
PrintWriter out = new PrintWriter (
                  new BufferedWriter (
                      new FileWriter( filename)));
```

сокращенная форма (JDK 5)

```
PrintWriter out = PrintWriter(filename);
```

Файлы с произвольным доступом

- RandomAccessFile
- конструктор требует указания режима
  - “r”
  - “rw”
- методы read() b write() и для разных типов
- дополнительно
  - seek()
  - skipBytes()
  - getFilePointer()

Стандартный ввод-вывод

- System.in (InputStream)
- System.out (PrintStream)
- System.err (PrintStream)

Текстовый ввод из System.in

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

String s=stdin.readLine();
```

Перенаправление

System.setIn(InputStream);

System.setOut(PrintStream);

System.setErr(PrintStream);

Новый ввод/вывод

- Пакет java.nio.*

начиная с версии JDK 1.4

- Основные структуры
  - каналы (channels) и
  - буферы (buffers)

  близкие к средствам операционной системы

Пакеты NIO

Пакет	Назначение
`java.nio`	Это пакет верхнего уровня в системе ввода-вывода NIO. Он инкапсулирует различные типы буферов, содержащих данные, которыми оперирует система ввода-вывода NIO
`java.nio.channels`	Поддерживает каналы, открывающие соединения для ввода-вывода
`java.nio.channels.spi`	Поддерживает поставщики услуг для каналов
`java.nio.charset`	Инкапсулирует наборы символов. Поддерживает также функционирование кодеров и декодеров для взаимного преобразования символов и байтов
`java.nio.charset.spi`	Поддерживает поставщики услуг для наборов символов
`java.nio.file`	Поддерживает ввод-вывод в файлы
`java.nio.file.attribute`	Поддерживает атрибуты файлов
`java.nio.file.spi`	Поддерживает поставщики услуг для файловых систем

Основные положения

- В буфере хранятся данные
- Канал представляет открытое соединение с устройством ввода-вывода
- Для применения системы ввода-вывода NIO нужно получить канал для устройства ввода –вывода и буфер для хранения данных
- Все операции реализуются через ввод-вывод данных в буфер

Buffer

- Базовый абстрактный класс
- Набор операций, позволяющий определять и изменять организацию буфера

Наследники

ByteBuffer
IntBuffer
CharBuffer
LongByffer
DoubleBuffer
FloatBuffer
ShortBuffer
MappedByteBuffer

- Методы
  - get()
  - put()
  - allocate()
  - wrap()
  - slice()

Каналы

- Интерфейс Channel – наследник Closeable
- с версии JDK 7 – наследник AutoCloseable (для try with resource)
- канал может быть получен от потокового класса java.io методом getChannel()
- в JDK7 через статический метод класса Files

Наборы символов и селекторы

- Набор символов определяет способ сопоставления байтов с символами
- С помощью селекторов можно выполнять операции ввода-вывода через несколько каналов

Новое с версии JDK 7

- Интерфейс Path
  - можно получить из объекта класса File методом toPath()
  - обратно – метод toFile()
- Класс Files
  - статические методы для операций с файлами

Чтение файла через канал

- Создать объект Path, связанный с файлом (возможно InvalidPathException)

- Создать канал, связанный с объектом Path с заданными характеристиками в блоке try-with-resouce

  SeekableByteChannel fChan =Files.newByteChannel(filepath)

- Создать буфер для канала или обернув массив или выделив необходимое количество памяти

  ByteBuffer mBuf = ByteBuffer.allocate(128);

- Читать данные в буфер (при достижении конца файла количество прочитанных байтов «-1». Возможно IOException)

  count = fChan.read(mBuf);

Чтение файла через канал

- Если конец файла не достигнут, читать данные из буфера

```
if(count != -1) {
// подготовить буфер к чтению из него данных
        mBuf.rewind();
// читать байты данных из буфера и
// выводить их на экран как символы
        for(int i=0; i < count; i++)
                System.out.print((char)mBuf.get());
}
```

Сопоставление файла с буфером

```
// получить канал к файлу в блоке оператора try с ресурсами
try ( FileChannel fChan =
        (FileChannel) Files.newByteChannel(Paths.get("test.txt")) )
{
// получить размер файла
long fSize = fChan.size();
// а теперь сопоставить файл с буфером
MappedByteBuffer mBuf =
                fChan.map(FileChannel.MapMode.READ_ONLY, 0, fSize);
```

Сопоставление файла с буфером

```
// читать байты из буфера и выводить их на экран
for(int i=0; i < fSize; i++)
        System.out.print((char)mBuf.get());
        System.out.println();
} catch(InvalidPathException e) {
System.out.println("Ошибка указания пути " + e);
} catch (IOException e) {
System.out.println("Ошибка ввода-вывода " + e);
}
```

Копирование файлов

```
try {
    Path source = Paths.get(. . .);
    Path target = Paths.get(. . .);
// скопировать файл
    Files.copy(source, target,
                    StandardCopyOption.REPLACE_EXISTING);
} catch(InvalidPathException e) {
    System.out.println("Ошибка указания пути " + e);
} catch (IOException e) {
    System.out.println("Ошибка ввода-вывода " + e);
}
```

Потоковый ввод-вывод

```
// открыть файл и получить связанный с ним поток ввода-вывода
try (InputStream fin = Files.newInputStream(Paths.get(. . .)))
{
 . . .
} catch(InvalidPathException e) {
   System.out.println("Ошибка указания пути " + e);
} catch(IOException e) {
   System.out.println("Ошибка ввода-вывода " + e);
}
```

Потоковый ввод-вывод

```
// открыть файл и получить связанный с ним поток вывода
try (OutputStream fout =
        new BufferedOutputStream(
                Files.newOutputStream(Paths.get("test.txt"))) )
{
 . . .
} catch(InvalidPathException e) {
    System.out.println("Ошибка указания пути " + e);
} catch(IOException e) {
    System.out.println("Ошибка ввода-вывода: " + e);
}
```

Получение содержимого каталога

```
try ( DirectoryStream<Path> dirstrm =  Files.newDirectoryStream(Paths.get(dirname)) )
{    System.out.println("Каталог " + dirname);
     for(Path entry : dirstrm) {
          BasicFileAttributes attribs = Files.readAttributes(entry,
     BasicFileAttributes.class);
          if(attribs.isDirectory())   System.out.print("<DIR> ");
                    else    System.out.print(" ");
          System.out.println(entry.getName(1));
     }
} catch(InvalidPathException e) { System.out.println("Ошибка указания пути " + e);
} catch(NotDirectoryException e) { System.out.println(dirname + " не является
     каталогом.");
} catch (IOException e) { System.out.println("Ошибка ввода-вывода: " + e);
}
```

Обход дерева каталогов

```
try {
    Files.walkFileTree(Paths.get(dirname), new MyFileVisitor());
} catch (IOException exc) {
    System.out.println("Ошибка ввода-вывода");
}
```

- класс MyFileVisitor расширяет класс SimpleFileVisitor
- должен определять в методе visitFile(), что делать для каждого файла

Пример «посетителя»

```
class MyFileVisitor extends SimpleFileVisitor<Path> {
  public FileVisitResult visitFile(
                                Path path, BasicFileAttributes attribs)
                    throws IOException
{
        System.out.println(path);
        return FileVisitResult.CONTINUE;
}
}
```