Fluxo de Bytes

pacote java.io

Oferece abstrações que permitem ao programador lidar com arquivos, diretórios e seus dados de uma maneira independente de plataforma

File, RandomAccessFile

Oferecem recursos para facilitar a manipulação de dados durante o processo de leitura ou gravação

- bytes sem tratamento
- caracteres Unicode
- dados filtrados de acordo com certo critério
- dados otimizados em buffers
- leitura/gravação automática de objetos

Pacote java.nio (New I/O): a partir do J2SDK 1.4.0

- Suporta mapeamento de memória e bloqueio de acesso

classe File

Usada para representar o sistema de arquivos

- É apenas uma abstração: a existência de um objeto File não significa a existência de um arquivo ou diretório
- Contém métodos para testar a existência de arquivos, para definir permissões (nos S.O.s onde for aplicável), para apagar arquivos, criar diretórios, listar o conteúdo de diretórios, etc.

Alguns métodos

- String getAbsolutePath()
- String getParent(): retorna o diretório (objeto File) pai
- boolean exists()
- boolean isFile()
- boolean isDirectory()
- boolean delete(): tenta apagar o diretório ou arquivo
- long length(): retorna o tamanho do arquivo em bytes
- boolean mkdir(): cria um diretório com o nome do arquivo
- String[] list(): retorna lista de arquivos contido no diretório

```
File diretorio = new File("c:\tmp\cesto");
diretorio.mkdir(); // cria, se possível
File arquivo = new File(diretorio, "lixo.txt");
FileOutputStream out =
            new FileOutputStream(arquivo);
// se arquivo não existe, tenta criar
out.write( new byte[]{'l','i','x','o'} );

File subdir = new File(diretorio, "subdir");
subdir.mkdir();
String[] arquivos = diretorio.list();
for (int i = 0; arquivos.length; i++) {
   File filho = new File(diretorio, arquivos[i]);
   System.out.println(filho.getAbsolutePath());
}
if (arquivo.exists()) {
   arquivo.delete();
}
```

O bloco de código acima precisa tratar IOException

Há várias fontes de onde se deseja ler ou destinos para onde se deseja gravar ou enviar dados

- *Arquivos*
- *Conexões de rede*
- *Console (teclado / tela)*
- *Memória*

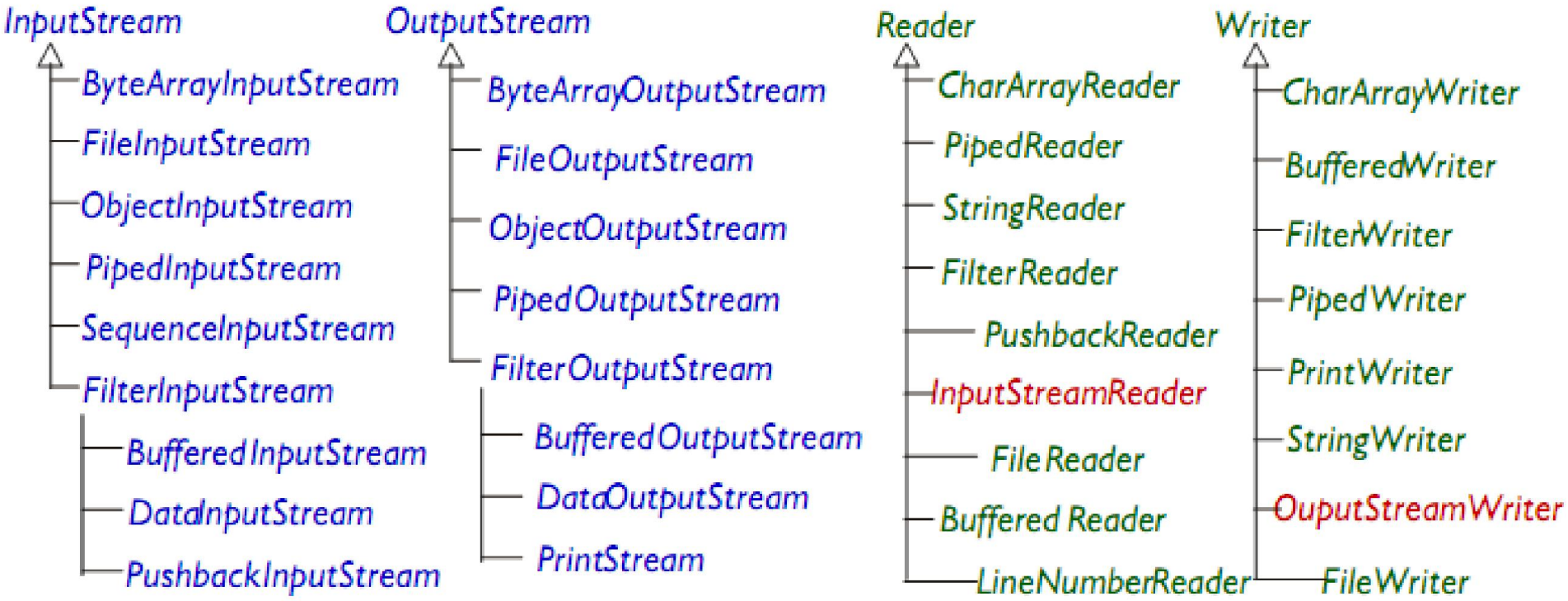
Há várias formas diferentes de ler/escrever dados

- *Seqüencialmente / aleatoriamente*
- *Como bytes / como caracteres*
- *Linha por linha / palavra por palavra, ...*

APIs Java para I/O oferecem objetos que abstraem fontes/destinos (nós) e fluxos de bytes e caracteres

Dois grupos:

- e/s de bytes: InputStream e OutputStream
- e/s de chars: Reader e Writer



InputStream

- Classe genérica (abstrata) para lidar com fluxos de bytes de entrada e nós de fonte (dados para leitura).
- Método principal: read()

OutputStream

- Classe genérica (abstrata) para lidar com fluxos de bytes de saída e nós de destino (dados para gravação).
- Método principal: write()

Principais implementações

- Nós (fontes): FileInputStream (arquivo), ByteArrayInputStream (memória) e PipedInputStream (pipe).
- Processamento de entrada: FilterInputStream (abstract) e subclasses
- Nós (destinos): FileOutputStream (arquivo), ByteArrayOutputStream (memória) e PipedOutputStream (pipe).
- Processamento de saída: FilterOutputStream (abstract) e subclasses.

Principais métodos de InputStream

- int read(): retorna um byte (ineficiente)
- int read(byte[] buffer): coloca bytes lidos no vetor passado como parâmetro e retorna quantidade lida
- int read(byte[] buffer, int offset, int length): idem
- void close(): fecha o stream
- int available(): número de bytes que há para ler agora

Métodos de OutputStream

- void write(int c): grava um byte (ineficiente)
- void write(byte[] buffer)
- void write(byte[] buffer, int offset, int length)
- void close(): fecha o stream (essencial)
- void flush(): esvazia o buffer

- *Trecho de programa que copia um arquivo**

```
String nomeFonte = args[0];
String nomeDestino = args[1];
File fonte = new File(nomeFonte);
File destino = new File(nomeDestino);
if (fonte.exists() && !destino.exists()) {
    FileInputStream in = new FileInputStream(fonte);
    FileOutputStream out = new FileOutputStream(destino);
    byte[] buffer = new byte[8192];
    int length = in.read(buffer);
    while ( length != -1) {
        out.write(buffer, 0, length);
        in.read(buffer);
    }
    in.close();
    out.flush();
    out.close();
}
```

-1 sinaliza EOF

Grava apenas os bytes lidos (e não o buffer inteiro)

```java
// objeto do tipo File
File tanque = new File("agua.txt");

// referência FileInputStream
// cano conectado no tanque
FileInputStream cano =
                    new FileInputStream(tanque);

// lê um byte a partir do cano
byte octeto = cano.read();
```

InputStreamReader é um filtro que converte bytes em chars

- *Para ler chars de um arquivo pode-se usar diretamente um FileWriter em vez de concatenar os filtros abaixo.*

```
// objeto do tipo File
File tanque = new File("agua.txt");

// referência FileInputStream
// cano conectado no tanque
FileInputStream cano =
                  new FileInputStream(tanque);

// filtro chf conectado no cano
InputStreamReader chf =
              new InputStreamReader(cano);

// lê um char a partir do filtro chf
char letra = chf.read();
```

```
// filtro chf conectado no cano
InputStreamReader chf = new InputStreamReader(cano);
// filtro br conectado no chf
BufferedReader br = new BufferedReader (chf);
// lê linha de texto a de br
String linha = br.readLine();
```

Reader

- Classe abstrata para lidar com fluxos de caracteres de entrada: método read() lê um caractere (16 bits) por vez

Writer

- Classe abstrata para lidar com fluxos de bytes de saída: método write() grava um caractere (16 bits) por vez

Principais implementações

- Nós (destinos): FileWriter (arquivo), CharArrayWriter (memória), PipedWriter (pipe) e StringWriter (memória).
- Processamento de saída: FilterWriter (abstract), BufferedWriter, OutputStreamWriter (conversor de bytes para chars), PrintWriter
- Nós (fontes): FileReader (arquivo), CharArrayReader (memória), PipedReader (pipe) e StringReader (memória).
- Processamento de entrada: FilterReader (abstract), BufferedReader, InputStreamReader (conversor bytes p/ chars), LineNumberReader

Principais métodos de Reader

- int read(): lê um char (ineficiente)
- int read(char[] buffer): coloca chars lidos no vetor passado como parâmetro e retorna quantidade lida
- int read(char[] buffer, int offset, int length): idem
- void close(): fecha o stream
- int available(): número de chars que há para ler agora

Métodos de Writer

- void write(int c): grava um char (ineficiente)
- void write(char[] buffer)
- void write(char[] buffer, int offset, int length)
- void close(): fecha o stream (essencial)
- void flush(): esvazia o buffer

A maneira mais eficiente de ler um arquivo de texto é usar FileReader decorado por um BufferedReader. Para gravar, use um PrintWriter decorando o FileWriter

```
BufferedReader in = new BufferedReader(
    new FileReader("arquivo.txt"));
StringBuffer sb =
    new StringBuffer(new File("arquivo.txt").length());
String linha = in.readLine();
while( linha != null ) {
    sb.append(linha).append('\n');
    linha = in.readLine();
}
in.close();
String textoLido = sb.toString();
// (...)
PrintWriter out = new PrintWriter(
    new FileWriter("ARQUIVO.TXT"));
out.print(textoLido.toUpperCase());
out.close();
```

A entrada padrão (System.in) é representada por um objeto do tipo InputStream.
O exemplo abaixo lê uma linha de texto digitado na entrada padrão e grava em uma String. Em seguida lê seqüencialmente a String e imprime uma palavra por linha

```
BufferedReader stdin = new BufferedReader(
    new InputStreamReader(System.in));
System.out.print("Digite uma linha:");
String linha = stdin.readLine());

StringReader rawIn = new StringReader(linha);
int c;
while((c = rawIn.read()) != -1)
  if ( c == ' ') System.out.println();
  else System.out.print((char)c);
}
```