TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN
KHOA CÔNG NGHỆ PHẦN MỀM

Chương 5
LUỒNG DỮ LIỆU

Quản Lý Tập Tin & Thư Mục

- java.lang.Object
  - +--**java.io.File**
- Lớp File không phục vụ cho việc nhập/xuất dữ liệu trên luồng. Lớp File thường được dùng để biết được các thông tin chi tiết về tập tin cũng như thư mục (tên, ngày giờ tạo, kích thước, …)

Xử lý thư mục – Lớp File

- **Các Constructor:**
  - Tạo đối tượng File từ đường dẫn tuyệt đối

    public ***File****(String pathname)*

    ví dụ: *File f = new File("C:\\Java\\vd1.java ");*
  - Tạo đối tượng File từ tên đường dẫn và tên tập tin tách biệt

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

    ví dụ: *File f = new File("C:\\Java ", "vd1.java ");*
  - Tạo đối tượng File từ một đối tượng File khác

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

    ví dụ: *File dir = new File ("C:\\Java ");*

    File f = new File(dir, "vd1.java ");

Xử lý thư mục – Lớp File

- Một số phương thức thường gặp của lớp File

public String ***getName()***	Lấy tên của đối tượng File
public String ***getPath()***	Lấy đường dẫn của tập tin
public boolean ***isDirectory()***	Kiểm tra xem tập tin có phải là thư mục không?
public boolean ***isFile()***	Kiểm tra xem tập tn có phải là một file không?
…	
public String[] ***list()***	Lấy danh sách tên các tập tin và thư mục con của đối tượng File đang xét và trả về trong một mảng.

Xử lý trên thư mục, tập tin

```
public void copyDirectory(File srcDir, File dstDir) throws IOException {
    if (srcDir.isDirectory()) {
            if (!dstDir.exists()) { dstDir.mkdir(); }
            String[] children = srcDir.list();
            for (int i=0; i<children.length; i++) {
                    copyDirectory(new File(srcDir, children[i]), new
                                            File(dstDir, children[i]));
            }
     }
    else {
            copyFile(srcDir, dstDir);
    }
}
```

I/O Streams

- Perform input/output
- Các nội dụng
  - Byte stream
  - Character stream
- **java.io**

I/O Streams

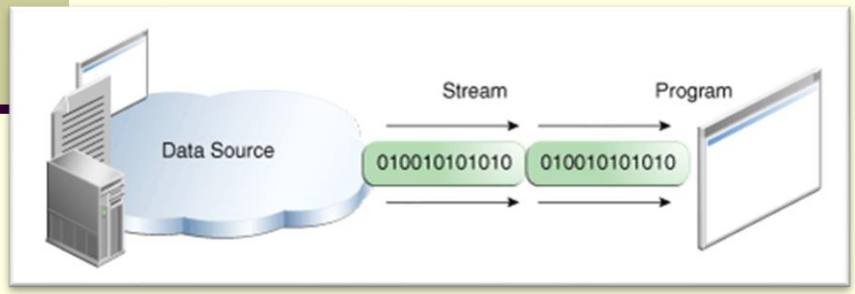
- A stream is a sequence of data
- Input source or an output destination
- represent many different kinds of sources and destinations
  - disk files
  - Devices
  - other programs
  - memory arrays

I/O Streams

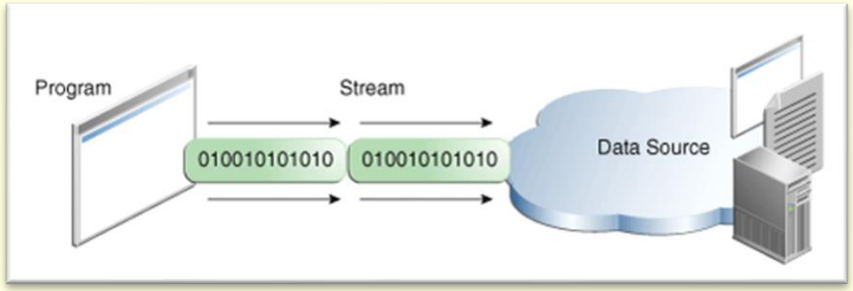
- Support many different kinds of data:
  - simple bytes
  - primitive data types
  - localized characters
  - objects

Model

- ***input stream*** to read data from a source, one item at a time
- ***output stream*** to write data to a destination, one item at time



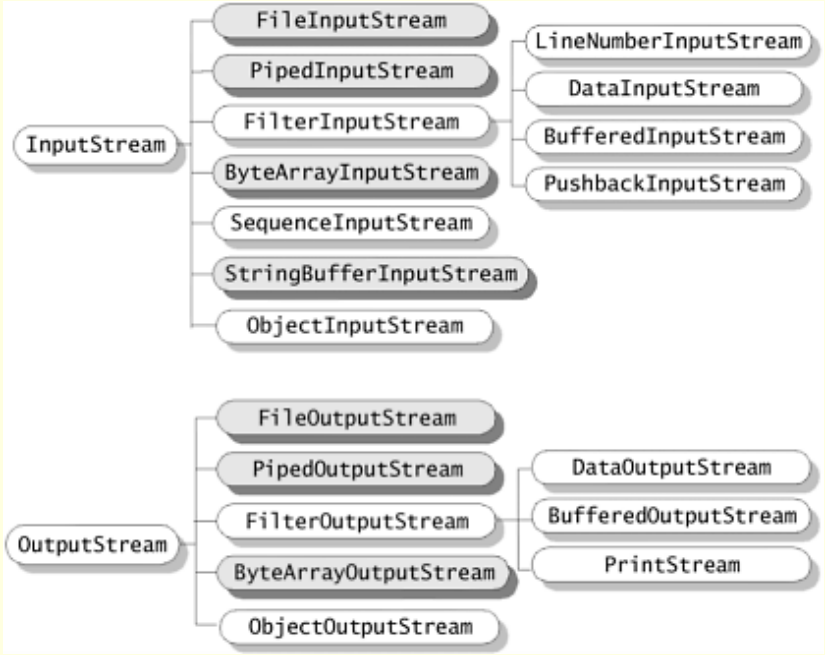
Reading information into a program



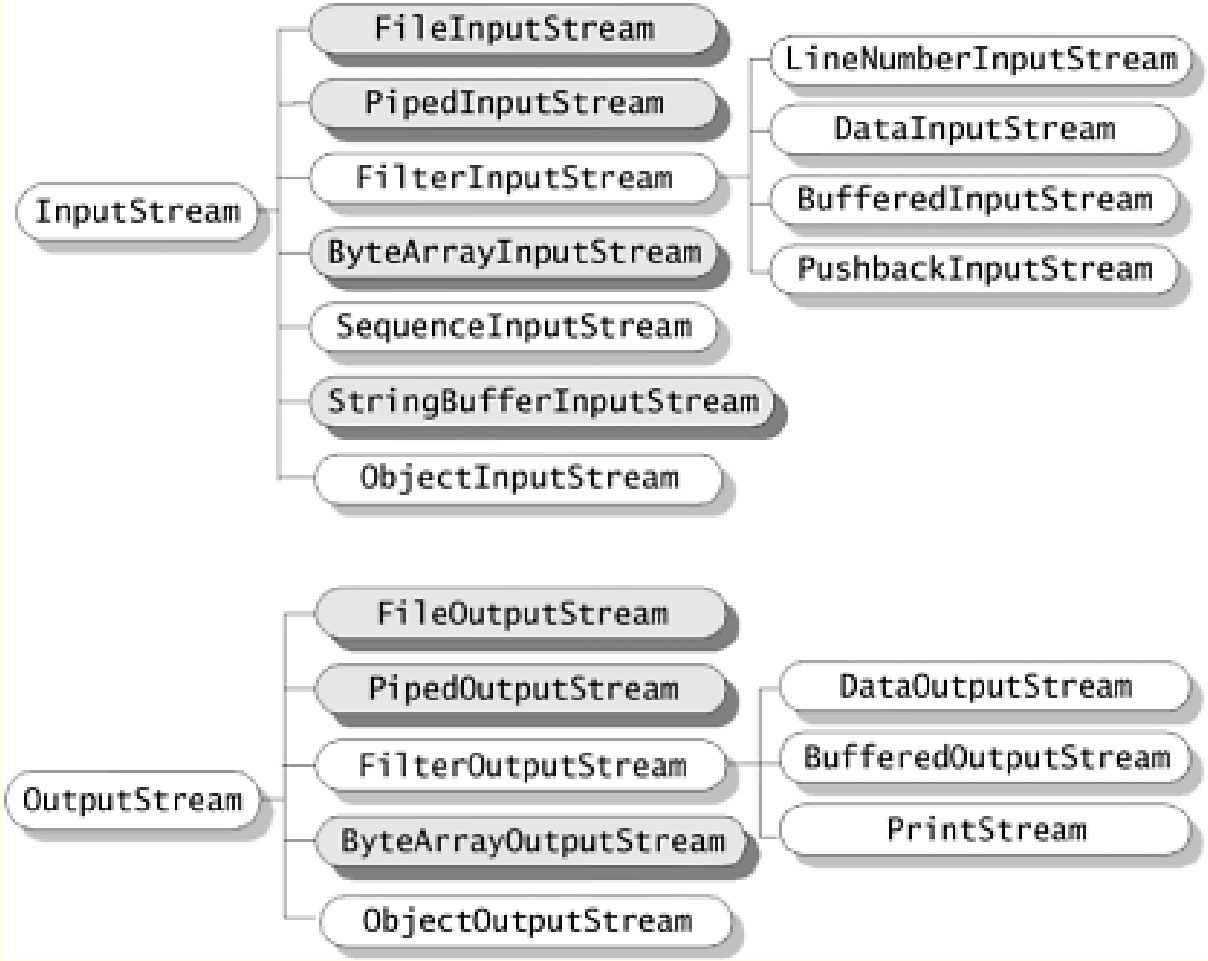
Writing information from a program

Byte Streams

- Perform input and output of 8-bit bytes
- All byte stream classes are descended from **InputStream** and **OutputStream**
- There are many byte stream classes



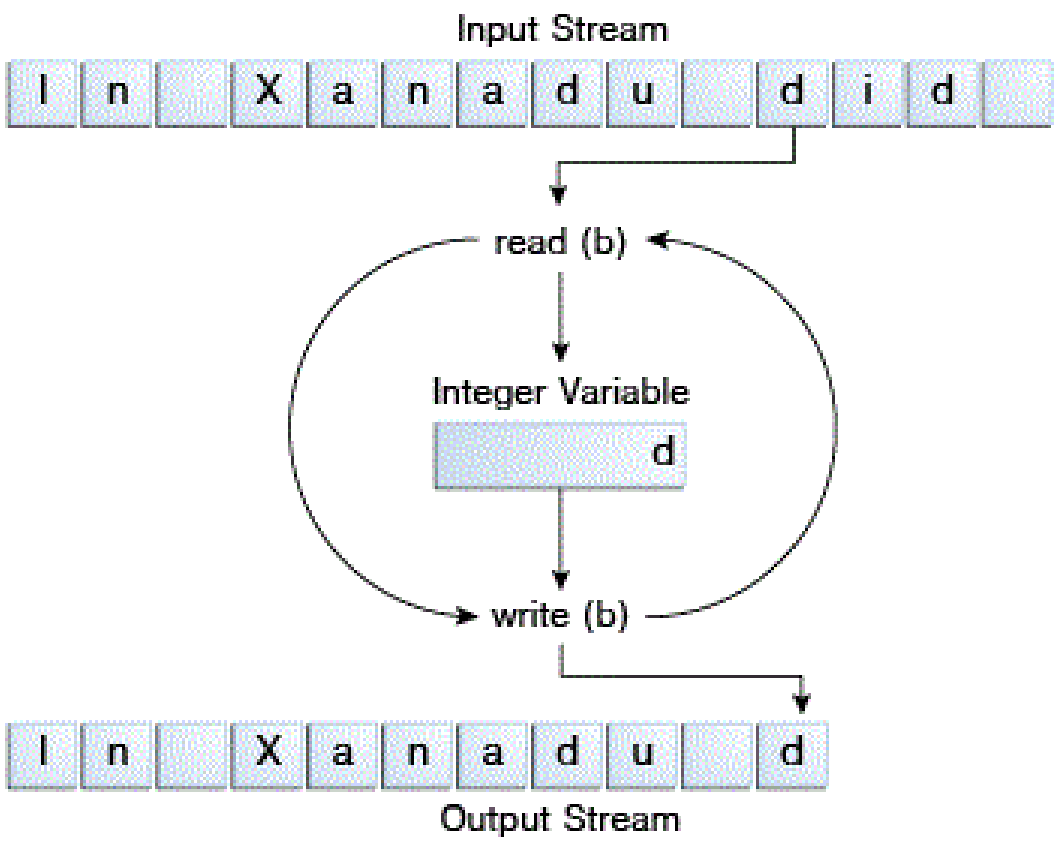
Byte Streams



Using Byte Streams

```
FileInputStream in = null;
FileOutputStream out = null;
try {
        in = new FileInputStream("xanadu.txt");
        out = new FileOutputStream("outagain.txt");
        int c;
        while ((c = in.read()) != -1) {
            out.write(c);
    }
} finally {
    if (in != null) {
        in.close();
    }
    if (out != null) {
        out.close();
    }
}
```

Using Byte Streams



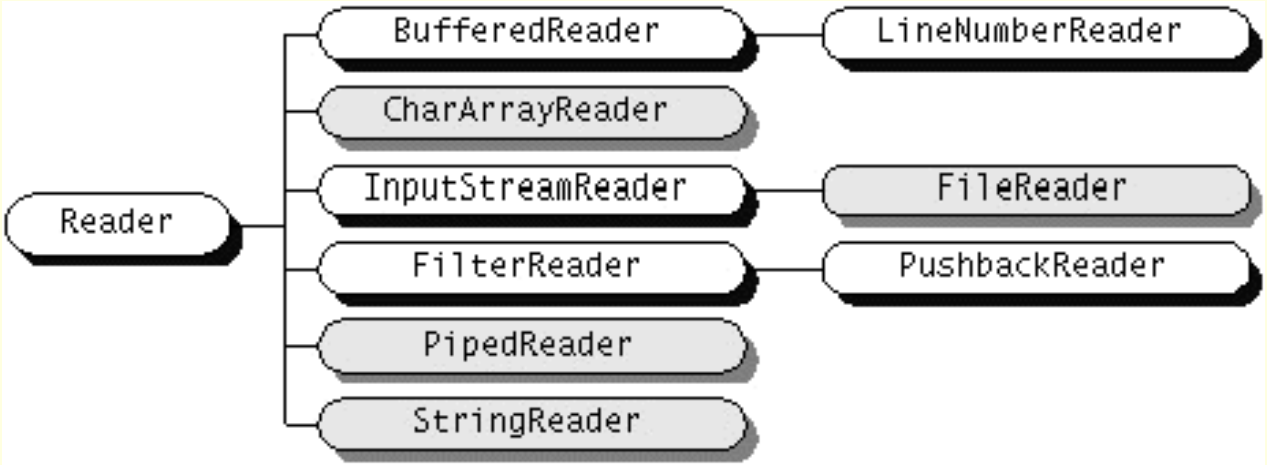
Notices

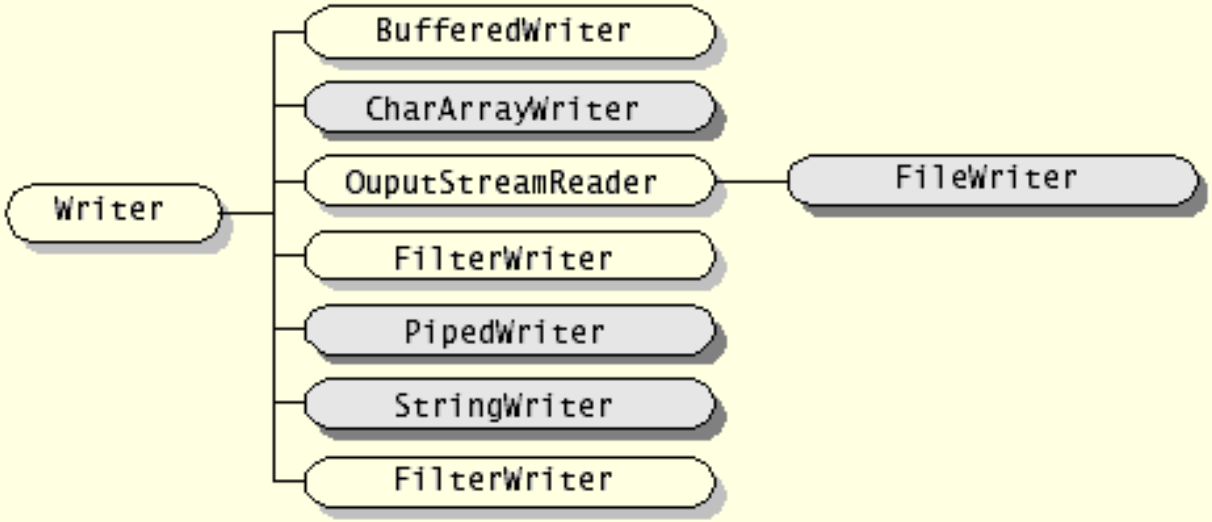
- Initial null value for stream
- Closing a stream when it's no longer needed

When Not to Use

- Contains character data
- Use **character streams**
- Byte streams should only be used for the **most primitive I/O**
- All other stream types are built on byte streams

Character Streams





Character Streams

- Java platform stores character values using Unicode conventions
- Character streams is no more complicated than byte streams
- Adapts to the local character set and is ready for internationalization

Character Streams

- Uses the byte stream to perform the physical I/O
- Handles translation between characters and bytes
- FileReader uses FileInputStream, while FileWriter uses FileOutputStream

Using Character Streams

```
FileReader inputStream = null;
  FileWriter outputStream = null;
  try {
      inputStream = new FileReader("xanadu.txt");
      outputStream = new FileWriter("characteroutput.txt");
      int c;
      while ((c = inputStream.read()) != -1) {
          outputStream.write(c);
      }
  } finally {
      if (inputStream != null) {
          inputStream.close();
      }
      if (outputStream != null) {
          outputStream.close();
      }
  }
```

CopyCharacters vs CopyBytes

- CopyCharacters is very similar to CopyBytes
- Uses **FileReader** and **FileWriter**
- CopyBytes and CopyCharacters use an **int** variable to read to and write from
- CopyCharacters: holds a character value in its last 16 bits; in CopyBytes: holds a byte value in its last 8 bits

“Bridge” streams

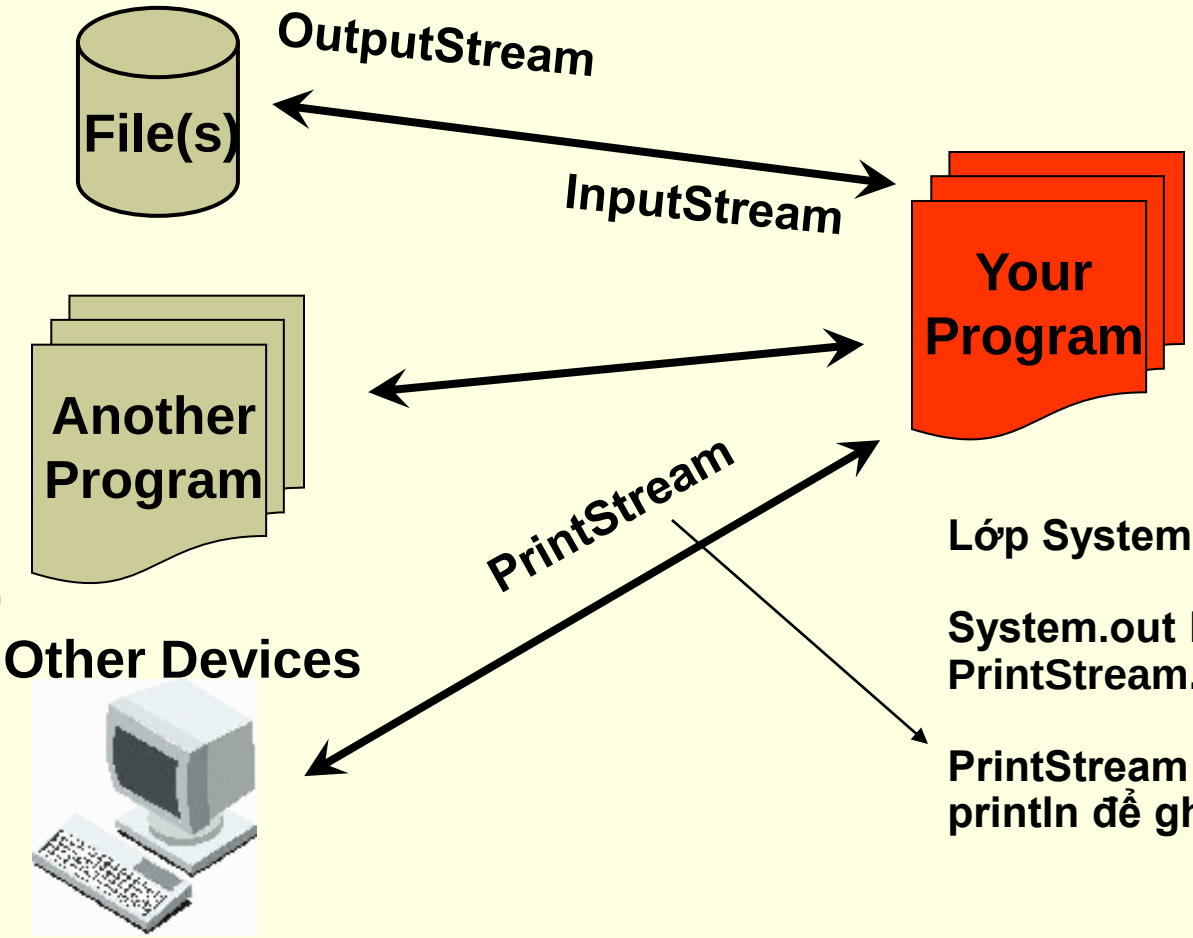
- Two general-purpose byte-to-character
- **InputStreamReader** and **OutputStreamWriter**
- Create character streams when there are no prepackaged character stream classes
- **Ex:** create character streams from the byte streams provided by socket classes

More

- Buffered Streams
- Data Streams
- Object Streams

Nhập/xuất dữ liệu

Nhập xuất dữ liệu trong Java dựa trên mô hình luồng dữ liệu



Lớp System có: in, out

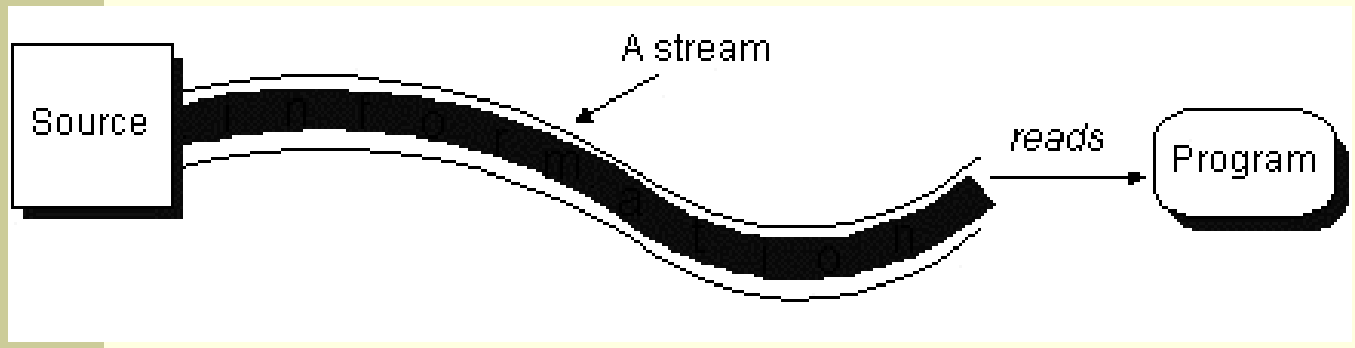
System.out là 1 thể hiện của lớp PrintStream.

PrintStream có phương thức print, println để ghi dữ liệu xuống luồng.

Nhập xuất dữ liệu

- Dữ liệu có thể đến từ bất kỳ nguồn nào và xuất ra bất kỳ nơi nào
  - Memory
  - Disk
  - Network
- Bất kể nguồn/đích loại nào đều có thể sử dụng luồng để đọc/ghi dữ liệu.

Nhập xuất dữ liệu



Đọc dữ liệu

Open a Stream

While more Information

Read

Close the Stream

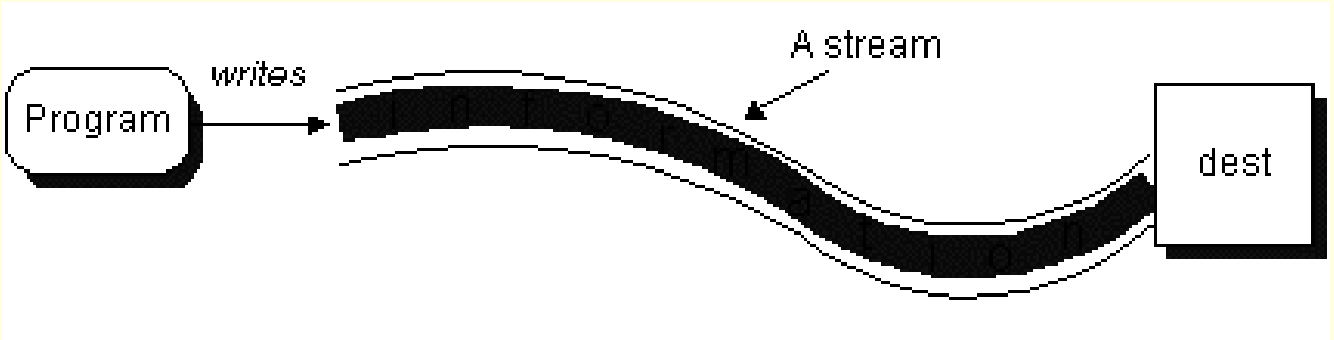
Ghi dữ liệu

Open a Stream

While more Information

Write

Close the Stream

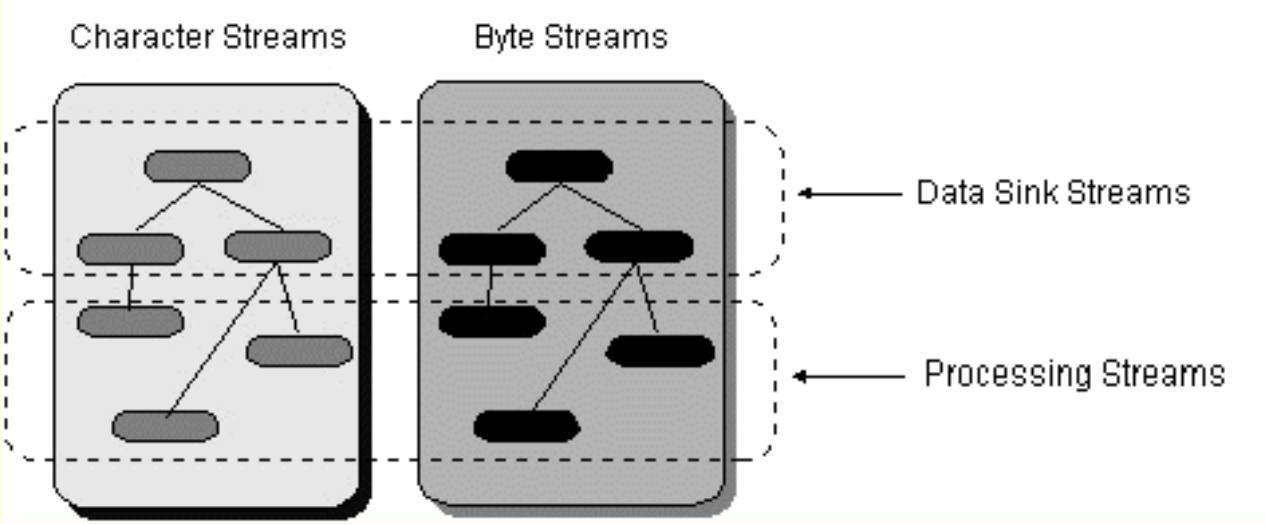


Luồng dữ liệu

- Các luồng là những đường ống dẫn để gửi và nhận thông tin trong các chương trình java.
- Khi một luồng đọc hoặc ghi, các luồng khác bị khoá.
- Nếu lỗi xảy ra trong khi đọc hoặc ghi luồng, một ngoại lệ sẽ kích hoạt.

Luồng dữ liệu

- Gói java.io cung cấp các lớp cài đặt luồng dữ liệu.
- Phân loại luồng



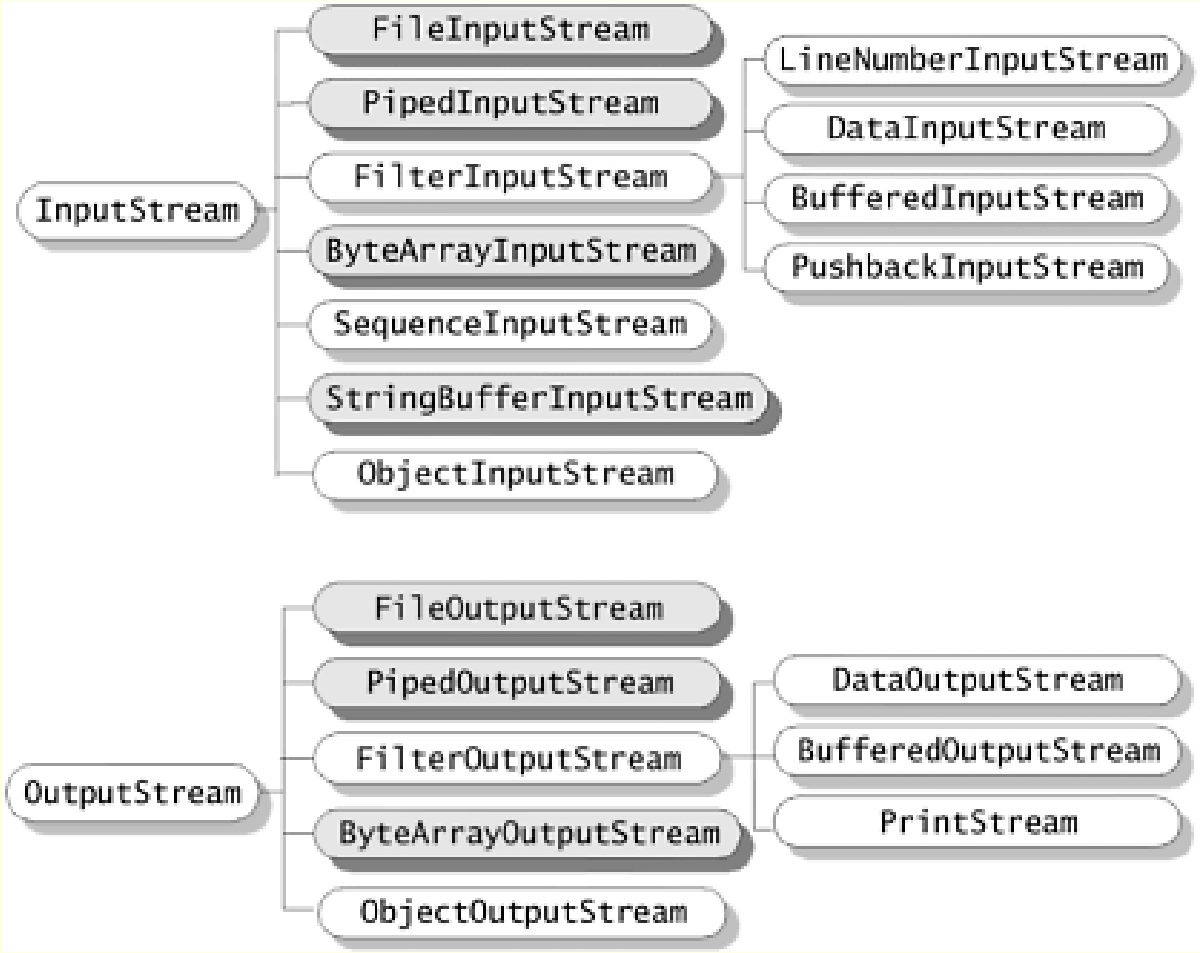
Luồng dữ liệu

- *Luồng Character* được dùng khi thao tác trên ký tự (16 bits) – Sử dụng lớp *Reader* & *Writer*
- *Byte Streams* are được dùng khi thao tác dữ liệu nhị phân (8 bits) – Sử dụng *InputStream* & *OutputStream* Classes
- Data Sinks
  - Files
  - Memory
  - Pipes
- Processing
  - Buffering
  - Filtering

Các lớp luồng dữ liệu

- Lớp System.out.
- Lớp System.in.
- Lớp System.err.

Luồng Byte


InputStream
FileInputStream
PipedInputStream
FilterInputStream
ByteArrayInputStream
SequenceInputStream
StringBufferInputStream
ObjectInputStream
LineNumberInputStream
DataInputStream
BufferedInputStream
PushbackInputStream
OutputStream
FileOutputStream
PipedOutputStream
FilterOutputStream
ByteArrayOutputStream
ObjectOutputStream
DataOutputStream
BufferedOutputStream
PrintStream

Lớp InputStream

- Là lớp trừu tượng
- Định nghĩa cách nhận dữ liệu
- Cung cấp số phương thức dùng để đọc và các luồng dữ liệu làm đầu vào.
- Các phương thức:
  - **int read()**
  - **int read(byte[] buffer)**
  - **int read(byte[] buffer, int offset, int length)**
  - **int available( )**
  - **void close ( )**
  - **void reset( )**
  - **long skip( )**

Lớp InputStream

- int read()

```
byte[] b = new byte[10];
for (int i = 0; i < b.length; i++) {
    b[i] = (byte) System.in.read();
}
```

Lớp InputStream

- int read()

```
public class StreamPrinter {
    public static void main(String[] args) {
        try {
           while (true) {
              int datum =  System.in.read( );
              if (datum == -1) break;
              System.out.println(datum);
            }
        } catch (IOException ex) {
            System.err.println("Couldn't read from System.in!");
        }
    }
}
```

Lớp InputStream

- int read(byte[] buffer, int offset, int length)

```
try {
    byte[] b = new byte[100];
    int offset = 0;
    while (offset < b.length) {
        int bytesRead = System.in.read(b, offset, b.length - offset);
        if (bytesRead == -1) break; // end of stream
        offset += bytesRead;
    }
}
catch (IOException ex) {
    System.err.println("Couldn't read from System.in!");
}
```

Lớp InputStream

- int available()

```
try {
    byte[] b = new byte[System.in.available( )];
    System.in.read(b);
}
 catch (IOException ex) {
    System.err.println("Couldn't read from System.in!");
}
```

Lớp InputStream

- long skip()

```
try {
    long bytesSkipped = 0;
    long bytesToSkip = 80;
    while (bytesSkipped < bytesToSkip) {
            long n = in.skip(bytesToSkip - bytesSkipped);
            if (n == -1) break;
            bytesSkipped += n;
    }
 }
catch (IOException ex) { System.err.println(ex); }
```

Lớp OutputStream

- Là lớp trừu tượng.
- Định nghĩa cách ghi dữ liệu vào luồng.
- Cung cấp tập các phương thức trợ giúp. trong việc tạo, ghi và xử lý các luồng xuất.
- Các phương thức:
  - **void write(int)**
  - **void write(byte[ ])**
  - **write(byte[ ], int, int)**
  - **void flush( )**
  - **void close( )**

Lớp OutputStream

- void write(int i)

```
public class AsciiChart {
    public static void main(String[] args) {
        for (int i = 32; i < 127; i++) {
            System.out.write(i);
            // break line after every eight characters.
            if (i % 8 == 7) System.out.write('\n');
            else System.out.write('\t');
        }
        System.out.write('\n');
    }
}
```

Lớp OutputStream

- void write(byte[] buff)
- void write(byte[] buff, int offset, int length)

```
public class WriteBytes
    public static void main(String[] args){
        try{
            String message = “Hello World”;
            byte[] data = message.getBytes();
            System.out.write(data);
        catch(IOException e)
        {
            System.out.println(“IO errors”);
        }
    }
}
```

Lớp OutputStream

```
public class StreamCopier {
    public static void main(String[] args) {
        try {
            copy(System.in, System.out); }
        catch (IOException ex) {
            System.err.println(ex);
        }
    }
    public static void copy(InputStream in, OutputStream out) throws
                                                        IOException {
        byte[] buffer = new byte[1024];
        while (true) {
            int bytesRead = in.read(buffer);
            if (bytesRead == -1) break;
            out.write(buffer, 0, bytesRead);
        }
    }
}
```

Lớp FileOutputStream

- Cho phép kết xuất để ghi ra một luồng tập tin
- Các đối tượng cũng tạo ra sử dụng một chuỗi tên tập tin, tạp tin, hay đối tượng FileDescriptor như một tham số.
- Lớp này nạp chồng các phương thức của lớp OutputStream và cung cấp phương thức ‘finalize( )’ và ‘getFD( )’

Sử Dụng FileOutputStream

```
byte[] originalData = new byte[10];
for (int i=0; i<originalData.length; i++)
  {
  originalData[i]=(byte)(Math.random()*128.0);
  }
FileOutputStream fw=null;
try { fw = new FileOutputStream("io1.dat"); }
catch (IOException fe )
  { System.out.println(fe); }
for (int i=0; i<originalData.length; i++)
  {
  try { fw.write(originalData[i]); }
  catch (IOException ioe)
  {System.out.println(ioe); }
  }
try { fw.close(); }
catch (IOException ioe)
  {System.out.println(ioe); }
```

Lớp FileInputStream

- Cho phép đầu vào đọc từ một tập tin trong một mẫu của một dòng
- Các đối tượng được tạo ra sử dụng chuỗi tên tập tin, tập tin, đối tượng FileDescriptor như một tham số.
- Các phương thức nạp chồng của lớp InputStream. nó cung cấp phương thức ‘finalize( )’ và ‘getFD( )’

Sử Dụng FileInputStream

```
byte data=0;
int bytesInFile=0;
FileInputStream fr = null;
try
  {
  fr = new FileInputStream("io1.dat");
  bytesInFile = fr.available();
  for (int i=0; i<bytesInFile; i++)
      {
      data=(byte)fr.read();
      System.out.println("Read "+data);
      }
  fr.close();
  }
catch (IOException ioe)
  {
  System.out.println("Problem with file");
  }
```

ByteArrayInput

- Sử dụng các đệm bộ nhớ
- Lớp **ByteArrayInputStream**
  - **ByteArrayInputStream(byte[] buf)**
- Tạo ra một luồng nhập từ đệm bộ nhớ mảng các byte.
  - Không hỗ trợ các phương thức mới
  - Các phương thức nộp chồng của lớp InputStream, giống như ‘read()’, ‘skip()’, ‘available()’ và ‘reset()’.

Byte Array Output

- Sử dụng các vùng đệm bộ nhớ
- Lớp **ByteArrayOutputStream**
  - Tạo ra một luồng kết xuất trên mảng byte
  - Cung cấp các khả năng bổ sung cho mảng kết xuất tăng trưởng nhằm chừa chổ cho dữ liệu mới ghi vào.
  - Cũng cung cấp các phương thức để chuyển đổi luồng tới mảng byte, hay đối tượng String.

Phương thức của lớp

- **ByteArrayOutputStream** :
  - **ByteArrayOutputStream()**
  - **void reset( )**
  - **int size( )**
  - **byte[] toByteArray()**
  - **String toString()**

Bộ lọc

- Lọc:
  - Là kiểu luồng sửa đổi cách điều quản một luồng hiện có.
  - Về cơ bản được sử dụng để thích ứng các luồng theo các nhu cầu của chương trình cụ thể.
  - Bộ lọc nằm giữa luồng cơ sở và CT.
  - Thực hiện một số tiến trình đặt biệt trên các byte được chuyển giao từ đầu vào đến kết xuất.
  - Có thể phối hợp để thực hiện một dãy các tuỳ chọn lọc.

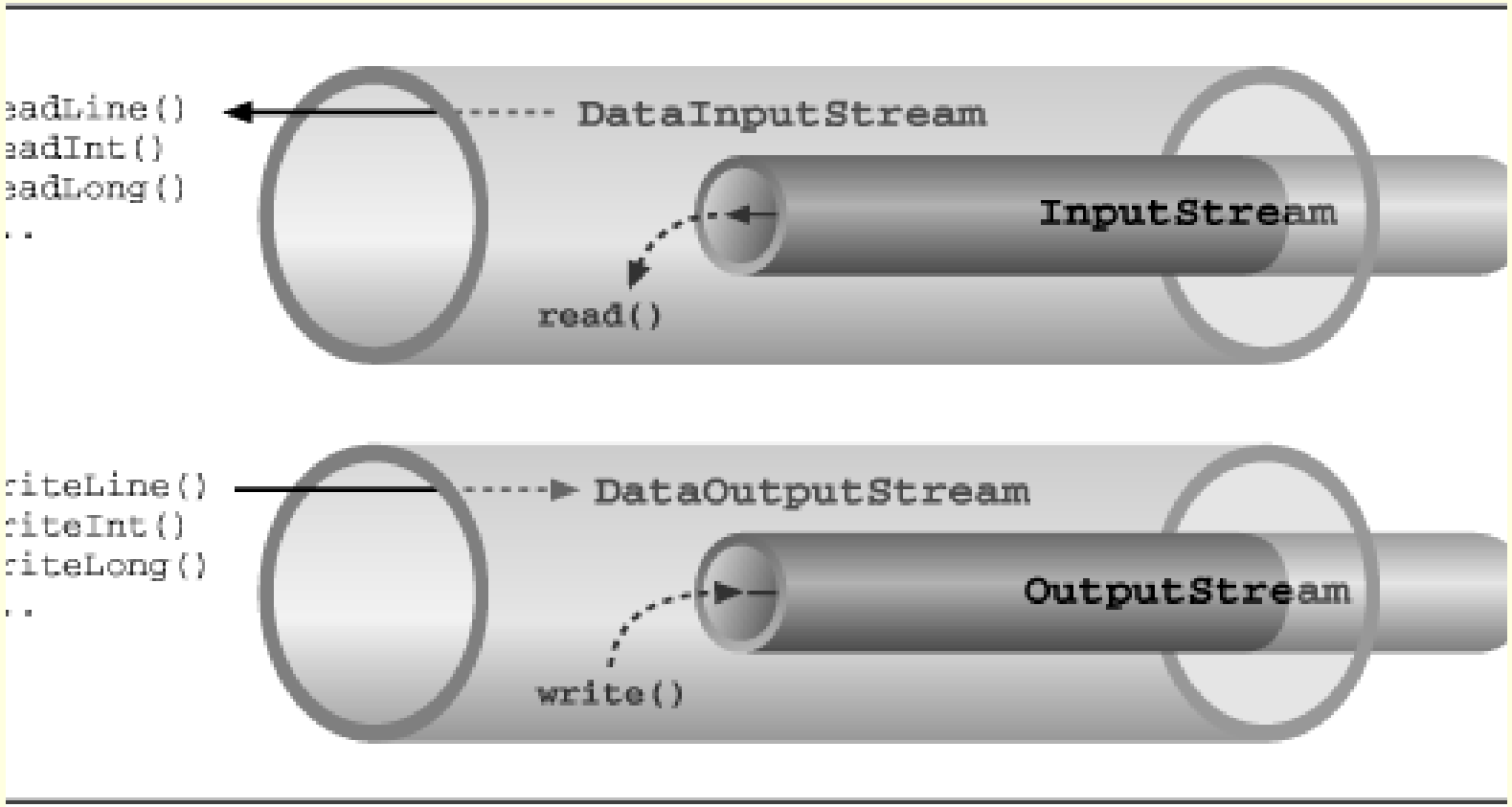
Lớp FilterInputStream

- Là lớp trừu tượng.
- Là cha của tất cả các lớp luồng nhập đã lọc.
- Cung cấp khả năng tạo ra một luồng từ luồng khác.
- Một luồng có thể đọc và cung cấp cung cấp dưới dạng kết xuất cho luồng khác.
- Duy trì một dãy các đối tượng của lớp ‘InputStream’
- Cho phép tạo ra nhiều bộ lọc kết xích (chained filters).

Lớp FilterOutputStream

- Là dạng bổ trợ cho lớp ‘FilterInputStream’.
- Là cha của tất cả các lớp luồng kết xuất.
- Duy trì đối tượng của lớp ‘OutputStream’ như là một biến ‘out’.
- Dữ liệu ghi ra lớp này có thể sửa đổi để thực hiện các thao tác lọc, và sau đó phản hồi đến đối tượng ‘OutputStream’.

Luồng Lọc


eadLine()
eadInt()
eadLong()
..
DataInputStream
InputStream
read()
riteLine()
riteInt()
riteLong()
..
DataOutputStream
OutputStream
write()

Vùng đệm nhập/xuất

- Vùng đệm:
  - Là kho lưu trữ dữ liệu.
  - Có thể cung cấp dữ liệu thay vì quay trợ lại nguồn dữ liệu gốc ban đầu.
  - Java sử dụng vùng đệm nhập và kết xuất để tạm thời lập cache dữ liệu được đọc hoặc ghi vào một luồng.
- Trong khi thực hiện vùng đệm nhập:
  - Số lượng byte lớn được đọc cùng thời điểm, và lưu trữ trong một vùng đệm nhập.
  - Khi chương trình đọc luồng nhập, các byte nhập được đọc vào vùng đệm nhập.

Vùng đệm nhập/xuất (tt…)

- Trong trường hợp vùng đệm kết xuất, một chương trình ghi ra một luồng.
- Dữ liệu kết xuất đựơc lưu trữ trong một vùng đệm kết xuất.
- Dữ liệu được lưu trữ cho đến khi vùng đệm trợ nên đầy, hay luồng kết xuất được xả trống.
- Kết thúc, vùng đệm kết xuất được chuyển gửi đến đích của luồng xuất.

Lớp BufferedInputStream

- Tự động tạo ra và duy trì vùng đệm để hổ trợ vùng đệm nhập.
- bởi lớp ‘BufferedInputStream’ là một bộ đệm, nó có thể áp đụng cho một số các đối tượng nhất định của lớp ‘InputStream’.
- Cũng có thể phối hợp các tập tin đầu vào khác.
- Sử dụng vài biến để triển khai vùng đệm nhập.

Lớp BufferedInputStream (Contd…)

- Định nghĩa hai phương thức thiết lập:
  - Một cho phép chỉ định kích thước của vùng đệm nhấp.
  - phương thức kia thì không.
- Cả hai phương thức thiết lập đều tiếp nhận một đối tượng của lớp ‘InputStream’ như một tham số.
- Nạp chồng các phương thức truy cập mà InputStream cung cấp, và không đưa vào bất kỳ phương thức mới nào.

Lớp BufferedOutputStream

- Thực hiện vùng đệm kết xuất theo cách tương ứng với lớp ‘BufferedInputStream’.
- Định nghĩa hai phương thức thiết lập. Nó cho phép chúng ta ấn định kích thước của vùng đệm xuất trong một phương thức thiết lập, cũng giống như cung cấp kích thước vùng đệm mặc định.
- Nạp chồng tất cả phương thức của lớp ‘OutputStream’ và không đưa vào bất kỳ phương thức nào.

Giao diện DataInput

- Được sử dụng để đọc các byte từ luồng nhị phân, và xây dựng lại dữ liệu trong một số kiểu dữ liệu nguyên thuỷ.
- Cho phép chúng ta chuyển đổi dữ liệu từ từ khuôn dạng UTF-8 được sửa đổi Java đến dạng chuỗi
- Định nghiã số phương thức, bao gồm các phương thức để đọc các kiểu dữ liệu nguyên thuỷ.

Những phương thức giao diện DataInput

- boolean readBoolean( )
- byte readByte( )
- char readChar( )
- short readShort( )
- long readLong( )
- float readFloat( )
- int readInt( )
- double readDouble( )
- String readUTF( )
- String readLine( )

Giao diện DataOutput

- Được sử dụng để xây dựng lại dữ liệu một số kiểu dữ liệu nguyên thuỷ vào trong dãy các byte
- Ghi các byte dữ liệu vào luồng nhị phân
- Cho phép chúng ta chuyển đổi một chuỗi vào khuôn dạng UTF-8 được sửa đổi Java và viết nó vào trong một dãy.
- Định nghĩa một số phương thức và tất cả phương thức kích hoạt IOException trong trường hợp lỗi.

Các phương thức giao diện DataOutput

- void writeBoolean(boolean b)
- void writeByte(int value)
- void writeChar(int value)
- void writeShort(int value)
- void writeLong(long value)
- void writeFloat(float value)
- void writeInt(int value)
- void writeDouble(double value)
- void writeUTF(String value)

Mảng byte sang int

```
public class ArrayCopy {

    public static int[] byte2int(byte[]src) {
        int dstLength = src.length >>> 2;
        int[]dst = new int[dstLength];

        for (int i=0; i<dstLength; i++) {
            int j = i << 2;
            int x = 0;
            x += (src[j++] & 0xff) << 0;
            x += (src[j++] & 0xff) << 8;
            x += (src[j++] & 0xff) << 16;
            x += (src[j++] & 0xff) << 24;
            dst[i] = x;
        }
        return dst;
    }
}
```

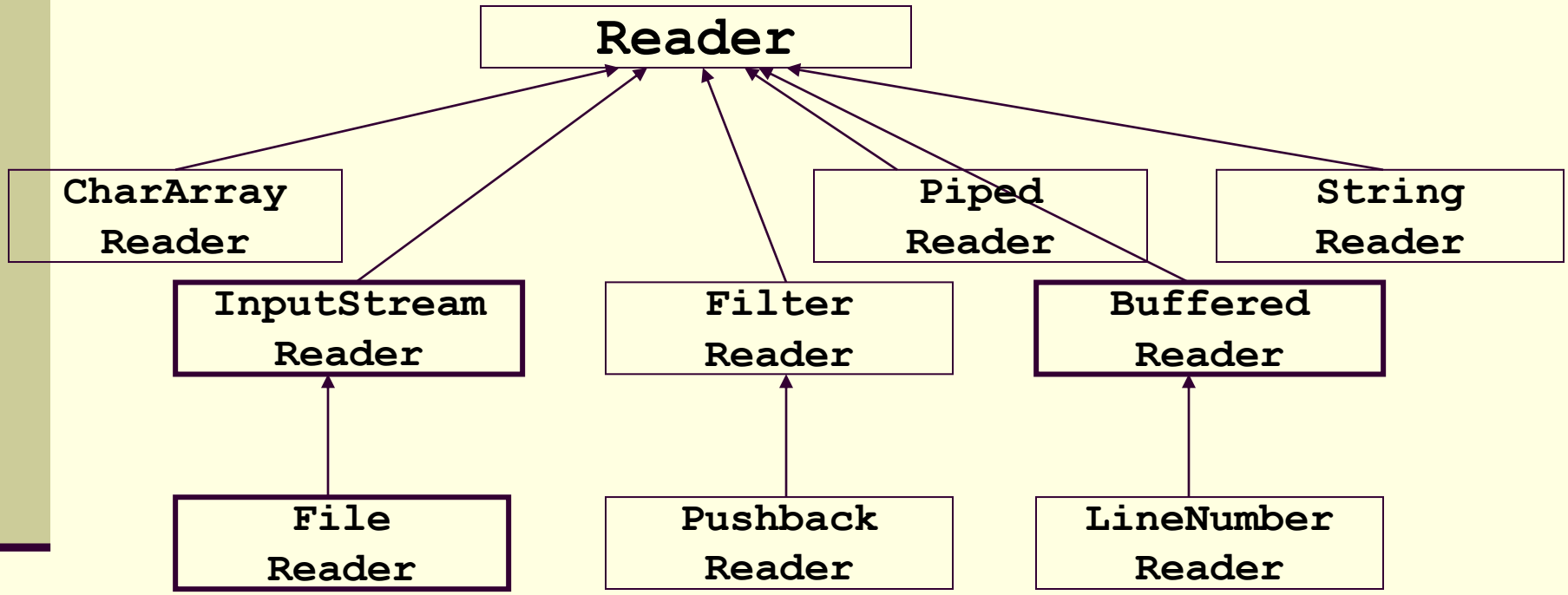
Sử Dụng DataOutputStream

```
int[] originalData = new int[10];
for (int i=0; i<originalData.length; i++)
  originalData[i]=(int)(Math.random()*1000.0);
FileOutputStream fos=null;
try { fos = new FileOutputStream("io2.dat"); }
catch (IOException fe )
  { System.out.println("Cant make new file"); }
DataOutputStream dos = new DataOutputStream(fos);
for (int i=0; i<originalData.length; i++)
  {
  try { dos.writeInt(originalData[i]); }
  catch (IOException ioe)
   {System.out.println("Cant write to file"); }
  }
```

Lớp Reader và Writer

- Là các lớp trừu tượng.
- Chúng nằm tại đỉnh của hệ phân cấp lớp, hỗ trợ việc đọc và ghi các luồng ký tự unicode.

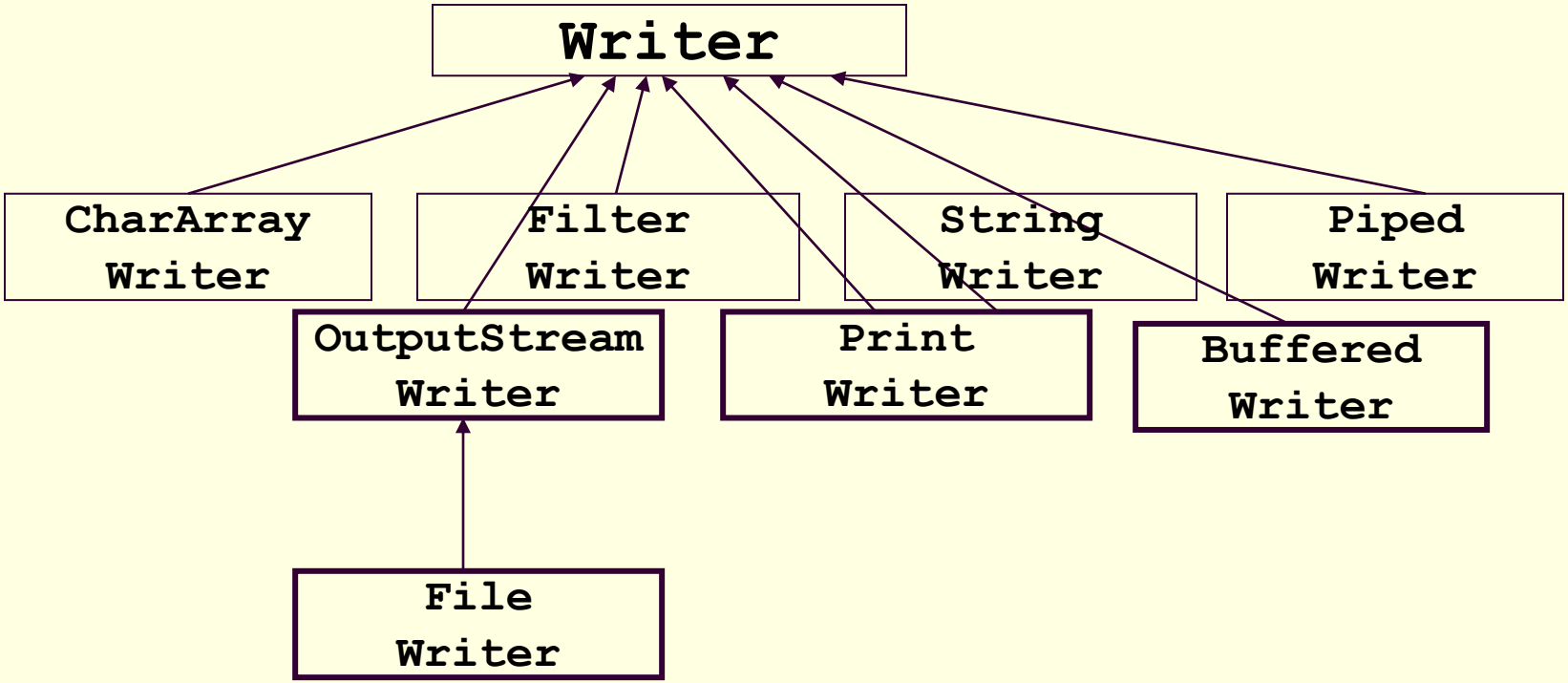
Lớp Reader


Reader
CharArray Reader
InputStream Reader
Filter Reader
Piped Reader
Buffered Reader
String Reader
File Reader
Pushback Reader
LineNumber Reader

Lớp Reader

- Hỗ trợ các phương thức sau:
  - **int read( )**
  - **int read(char[] data)**
  - **int read(char[] data, int offset, int len)**
  - **void reset( )**
  - **long skip( )**
  - **void close( )**
  - **boolean ready( )**

Lớp Writer


Writer
CharArray Writer
Filter Writer
String Writer
Piped Writer
OutputStream Writer
Print Writer
Buffered Writer
File Writer

Lớp Writer

- Hỗ trợ các phương thức sau :
  - **void write(int ch)**
  - **void write(char[] text)**
  - **void write(String str)**
  - **void write(String str, int offset, int len)**
  - **void flush( )**
  - **void close( )**

Đọc Tập Tin Văn Bản

```
import java.io.*;
public class FileRead  throws IOException
{
  public static void main(String[] args)
  {
       //all the following, up to the definition of the reader, are in
       //the class java.io.File, which contains a number of methods
       //related to the file attributes and the directory it resides in.

       String fileName = args[0];     // args[0] for file name

      //the next statement creates a reader for the file
      BufferedReader dat = new BufferedReader(new FileReader(fileName));

      System.out.println("DATA FROM THE FILE: " + fileName);
        String line = dat.readLine();
        while (line != null)
        {
            System.out.println(line);
            line = dat.readLine();
        }
       System.out.println("END OF FILE REACHED");
        dat.close();

  }
}
```

//If there is no more data in the file, readLine returns null

Đọc Tập Tin

```
import java.io.*;
public class IntFileRead
{
  public static void main(String[] args)  throws IOException
  {
    File dataf = new File (”vd.txt");
    int number;

    //Create a reader for the file
    FileReader fdat = new FileReader(dataf);
    BufferedReader dat = new BufferedReader(fdat);
    System.out.println("DATA FROM THE FILE:");

    String line = dat.readLine();
    while (line != null)
    {
        number=Integer.parseInt(line);
        System.out.println(number);
        line = dat.readLine();
    }

    System.out.println("END OF FILE REACHED");
    dat.close();
  }//end main
}//end IntFileRead
```

Lớp PrinterWriter

- Thực hiện một kết xuất.
- Lớp này có phương thức bổ sung , trợ giúp in các kiểu dữ liệu cơ bản .
- Lớp PrintWriter thay thế lớp ‘PrintStream’
- Thực tế cải thiện lớp ‘PrintStream’; lớp này dùng một dấu tách dòng phụ thuộc nền tảng điểm các dòng thay vì ký tự ‘\n’.
- Cung cấp phần hỗ trợ cho các ký tự unicode so với PrintStream.
- Các phương thức:
  - **checkError( )**
  - **setError( )**

Ghi Xuống Tập Tin

```
import java.io.*;
public class TextFileWrite
{
   public static void main(String[] args)  throws IOException
   {
        //for this example, set up data we want to write as a four element array of strings
        String [] song = new String [4];
        song[0]="Mary had a little lamb";
        song[1]="Its fleece was white as snow";
        song[2]="And everywhere that Mary went";
        song[3]="The lamb was sure to go";

        //set up the output file to be written to
      String outFileName = “vd.txt”;
        //using PrintWriter allows us to use the print and println commands for files
      File outData = new File(outFileName);
      PrintWriter outDat = new PrintWriter(new FileWriter(outData));

      //Now write the data .....
       for (int line=0; line<song.length; line++)
            outDat.println(song[line]);
      System.out.println("DATA WRITTEN ON OUTPUT FILE: "+ outFileName);
      outDat.close();
   }
}
```

Lớp RandomAccessFile

- Cung cấp khả năng thực hiện I/O theo các vị trí cụ thể bên trong một tập tin.
- Dữ liệu có thể đọc hoặc ghi ngẫu nhiên ở những vị trí bên trong tập tin thay vì một kho lưu trữ thông tin liên tục.
- Phương thức ‘seek( )’ hỗ trợ truy cập ngẫu nhiên.
- Thực hiện cả đầu vào và đầu ra dữ liệu.
- Hỗ trợ các cấp phép đọc và ghi tập tin cơ bản.
- Kế thừa các phương thức từ các lớp ‘DataInput’ và ‘DataOutput’

Các phương thức của lớp RandomAccessFile

- RandomAccessFile(String fn,String mode)
  “r”, “rw”, …..
- seek( )
- getFilePointer( )
- length( )
- readBoolean()
- ….
- writeBoolean()
- …..

Ví dụ

```
RandomAccessFile rf = new
    RandomAccessFile(“doubles.dat”,”rw”);

// read third double: go to 2 * 8 (size of one)
rf.seek(8*2);
double x = rf.readDouble();

// overwrite first double value
rf.seek(0);
rf.writeDouble(x);
rf.close();
```

Câu hỏi ôn tập

- Stream là gì? Khác nhau giữa byte và character stream
- Trình bày mục đích, khởi tạo và cách sử dụng I/O với các lớp?
  - Byte stream
    - FileInputStream, FileOutputStream
    - ByteArrayInputStream, ByteArrayOutputStream
    - BufferedInputStream, BufferedOutputStream
    - DataOutputStream , DataInputStream
    - RandomAccessFile
  - Character stream
    - FileReader, FileWriter
    - CharArrayReader, CharArrayWriter