

NHẬP MÔN JAVA  
**BÀI 8**

# LUỒNG (STREAMS)



# NỘI DUNG

Phần này sẽ cung cấp cho chúng ta những kiến thức cơ bản về luồng (streams) và files:

- Thư viện các lớp về luồng trong java: luồng byte, luồng ký tự.
- Xuất nhập Console dùng luồng byte, luồng ký tự.
- Xuất nhập files dùng luồng ký tự và luồng byte.
- Vấn đề xử lý files truy cập ngẫu nhiên dùng lớp `RandomAccessFile`.
- Xử lý file và thư mục dùng lớp `File`.

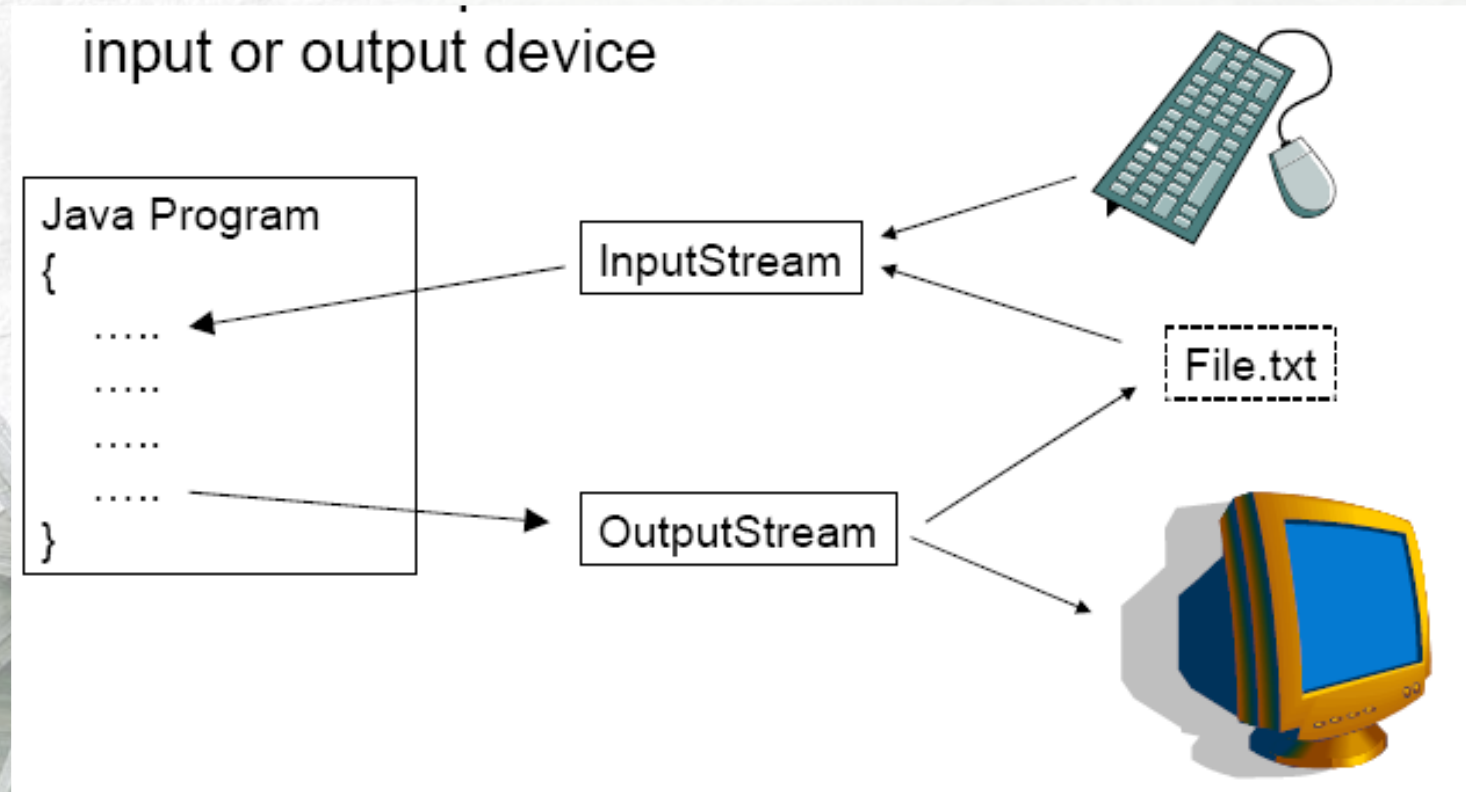


# PHẦN 1

# KHÁI NIỆM LUỒNG

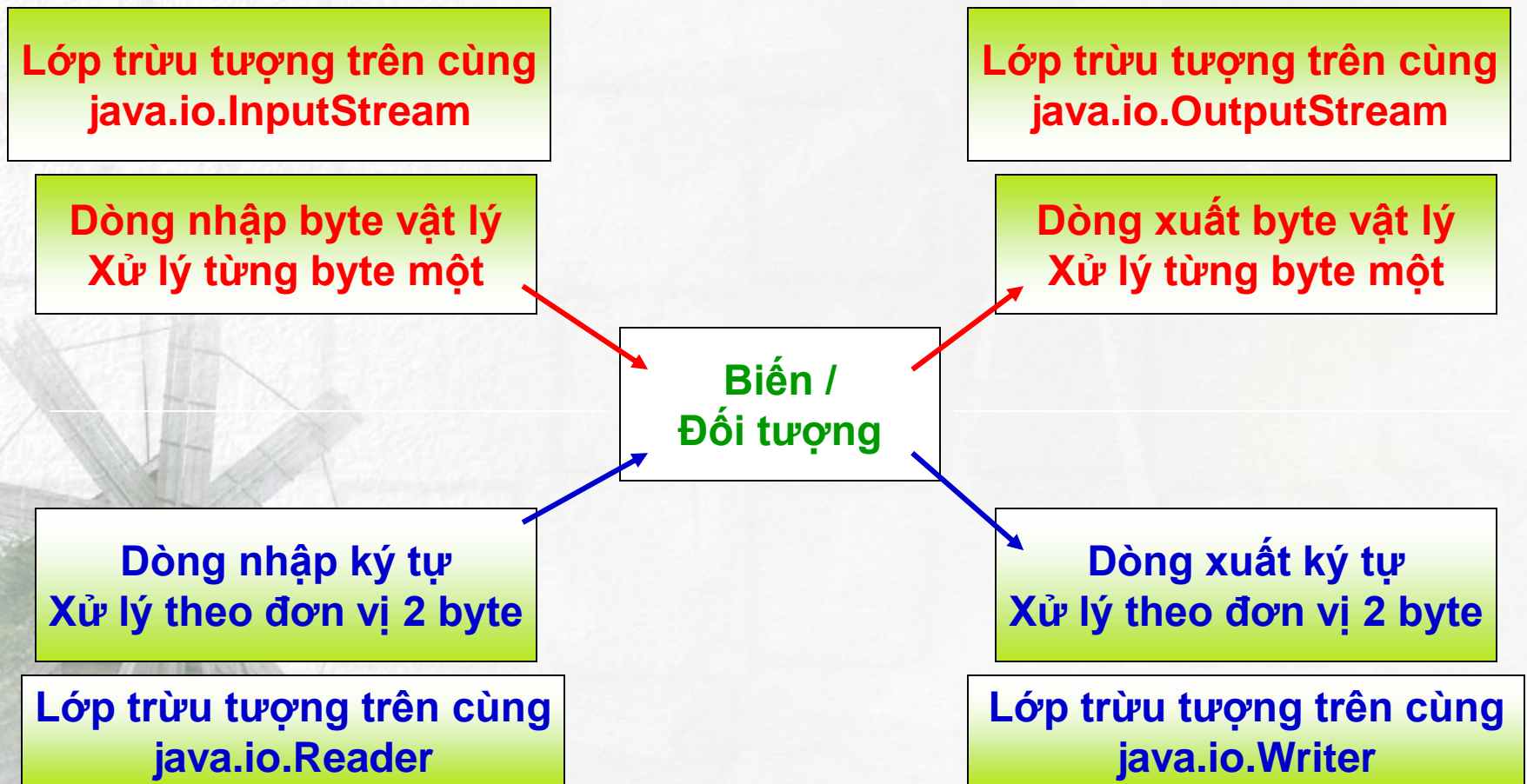
# KHÁI NIỆM LUỒNG (STREAMS)

- Luồng (stream) là một sự biểu diễn trừu tượng việc xuất nhập dữ liệu được kết nối với một số thiết bị vào hay ra



# KHÁI NIỆM LUỒNG (STREAMS)

- Java hiện thực luồng bằng tập hợp các lớp phân cấp trong gói *java.io*.





# KHÁI NIỆM LUỒNG

- **Luồng byte** (hay luồng dựa trên byte) hỗ trợ việc xuất nhập dữ liệu trên byte, thường được dùng khi đọc ghi dữ liệu nhị phân.
- **Luồng ký tự** được thiết kế hỗ trợ việc xuất nhập dữ liệu kiểu ký tự (Unicode). Luồng ký tự hỗ trợ hiệu quả chỉ đối với việc quản lý, xử lý các ký tự.



# LUỒNG BYTE (Byte Streams)

Các luồng byte được định nghĩa dùng hai lớp phân cấp.

- Mức trên cùng là hai lớp trừu tượng *InputStream* và *OutputStream*.
- *InputStream* định nghĩa những đặc điểm chung cho những luồng nhập byte.
- *OutputStream* mô tả cách xử lý của các luồng xuất byte.

# CÂY THỪA KẾ CỦA INPUTSTREAM

java.io Class Hierarchy (Java 2 Platform SE 5.0) - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Back Forward Stop Home Search Favorites Refresh Print Mail News RSS Feeds

Address <C:\Program Files\Java\docs\api\java\io\package-tree.html> Go

- o java.lang.[Object](#)
  - o java.io.[File](#) (implements java.lang.[Comparable](#)<T>, java.io.[Serializable](#))
  - o java.io.[FileDescriptor](#)
  - o java.io.[InputStream](#) (implements java.io.[Closeable](#))
    - o java.io.[ByteArrayInputStream](#)
    - o java.io.[FileInputStream](#)
    - o java.io.[FilterInputStream](#)
      - o java.io.[BufferedInputStream](#)
      - o java.io.[DataInputStream](#) (implements java.io.[DataInput](#))
      - o java.io.[LineNumberInputStream](#)
      - o java.io.[PushbackInputStream](#)
    - o java.io.[ObjectInputStream](#) (implements java.io.[ObjectInput](#), java.io.[ObjectStreamConstants](#))
    - o java.io.[PipedInputStream](#)
    - o java.io.[SequenceInputStream](#)
    - o java.io.[StringBufferInputStream](#)
  - o java.io.[ObjectInputStream.GetField](#)
  - o java.io.[ObjectOutputStream.PutField](#)
  - o java.io.[ObjectStreamClass](#) (implements java.io.[Serializable](#))
  - o java.io.[ObjectStreamField](#) (implements java.lang.[Comparable](#)<T>)
  - o java.io.[OutputStream](#) (implements java.io.[Closeable](#), java.io.[Flushable](#))
    - o java.io.[ByteArrayOutputStream](#)

My Computer



# CÂY THỪA KẾ CỦA OUTPUTSTREAM

java.io Class Hierarchy (Java 2 Platform SE 5.0) - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Back Forward Stop Home Search Favorites Reload Print View Source Help

Address C:\Program Files\Java\docs\api\java\io\package-tree.html

- o java.lang.[Object](#)
  - o java.io.[OutputStream](#) (implements java.io.[Closeable](#), java.io.[Flushable](#))
    - o java.io.[ByteArrayOutputStream](#)
    - o java.io.[FileOutputStream](#)
    - o java.io.[FilterOutputStream](#)
      - o java.io.[BufferedOutputStream](#)
      - o java.io.[DataOutputStream](#) (implements java.io.[DataOutput](#))
      - o java.io.[PrintStream](#) (implements java.lang.[Appendable](#), java.io.[Closeable](#))
    - o java.io.[ObjectOutputStream](#) (implements java.io.[ObjectOutput](#), java.io.[ObjectStreamConstants](#))
    - o java.io.[PipedOutputStream](#)
  - o java.security.[Permission](#) (implements java.security.[Guard](#), java.io.[Serializable](#))
    - o java.security.[BasicPermission](#) (implements java.io.[Serializable](#))
      - o java.io.[SerializablePermission](#)
    - o java.io.[FilePermission](#) (implements java.io.[Serializable](#))
  - o java.io.[RandomAccessFile](#) (implements java.io.[Closeable](#), java.io.[DataInput](#), java.io.[DataOutput](#))
  - o java.io.[Reader](#) (implements java.io.[Closeable](#), java.lang.[Readable](#))
    - o java.io.[BufferedReader](#)
      - o java.io.[LineNumberReader](#)
    - o java.io.[CharArrayReader](#)
    - o java.io.[FilterReader](#)

My Computer

# LUỒNG KÝ TỰ (Character Streams)

- Các luồng ký tự được định nghĩa dùng hai lớp phân cấp.
- Mức trên cùng là hai lớp trừu tượng *Reader* và *Writer*.
- Lớp *Reader* dùng cho việc nhập dữ liệu của luồng.
- Lớp *Writer* dùng cho việc xuất dữ liệu của luồng.
- Những lớp dẫn xuất từ *Reader* và *Writer* thao tác trên các luồng ký tự Unicode.

# CÂY THỪA KẾ CỦA READER & WRITER

- java.io.**Reader** (implements java.io.[Closeable](#), java.lang.[Readable](#))
  - java.io.**BufferedReader**
    - java.io.[LineNumberReader](#)
  - java.io.[CharArrayReader](#)
  - java.io.[FilterReader](#)
    - java.io.[PushbackReader](#)
  - java.io.**InputStreamReader**
    - java.io.[FileReader](#)
  - java.io.[PipedReader](#)
  - java.io.[StringReader](#)
- java.io.**Writer** (implements java.lang.[Appendable](#), java.io.[Closeable](#), java.io.[Flushable](#))
  - java.io.**BufferedWriter**
  - java.io.[CharArrayWriter](#)
  - java.io.[FilterWriter](#)
  - java.io.[OutputStreamWriter](#)
    - java.io.[FileWriter](#)
  - java.io.[PipedWriter](#)
  - java.io.[PrintWriter](#)
  - java.io.[StringWriter](#)





# CÁC LUỒNG ĐỊNH NGHĨA TRƯỚC

- Tất cả các chương trình viết bằng java luôn tự động import gói *java.lang*. Gói này có định nghĩa lớp *System*, nó có ba biến luồng được định nghĩa trước là ***in***, ***out*** và ***err***, chúng là các fields được khai báo *static* trong lớp *System*.
- **System.out**: luồng xuất chuẩn, mặc định là console. *System.out* là một đối tượng kiểu *PrintStream*.
- **System.in**: luồng nhập chuẩn, mặc định là bàn phím. *System.in* là một đối tượng kiểu *InputStream*.
- **System.err**: luồng lỗi chuẩn, mặc định cũng là console. *System.err* cũng là một đối tượng kiểu *PrintStream* giống *System.out*.

## PHẦN 2

# SỬ DỤNG LUỒNG BYTE



# SỬ DỤNG LUỒNG BYTE

- Như chúng ta đã biết hai lớp *InputStream* và *OutputStream* là hai siêu lớp (cha) đối với tất cả những lớp luồng xuất nhập kiểu byte.
- Những phương thức trong hai siêu lớp này ném ra các lỗi kiểu *IOException*.
- Những phương thức định nghĩa trong hai siêu lớp này có thể dùng trong các lớp con của chúng. Vì vậy tập các phương thức này là tập tối thiểu các chức năng nhập xuất mà những luồng nhập xuất kiểu byte có thể sử dụng.



# CÁC PHƯƠNG THỨC CỦA INPUTSTREAM

<code>int available( )</code>	Trả về số lượng bytes có thể đọc được từ luồng nhập
<code>void close( )</code>	Đóng luồng nhập và giải phóng tài nguyên hệ thống gắn với luồng. Không thành công sẽ ném ra một lỗi <code>IOException</code>
<code>void mark(int numBytes)</code>	Đánh dấu ở vị trí hiện tại trong luồng nhập
<code>boolean markSupported( )</code>	Kiểm tra xem luồng nhập có hỗ trợ phương thức <i>mark()</i> và <i>reset()</i> không.
<code>int read( )</code>	Đọc byte tiếp theo từ luồng nhập
<code>int read(byte buffer[ ])</code>	Đọc <code>buffer.length</code> bytes và lưu vào trong vùng nhớ <code>buffer</code> . Kết quả trả về số bytes thật sự đọc được
<code>int read(byte buffer[ ], int offset, int numBytes)</code>	Đọc <code>numBytes</code> bytes bắt đầu từ địa chỉ <code>offset</code> và lưu vào trong vùng nhớ <code>buffer</code> . Kết quả trả về số bytes thật sự đọc được
<code>void reset( )</code>	Nhảy con trỏ đến vị trí được xác định bởi việc gọi hàm <i>mark()</i> lần sau cùng.
<code>long skip(long numBytes)</code>	Nhảy qua <code>numBytes</code> dữ liệu từ luồng nhập

# CÁC PHƯƠNG THỨC CỦA OUTPUTSTREAM

<code>void close( )</code>	Đóng luồng xuất và giải phóng tài nguyên hệ thống gắn với luồng. Không thành công sẽ ném ra một lỗi <code>IOException</code>
<code>void flush( )</code>	Ép dữ liệu từ bộ đệm phải ghi ngay xuống luồng (nếu có)
<code>void write(int b)</code>	Ghi byte dữ liệu chỉ định xuống luồng
<code>void write(byte buffer[ ])</code>	Ghi <code>buffer.length</code> bytes dữ liệu từ mảng chỉ định xuống luồng
<code>void write(byte buffer[ ], int offset, int numBytes)</code>	Ghi <code>numBytes</code> bytes dữ liệu từ vị trí <code>offset</code> của mảng chỉ định <code>buffer</code> xuống luồng

# ĐỌC DỮ LIỆU TỪ Console

- Ví dụ sau đây minh họa cách dùng luồng byte thực hiện việc nhập xuất Console. Chương trình minh họa việc đọc một mảng bytes từ System.in

```
import java.io.*;
class ReadBytes
{
    public static void main(String args[]) throws IOException
    {
        byte data[] = new byte[100];
        System.out.print("Enter some characters.");
        System.in.read(data);
        System.out.print("You entered: ");
        for(int i=0; i < data.length; i++)
            System.out.print((char) data[i]);
    }
}
```



# XUẤT DỮ LIỆU RA Console

Chúng ta đã khá quen thuộc với phương thức `print()` và `println()`, dùng để xuất dữ liệu ra Console. Bên cạnh đó chúng ta cũng có thể dùng phương thức `write()`. **Ví dụ:** minh họa sử dụng phương thức `System.out.write()` để xuất ký tự 'X' ra Console

```
import java.io.*;
class WriteDemo
{
    public static void main(String args[])
    {
        int b;
        b = 'X';
        System.out.write(b);
        System.out.write('\n');
    }
}
```

# ĐỌC VÀ GHI FILE DÙNG LUỒNG BYTE

- Tạo một luồng Byte gắn với file chỉ định dùng `FileInputStream` và `FileOutputStream`.
- Để mở một file, đơn giản chỉ cần tạo một đối tượng của những lớp này, tên file cần mở là thông số trong constructor.
- Khi file mở, việc đọc và ghi dữ liệu trên file được thực hiện một cách bình thường thông qua các phương thức cung cấp trong luồng.



# ĐỌC VÀ GHI FILE DÙNG LUỒNG BYTE

## Đọc dữ liệu từ file:

- **Mở một file để đọc dữ liệu**

*FileInputStream(String fileName) throws*

*FileNotFoundException*

*Nếu file không tồn tại: thì ném ra FileNotFoundException*

- **Đọc dữ liệu:** dùng phương thức *read()*

*int read( ) throws IOException:* đọc từng byte từ file và trả về giá trị của byte đọc được. Trả về -1 khi hết file, và ném ra *IOException* khi có lỗi đọc.

- **Đóng file:** dùng phương thức *close()*

*void close( ) throws IOException:* sau khi làm việc xong cần đóng file để giải phóng tài nguyên hệ thống đã cấp phát cho file.



# ĐỌC VÀ GHI FILE DÙNG LUỒNG BYTE

- Ví dụ Đọc dữ liệu từ file, hiển thị nội dung của một file tên test.txt lưu tại D:\test.txt

```
import java.io.*;
class ShowFile
{   public static void main(String args[]) throws IOException
    {   int i; FileInputStream fin;
        try { fin = new FileInputStream("D:\\test.txt"); }
        catch(FileNotFoundException exc){
            System.out.println("File Not Found"); return;
        }
        catch(ArrayIndexOutOfBoundsException exc){
            System.out.println("Usage: ShowFile File"); return;
        }
        // read bytes until EOF is encountered
        do
        {
            i = fin.read();
            if(i != -1) System.out.print((char) i);
        } while(i != -1);
        fin.close();
    }
}
```

# ĐỌC VÀ GHI FILE DÙNG LUỒNG BYTE

## Ghi dữ liệu xuống file:

- **Mở một file để ghi dữ liệu**

*FileOutputStream(String fileName) throws*

*FileNotFoundException*

*Nếu file không tạo được: thì ném ra FileNotFoundException*

- **Ghi dữ liệu xuống:** dùng phương thức *write()*

*void write(int byteval) throws IOException:* ghi một byte xác định bởi tham số *byteval* xuống file, và ném ra *IOException* khi có lỗi ghi.

- **Đóng file:** dùng phương thức *close()*

*void close( ) throws IOException:* sau khi làm việc xong cần đóng file để giải phóng tài nguyên hệ thống đã cấp phát cho file.



# ĐỌC VÀ GHI FILE DÙNG LUỒNG BYTE

- Ví dụ Ghi dữ liệu xuống file, copy nội dung một file text đến một file text khác.

```
import java.io.*;
class CopyFile
{   public static void main(String args[])throws IOException
    {       int i; FileInputStream fin; FileOutputStream fout;
        try { // open input file
            try { fin = new FileInputStream("D:\\source.txt");}
            catch(FileNotFoundException exc) { System.out.println("Input File Not Found"); return; }
            // open output file
            try { fout = new FileOutputStream("D:\\dest.txt"); }
            catch(FileNotFoundException exc) { System.out.println("Error Opening Output File"); return; }
        } catch(ArrayIndexOutOfBoundsException exc)
        { System.out.println("Usage: CopyFile From To"); return; }
        try { // Copy File
            do { i = fin.read();
                if(i != -1) fout.write(i);
            } while(i != -1);
        } catch(IOException exc)
        { System.out.println("File Error"); }
        fin.close(); fout.close();
    }
}
```

} //Kết quả, chương trình sẽ copy nội dung của file D:\source.txt và ghi vào một file mới D:\dest.txt.



# ĐỌC VÀ GHI DỮ LIỆU NHỊ PHÂN

- Phần trên chúng ta đã đọc và ghi các bytes dữ liệu là các ký tự mã ASCII. Để đọc và ghi những giá trị nhị phân của các kiểu dữ liệu trong java, chúng ta sử dụng:
  - ***DataInputStream***
  - ***DataOutputStream***.

# ĐỌC VÀ GHI DỮ LIỆU NHỊ PHÂN

- Phần trên chúng ta đã đọc và ghi các bytes dữ liệu là các ký tự. Để đọc và ghi những giá trị nhị phân của các kiểu dữ liệu trong java, chúng ta sử dụng *DataInputStream* và *DataOutputStream*.
- DataOutputStream**: hiện thực interface *DataOutput*. Interface *DataOutput* có các phương thức cho phép ghi tất cả những kiểu dữ liệu cơ sở của java đến luồng (theo định dạng nhị phân).

<i>void writeBoolean(boolean val)</i>	Ghi xuống luồng một giá trị boolean được xác định bởi val.
<i>void writeByte (int val)</i>	Ghi xuống luồng một byte được xác định bởi val.
<i>void writeChar (int val)</i>	Ghi xuống luồng một Char được xác định bởi val.
<i>void writeDouble(double val)</i>	Ghi xuống luồng một giá trị Double được xác định bởi val.
<i>void writeFloat (float val)</i>	Ghi xuống luồng một giá trị float được xác định bởi val.
<i>void writeInt (int val)</i>	Ghi xuống luồng một giá trị int được xác định bởi val.
<i>void writeLong (long val)</i>	Ghi xuống luồng một giá trị long được xác định bởi val.
<i>void writeShort (int val)</i>	Ghi xuống luồng một giá trị short được xác định bởi val.
<p>Constructor: <i>DataOutputStream(OutputStream outputStream)</i> <i>OutputStream</i>: là luồng xuất dữ liệu. Để ghi dữ liệu ra file thì đối tượng <i>outputStream</i> có thể là <i>FileOutputStream</i>.</p>	



# ĐỌC VÀ GHI DỮ LIỆU NHỊ PHÂN

- **DataInputStream**: hiện thực interface *DataInput*. Interface *DataInput* có các phương thức cho phép đọc tất cả những kiểu dữ liệu cơ sở của java (theo định dạng nhị phân).

<i>boolean readBoolean( )</i>	Đọc một giá trị boolean
<i>Byte readByte( )</i>	Đọc một byte
<i>char readChar( )</i>	Đọc một Char
<i>double readDouble( )</i>	Đọc một giá trị Double
<i>float readFloat( )</i>	Đọc một giá trị float
<i>int readInt( )</i>	Đọc một giá trị int
<i>long readLong( )</i>	Đọc một giá trị long
<i>short readShort( )</i>	Đọc một giá trị short
Contructor: <i>DataStream(InputStream inputStream)</i> <i>InputStream</i> : là luồng nhập dữ liệu. Để đọc dữ liệu từ file thì đối tượng <i>InputStream</i> có thể là <i>FileInputStream</i> .	



# ĐỌC VÀ GHI DỮ LIỆU NHỊ PHÂN

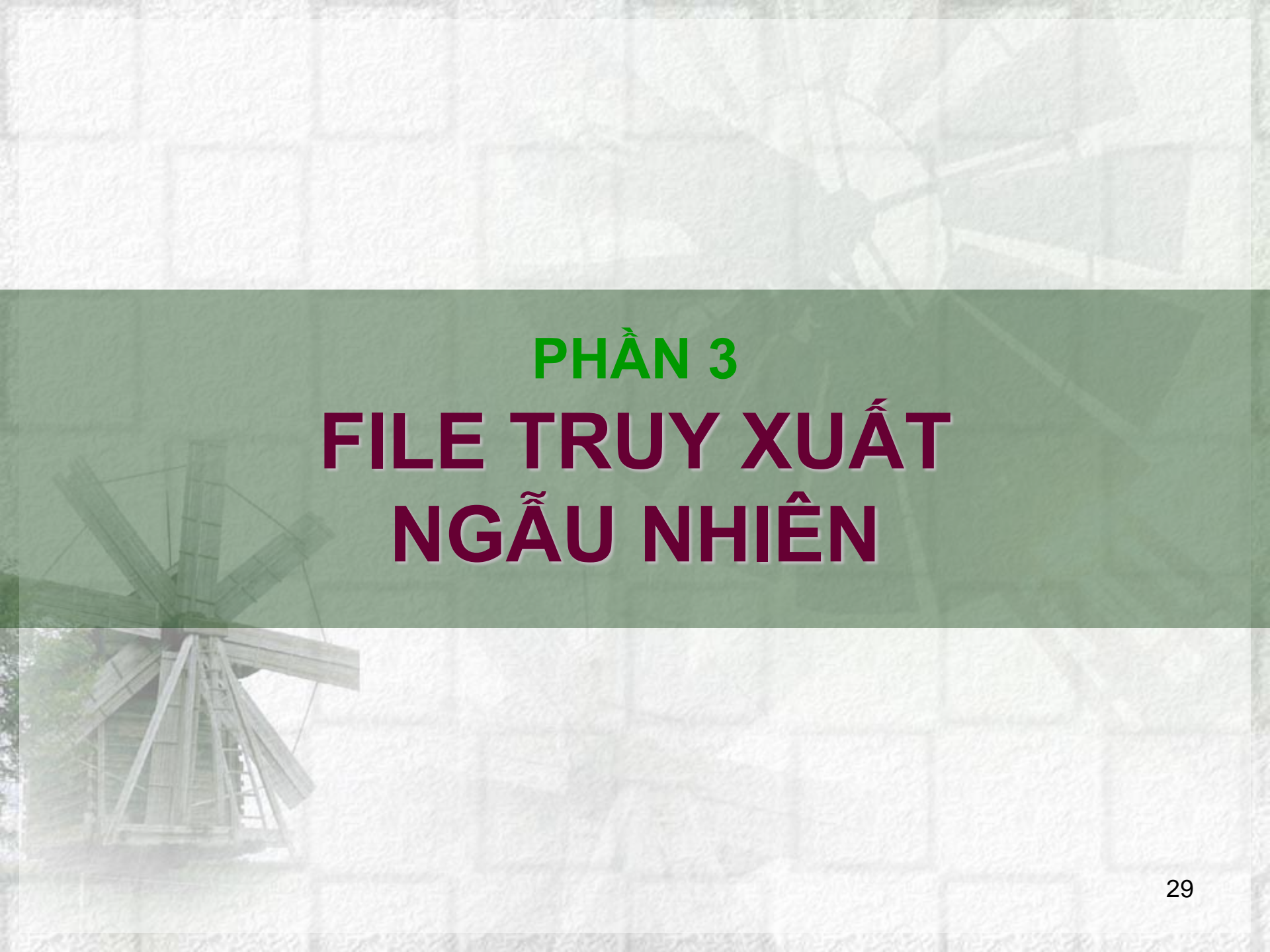
- **Ví dụ:** dùng `DataOutputStream` và `DataInputStream` để ghi và đọc những kiểu dữ liệu khác nhau trên file.

```
import java.io.*;
class RWData
{
    public static void main(String args[]) throws IOException
    {
        DataOutputStream dataOut; DataInputStream dataIn; int i = 10; double d = 1023.56; boolean b = true;
        try {dataOut = new DataOutputStream(new FileOutputStream("D:\\testdata"));}
        catch(IOException exc) { System.out.println("Cannot open file."); return;}
        try {
            System.out.println("Writing " + i);
            dataOut.writeInt(i);
            System.out.println("Writing " + d);
            dataOut.writeDouble(d);
            System.out.println("Writing " + b);
            dataOut.writeBoolean(b);
            System.out.println("Writing " + 12.2 * 7.4);
            dataOut.writeDouble(12.2 * 7.4);
        }
        catch(IOException exc)
        {
            System.out.println("Write error.");
        }
    } //XEM TIẾP Ở SLIDE TIẾP THEO
}
```

# ĐỌC VÀ GHI DỮ LIỆU NHỊ PHÂN

```
dataOut.close(); System.out.println();  
// Now, read them back.  
try {    dataIn = new DataInputStream(  
        new FileInputStream("D:\\testdata"));  
}  
catch(IOException exc) {  
    System.out.println("Cannot open file."); return;  
}  
try {    i = dataIn.readInt();  
        System.out.println("Reading " + i);  
        d = dataIn.readDouble();  
        System.out.println("Reading " + d);  
        b = dataIn.readBoolean();  
        System.out.println("Reading " + b);  
        d = dataIn.readDouble();  
        System.out.println("Reading " + d);  
}  
catch(IOException exc) { System.out.println("Read error."); }  
dataIn.close();  
}  
}
```



The background of the slide features a faded, light green image of a traditional wooden windmill. The windmill has a tall, conical body and several long, thin sails extending from the top. It is positioned on the left side of the slide, with its base partially visible. The overall aesthetic is clean and academic.

# **PHẦN 3**

# **FILE TRUY XUẤT**

# **NGẪU NHIÊN**



# FILE TRUY XUẤT NGẪU NHIÊN

- Bên cạnh việc xử lý xuất nhập trên file theo kiểu tuần tự thông qua các luồng, java cũng hỗ trợ truy cập ngẫu nhiên nội dung của một file nào đó dùng *RandomAccessFile*.
- *RandomAccessFile* không dẫn xuất từ *InputStream* hay *OutputStream* mà nó hiện thực các interface *DataInput*, *DataOutput* (có định nghĩa các phương thức I/O cơ bản).
- *RandomAccessFile* hỗ trợ vấn đề định vị con trỏ file bên trong một file dùng phương thức *seek(long newPos)*.

# FILE TRUY XUẤT NGẪU NHIÊN

- **Ví dụ:** minh họa việc truy cập ngẫu nhiên trên file. Chương trình ghi 6 số kiểu double xuống file, rồi đọc lên theo thứ tự ngẫu nhiên.

```
import java.io.*;

class RandomAccessDemo
{
    public static void main(String args[]) throws IOException
    {
        double data[] = {19.4, 10.1, 123.54, 33.0, 87.9, 74.25}; double d; RandomAccessFile raf;
        try { raf = new RandomAccessFile("D:\\random.dat", "rw"); }
        catch (FileNotFoundException exc) { System.out.println("Cannot open file."); return ;}
        // Write values to the file.
        for(int i=0; i < data.length; i++)
        {
            try {
                raf.writeDouble(data[i]);
            }
            catch (IOException exc)
            {
                System.out.println("Error writing to file.");
                return ;
            }
        }
    }
}
```

//XEM TIẾP Ở SLIDE TIẾP THEO



# FILE TRUY XUẤT NGẪU NHIÊN

```
try { // Now, read back specific values
    raf.seek(0); // seek to first double
    d = raf.readDouble();
    System.out.println("First value is " + d);
    raf.seek(8); // seek to second double
    d = raf.readDouble();
    System.out.println("Second value is " + d);
    raf.seek(8 * 3); // seek to fourth double
    d = raf.readDouble();
    System.out.println("Fourth value is " + d);
    System.out.println();
    // Now, read every other value.
    System.out.println("Here is every other value: ");
    for(int i=0; i < data.length; i+=2)
    {
        raf.seek(8 * i); // seek to ith double
        d = raf.readDouble();
        System.out.print(d + " ");
    }
    System.out.println("\n");
} catch(IOException exc) { System.out.println("Error seeking or reading."); }
raf.close();
}
```

```
}
```



# PHẦN 4

## SỬ DỤNG

### LUỒNG KÝ TỰ



# LỚP READER

Reader	
<i>abstract void close()</i>	Đóng luồng
<i>void mark(int numChars)</i>	Đánh dấu vị trí hiện tại trên luồng
<i>boolean markSupported()</i>	Kiểm tra xem luồng có hỗ trợ thao tác đánh dấu <i>mark()</i> không?
<i>int read()</i>	Đọc một ký tự
<i>int read(char buffer[ ])</i>	Đọc <i>buffer.length</i> ký tự cho vào <i>buffer</i>
<i>abstract int read(char buffer[ ], int offset, int numChars)</i>	Đọc <i>numChars</i> ký tự cho vào vùng đệm <i>buffer</i> tại vị trí <i>buffer[offset]</i>
<i>boolean ready()</i>	Kiểm tra xem luồng có đọc được không?
<i>void reset()</i>	Dời con trỏ nhập đến vị trí đánh dấu trước đó
<i>long skip(long numChars)</i>	Bỏ qua <i>numChars</i> của luồng nhập

# LỚP WRITE

Writer	
<i>abstract void close()</i>	Đóng luồng xuất. Có lỗi ném ra IOException
<i>abstract void flush()</i>	Dọn dẹp luồng (buffer xuất)
<i>void write(int ch)</i>	Ghi một ký tự
<i>void write(byte buffer[ ])</i>	Ghi một mảng các ký tự
<i>abstract void write(char buffer[ ], int offset, int numChars)</i>	Ghi một phần của mảng ký tự
<i>void write(String str)</i>	Ghi một chuỗi
<i>void write(String str, int offset, int numChars)</i>	Ghi một phần của một chuỗi ký tự



# NHẬP CONSOLE DÙNG LUỒNG KÝ TỰ

- Muốn nhập dữ liệu từ Console là lớp *BufferedReader* thì chúng ta không thể xây dựng một lớp *BufferedReader* trực tiếp từ *System.in*. Thay vào đó chúng ta phải chuyển nó thành một luồng ký tự bằng cách dùng *InputStreamReader* chuyển bytes thành ký tự.
- Để có được một đối tượng *InputStreamReader* gắn với *System.in* ta dùng constructor của *InputStreamReader*.
  - ***InputStreamReader(InputStream inputStream)***
- Tiếp theo dùng đối tượng *InputStreamReader* đã tạo ra để tạo ra một *BufferedReader* dùng constructor *BufferedReader*.
  - ***BufferedReader(Reader inputReader)***
- Ví dụ: Tạo một *BufferedReader* gắn với Keyboard
  - ***BufferedReader br = new BufferedReader(new InputStreamReader(System.in));***
- Sau khi thực hiện câu lệnh trên, *br* là một luồng ký tự gắn với Console thông qua *System.in*.

# NHẬP CONSOLE DÙNG LUỒNG KÝ TỰ

- **Ví dụ:** Dùng `BufferedReader` đọc từng ký tự từ Console. Việc đọc kết thúc khi gặp dấu chấm (dấu chấm để kết thúc chương trình).

```
import java.io.*;
class ReadChars
{
    public static void main(String args[]) throws IOException
    {
        char c;
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        System.out.println("Nhap chuoi ky tu, gioi han dau cham.");
        // read characters
        do
        {
            c = (char) br.read();
            System.out.println(c);
        } while(c != '.');
    }
}
```



# NHẬP CONSOLE DÙNG LUỒNG KÝ TỰ

- **Ví dụ:** Dùng `BufferedReader` đọc chuỗi ký tự từ Console. Chương trình kết thúc khi gặp chuỗi đọc là chuỗi "stop"

```
import java.io.*;
class ReadLines
{
    public static void main(String args[]) throws IOException
    {
        // create a BufferedReader using System.in
        BufferedReader br = new BufferedReader(new
        InputStreamReader(System.in));
        String str;
        System.out.println("Nhap chuoi.");
        System.out.println("Nhap 'stop' ket thuc chuong trinh.");
        do
        {
            str = br.readLine();
            System.out.println(str);
        } while(!str.equals("stop"));
    }
}
```

# XUẤT CONSOLE DÙNG LUỒNG KÝ TỰ

- Trong ngôn ngữ java, bên cạnh việc dùng *System.out* để xuất dữ liệu ra Console (thường dùng để debug chương trình), chúng ta có thể dùng luồng *PrintWriter* đối với các chương trình “chuyên nghiệp”.
- *PrintWriter* là một trong những lớp luồng ký tự. Việc dùng các lớp luồng ký tự để xuất dữ liệu ra Console thường được “ưa chuộng” hơn.
- Để xuất dữ liệu ra Console dùng *PrintWriter* cần thiết phải chỉ định *System.out* cho luồng xuất.
- **Ví dụ**, tạo đối tượng *PrintWriter* để xuất dữ liệu ra Console:  
*PrintWriter pw = new PrintWriter(System.out, true);*



# XUẤT CONSOLE DÙNG LUỒNG KÝ TỰ

- **Ví dụ:** minh họa dùng *PrintWriter* để xuất dữ liệu ra Console

```
import java.io.*;

public class PrintWriterDemo
{
    public static void main(String args[])
    {
        PrintWriter pw = new PrintWriter(System.out, true);
        int i = 10;
        double d = 123.67;
        double r = i+d
        pw.println("Using a PrintWriter.");
        pw.println(i);
        pw.println(d);
        pw.println(i + " + " + d + " = " + r);
    }
}
```

# ĐỌC GHI FILE DÙNG LUỒNG KÝ TỰ

- Thông thường để đọc/ghi file người ta thường dùng luồng byte, nhưng đối với luồng ký tự chúng ta cũng có thể thực hiện được. Ưu điểm của việc dùng luồng ký tự là chúng thao tác trực tiếp trên các ký tự Unicode. Vì vậy luồng ký tự là chọn lựa tốt nhất khi cần lưu những văn bản Unicode.
- Hai lớp luồng thường dùng cho việc đọc/ghi dữ liệu ký tự xuống file là *FileReader* và *FileWriter*.



# ĐỌC GHI FILE DÙNG LUỒNG KÝ TỰ

**Ví dụ:** Đọc những dòng văn bản nhập từ bàn phím và ghi chúng xuống file tên là "test.txt". Việc đọc và ghi kết thúc khi người dùng nhập vào chuỗi "stop".

```
import java.io.*;
class KtoD
{
    public static void main(String args[]) throws IOException
    {
        String str;
        FileWriter fw;
        BufferedReader br = new BufferedReader(new
            InputStreamReader(System.in));

        try{
            fw = new FileWriter("D:\\test.txt");
        }
        catch(IOException exc){
            System.out.println("Khong the mo file.");
            return;
        }

        System.out.println("Nhap ('stop' de ket thuc chuong trinh).");

        do
        {
            System.out.print(": ");
            str = br.readLine();
            if(str.compareTo("stop") == 0) break;
            str = str + "\r\n";
            fw.write(str);
        } while(str.compareTo("stop") != 0);

        fw.close();
    }
}
```

# ĐỌC GHI FILE DÒNG LUỒNG KÝ TỰ

- **Ví dụ:** đọc và hiển thị nội dung của file “test.txt” lên màn hình.

```
import java.io.*;
class DtoS
{
    public static void main(String args[]) throws Exception
    {
        FileReader fr = new FileReader("D:\\test.txt");
        BufferedReader br = new BufferedReader(fr);
        String s;
        while((s = br.readLine()) != null)
        {
            System.out.println(s);
        }
        fr.close();
    }
}
```



# PHẦN 5

# LỚP FILE





# LỚP 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, ...)

java.lang.Object

+++**java.io.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");*

# MỘT SỐ PHƯƠNG THỨC THỨC LỚP FILE

Một số phương thức thường gặp của lớp File (chi tiết về các phương thức đọc thêm trong tài liệu J2SE API Specification)

<i>public <a href="#">String</a> getName()</i>	Lấy tên của đối tượng File
<i>public <a href="#">String</a> getPath()</i>	Lấy đường dẫn của tập tin
<i>public boolean isDirectory()</i>	Kiểm tra xem tập tin có phải là thư mục không?
<i>public boolean isFile()</i>	Kiểm tra xem tập tin có phải là một file không?
...	
<i>public <a href="#">String</a>[] list()</i>	Lấy danh sách tên các tập tin và thư mục con của đối tượng File đang xét và trả về trong một mảng.



# VÍ DỤ VỀ LỚP FILE

```
import java.awt.*;  
import java.io.*;  
public class FileDemo  
{  
    public static void main(String args[])  
    {  
        Frame fr = new Frame ("File Demo");  
        fr.setBounds(10, 10, 300, 200);  
        fr.setLayout(new BorderLayout());  
        Panel p = new Panel(new GridLayout(1,2));  
        List list_C = new List();  
        list_C.add("C:\\");  
        File driver_C = new File ("C:\\");  
        String[] dirs_C = driver_C.list();  
        for (int i=0;i<dirs_C.length;i++)  
        {  
            File f = new File ("C:\\\" + dirs_C[i]);  
            if (f.isDirectory())  
                list_C.add("<DIR>" + dirs_C[i]);  
            else  
                list_C.add(" " + dirs_C[i]);  
        }  
    }  
}
```

//XEM TIẾP Ở SLIDE TIẾP THEO

# VÍ DỤ VỀ LỚP FILE

```
List list_D = new List();
list_D.add("D:\\");
File driver_D = new File ("D:\\");
String[] dirs_D = driver_D.list();
for (int i=0;i<dirs_D.length;i++)
{
    File f = new File ("D:\\\" + dirs_D[i]);
    if (f.isDirectory())
        list_D.add("<DIR>" + dirs_D[i]);
    else
        list_D.add(" " + dirs_D[i]);
}
p.add(list_C);
p.add(list_D);
fr.add(p, BorderLayout.CENTER);
fr.setVisible(true);
}
}
```



# VÍ DỤ VỀ LỚP FILE

- Kết quả thực thi chương trình:



# HẾT BÀI 8

