Hackademics Hanoi
Hack your passion, Code your life!

CHƯƠNG 4

LUỒNG (STREAMS)

NỘI DUNG

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

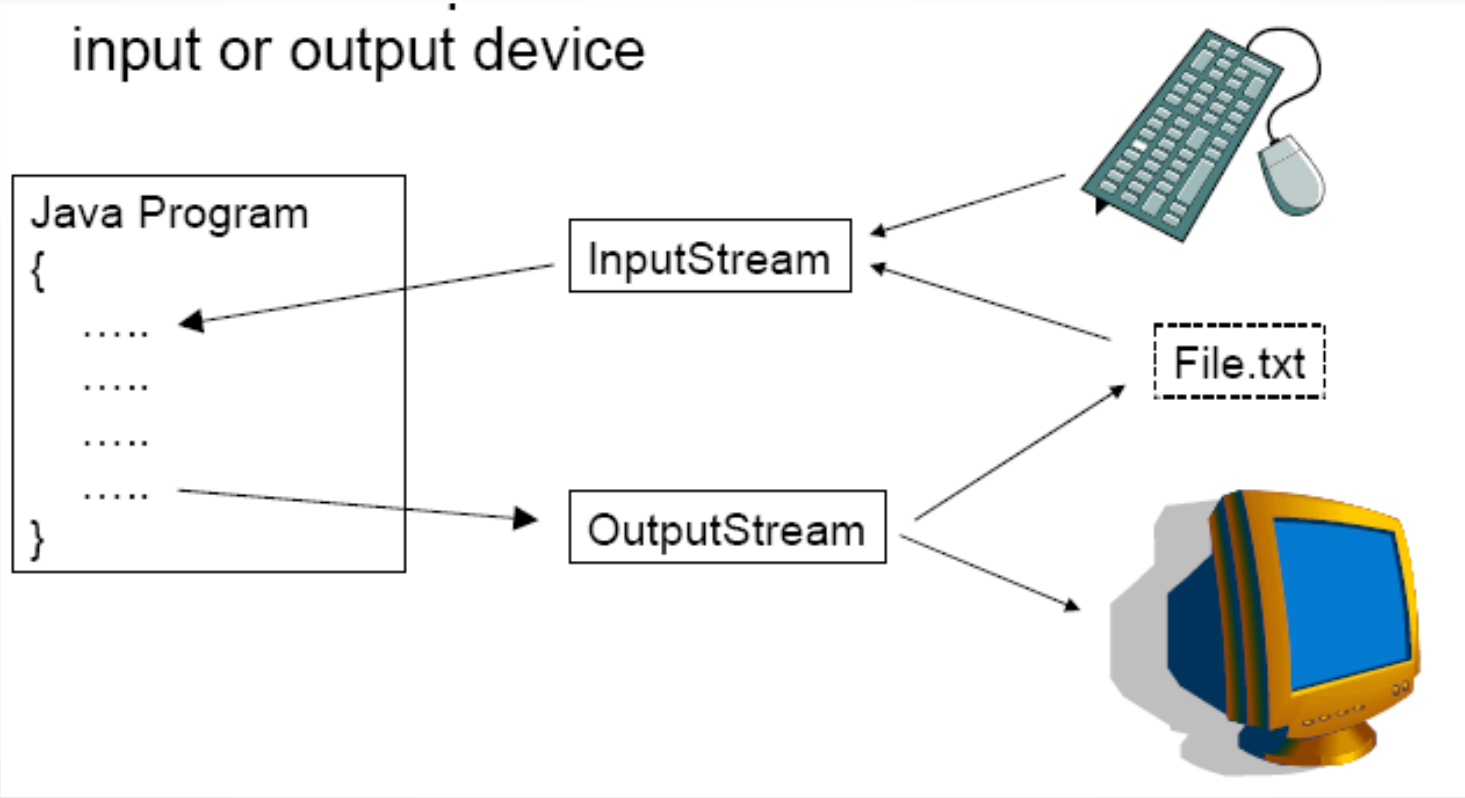
- Thư viện các lớp về luồng trong java: luồng byte, luồng ký tự.
- Xuất nhập console dùng luồng byte, luồng ký tự.
- Xuất nhập files dùng luồng ký tự và luồng byte.
- Vấn đề xử lý files truy cập ngẫu nhiên dùng lớp randomaccessfile.
- Xử lý file và thư mục dùng lớp file.

Hackademics Hanoi
Hack your passion, Code your life!

PHẦN 1
KHÁI NIỆM LUỒNG

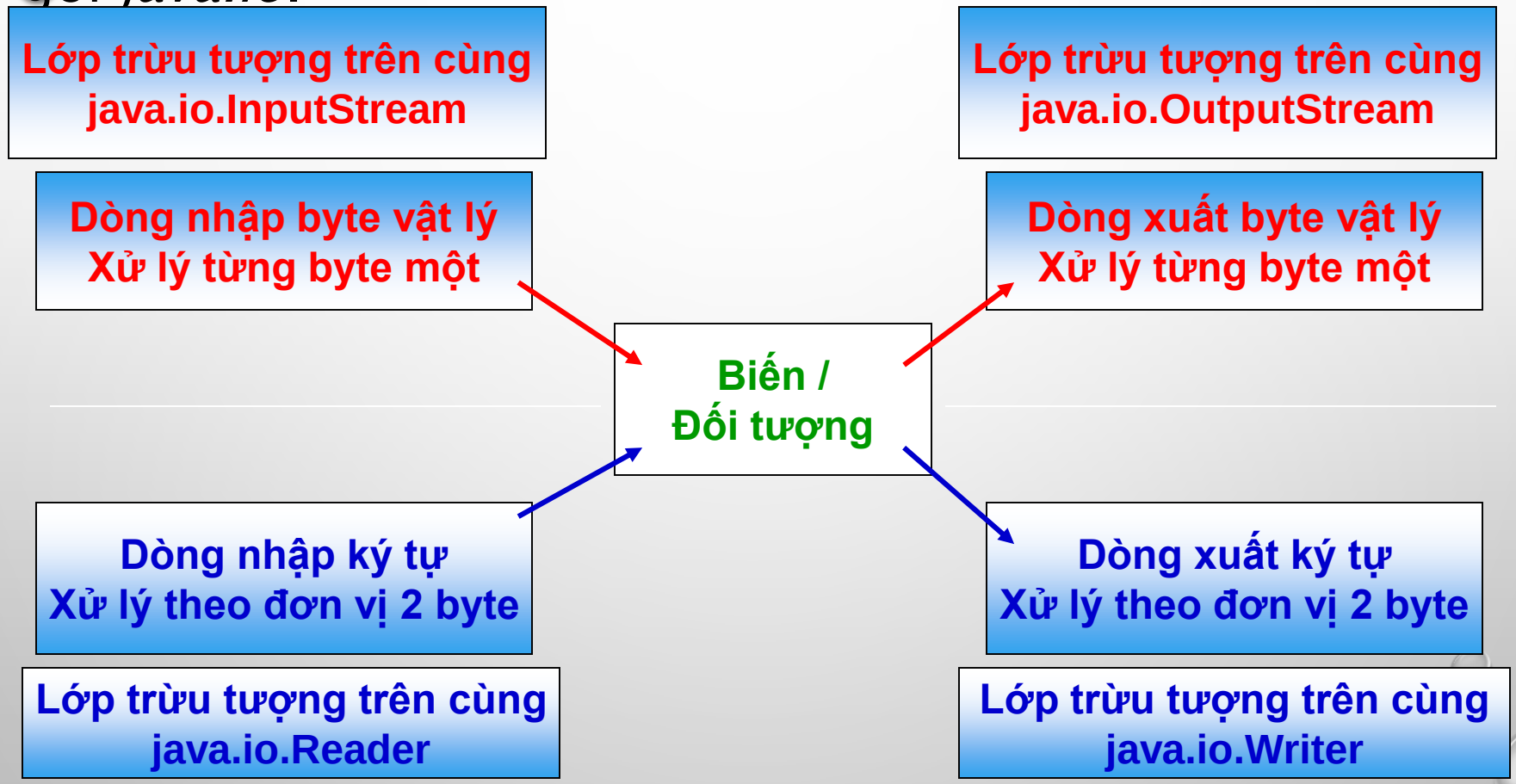
KHÁI NIỆM LUỒNG (STREAMS)

- 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.lo*.



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

CÂY THỪA KẾ CỦA OUTPUTSTREAM

- java.io.**OutputStream** (implements java.io.Closeable, java.io.Flushable)
  - java.io.**ByteArrayOutputStream**
  - java.io.**FileOutputStream**
  - java.io.**FilterOutputStream**
    - java.io.**BufferedOutputStream**
    - java.io.**DataOutputStream** (implements java.io.DataOutput)
    - java.io.**PrintStream** (implements java.lang.Appendable, java.io.Closeable)
  - java.io.**ObjectOutputStream** (implements java.io.ObjectOutput, java.io.ObjectStreamConstants)
  - java.io.**PipedOutputStream**

https://docs.oracle.com/javase/7/docs/api/java/io/package-tree.html

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

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

CÁC PHƯƠNG THỨC CỦA OUTPUTSTREAM

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

- **Dataoutputstream:** hiện thực interface dataouput. 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).

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

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).

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

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();
  }
}
```

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

LỚP WRITE

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

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.

VÍ Dụ VỀ LỚP FILE

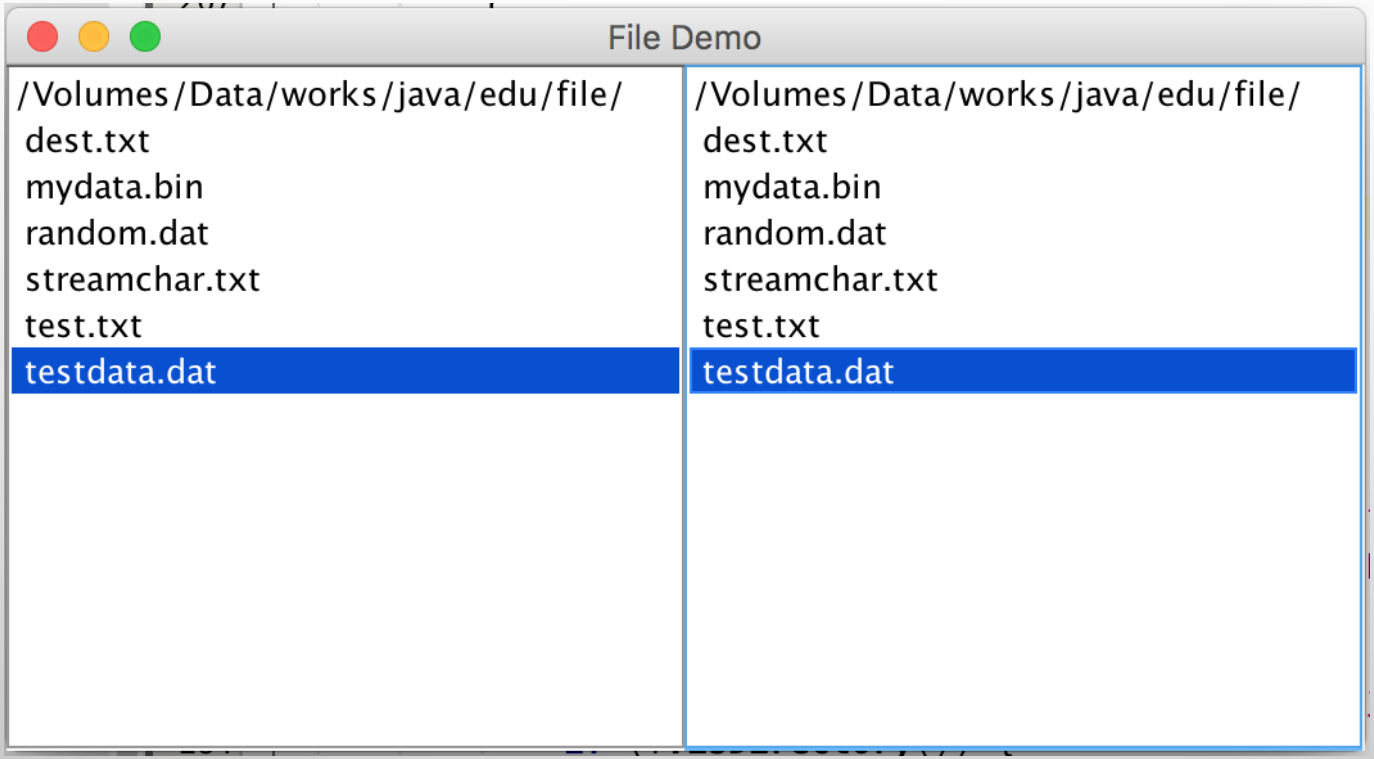
```
import java.awt.*;
import java.io.*;
public class FileDemo
{
    public static void main(String args[])
    {
        Frame fr = new Frame ("File Demo");
        fr.setBounds(10, 10, 300, 200);
        fr.setLayout(new BorderLayout());
        Panel p = new Panel(new GridLayout(1,2));
        List 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
CHƯƠNG 4