Nội dung môn học

Chương 1: Giới thiệu về NNLT Java

Chương 2: Hướng đối tượng trong Java

Chương 3: Các lớp tiện ích trong Java

Chương 4: Quản lý Exception

Chương 5: Nhập xuất dữ liệu (I/O Streams)

Chương 6: Xử lý đa luồng

Chương 7: Kết nối và thao tác CSDL với JDBC

Chương 8: Lập trình GUI với AWT & Swing

NHẬP XUẤT TRONG JAVA

Nội dung

1. Khái niệm luồng I/O
2. Các loại luồng I/O & ứng dụng (I/O Streams)
3. Tập tin & Thư mục
4. Java Socket I/O

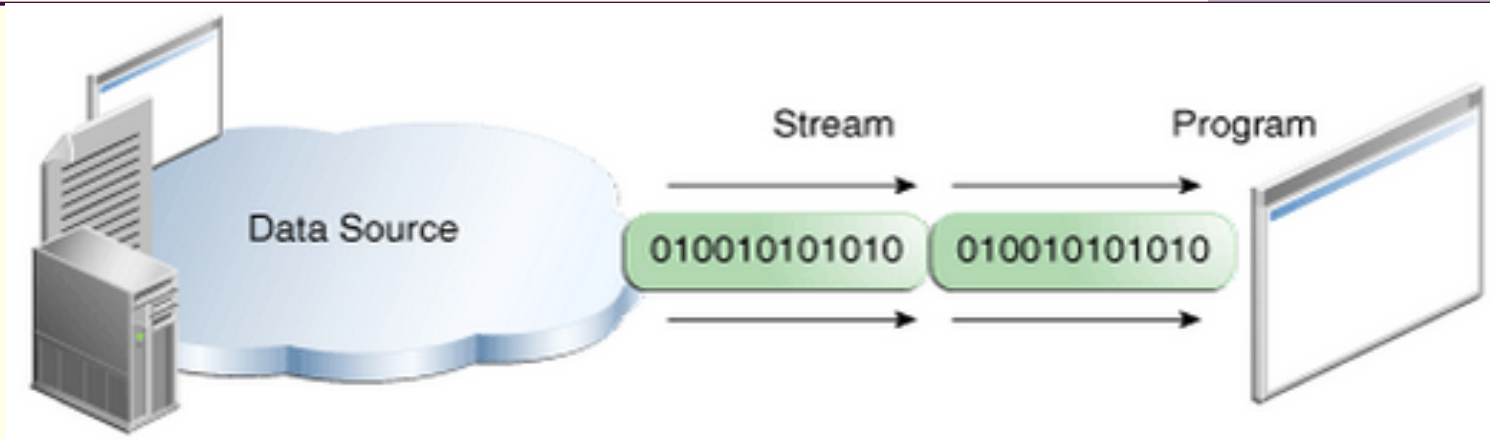
Khái niệm luồng (I/O Stream)

- ***Luồng*:** là nơi có thể "sản xuất" và "tiêu thụ" thông tin. Luồng thường được hệ thống xuất nhập trong java gắn kết với một thiết bị vật lý.
- Tất cả những hoạt động nhập/xuất dữ liệu (nhập dữ liệu từ bàn phím, lấy dữ liệu từ mạng về, ghi dữ liệu ra đĩa, xuất dữ liệu ra màn hình, máy in, …) đều được quy về một khái niệm gọi là luồng (stream)

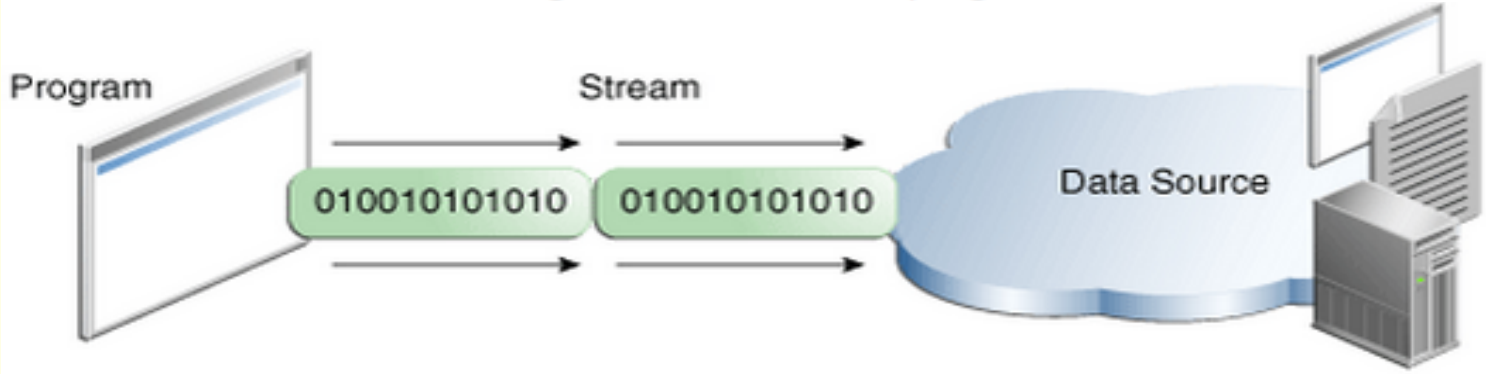
Khái niệm luồng (I/O Stream)

- Có thể hiểu ***luồng*** là một lộ trình dữ liệu được truyền trong chương trình.
- Chức năng của ***luồng***: vận chuyển dữ liệu từ nơi này qua nơi khác. Các chương trình muốn chuyển dữ liệu cho nhau phải tạo các ***luồng*** để gửi và nhận dữ liệu.
- Tất cả các task liên quan đến các lớp nhập xuất trong java nếu có lỗi sẽ quăng ra một IOException.

Khái niệm luồng (I/O Stream)



Reading information into a program.



Writing information from a program.

Khái niệm luồng (I/O Stream)

- Luồng là dãy thứ tự các thành phần dữ liệu.
- Một luồng I/O có một nguồn Input và một đích Output. Có nhiều kiểu nguồn và đích khác nhau như: tập tin, thiết bị, chương trình, bộ nhớ, socket ….
- Kiểu dữ liệu: byte, primitive data types, localized character, objects.

Ref: https://docs.oracle.com/javase/tutorial/essential/io/streams.html

Các loại luồng

- **Luồng byte (Byte Streams)**: I/O dữ liệu nhị phân
- **Luồng ký tự (Character Streams)**: I/O dữ liệu ký tự
- **Luồng đệm (Buffered Streams)**: tối ưu I/O, giảm số lần gọi đến các Native API.
- **Luồng dữ liệu (Data Streams)**: I/O String values & các kiểu dữ liệu cơ sở.
- **Luồng đối tượng (Object Streams)**: I/O binary của các Objects.

Luồng Byte (Byte Streams)

- Input/Output cho các bytes 8-bit.
- Tất cả luồng byte dẫn xuất từ ***InputStream*** và ***OutputStream***.
- I/O ở mức thấp
- ***Ví dụ***: I/O tập tin dùng luồng byte: ***FileInputStream***, ***FileOutputStreams***.

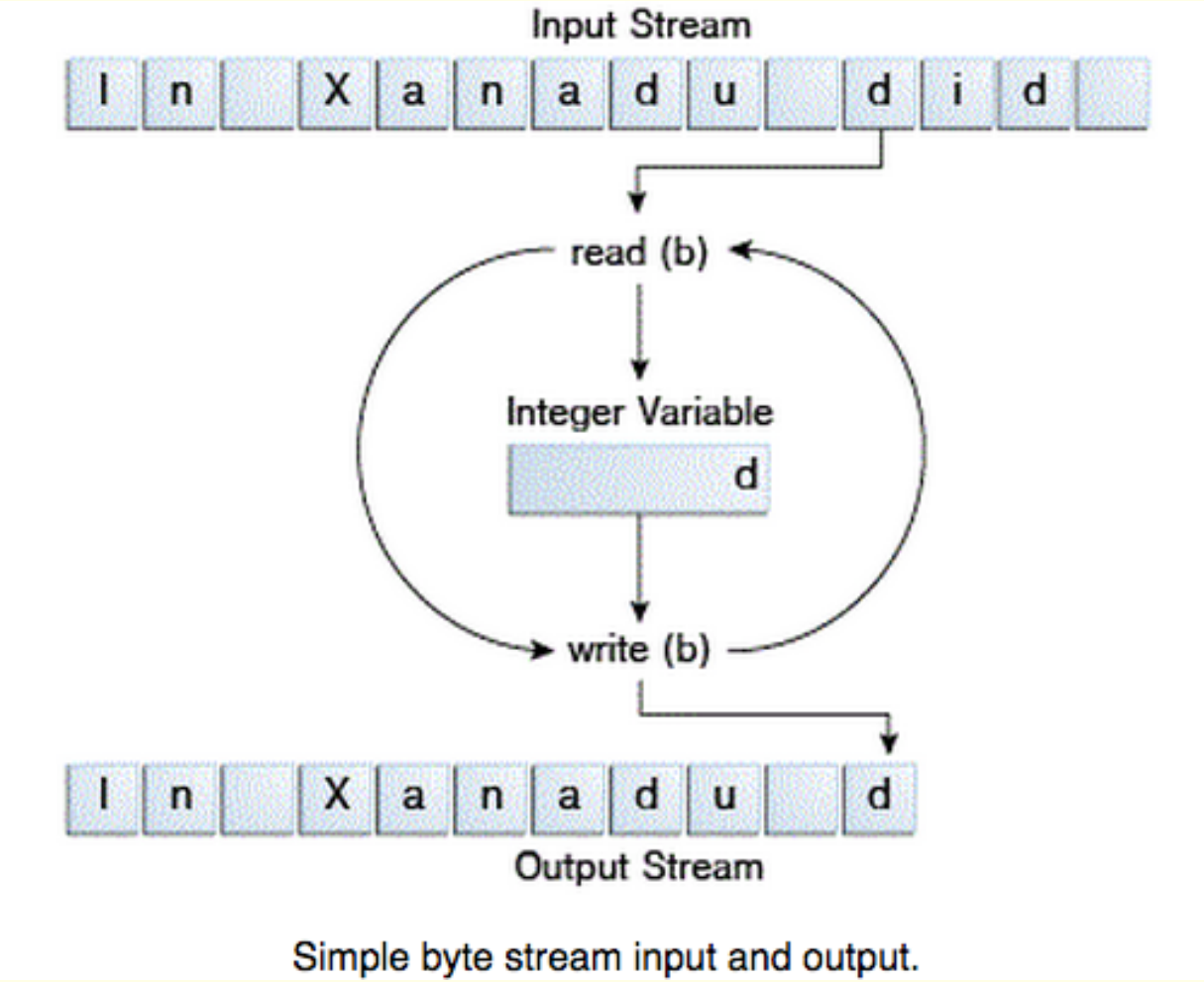
Sử dụng luồng byte (Byte Streams)

Ví dụ 1: I/O file dùng luồng Byte

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

Sử dụng luồng byte (Byte Streams)

- Mở luồng.
- Thao tác I/O.
- Đóng luồng: rất quan trọng → tránh "rò rỉ" tài nguyên



Simple byte stream input and output.

Sử dụng luồng byte (Byte Streams)

- Khi nào không dùng Byte Streams?
  - I/O character tốt nhất dùng character streams.
  - Byte Streams dùng cho các I/O nền tảng nhất. Byte Streams là nền tảng để xây dựng nên các Streams khác.

Luồng ký tự (Character Streams)

- Java lưu trữ giá ký tự dùng qui ước Unicode.
- Sử dụng luồng Byte để I/O mức vật lý. Luồng Chracter tập trung vào việc chuyển đổi giữa Bytes và Characters. I/O Character Streams tự động chuyển định dạng bên trong luồng thành tập các ký tự.
  - *Ví dụ: FileReader dùng FileInputStream, FileWriter dùng FileOutputStream*.
- Tất cả kế thừa từ 2 lớp: ***Reader*** & ***Writer***.
- I/O ký tự nói chung: ***InputStreamReader***, ***OutputStreamWriter***
- Hỗ trợ Internationalization.

Sử dụng luồng ký tự (Character Streams)

Ví dụ 2: I/O file dùng luồng Chracter

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

- *Ví dụ 1, Ví dụ 2 dùng biến int để đọc và ghi. Biến int trong ví dụ 2 lưu giá trị Character 16 bits. Biến int trong ví dụ 1 lưu giá trị Byte 8 bits.*

Sử dụng luồng ký tự (Character Streams)

- *I/O với đơn vị line: 1 chuỗi ký tự với kết thúc dòng ở cuối*

```
BufferedReader inputStream = null;
PrintWriter outputStream = null;
try {
    inputStream = new BufferedReader(new FileReader("input.txt"));
    outputStream = new PrintWriter(new FileWriter("output.txt"));
    String l;
    while ((l = inputStream.readLine()) != null) {
        outputStream.println(l);
    }
} finally {
    if (inputStream != null) {
        inputStream.close();
    }
    if (outputStream != null) {
        outputStream.close();
    }
}
```

Luồng đệm (Buffered Streams)

- I/O của luồng không đệm → kiểm soát trực tiếp bởi HĐH, ràng buộc với thao tác vật lý. → Tốn chi phí, Ít hiệu quả.
- Luồng đệm → dùng bộ nhớ như vùng đệm để I/O data. Native API được gọi khi Buffer trống/đầy tướng ứng thao tác đọc/ghi.

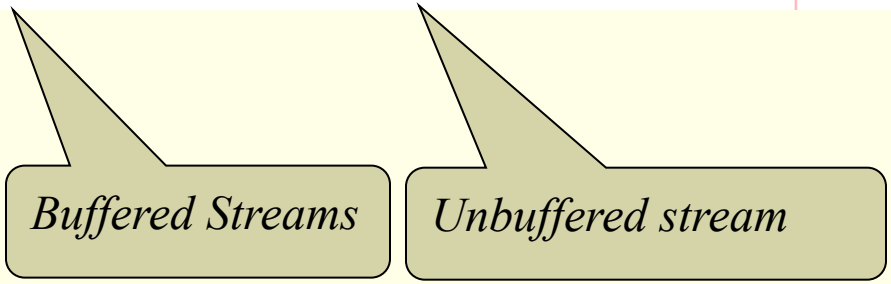
```
for(int i = 0; i < 100; i++) {
    writer.write("foorbar");
    writer.write(NEW_LINE);
}
writer.close();
```

- *Ví dụ này sẽ gọi Native API (System calls) 200 lần để ghi dữ liệu. Trong khi dùng luồng đệm thì gọi Native API 1 lần.*

Luồng đệm (Buffered Streams)

- Chuyển luồng không đệm → luồng đệm dùng kiểu bao

```
BufferedReader inputStream = new BufferedReader(new FileReader("input.txt"));
BufferedWriter outputStream = new BufferedWriter(new FileWriter("output.txt"));
```



- **BufferedInputStream & BufferedOutputStream**: *đệm byte*
- **BufferedReader & BufferedWriter**: *đệm character*

Sử dụng luồng đệm (Buffered Streams)

```
public static void copyFile(String fileIn, String fileOut) throws IOException {
    BufferedReader br = new BufferedReader(new FileReader(fileIn));
    BufferedWriter bw = new BufferedWriter(new FileWriter(fileOut));
    String line = br.readLine();
    while (line != null) {
        bw.write(line);
        bw.newLine();
        line = br.readLine();
    }
    bw.flush(); // Flush a stream manually
    br.close();
    bw.close();
}
```

Sử dụng luồng đệm (Buffered Streams)

```
public static void copyFile(String fileIn, String fileOut) throws IOException {
    BufferedReader br = new BufferedReader(new FileReader(fileIn));
    // Some buffered output classes support autoflush,
    // specified by an optional constructor argument
    PrintWriter bw = new PrintWriter(new FileWriter(fileOut), true);
    String line = br.readLine();
    while (line != null) {
        bw.println(line);
        line = br.readLine();
    }
    br.close();
    bw.close();
}
```

- *Đối tượng PrintWriter tự động flush vùng đệm mỗi lần gọi phương thức println(...).*

Scanning & Formatting

- Chuyển đổi giữa các định dạng dữ liệu khi I/O.
- Scanner API:
  - Tách tokens dùng white space (blanks, tabs, line-terminators).
  - Chỉ định separator khác: *useDelimiter(<Pattern>)*

Ví dụ: tách tokens dùng Scanner

```
try (Scanner s = new Scanner(new BufferedReader(
        new FileReader("C:\\Input.txt")))) {
    while (s.hasNext()) {
        System.out.println(s.next());
    }
}
```

Scanning & Formatting

- Formatting API: đưa dữ liệu về kiểu phù hợp.
- Các luồng hiện thực việc định dạng dữ liệu: bổ sung thêm tập các phương thức cho phép chuyển đổi kiểu dữ liệu nội tại bên trong thành dữ liệu định dạng tương ứng đầu ra.
  - **PrintWriter**: luồng ký tự
  - **PrintStream**: luồng byte

Scanning & Formatting

- Phương thức **format()**: định dạng hầu hết các giá trị số dựa trên một chuỗi với nhiều options định dạng trước.

```
int i = 3;
double r = Math.sqrt(i);
System.out.format("The square root of %d is %f.%n", i, r);
```

  - *d: formats an integer value as a decimal value.*
  - *f formats a floating point value as a decimal value.*
  - *n outputs a platform-specific line terminator.*
  - *x formats an integer as a hexadecimal value.*
  - *s formats any value as a string.*
  - *tB formats an integer as a locale-specific month name.*

Luồng dữ liệu (Data Streams)

- Hỗ trợ I/O String values & các kiểu dữ liệu cơ sở.
- *Ví dụ: ghi và đọc lên các mẫu tin từ file.*

```
static final String dataFile = "C:\\invoicedata";
static final int[] units = {1, 2, 3, 4, 5};
static final double[] prices = {100.00, 105.99, 15.99, 30.99, 4.99};
static final String[] descs = {
    "MotherBoard",
    "CPU",
    "Ram",
    "HDD",
    "Mouse"
};
```

Luồng dữ liệu (Data Streams)

```
try (DataOutputStream out = new DataOutputStream(new BufferedOutputStream(
        new FileOutputStream(dataFile)))) {
    // DataStreams writes out the records and closes the output stream.
    for (int i = 0; i < prices.length; i++) {
        // writeUTF method writes out String values in a modified form of UTF-8.
        // This is a variable-width character encoding
        // that only needs a single byte for common Western characters.
        out.writeUTF(descs[i]);
        out.writeInt(units[i]);
        out.writeDouble(prices[i]);
    }
}
```

Luồng dữ liệu (Data Streams)

```
double price;
int unit;
String desc;
double total = 0.0;
// DataStreams can read each record in the stream,
// reporting on the data it encounters.
try (DataInputStream in = new DataInputStream(
        new BufferedInputStream(new FileInputStream(dataFile)))) {
    while (true) {
        desc = in.readUTF();
        unit = in.readInt();
        price = in.readDouble();
        System.out.format("You ordered %d" + " units of %s at $%.2f%n",
                unit, desc, price);
        total += unit * price;
    }

} catch (EOFException e) {
}
```

Luồng đối tượng (Object Streams)

- Object Streams hỗ trợ I/O binary của các Objects, Complex Objects.
- Luồng I/O Object: **ObjectInputStream** và **ObjectOutputStream**.

Class ObjectInputStream

java.lang.Object
    java.io.InputStream
        java.io.ObjectInputStream

All Implemented Interfaces:

Closeable, DataInput, ObjectInput, ObjectStreamConstants, AutoCloseable

Class ObjectOutputStream

java.lang.Object
    java.io.OutputStream
        java.io.ObjectOutputStream

All Implemented Interfaces:

Closeable, DataOutput, Flushable, ObjectOutput, ObjectStreamConstants, AutoCloseable

Tất cả phương thức I/O kiểu cơ sở trong Data Streams cũng được hiện thực trong Object Streams, do implement DataInput/DataOutput

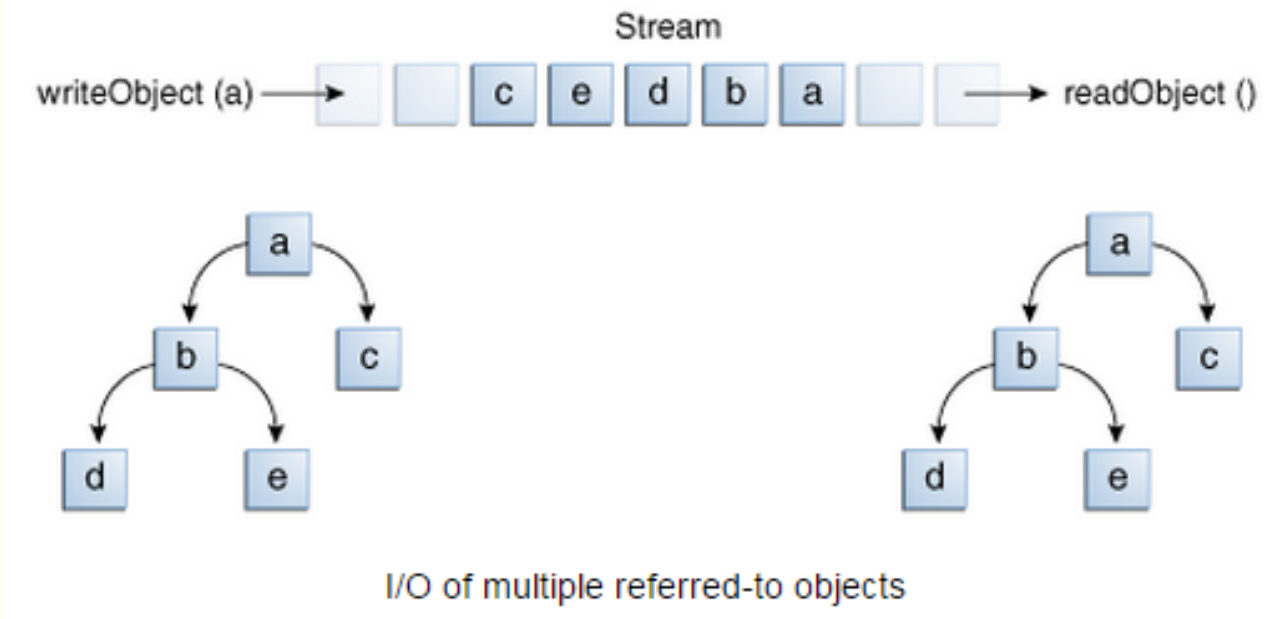
Luồng đối tượng (Object Streams)

```
String s1 = "Trường Đại học CNTT";
String s2 = "Nhập môn Lập trình Java";
int i = 897648764;
try {
    // create a new file with an ObjectOutputStream
    ObjectOutputStream out = new ObjectOutputStream(
            new FileOutputStream("C:\\OutputStream.txt"));
    // write something in the file
    out.writeObject(s1);
    out.writeObject(s2);
    out.writeObject(i);
    // close the stream
    out.close();

    // create an ObjectInputStream for the file we created before
    ObjectInputStream ois
            = new ObjectInputStream(new FileInputStream("C:\\OutputStream.txt"));
    // read and print what we wrote before
    System.out.println("" + (String) ois.readObject());
    System.out.println("" + (String) ois.readObject());
    System.out.println("" + ois.readObject());
} catch (Exception ex) {
    ex.printStackTrace();
}
```

Luồng đối tượng (Object Streams)

- I/O các complex Objects: gọi ***writeObject(a***) không chỉ ghi ***a***, mà tất cả các objects tạo nên/liên quan ***a (là b,c,d,e)*** cũng được ghi. Khi ***a*** được đọc bởi ***readObject***, các object liên quan ***(b,c,d,e)*** cũng được đọc. Tất cả các references được giữ nguyên. (hình vẽ)



I/O of multiple referred-to objects

Luồng đối tượng (Object Streams)

- Khi 2 Objects (**A & B**) refer đến cùng 1 object nào đó (**C**)? Khi 2 Objects A & B được đọc lên trong luồng thì sao?
  - *Luồng chỉ chứa 1 copy của object **C**, nhưng có thể có nhiều references đến **C**.*
  - *Ghi 1 Object xuống stream 2 lần → chỉ ghi 2 lần reference.*

```
Object ob = new Object();
out.writeObject(ob);
out.writeObject(ob);
Object ob1 = in.readObject();
Object ob2 = in.readObject();
```

- *Hai biến ob1, ob2 refer đến cùng 1 single object.*
- *Nhưng nếu 1 Object được ghi xuống 2 luồng khác nhau → thì duplicate → đọc từ 2 luồng sẽ thấy 2 object khác nhau.*

Làm việc thư mục – Lớp File

- Java.lang.Object

  + **java.io.File**

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

Làm việc 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");

Làm việc thư mục – Lớp File

- **Một số phương thức thường dùng:**

*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.

Java Socket I/O

http://www.oracle.com/technetwork/java/socket-140484.html

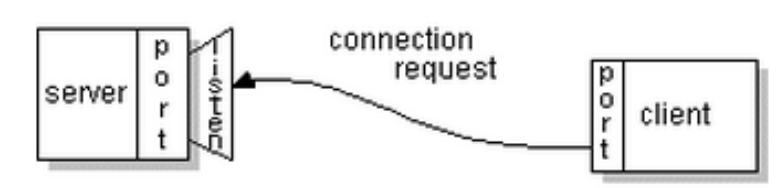
What is a socket?

- *A socket is one end-point of a two-way communication link between two programs running on the network. Socket classes are used to represent the connection between a client program and a server program. The java.net package provides two classes--Socket and ServerSocket--that implement the client side of the connection and the server side of the connection, respectively.*
- *An end-point is a combination of an IP address and a port number. Every TCP connection can be uniquely identified by its two endpoints. That way you can have multiple connections between your host and the server.*

http://www.oracle.com/technetwork/java/socket-140484.html

What is a Socket?

- Normally, a server runs on a specific computer and has a socket that is bound to a specific port number. The server just waits, listening to the socket for a client to make a connection request.
- If everything goes well, the server accepts the connection.
- The client and server can now communicate by writing to or reading from their sockets



Reading from and Writing to a Socket

```
public class EchoServer {
    public static void main(String[] args) throws IOException {
        if (args.length != 1) {
            System.err.println("Usage: java EchoServer <port number>");
            System.exit(1);
        }
        int portNumber = Integer.parseInt(args[0]);
        try (   ServerSocket serverSocket = new ServerSocket(Integer.parseInt(args[0]));
                Socket clientSocket = serverSocket.accept();
                PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);
                BufferedReader in = new BufferedReader(
                        new InputStreamReader(clientSocket.getInputStream()));) {
            String inputLine;
            while ((inputLine = in.readLine()) != null) {
                System.out.println("From Client:" + inputLine);
                out.println(inputLine);
            }
        } catch (IOException e) {
            System.out.println("Exception caught when trying to listen on port "
                    + portNumber + " or listening for a connection");
            System.out.println(e.getMessage());
        }
    }
}
```

```java
public class EchoClient {
    public static void main(String[] args) throws IOException {
        if (args.length != 2) {
            System.err.println("Usage: java EchoClient <host name> <port number>");
            System.exit(1);
        }
        String hostName = args[0];
        int portNumber = Integer.parseInt(args[1]);
        try (   Socket echoSocket = new Socket(hostName, portNumber);
                PrintWriter out = new PrintWriter(echoSocket.getOutputStream(), true);
                BufferedReader in = new BufferedReader(
                        new InputStreamReader(echoSocket.getInputStream()));
                BufferedReader stdIn = new BufferedReader(
                        new InputStreamReader(System.in))) {
            String userInput;
            while ((userInput = stdIn.readLine()) != null) {
                out.println(userInput);
                System.out.println("echo: " + in.readLine());
            }
        } catch (UnknownHostException e) {
            System.err.println("Don't know about host " + hostName);
            System.exit(1);
        } catch (IOException e) {
            System.err.println("Couldn't get I/O for the connection to " + hostName);
            System.exit(1);
        }
    }
}
```

Tài liệu tham khảo

1. https://docs.oracle.com/javase/tutorial/essential/io/streams.html
2. https://docs.oracle.com/javase/tutorial/essential/io/index.html
3. https://docs.oracle.com/javase/tutorial/essential/io/QandE/questions.html
4. http://www.oracle.com/technetwork/java/socket-140484.html
5. https://docs.oracle.com/javase/tutorial/networking/sockets/readingWriting.html