



Các luồng vào/ra (I/O Streams)

Lê Khánh Trình

Nội dung



- Tổng quan
- File
- I/O với file văn bản
- I/O với file nhị phân
- Một số luồng trong Java

- **Tổng quan**
- File
- I/O với file văn bản
- I/O với file nhị phân
- Một số luồng trong Java

Tổng quan



- Luồng nhập dữ liệu vào (input) và xuất dữ liệu ra (output) của chương trình
 - Input: Bàn phím, file
 - Output: Màn hình, file

Luồng (stream)

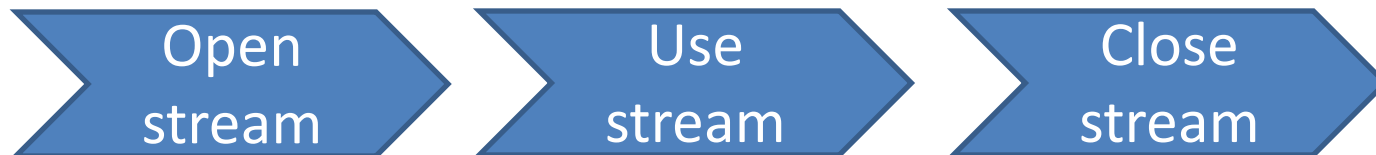


- Là đối tượng đưa/lấy dữ liệu
 - Là trung gian giữa nguồn vào và ra của dữ liệu
 - System.in là input stream
 - System.out là output stream
 - Kết nối chương trình với một đối tượng I/O
 - System.in: từ bàn phím
 - System.out: tới màn hình

Mô hình I/O



- Mô hình luồng (stream)
 - Mở stream
 - Sử dụng (read, write hoặc cả hai)
 - Đóng stream



Mở stream

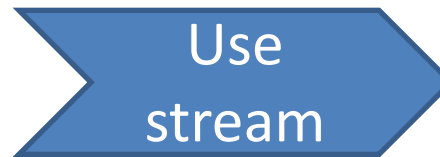


Open
stream

Khi cần đọc/ghi dữ liệu:

- Kết nối tới nguồn dữ liệu
- Thao tác thông qua đối tượng stream

```
FileReader fileReader = new FileReader(fileName);
```



```
int charAsInt;  
charAsInt = fileReader.read();
```

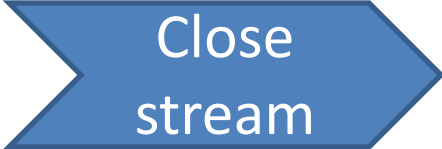
- `fileReader.read()`:
 - Đọc một ký tự, trả về dạng `int`
 - Trả về -1 nếu đọc hết



- Có thể ép kiểu về `char`:

```
char ch = (char) fileReader.read();
```


Đóng stream

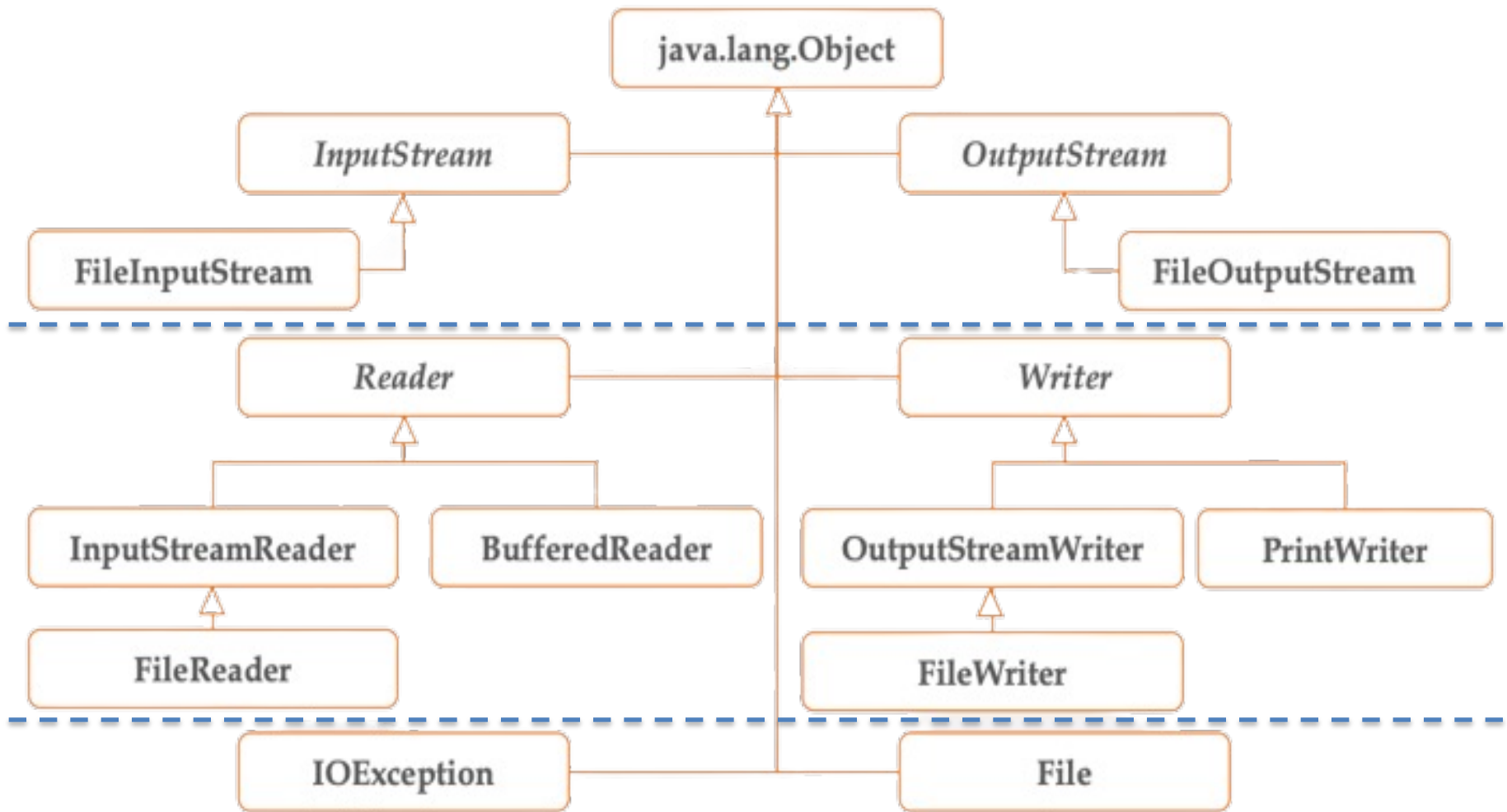
A blue arrow pointing to the right, containing the text "Close stream" in white. The arrow has a white outline and a slight 3D effect.

Close
stream

- Thông thường Java sẽ tự đóng các luồng khi chương trình kết thúc
- Không nên quá phụ thuộc vào điều này

```
fileReader.close();
```

Các lớp xử lý I/O trong Java



Nội dung



- Tổng quan
- **File**
- I/O với file văn bản
- I/O với file nhị phân
- Một số luồng trong Java

Lớp File



- Là đối tượng thao tác luồng nhập/xuất dữ liệu phổ biến
- Lớp File cung cấp các chức năng cơ bản để thao tác với tệp
 - tạo file,
 - mở file,
 - các thông tin về file và thư mục

Tạo đối tượng File



```
File myFile;
```

```
myFile = new File("data.txt");
```

```
myFile = new File("myDocs", "data.txt");
```

Thư mục cũng được coi như là một tệp đặc biệt

```
File myDir = new File("myDocs");
```

```
File myFile = new File(myDir, "data.txt");
```

có phương thức riêng để thao tác với thư mục

Một số phương thức



- Tên file
 - `String getName()`
 - `String getPath()`
 - `String getAbsolutePath()`
 - `String getParent()`
 - `boolean renameTo(File newName)`
- Kiểm tra file
 - `boolean exists()`
 - `boolean canWrite()`
 - `boolean canRead()`
 - `boolean isFile()`
 - `boolean isDirectory()`
 - `boolean isAbsolute()`

File nhị phân và file văn bản



Tất cả các dữ liệu và chương trình bản chất là các số 0 và 1

- Mỗi chữ số chỉ có thể mang 2 giá trị này, do đó chúng ta gọi là nhị phân
- bit là một chữ số nhị phân
- byte là một tập 8 bits

File nhị phân và file văn bản



- File nhị phân: các bit thể hiện các kiểu thông tin được mã hoá (chỉ lệnh - *instruction* hoặc dữ liệu số)
 - Dễ đọc bằng máy tính, khó với con người
- File văn bản: các bit biểu diễn ký tự chữ cái
 - Mỗi chữ cái ASCII là một byte
 - VD: Java code file, notepad, ...

Nội dung

- Tổng quan
- File
- **I/O với file văn bản**
- I/O với file nhị phân
- Một số luồng trong Java

Xử lý file văn bản



- FileReader: Luồng đọc các ký tự
 - `FileReader(String fileName)`
 - `read()`: xử lý `char` và `char[]`
- FileWriter: Luồng ghi các ký tự
 - `FileWriter(String fileName)`
 - `write()`

Ví dụ: Đọc từng ký tự

```
static void copyFile(FileReader inputFile, FileWriter outputFile)
{
    try{
        // Đọc ký tự đầu tiên
        int nextChar = inputFile.read();

        // Kiểm tra đã đọc hết file chưa?
        while(nextChar != -1) {
            outputFile.write(nextChar);
            // Đọc ký tự tiếp theo
            nextChar = inputFile.read();
        }
        outputFile.flush();
    } catch(IOException e) {
        System.out.println("Unable to copy file");
    }
}
```

Ví dụ: Đọc/ghi dữ liệu theo lô



```
static void copyFile(FileReader inputFile, FileWriter outputFile)
    throws IOException
{
    final int bufferSize = 1024;
    //Tạo bộ đệm với kích thước cụ thể
    char[] buffer = new char[bufferSize];

    // Đọc đoạn ký tự đầu tiên
    int numberRead = inputFile.read(buffer);

    while(numberRead > 0) {
        // Ghi những thông tin được lưu ở numberRead
        outputFile.write(buffer, 0, numberRead);
        numberRead = inputFile.read(buffer);
    }
    outputFile.flush();
}
```

Đọc/ghi theo dòng



- Đôi khi cần thao tác với từng dòng
 - Các file config.ini, .csv, ...
- Java hỗ trợ **BufferedReader** và **BufferedWriter** thông qua các đối tượng **FileReader** và **FileWriter** tương ứng

Ví dụ: copy file theo từng dòng



```
BufferedReader reader = new BufferedReader(  
    new FileReader("source.txt"));  
BufferedWriter writer = new BufferedWriter(  
    new FileWriter("dest.txt"));  
  
// Đọc dòng đầu tiên  
String line = reader.readLine();  
// trả về null nếu kết thúc file  
while(line != null) {  
    // Ghi dòng đã đọc  
    writer.write(line);  
    // Ghi ký tự xuống dòng  
    writer.newLine();  
    // Đọc dòng tiếp theo  
    line = reader.readLine();  
}
```

Sử dụng bộ đệm



```
// Sử dụng Buffered Reader và Buffered Writer (có bộ đệm)
long startTime = System.nanoTime();
try {
    BufferedReader reader = new BufferedReader(new FileReader(sourceFile));
    BufferedWriter writer = new BufferedWriter(new FileWriter(destFile));
    String line;
    while ((line = reader.readLine()) != null) {
        writer.write(line);
        writer.newLine();
    }
} catch (IOException e) {
    e.printStackTrace();
}
long bufferedTime = System.nanoTime() - startTime;
```

Không sử dụng bộ đệm



```
// Sử dụng FileReader và FileWriter (không có bộ đệm)
startTime = System.nanoTime();
try (FileReader reader = new FileReader(sourceFile);
    FileWriter writer = new FileWriter(destFile)) {
    int c;
    while ((c = reader.read()) != -1) {
        writer.write(c);
    }
} catch (IOException e) {
    e.printStackTrace();
}
long nonBufferedTime = System.nanoTime() - startTime;
```


Nội dung



- Tổng quan
- File
- I/O với file văn bản
- **I/O với file nhị phân**
- Một số luồng trong Java

Luồng input, output nhị phân



- Sử dụng lớp `FileInputStream` và `FileOutputStream`
- `FileInputStream/FileOutputStream` liên kết một luồng input/output nhị phân với một file

FileInputStream



- Phương thức khởi tạo

```
public FileInputStream(String filename)  
public FileInputStream(File file)
```

- Ngoại lệ `java.io.FileNotFoundException` có thể xảy ra file không tồn tại

InputStream



- **int read()**
 - đọc byte kế tiếp từ input stream. Trả về -1 nếu hết file
- **int read(byte buf[])**
 - đọc tất cả các byte hiện có trong stream vào mảng buf[]
- **int read(byte buf[], int offset, int length)**
 - Đọc số lượng (length) byte từ Stream hiện tại, lưu vào trong mảng byte (buf) bắt đầu từ vị trí (offset) được chỉ định.
- **void close()**
 - đóng inputstream hiện tại

FileOutputStream



- Phương thức khởi tạo

```
public FileOutputStream(String filename)
public FileOutputStream(File file)
public FileOutputStream(String filename, boolean append)
public FileOutputStream(File file, boolean append)
```

- Nếu file chưa tồn tại thì tạo file mới
- Nếu đã tồn tại, 2 phương thức đầu sẽ xoá nội dung đã có

OutputStream



- **int write(int c)**
 - ghi một byte đến output stream hiện tại
- **int write(byte buf[])**
 - ghi một mảng các byte (buf[]) đến output stream hiện tại
- **int write(byte buf[], int offset, int length)**
 - ghi một mảng các byte có độ dài (length) xác định đến output stream hiện tại, bắt đầu từ vị trí (offset) được chỉ định
- **void close()**
 - đóng output stream hiện tại
- **void flush()**
 - Đẩy các byte được lưu trong vùng đệm của Stream ra thiết bị ngoại vi

Vì sao không sử dụng luôn các lớp hỗ trợ luồng I/O nhị phân để đọc/ghi các file văn bản?



Giải thích



- Byte stream:
 - Audio/video/image
 - File nhị phân

Le Khanh Trinh
Lê Khánh Trình

- Character stream:
 - File văn bản
 - Hỗ trợ mã hoá bộ ký tự khác với bộ ký tự mặc định của hệ thống

Le Khanh Trinh
LÃ^a KhÃ;nh TrÃ¬nh

- **I/O với file nhị phân**
 - Đọc/ghi các *đối tượng (objects)*

Thao tác với đối tượng

- Đọc/ghi các đối tượng vào file
- Gọi là quá trình tuần tự – **Serialization**
- Ở các ngôn ngữ khác có thể rất khó khăn do đối tượng đó có thể tham chiếu tới đối tượng khác
 - Java hỗ trợ thực hiện quá trình này dễ dàng hơn

- Để một đối tượng có thể được tuần tự hoá
 - Là một lớp **public**
 - Thực thi **interface Serializable**
 - Các thành phần phải có thể tuần tự hoá được
 - Kiểu nguyên thuỷ
 - Các đối tượng **serializable**

interface Serializable



- **interface Serializable** không định nghĩa bất kỳ phương thức nào
 - Java sử dụng **interface Serializable** như là một flag để xử lý các lớp liên quan

Thao tác với đối tượng



- Sử dụng các lớp `ObjectStreamReader` và `ObjectStreamWriter`
- Khởi tạo bằng các đối tượng `FileInputStream` và `FileOutputStream` tương ứng

Ví dụ



```
public class Person implements Serializable {  
    private String name;  
    private int age;  
  
    public Person(String name, int age) {  
        this.name = name;  
        this.age = age;  
    }  
  
    @Override  
    public String toString() {  
        return "Person [name=" + name + ", age=" + age + " ]";  
    }  
}
```

Ghi đối tượng ra file



```
public class ObjectSerializationExample {
    public static void main(String[] args) {
        // Tạo một danh sách các đối tượng Person
        Person person1 = new Person("An", 30);
        Person person2 = new Person("Bình", 25);
        Person person3 = new Person("Công", 35);

        try {
            // Ghi danh sách các đối tượng vào tệp "people.ser"
            ObjectOutputStream oos = new ObjectOutputStream(new
                FileOutputStream("people.ser"));

            oos.writeObject(person1);
            oos.writeObject(person2);
            oos.writeObject(person3);
            oos.close();
        }
    }
}
```

Đọc đối tượng từ file



```
// Đọc danh sách các đối tượng từ tệp "people.ser"
ObjectInputStream ois = new ObjectInputStream(new
    FileInputStream("people.ser"));
Person p1 = (Person) ois.readObject();
Person p2 = (Person) ois.readObject();
Person p3 = (Person) ois.readObject();
ois.close();
```

```
System.out.println("Đọc các đối tượng từ tệp:");
System.out.println(p1);
System.out.println(p2);
System.out.println(p3);
} catch (IOException | ClassNotFoundException e) {
    e.printStackTrace();
}
}
```


Nội dung



- Tổng quan
- File
- I/O với file nhị phân
- I/O với file văn bản
- **Một số luồng trong Java**

Các luồng có sẵn trong Java



- Tất cả các chương trình Java đều tự động `import java.lang`
- Gói `java.lang` định nghĩa một lớp gọi là `System` đóng gói nhiều thành phần khác nhau của môi trường làm việc
- Lớp `System` chứa 3 biến stream có sẵn
 - `in`, `out`, `err` (`System.in`, `System.out`, `System.err`)
- Nhưng biến này được khai báo `public` và `static` trong lớp `System`.

Các luồng có sẵn trong Java



- **System.out** sử dụng cho luồng output tiêu chuẩn đưa dữ liệu mặc định ra console (màn hình)
- **System.in** sử dụng luồng input tiêu chuẩn mặc định là bàn phím (bàn phím)
- **System.err** sử dụng luồng in lỗi, cũng đưa dữ liệu mặc định ra console (màn hình)
 - Những luồng này có thể được định hướng lại đến bất kỳ thiết bị I/O phù hợp nào

Tổng kết



- Mô hình làm việc với luồng I/O
 - Mở luồng -> Sử dụng -> Đóng luồng
- I/O với file text
 - Làm việc với char: **FileReader** và **FileWriter**
 - Làm việc với từng dòng: **BufferedReader** và **BufferedWriter** (khởi tạo bằng đối tượng **FileReader** và **FileWriter**)
- I/O với file nhị phân
 - Làm việc với byte: **FileInputStream** và **FileOutputStream**
 - Làm việc với đối tượng: **ObjectInputStream** và **ObjectOutputStream** (khởi tạo bằng đối tượng **FileInputStream** và **FileOutputStream**)
 - Cần thực thi giao diện **Serializable**

