Lập trình hướng đối tượng

Vào ra file

GV: ThS. Ngô Tiến Đức

Nội dung chính

- Vào ra file trong Java
- Đọc dữ liệu từ file văn bản
- Ghi dữ liệu ra file văn bản
- Vào ra file văn bản theo luồng
- Vào ra file kiểu nhị phân

Vào ra file trong Java

- I/O = Input/Output
  - Input: Đưa dữ liệu vào cho chương trình đọc
  - Output: Ghi dữ liệu từ chương trình ra
- File I/O: Thao tác đọc và ghi dữ liệu với các tập tin
  - Được lưu trữ trên phần cứng máy tính
  - Tự động nhập dữ liệu vào thay vì gõ thủ công từng giá trị mỗi lần nhập
  - Đầu ra của chương trình này có thể làm đầu vào cho chương trình khác

Vào ra file trong Java

Lớp **File**:

- Cho phép thao tác với các file được lưu trên phần cứng máy tính (on disk)
- Nằm trong package `java.io`
- Khởi tạo đối tượng File với đường dẫn và tên file

```
import java.io.File;
…
File file = new File("file_name.txt");
// file nằm trong folder hiện tại (directory của project)
```

Vào ra file trong Java

Một số phương thức của lớp File:

Phương thức	Chức năng
`boolean canRead()`	Kiểm tra xem file có đọc được không
`boolean canWrite()`	Kiểm tra xem file có ghi được không
`boolean createNewFile()`	Tạo file mới (file trống)
`boolean delete()`	Xóa file
`boolean exists()`	Kiểm tra xem file có tồn tại không
`String getName()`	Trả về tên file
`String getAbsolutePath()`	Trả về đường dẫn của file
`long length()`	Trả về kích thước file (tính bằng byte)
`String[] list()`	Trả về mảng các file cùng folder (directory)

Đọc dữ liệu từ file văn bản

- Tạo một đối tượng Scanner để đọc file thay vì đọc từ console
- Phải xử lý ngoại lệ **FileNotFoundException** nếu không chương trình sẽ không thể biên dịch

```
public static void main(String[] args) throws FileNotFoundException
{
	File file = new File("input.txt");
	Scanner sc = new Scanner(file);
	…
}
```

Đọc dữ liệu từ file văn bản

Đọc file theo từng token

- Ví dụ dữ liệu trong file có tên "numbers.txt":

```
308.2 14.9
7.4 2.8

3.9 4.7    -15.4
    2.8
```

- Scanner sẽ coi dữ liệu đọc được như một luồng (stream) các kí tự và sử dụng con trỏ đầu vào để duyệt:

```
308.2 14.9\n7.4 2.8\n\n3.9 4.7 -15.4\n2.8\n
^
```

Scanner's input cusor

Đọc dữ liệu từ file văn bản

Đọc file theo từng token (tiếp)

Mỗi lời gọi tới các phương thức `next, nextInt, nextDouble,...` sẽ di chuyển con trỏ tới cuối token - bỏ qua các khoảng trắng

- VD: Gọi `sc.nextDouble()`

```
308.2 14.9\n7.4 2.8\n\n3.9 4.7 -15.4\n2.8\n
      ^
```

- Tiếp tục gọi `sc.nextDouble()`

```
308.2 14.9\n7.4 2.8\n\n3.9 4.7 -15.4\n2.8\n
           ^
```

Đọc dữ liệu từ file văn bản

- VD: Viết chương trình in ra 5 số đầu tiên trong file

```
public static void main(String[] args) throws FileNotFoundException
{
    File file = new File("numbers.txt");
    Scanner sc = new Scanner(file);
    for (int i = 0; i < 5; i++)  {
        double number = sc.nextDouble();
        System.out.println(number);
    }
}
```

Đọc dữ liệu từ file văn bản

- Các phương thức `hasNext, hasNextInt, hasNextDouble,...` để kiểm tra token tiếp theo -> Có thể sử dụng để đọc đến cuối file

```
while (sc.hasNextDouble())  {
    double number = sc.nextDouble();
    System.out.println(number);
}
```

Đọc dữ liệu từ file văn bản

- Nếu file “number.txt” là như thế này thì output in ra có giống như ví dụ trước không?

```
308.2 14.9 hello
7.4  bad code 2.8

3.9 4.7  oops  -15.4
:-)    2.8  @#*($&
```

Đọc dữ liệu từ file văn bản

```
while (sc.hasNext()) {
    if (sc.hasNextDouble()) {
        double number = sc.nextDouble();
        System.out.println(number);
    } else {
        sc.next(); // bỏ qua các token không phải double
    }
}
```

Đọc dữ liệu từ file văn bản

Đọc file theo từng dòng

- Sử dụng `hasNextLine` và `nextLine`

```
while (sc.hasNextLine()) {
    String line = sc.nextLine();
    …
}
```

- Phương thức `nextLine` sẽ trả về các kí tự từ vị trí hiện tại của con trỏ cho đến kí tự `\n` gần nhất

Đọc dữ liệu từ file văn bản

- Ví dụ file có tên "works.in"

```
An 12 8 7 3
Binh 4 11 6 2 12
Chi 8 8 8 8 7
```

- Viết chương trình in ra các dữ liệu trong file thêm ký tự ">" ở đầu dòng

```
> An 12 8 7 3
> Binh 4 11 6 2 12
> Chi 8 8 8 8 7
```

Đọc dữ liệu từ file văn bản

- Ví dụ dữ liệu trong file có tên “works.in”

```
An 12 8 7 3
Binh 4 11 6 2 12
Chi 8 8 8 8 7
```

- Viết chương trình in ra tổng số giờ làm của mỗi người

```
An worked 30 hours
Binh worked 35 hours
Chi worked 39 hours
```

Đọc dữ liệu từ file văn bản

- Scanner có thể chia String thành các token (tokenize a String)

```
String text = "1.4 3.2 hello 9 27.5";

Scanner scLine = new Scanner(text);

while(scLine.hasNext()){

    System.out.println(sc.next());

} // in ra 5 token
```

Đọc dữ liệu từ file văn bản

```
while (sc.hasNextLine()) {
    String line = sc.nextLine();
    Scanner scLine = new Scanner(line);
    String name = scLine.next();
    int sum = 0;
    while(scLine.hasNextInt()) {
        sum += scLine.nextInt();
    }
    System.out.println(name + " worked " + sum + " hours");
}
```

Ghi dữ liệu ra file văn bản

- Lớp **PrintWriter** nằm trong package `java.io`
- Cần xử lý ngoại lệ `FileNotFoundException`

```
import java.io.*;
…
public static void main(String[] args) throws FileNotFoundException
{
        PrintWriter out = new PrintWriter("output.txt");
        out.println("Hello, output file!");
        …
```

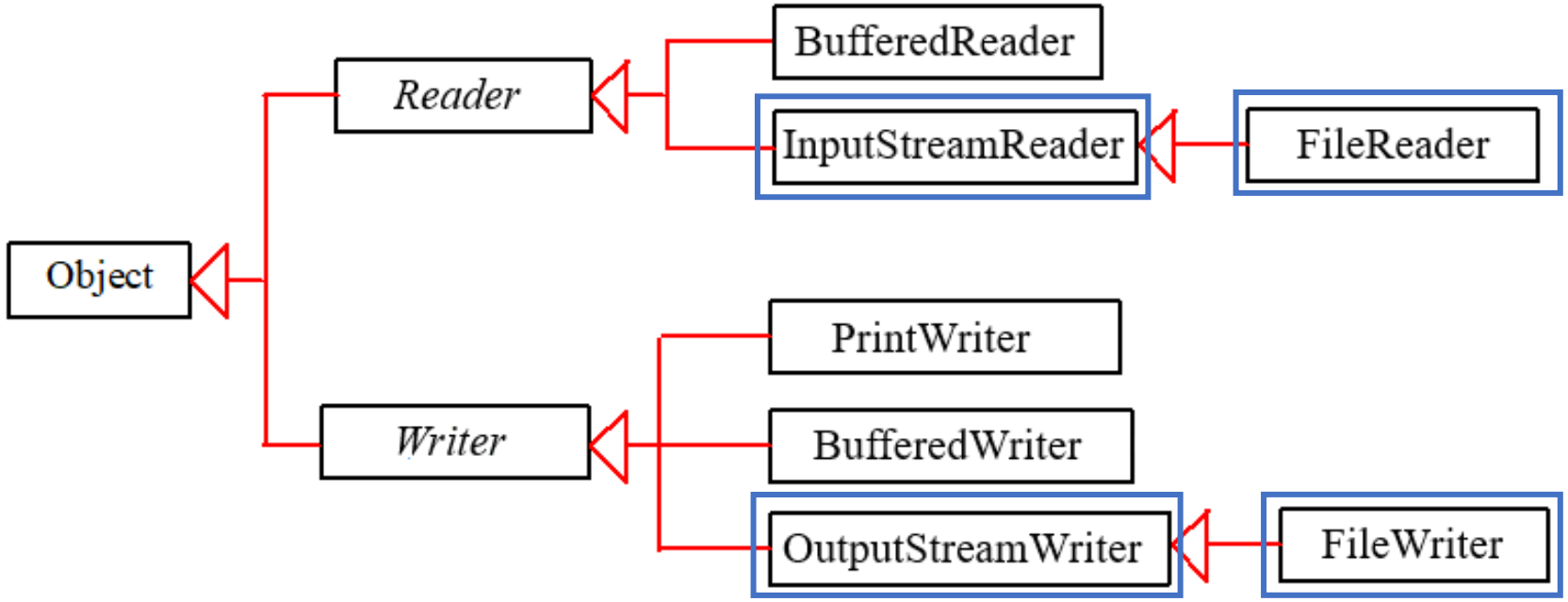
Ghi dữ liệu ra file văn bản

- Phương thức `close()`: Giải phóng tài nguyên hệ thống có liên quan tới luồng hiện tại
  - Scanner (và đọc file) cũng nên gọi `close()`
- Sau khi ghi file xong phải gọi lệnh đóng luồng xử lý:

```
PrintWriter pw = new PrintWriter("output.txt");
pw.println("Hello, output file!");
pw.close();
```

Vào ra file văn bản theo luồng

- Luồng vào ra (I/O Stream): Dòng lưu chuyển dữ liệu vào/ra
- Luồng ký tự: Đọc/ghi theo từng ký tự (char) Unicode
- Phải xử lý ngoại lệ `IOException`



Vào ra file văn bản theo luồng

Một số phương thức của Reader:

Phương thức	Chức năng
`int read()`	Đọc một ký tự từ luồng. Khi hết luồng trả về -1
`int read(char[] b)`	Đọc một mảng ký tự và lưu vào bộ nhớ đệm
`long skip(long n)`	Bỏ qua **n** ký tự khi đọc
`void close()`	Đóng luồng, giải phóng tài nguyên hệ thống liên quan tới luồng

Vào ra file văn bản theo luồng

Một số phương thức của Writer:

Phương thức	Chức năng
`void write(String s)`	Ghi các ký tự trong **s** vào luồng
`void write(char[] b)`	Ghi các ký tự từ luồng vào mảng **b**
`void close()`	Đóng luồng và giải phóng tài nguyên hệ thống liên quan tới luồng
`void flush()`	Ghi các dữ liệu trong bộ đệm luồng vào file đích

Vào ra file văn bản theo luồng

- **InputStreamReader** kế thừa Reader: Hỗ trợ chuyển từ luồng byte sang luồng ký tự
- **OutputStreamWriter** kế thừa Writer: Hỗ trợ chuyển từ luồng ký tự sang luồng byte
- **FileReader** kế thừa InputStreamReader: Đọc dữ liệu từ file văn bản
- **FileWriter** kế thừa OutputStreamWriter: Ghi dữ liệu ra file văn bản

Vào ra file văn bản theo luồng

Ví dụ sử dụng FileReader để đọc file "input.in"

```
public static void main(String[] args) throws IOException {
    FileReader fr = new FileReader("input.in");
    int character;
    while ((character = fr.read()) != -1) {
        System.out.print((char) character);
    }
}
```

Vào ra file văn bản theo luồng

- **BufferedReader** và **BufferedWriter**: Sử dụng bộ đệm để tăng tốc cho quá trình đọc/ghi
- Hỗ trợ đọc/ghi theo dòng với `readLine()` và `newline()`

```
FileReader fr = new FileReader("input.in");
BufferedReader br = new BufferedReader(fr);
String line;
while ((line = br.readLine()) != null) {
    System.out.println(line);
}
```

Vào ra file văn bản theo luồng

Ví dụ ghi file với FileWriter và BufferedWriter:

```
FileWriter fw = new FileWriter(“output.txt");
BufferedWriter br = new BufferedWriter(fw);
br.write(“Hello”);
br.newLine();
br.write(“This is file writer with buffer”);
br.close();
```

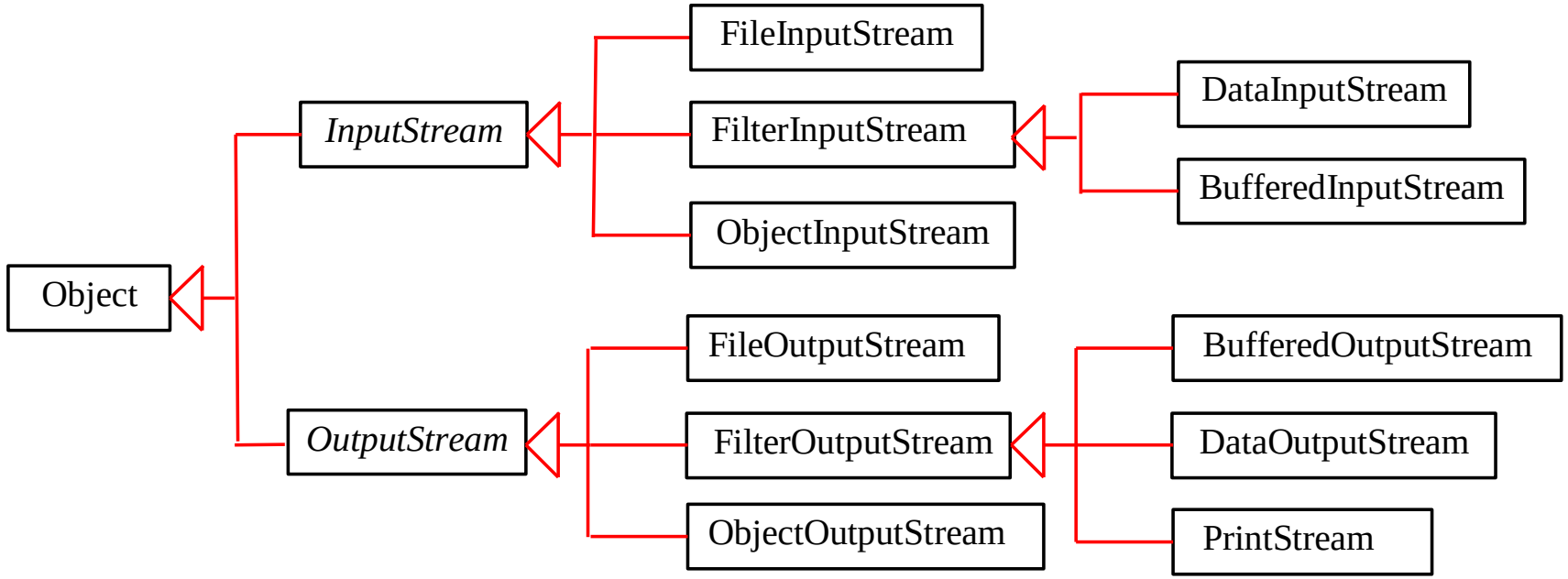
Vào ra file kiểu nhị phân

Text I/O thì làm sao?

- Nếu file không phải văn bản?
- File văn bản định dạng Unicode cần được mã hóa và giải mã trong quá trình đọc/ghi
  - Nếu file văn bản không chứa ký tự `\n`?

Vào ra file kiểu nhị phân

- Vào ra file kiểu nhị phân (Binary I/O) không cần mã hóa và giải mã: Dữ liệu được đọc/ghi theo luồng byte (byte stream)



Vào ra file kiểu nhị phân

Một số phương thức của InputStream:

Phương thức	Chức năng
`int read()`	Đọc một byte từ luồng. Khi hết luồng trả về -1
`int read(byte[] b)`	Đọc một mảng tối đa **b.length** byte dữ liệu từ luồng
`long skip(long n)`	Bỏ qua và loại bỏ **n** byte dữ liệu khỏi luồng
`void close()`	Đóng luồng, giải phóng tài nguyên hệ thống liên quan tới luồng

Vào ra file kiểu nhị phân

Một số phương thức của OutputStream:

Phương thức	Chức năng
`void write(int b)`	Ghi byte dữ liệu **b** vào luồng
`void write(byte[] b)`	Ghi các byte từ mảng **b** vào luồng
`void close()`	Đóng luồng và giải phóng tài nguyên hệ thống liên quan tới luồng
`void flush()`	Ghi các dữ liệu trong bộ đệm luồng vào file đích

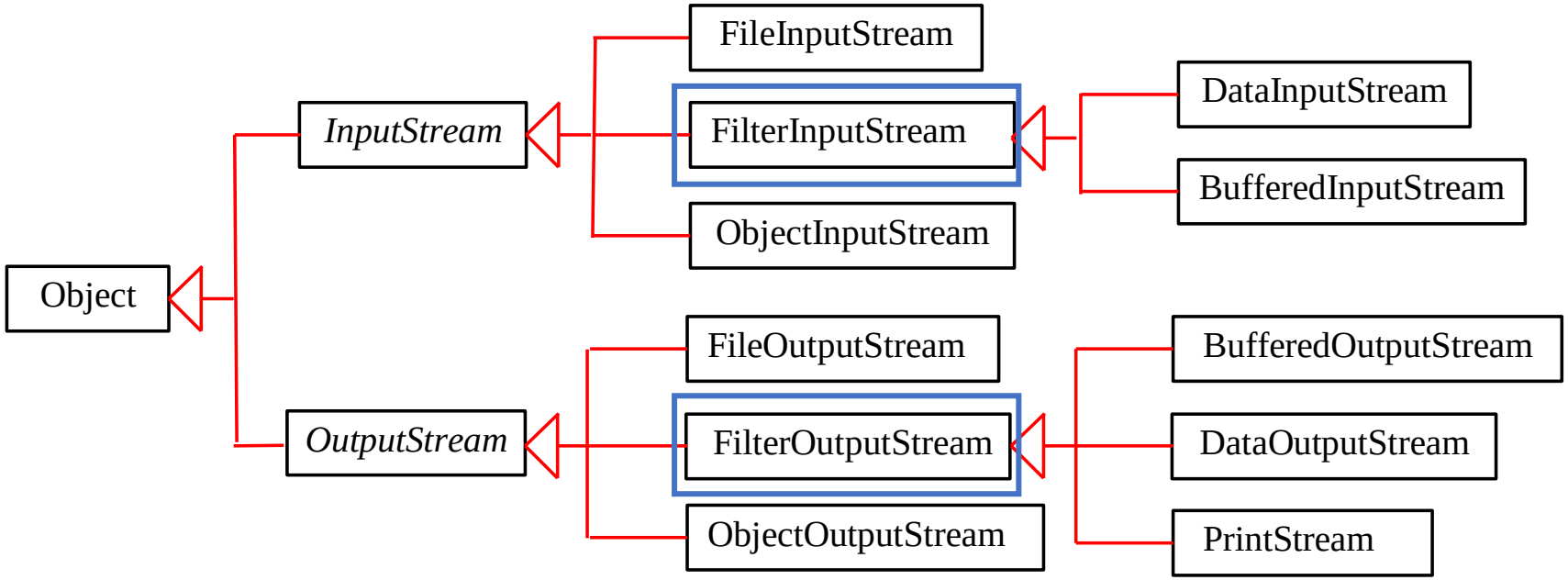
Vào ra file kiểu nhị phân

Ví dụ copy nội dung từ file “input.jpg” sang file “output.jpg”:

```
public static void main(String[] args) throws IOException {
    FileInputStream fis = new FileInputStream(“input.jpg");
    FileOutputStream fos = new FileOutputStream("output.jpg");
    int c;
    while ((c = fis.read()) != -1) {
        fos.write(c);
    }
    fis.close();
    fos.close();
}
```

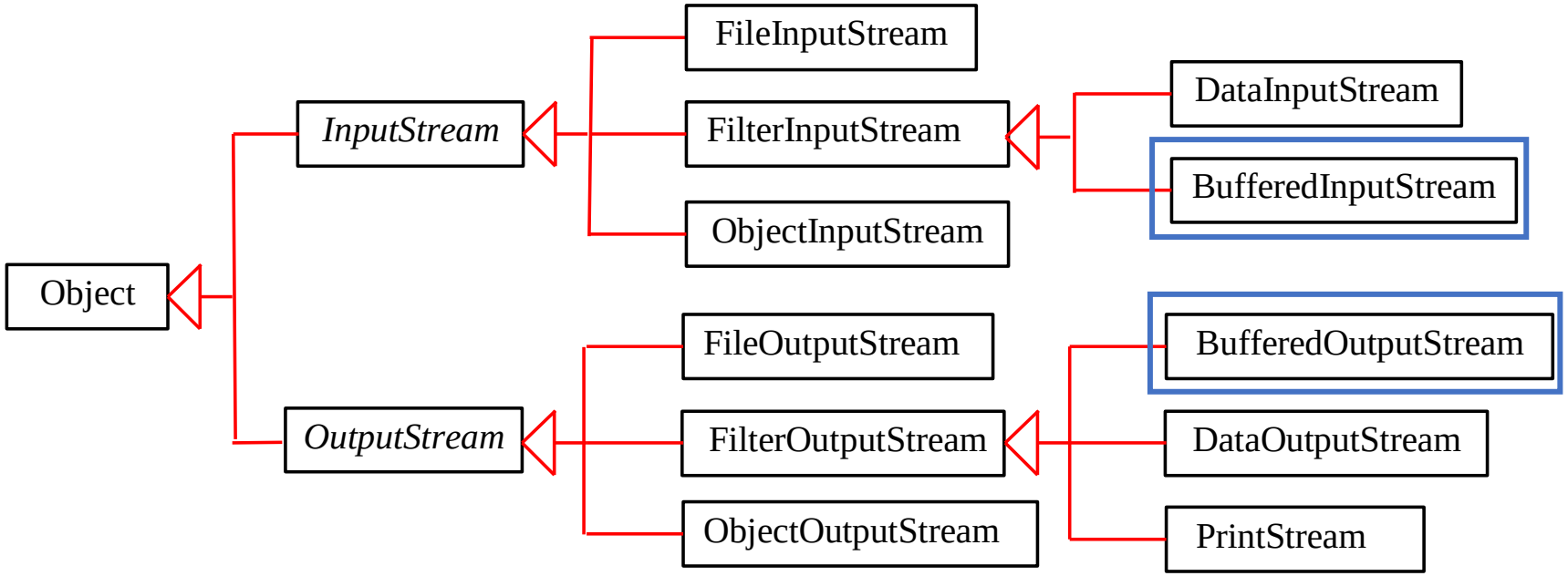
Vào ra file kiểu nhị phân

- **FilterInputStream** và **FilterOutputStream**: Các lớp trừu tượng
- Bổ sung tính năng cho các luồng đọc ghi chính



Vào ra file kiểu nhị phân

- **BufferedInputStream** và **BufferedOutputStream**: Sử dụng bộ đệm để tăng tốc cho quá trình vào ra file
- Không chứa phương thức nào mới

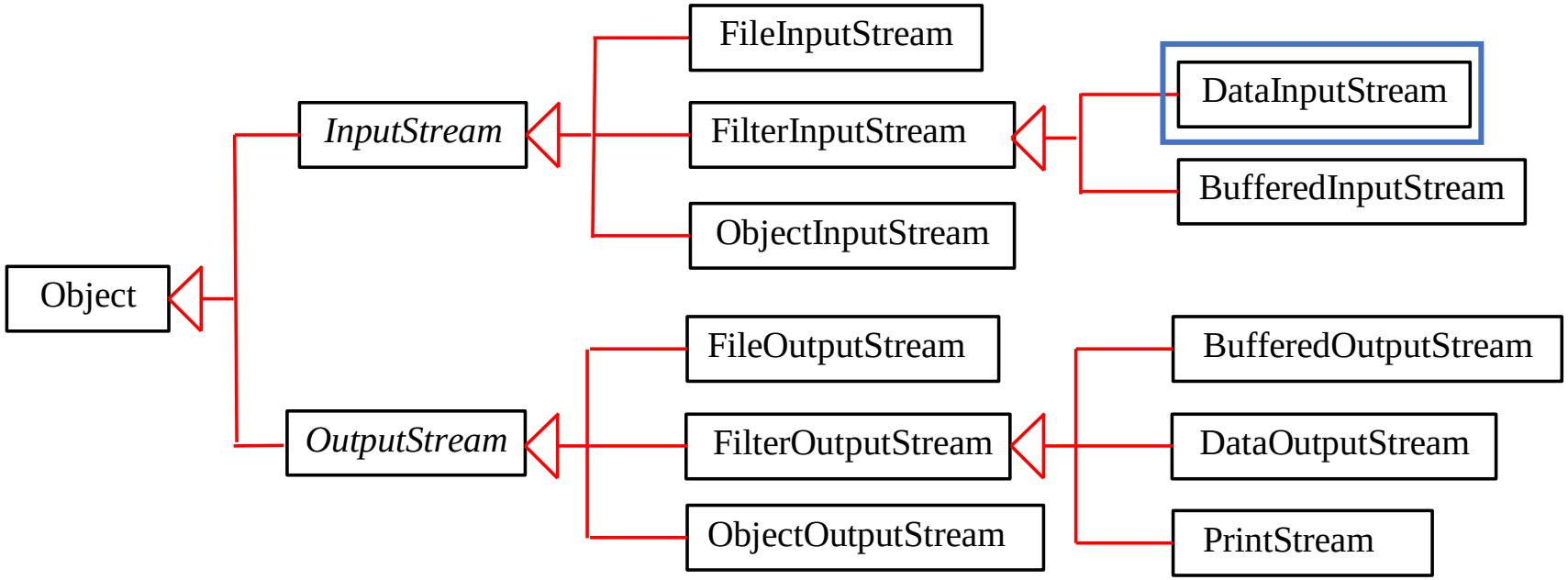


Vào ra file kiểu nhị phân

```
FileInputStream fis = new FileInputStream(“input.txt");
BufferedInputStream bis = new BufferedInputStream(fis);
FileOutputStream fos = new FileOutputStream("output.txt");
BufferedOutputStream bos = new BufferedOutputStream(fos);
…
bis.close();
bos.close();
```

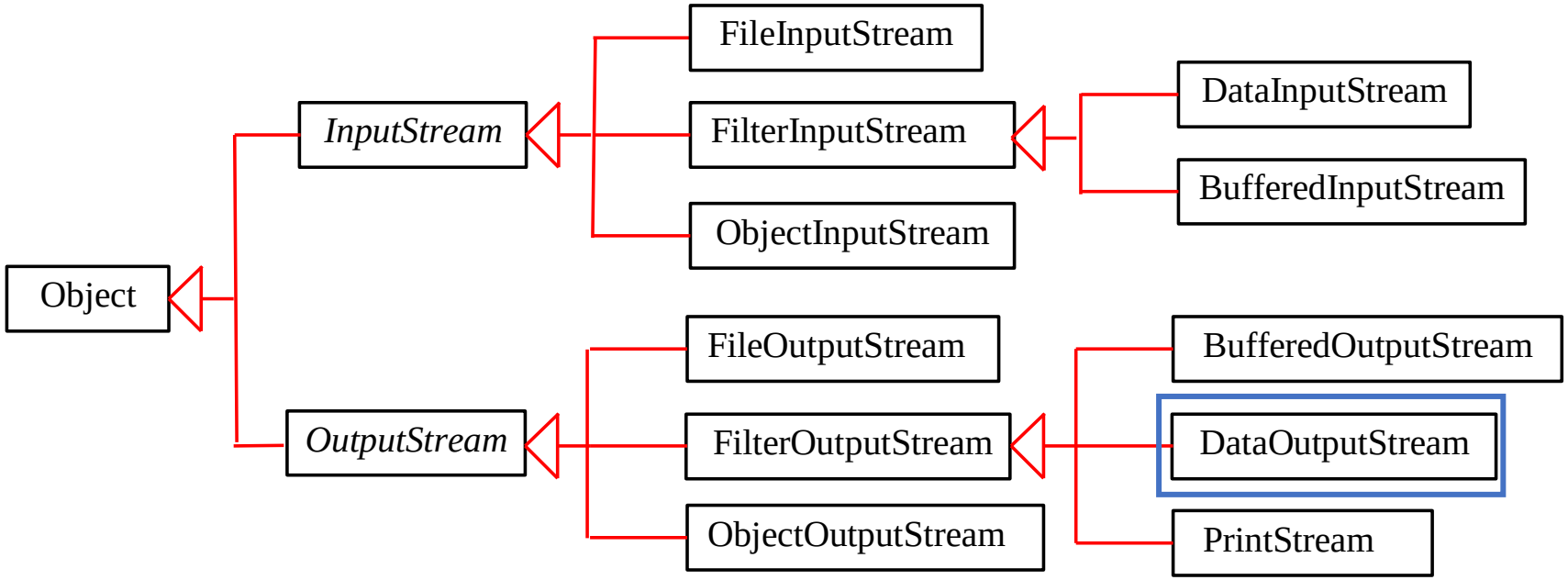
Vào ra file kiểu nhị phân

- **DataInputStream**: Đọc các byte từ luồng và chuyển thành các giá trị có kiểu nguyên thủy hoặc các xâu ký tự
- Các phương thức: `readInt, readDouble, readUTF,…`



Vào ra file kiểu nhị phân

- **DataOutputStream**: Chuyển các giá trị kiểu nguyên thủy hoặc xâu ký tự thành byte và ghi vào luồng ra
- Các phương thức: `writeInt, writeDouble, writeUTF,…`



Vào ra file kiểu nhị phân

```
// write
FileOutputStream fos = new FileOutputStream("binaryIO.dat");
DataOutputStream dos = new DataOutputStream(fos);


dos.writeInt(123);
dos.writeDouble(45.67);
dos.writeUTF("Hello, World!");
dos.close();
```

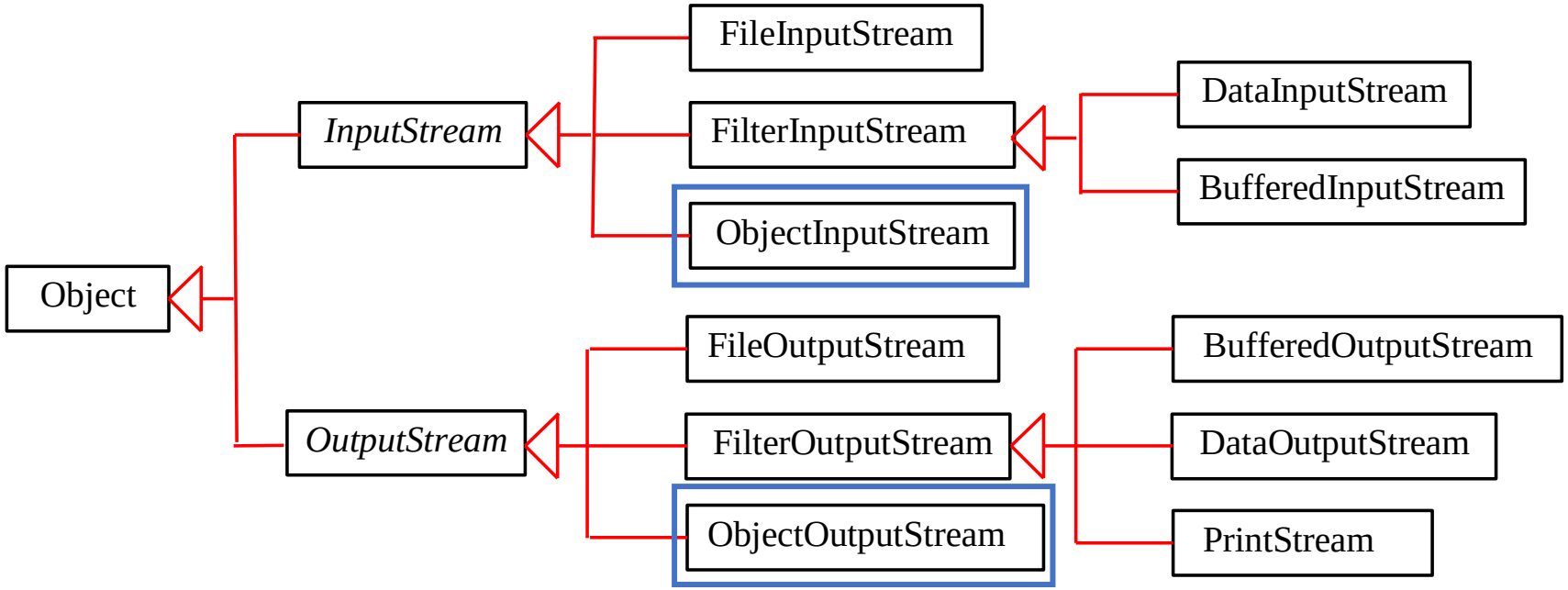
Vào ra file kiểu nhị phân

```
// read
FileInputStream fis = new FileInputStream("binaryIO.dat");
DataInputStream dis = new DataInputStream(fis);
int intValue = dis.readInt();
double doubleValue = dis.readDouble();
String stringValue = dis.readUTF();


System.out.println("Integer value: " + intValue);
System.out.println("Double value: " + doubleValue);
System.out.println("String: " + stringValue);
```

Vào ra file kiểu nhị phân

- **ObjectInputStream** và **ObjectOutputStream**: Thao tác với các đối tượng ngoài các kiểu dữ liệu nguyên thủy và xâu ký tự
- Các đối tượng phải là **Serializable**

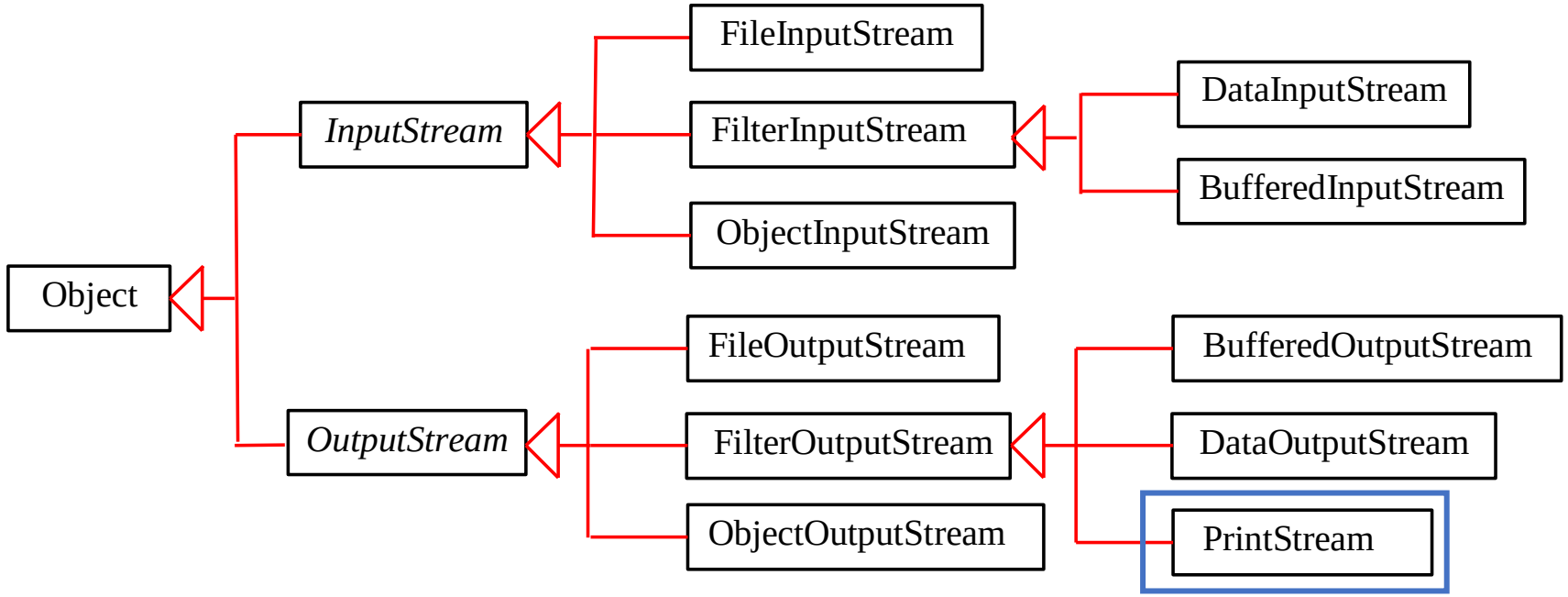


Vào ra file kiểu nhị phân

- Đối tượng được tuần tự hóa (serialize) sẽ chuyển thành các byte tuần tự và có thể được khôi phục (deserialize)
- ObjectOutputStream có thể tuần tự hóa đối tượng sử dụng phương thức `void writeObject(Object obj)`
- ObjectInputStream có thể khôi phục đối tượng đã tuần tự hóa sử dụng phương thức `<Object_type> readObject()`

Vào ra file kiểu nhị phân

- **PrintStream** cũng cho phép ghi dữ liệu ra file kiểu nhị phân
- `System.out` cũng là một đối tượng PrintStream



Tài liệu tham khảo

- Liang, Y. Daniel, Introduction to Java programming and Data Structures, 11th edition, Pearson Prentice Hall, 2019
- N. M. Sơn, Bài giảng Lập trình hướng đối tượng, HVCNBCVT, 2020
- N. M. Sơn, Slide giảng dạy môn Lập trình hướng đối tượng
- T. T. V. Anh, Slide giảng dạy môn Lập trình hướng đối tượng