Bài 11

Input và output

Trịnh Thành Trung

trungtt@soict.hust.edu.vn

Nội dung

1. Tổng quan
2. I/O với file text
3. I/O với file nhị phân
4. Một số luồng trong Java

1

Tổng quan

Mô hình input và output trong Java

Tổng quan

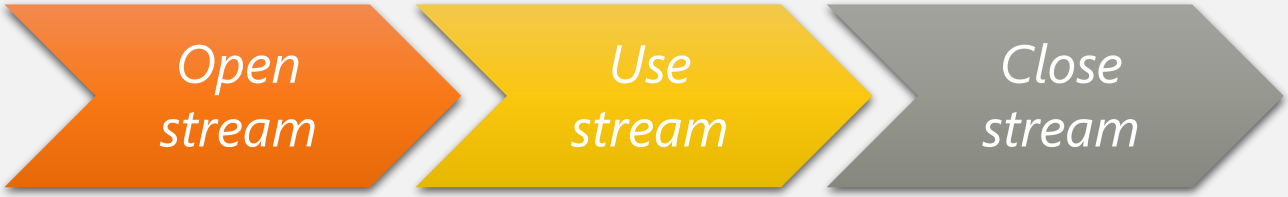
- I/O = Input/Output
- Ở đây là đưa dữ liệu vào (input) và lấy dữ liệu ra (output) từ chương trình
- Input có thể là từ bàn phím hoặc từ file
- Output có thể là ra thiết bị hiển thị (màn hình) hoặc ra file
- Ưu điểm của file I/O
  - Sao lưu trên máy
  - Output từ một chương trình có thể trở thành input cho một chương trình khác
  - Các giá trị input có thể được tự động nhập (thay vì phải gõ từng giá trị)

Luồng

- Luồng: Là một đối tượng đưa dữ liệu đến một đích đến (màn hình, file...) hoặc lấy dữ liệu từ một nguồn (bàn phím, file...)
  - Luồng hoạt động như một bộ đệm giữa nguồn dữ liệu và đích đến
  - Luồng vào - Input stream: Luồng đưa dữ liệu vào chương trình
    - System.in là input stream
  - Luồng ra - Output stream: Luồng nhận dữ liệu từ một chương trình
    - System.out là output stream
- Luồng kết nối chương trình với một đối tượng I/O
  - System.out kết nối chương trình với màn hình
  - System.in kết nối chương trình với bàn phím

Mô hình I/O

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



Mở luồng

- Sử dụng khi có dữ liệu bên ngoài chương trình mà chúng ta cần lấy, hoặc chúng ta cần đặt dữ liệu đâu đó bên ngoài chương trình
- Khi mở một luồng, chúng ta tạo ra một kết nối đến vị trị bên ngoài đó
- Khi kết nối đã được tạo, chúng ta có thể không cần quan tâm đến vị trí bên ngoài đó và chỉ cần thao tác với luồng

Ví dụ

- FileReader được sử dụng để kết nối với một file sẽ được sử dụng cho input:

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

- fileName xác định vị trí của file
- Chúng ta không cần phải sử dụng đến fileName nữa. Thay vào đó ta sử dụng fileReader

Sử dụng luồng

- Một số luồng chỉ có thể được sử dụng cho input, một số chỉ sử dụng cho output, trong khi một số có thể được sử dụng cho cả hai
- Sử dụng một luồng có nghĩa là thực hiện việc đưa dữ liệu vào hoặc lấy dữ liệu ra từ luồng đó
- Tuy nhiên, ta vẫn cần phải xử lý dữ liệu khi chúng đi vào hay đi ra theo một cách nào đó

Ví dụ

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

- `fileReader.read()`:
  - Đọc một ký tự
  - Trả ký tự về dạng **int**,
  - Trả về -1 nếu không còn ký tự để đọc

- Có thể ép kiểu int về char:

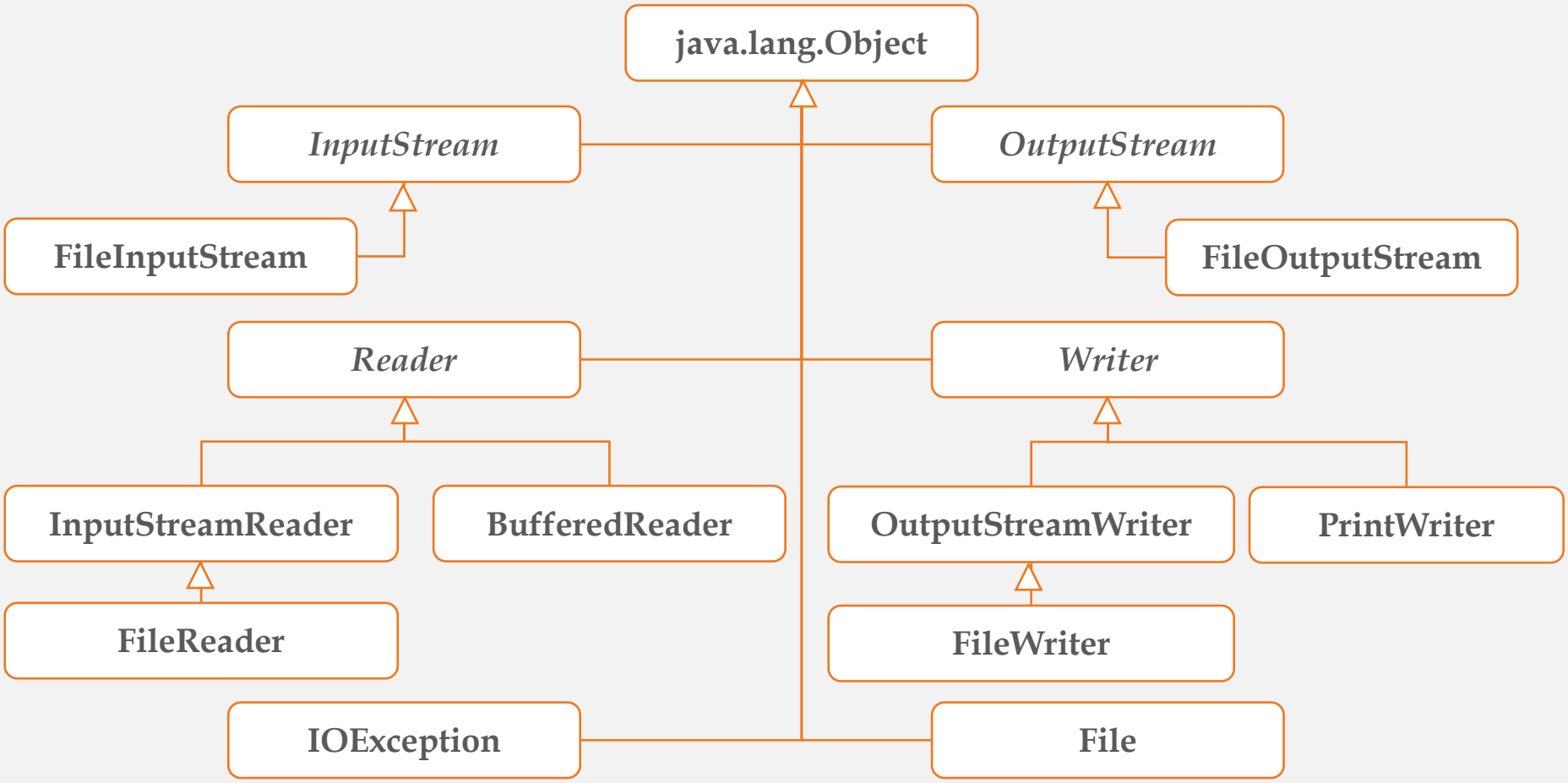
```
char ch = (char) fileReader.read( );
```

Đóng

```
bufferedReader.close( );
```

- Chúng ta không nên để hai luồng mở cùng lúc
- Cần phải đóng luồng trước khi chúng ta mở luồng trở lại
- Thông thường Java sẽ tự đóng các luồng khi chương trình kết thúc, tuy nhiên chúng ta không nên quá phụ thuộc vào điều này

Lớp xử lý io trong Java



Lớp File

- Một đối tượng File thể hiện một đường dẫn trừu tượng
  - Bao gồm cả đường dẫn và tên file
  - Phương thức khởi tạo lấy tham số là

```
File info = new File("Letter.txt");
```

  - Không ngoại lệ nào được tung ra nếu file không tồn tại
- Các thể hiện của lớp File là bất biến (immutable – một khi đã được khởi tạo thì không thể thay đổi đường dẫn file được)

Các phương thức của lớp File

```
File info = new File("Letter.txt");
if(info.exists()) {
    System.out.println("Size of " +
        info.getName()+ " is "+info.length());
    if(info.isDirectory()) {
        System.out.println("The file is a directory.");
    }
    if(info.canRead()) {
        System.out.println("The file is readable.");
    }
    if(info.canWrite()) {
        System.out.println("The file is writeable.");
    }
}
```

File text và file nhị phân

- 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 bit

File text và file nhị phân

- File nhị phân: Các bit thể hiện các kiểu khác nhau của thông tin đã được mã hóa, ví dụ các chỉ lệnh hoặc các dữ liệu số học
  - Những file này có thể dễ dàng được đọc bởi máy tính nhưng khó đọc đối với con người
  - Nhưng file này không «in» ra được (Có thể in ra được nhưng không đọc được)
- File Text: Các bit thể hiện các ký tự chữ cái
  - Mỗi chữ cái ASCII là 1 byte
  - Ví dụ: File mã nguồn Java hoặc các file tạo bởi Notepad, gedit...

File nhị phân Java

- File nhị phân Java có thể được sử dụng bởi Java trên nhiều máy khác nhau
- Đọc và viết các file nhị phân thường được thực hiện bởi chương trình
- Các file text chỉ được sử dụng để giao tiếp với con người

2

I/O với file text

FileReader và FileWriter

Xử lý file text

- Sử dụng FileReader và FileWriter
- FileReader: Luồng đọc các ký tự
  - `FileReader(String fileName)`
  - `read()`: làm việc với char và char[]
- FileWriter: Luồng ghi các ký tự
  - `FileWriter(String fileName)`
  - `write()`: làm việc với char và char[].

Mở và đóng file

```
try {
    // Try to open the file.
    FileReader inputFile = new FileReader(filename);
    // Process the file's contents.
    ...
    // Close the file now that it is finished with.
    inputFile.close();
}
catch(FileNotFoundException e) {
    System.out.println("Unable to open "+filename);
}
catch(IOException e) {
    // The file could not be read or closed.
    System.out.println("Unable to process "+filename);
}
```

Copy từng ký tự vào file

```
static void copyFile(FileReader inputFile,
                     FileWriter outputFile) {
    try{
        // Read the first character.
        int nextChar = inputFile.read();
        // Have we reached the end of file?
        while(nextChar != -1) {
            outputFile.write(nextChar);
            // Read the next character.
            nextChar = inputFile.read();
        }
        outputFile.flush();
    }
    catch(IOException e) {
        System.out.println("Unable to copy file");
    }
}
```

Copy nhiều ký tự

```
static void copyFile(FileReader inputFile,
                     FileWriter outputFile)
                                  throws IOException {
    final int bufferSize = 1024;
    char[] buffer = new char[bufferSize];
    // Read the first chunk of characters.
    int numberRead = inputFile.read(buffer);
    while(numberRead > 0) {
        // Write out what was read.
        outputFile.write(buffer,0,numberRead);
        numberRead = inputFile.read(buffer);
    }
    outputFile.flush();
}
```

Làm việc với dòng

- Trong rất nhiều trường hợp chúng ta làm việc với từng dòng thay vì làm việc với char
  - Ví dụ: Các file config.ini
- Java hỗ trợ hai lớp BufferedReader và BufferedWriter cho việc này
- Khởi tạo bằng các đối tượng FileReader và FileWriter tương ứng

Ví dụ

```
try {
    FileReader in = new FileReader(infile);
    BufferedReader reader = new BufferedReader(in);
    FileWriter out = new FileWriter(outfile);
    BufferedWriter writer = new BufferedWriter(out);
    ...
    reader.close();
    writer.close();
}
catch(FileNotFoundException e) {
    System.out.println(e.getMessage());
}
catch(IOException e) {
    System.out.println(e.getMessage());
}
```

Copy từng dòng

```
BufferedReader reader = new BufferedReader(...);
// Read the first line.
String line = reader.readLine();
// null returned on EOF.
while(line != null) {
    // Write the whole line.
    writer.write(line);
    // Add the newline character.
    writer.newLine();
    // Read the next line.
    line = reader.readLine();
}
```

PrintWriter

- In trực tiếp ra màn hình

```
try {
    FileWriter out = new FileWriter(outfile);
    PrintWriter writer = new PrintWriter(out);

    writer.println(…);
    writer.close();
}
catch(IOException e) {
    System.out.println(e.getMessage());
}
```

3

I/O với file nhị phân

`FileInputStream và FileOutputStream`

Luồng input, output nhị phân

- Sử dụng lớp FileInputStream và FileOutputStream
- FileInputStream/FileOutputStream liên kết một luồng input/output nhị phân với một file liên kết ngoài
- Tất các các phương thức trong FileInputStream và FileOutputStream đều được thừa kế từ các lớp cha

FileInputStream

- Phương thức khởi tạo

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

- Ngoại lệ java.io.FileNotFoundException có thể xảy ra nếu ta sử dụng FileInputStream với file không tồn tại

Các phương thức của FileInputStream

1. *read(): int*
2. read(b: byte[]): int
3. read(b: byte[], off: int, len: int): int
4. available(): int
5. close(): void
6. skip(n: long): long
7. markSupported(): boolean
8. mark(readlimit: int): void
9. reset(): void

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 không tồn tại thì file mới sẽ được tạo
- Nếu file tồn tại, 2 phương thức khởi tạo đầu tiên sẽ xóa nội dung hiện tại của file. Để có thể giữ lại nội dung và thêm dữ liệu vào file, ta sử dụng 2 phương thức khởi tạo ở dưới với tham số append là true

FileOutputStream

1. *write(int b): void*
2. write(b: byte[]): void
3. write(b: byte[], off: int, len: int): void
4. close(): void
5. *flush(): void*

Ví dụ FileOutputStream

```
import java.io.*;
class FileOutputStreamDemo {

  public static void main(String args[]) throws Exception {
    String source = "Now is the time for all good men\\n"
              + " to come to the aid of their country\\n"
              + " and pay their due taxes.";
    byte buf[] = source.getBytes();

    OutputStream f0 = new FileOutputStream("file1.txt");
```

Ví dụ FileOutputStream (tiếp)

```
        for (int i=0; i < buf.length; i += 2) {
            f0.write(buf[i]);
        }
        f0.close();

        OutputStream f1 = new FileOutputStream("file2.txt");
        f1.write(buf);
        f1.close();

        OutputStream f2 = new FileOutputStream("file3.txt");
        f2.write(buf,buf.length-buf.length/4,buf.length/4);
        f2.close();
    }
}
```

Ví dụ FileOutputStream (tiếp)

- **file1.txt**:
  Nwi h iefralgo e
  t oet h i ftercuty n a hi u ae.
- **file2.txt**:
  Now is the time for all good men
  to come to the aid of their country
  and pay their due taxes.
- **file3.txt**:
  nd pay their due taxes.

Làm việc với đối tượng

- Ta có thể đọc và ghi các *đối tượng* vào file
- Quá trình này được gọi là quá trình tuần tự - serialization
- Quá trình tuần tự ở các ngôn ngữ khác có thể rất khó khăn vì các đối tượng có thể chứa các tham chiếu đến các đối tượng khác. Java giúp cho quá trình này khá dễ dàng

Điều kiện cho quá trình tuần tự

- Để một đối tượng có thể được tuần tự hóa thì
  - Lớp đó phải được khai báo public
  - Thực thi giao diện Serializable
  - Phải có một phương thức khởi tạo không tham số
  - Tất cả các thành phần của lớp đó phải có thể tuần tự hóa được: Hoặc là kiểu nguyên thủy hoặc là các đối tượng Serializable

Giao diện Serializable

- Thông thường để thực thi một giao diện đồng nghĩa với việc phải viết tất cả các phương thức được khai báo bởi giao diện đó. Tuy nhiên, giao diện Serializable không định nghĩa bất kỳ phương thức nào
  - Lý do: Java sử dụng giao diện Serializable như là một flag để xử lý với các lớp

Làm việc với đối tượng

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

Ghi đối tượng ra file

```
FileOutputStream fos = new
  FileOutputStream("t.tmp");
ObjectOutputStream oos = new
  ObjectOutputStream(fos);
oos.writeObject(new Date());
oos.writeObject("Today");
oos.writeInt(12345);
oos.close();
```

Đọc đối tượng từ file

```
FileInputStream fis = new
  FileInputStream("t.tmp");
ObjectInputStream ois = new
  ObjectInputStream(fis);
Date date = (Date) ois.readObject();
String today = (String) ois.readObject();
int i = ois.readInt();
ois.close();
```

4

Một số luồng trong Java

Scanner & printf

Các luồng có sẵn trong Java

- Tất cả các chương trình Java đều tự động import gói 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

Các luồng có sẵn trong Java

- System.in là một đối tượng kiểu InputStream. (byte stream)
- System.out là một đối tượng kiểu PrintStream. (byte stream)
- System.err là một đối tượng kiểu PrintStream. (byte stream)
  - đều là luồng byte, KHÔNG PHẢI LÀ luồng character

java.util.Scanner

- Đọc input
- Đầu tiên, chúng ta cần tạo một đối tượng Scanner
  - Đọc từ bàn phím (System.in)

```
Scanner scanner = new Scanner(System.in);
```

  - Đọc từ file:

```
File myFile = new File("myFileName.txt");
Scanner scanner = new Scanner(myFile);
```

    + Ngoại lệ FileNotFound có thể được tung ra
  - Đọc một String:

```
Scanner scanner = new Scanner(myString);
```

    + Hữu ích khi cần parse một String

Sử dụng Scanner

- Đầu tiên, chúng ta phải bảo đảm Scanner có dữ liệu để đọc
  - `scanner.hasNext()` → boolean
  - Không cần phải dùng nếu chúng ta sử dụng bàn phím
- Có thể đọc từng dòng một
  - `scanner.nextLine()` → String
- Có thể đọc từng "khối"
  - Một khối là một chuỗi các ký tự không chứa khoảng trắng
  - `scanner.next ()` → String
- nextLine và next trả về String, chúng ta cần phải convert về số hoặc về kiểu tương ứng
- Bên cạnh đó cũng có những phương thức trả trực tiếp về các kiểu nguyên thủy

Sử dụng Scanner

- Các phương thức trả về các kiểu nguyên thủy
  - `boolean b = sc.nextBoolean();`
  - `byte by = sc.nextByte();`
  - `short sh = sc.nextShort();`
  - `int i = sc.nextInt();`
  - `long l = sc.nextLong();`
  - `float f = sc.nextFloat();`
  - `double d = sc.nextDouble();`
- Các phương thức kiểm tra
  - `hasNextBoolean()`
  - `hasNextByte()`
  - `hasNextShort()`
  - `hasNextInt()`
  - `hasNextLong()`
  - `hasNextFloat()`
  - `hasNextDouble()`

Định dạng output

- Java có phương thức printf tương tự C
- Các định dạng sử dụng ký tự % với
- s cho string, d cho integer, f cho số floating point
- Ví dụ:
  - ```
    double pi = Math.PI;
    System.out.printf("%8s %-8s %6d %-6d %8f %-8.2f :)\n",
                      "abc", "def", 123, 456, pi, pi);
    System.out.printf("%8s %-8s %6d %-6d",
                      "abcdef", "ghijkl", 12345, 6789);
    ```
  - Output:
    ```
    abc def          123 456    3.141593 3.14     :)
    abcdef ghijkl   12345 6789
    ```
    ```

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: BufferredReader và BufferredWriter (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

Thank you!

Any questions?