Lập trình Java cơ bản

Cao Đức Thông - Trần Minh Tuấn
cdthong@ifi.edu.vn, tmtuan@ifi.edu.vn

Bài 7. Luồng và xử lý file

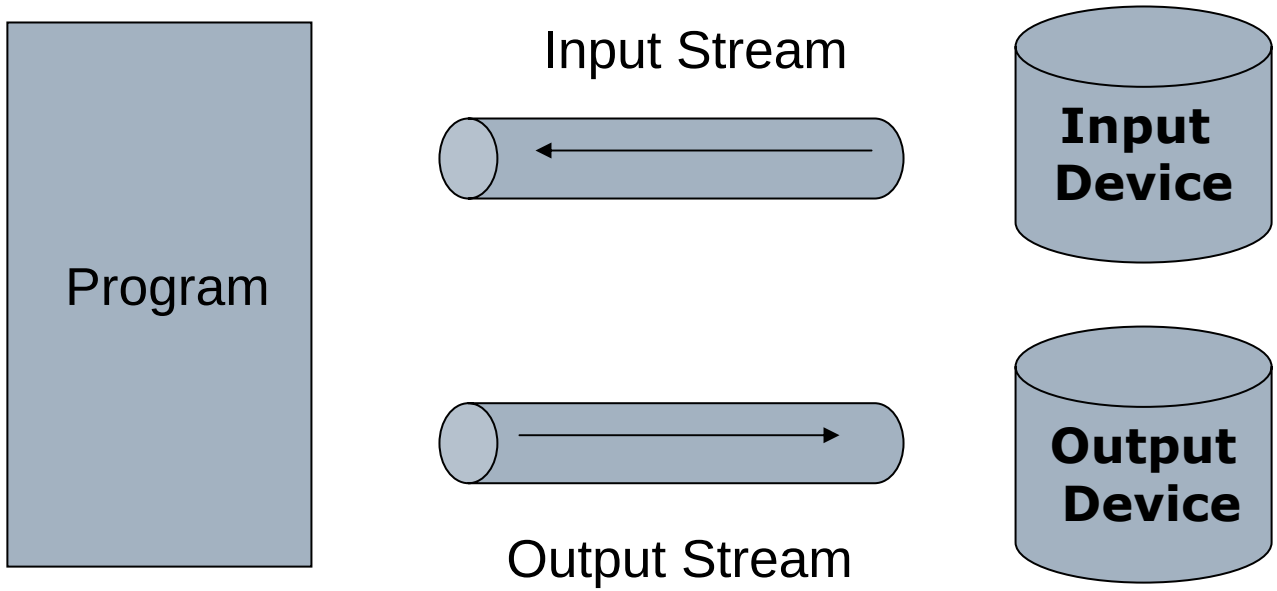
- Khái niệm luồng
- Các luồng byte
- Đối tượng serializable
- Các luồng ký tự
- File truy cập ngẫu nhiên
- Lớp File
- Bài tập

Khái niệm luồng (stream)

- Luồng là một "dòng chảy" của dữ liệu được gắn với các thiết bị vào ra.
- Hai loại luồng:
  - Luồng nhập: Gắn với các thiết bị nhập như bàn phím, máy scan, file...
  - Luồng xuất: Gắn với các thiết bị xuất như màn hình, máy in, file...
- Việc xử lý vào ra thông qua luồng giúp cho lập trình viên không phải quan tâm đến bản chất của thiết bị vào ra.

Khái niệm luồng (stream)

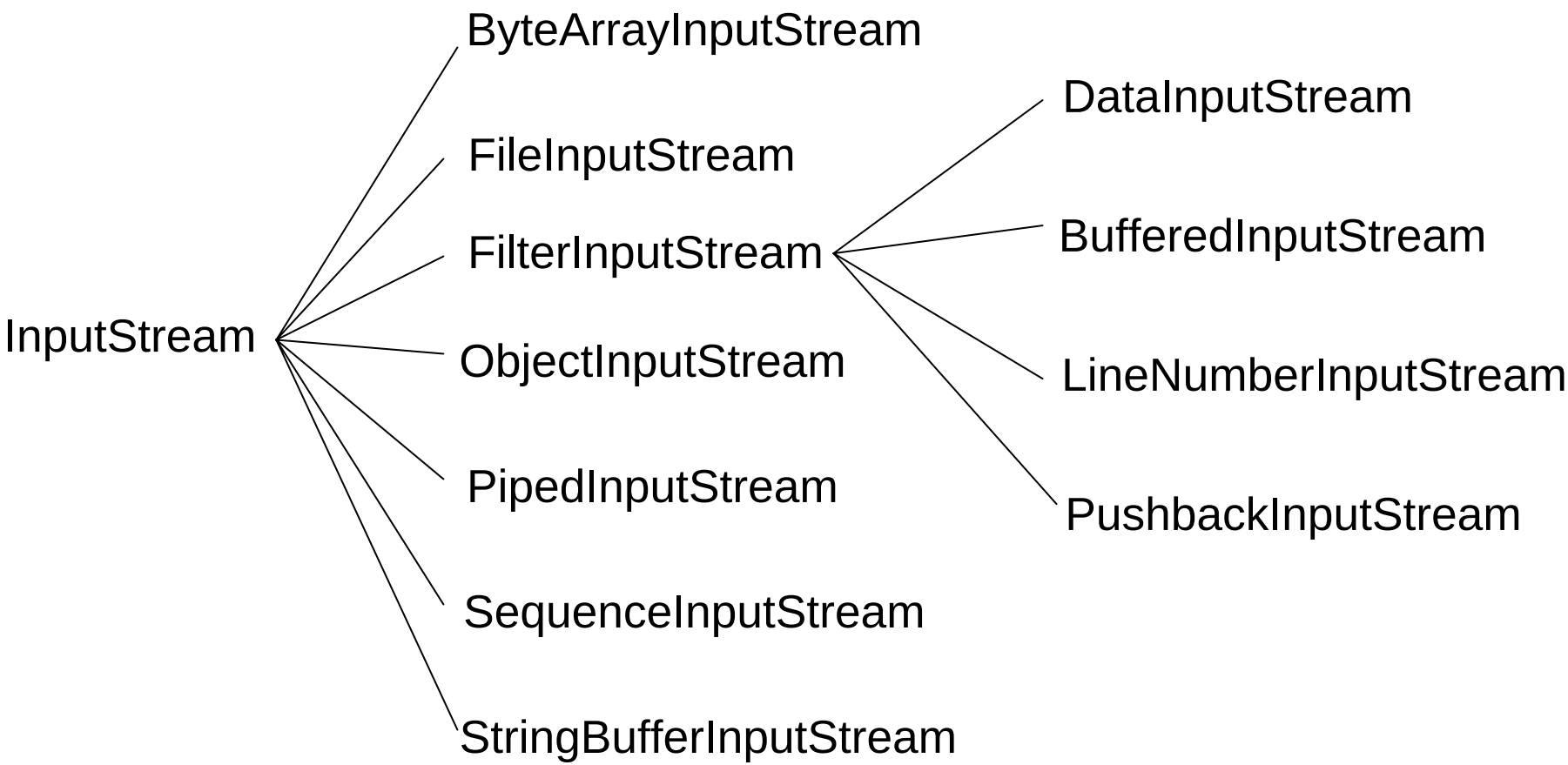
- Chương trình đọc trên luồng nhập để lấy dữ liệu từ thiết bị nhập, ghi vào luồng xuất để đưa dữ liệu ra thiết bị xuất



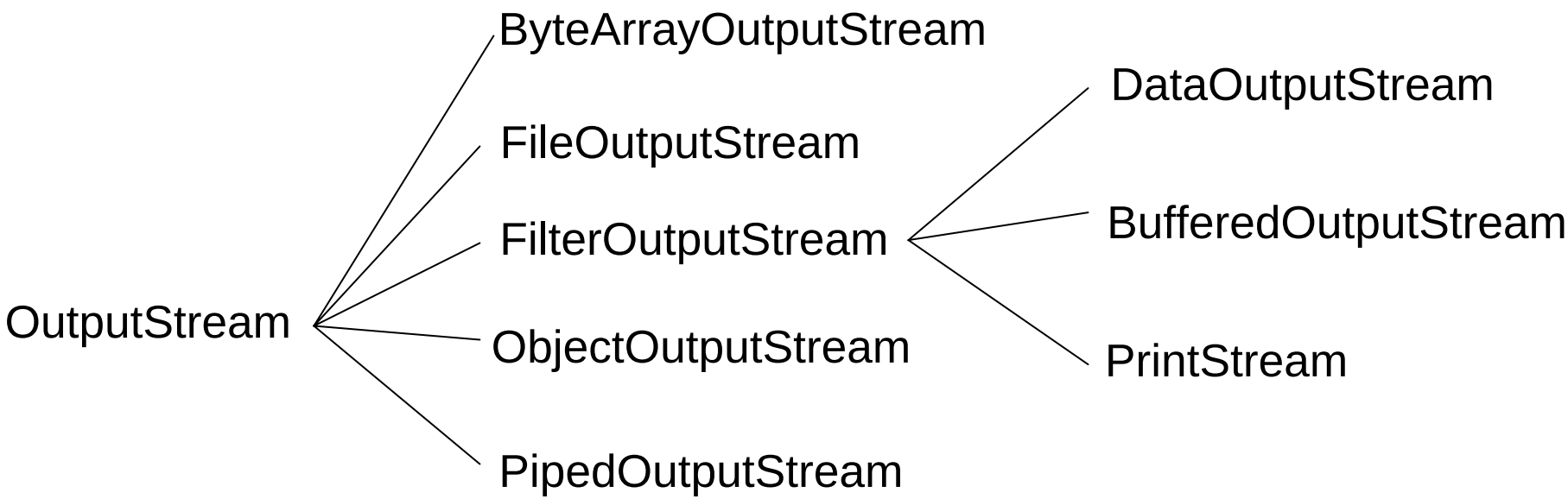
Các luồng cơ bản

- Luồng byte: Là dòng chảy các byte
  - InputStream: Luồng nhập byte cơ bản
  - OutputStream: Luồng xuất byte cơ bản
- Luồng ký tự: Là dòng chảy các ký tự (char)
  - Reader: Luồng nhập ký tự cơ bản
  - Writer: Luồng xuất ký tự cơ bản
- Các lớp luồng nằm trong gói java.io

Luồng byte

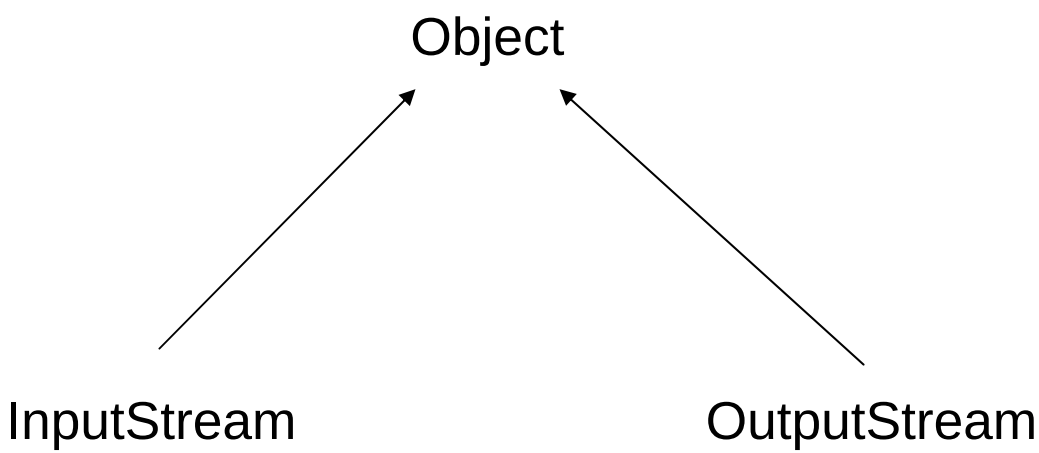

InputStream
ByteArrayInputStream
FileInputStream
FilterInputStream
ObjectInputStream
PipedInputStream
SequenceInputStream
StringBufferInputStream
DataInputStream
BufferedInputStream
LineNumberInputStream
PushbackInputStream

Luồng byte


OutputStream
ByteArrayOutputStream
FileOutputStream
FilterOutputStream
ObjectOutputStream
PipedOutputStream
DataOutputStream
BufferedOutputStream
PrintStream

Luồng nhập/xuất byte cơ bản

- InputStream và OutputStream là hai lớp gốc của mọi luồng nhập/xuất byte (abstract).



Lớp InputStream

- Một số phương thức của InputStream
  - abstract int read() throws IOException
    - Đọc một byte từ luồng.
    - Nếu cuối luồng sẽ trả về -1
  - int read(byte[] b) throws IOException
    - Đọc một dãy byte từ luồng
  - void close() throws IOException
    - Đóng luồng nhập
  - int available() throws IOException
    - Trả về số byte có thể đọc tiếp
  - long skip(long n) throws IOException
    - Bỏ qua n byte

Lớp OutputStream

- Một số phương thức của OutputStream
  - abstract void write(int b) throws IOException
    - Ghi một byte ra luồng
  - void write(byte[] b) throws IOException
    - Ghi một dãy byte ra luồng
  - void close() throws IOException
    - Đóng luồng
  - void flush() throws IOException
    - Dồn xuất luồng

Các luồng file

- Được sử dụng để xuất nhập với file.
- Luồng nhập từ file: FileInputStream
  - FileInputStream(String name)
  - FileInputStream(File f)
- Luồng xuất ra file: FileOutputStream
  - FileOutputStream(String name)
  - FileOutputStream(File f)
  - FileOutputStream(String name, boolean append)
- Phương thức nhập/xuất của các luồng file giống như của các luồng nhập xuất cơ bản

Ví dụ: Đọc và hiển thị file (v1)

```
import java.io.*;

public class ReadFile
{
    public static void main(String[] args)
    {
        try {
            FileInputStream f = new FileInputStream("readme.txt");
            int ch;
            while ( (ch = f.read()) != -1 )
            {
                System.out.print((char)ch);
            }
            f.close();
        } catch (FileNotFoundException d) {
            System.out.println("File not found");
        } catch (IOException d) {
            System.out.println("Can not read file");
        }
    }
}
```

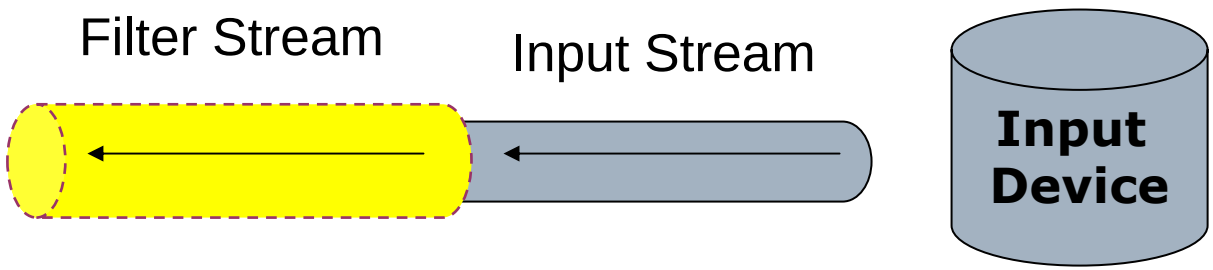
Ví dụ: Ghi dữ liệu ra file

```
import java.io.*;

public class WriteFile
{
    public static void main(String[] args)
    {
        byte buffer[] = new byte[80];
        try {
                System.out.println("Enter a string to write to file: ");
                int num = System.in.read(buffer);
                FileOutputStream f = new FileOutputStream("line.txt");
                f.write(buffer, 0, num);
                f.close();
        } catch (IOException e) {
            System.out.println("Error IO file");
        }
    }
}
```

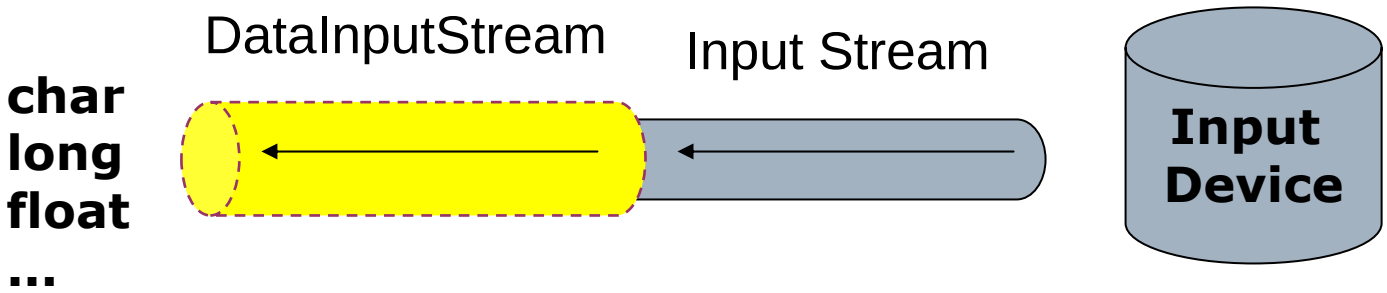
Luồng lọc (filter stream)

- Luồng lọc có khả năng kết nối với các luồng khác và xử lý dữ liệu "theo cách riêng" của nó.
- FilterInputStream và FilterOutputStream là 2 lớp luồng lọc cơ bản.



Luồng nhập/xuất dữ liệu sơ cấp

- DataInputStream và DataOutputStream là 2 lớp lọc cho phép nhập xuất dữ liệu thuộc các kiểu sơ cấp.



Luồng nhập/xuất dữ liệu sơ cấp

- Một số phương thức của DataInputStream
  - float readFloat() throws IOException
  - int readInt() throws IOException
  - long readLong() throws IOException
  - String readUTF() throws IOException
- Một số phương thức của DataOutputStream
  - void writeFloat(float v) throws IOException
  - void writeInt(int b) throws IOException
  - void writeLong(long v) throws IOException
  - void writeUTF(String s) throws IOException
  - …

Ví dụ: Tạo file các số ngẫu nhiên

```
try {
        FileOutputStream f = new FileOutputStream("randnum.dat");
        DataOutputStream outFile = new DataOutputStream(f);
        for(int i = 0; i < 20; i++)
            outFile.writeInt( (int) (Math.random()*1000) );
        outFile.close();
} catch (IOException e) { ... }
```

```
try {
        FileInputStream g = new FileInputStream("randnum.dat");
        DataInputStream inFile = new DataInputStream(g);
        int num;
        while (true)
        {
            num = inFile.readInt();
            System.out.println("num = " + num);
        }
} catch (EOFException e) {
    System.out.println("End of file");
} catch (IOException e) { ... }
```

Luồng đệm (buffered stream)

- Luồng đệm giúp giảm bớt số lần đọc ghi dữ liệu trên thiết bị vào ra, tăng tốc độ nhập/xuất.
- Các lớp luồng đệm
  - BufferedInputStream (đệm nhập)
  - BufferedOutputStream (đệm xuất)

Ví dụ: Đọc và hiển thị file (v2)

```
// version 2 này có thể đem lại hiệu quả đáng kể hơn version 1 trên
// những file có kích thước lớn
try
{
    FileInputStream f = new FileInputStream("readme.txt");
    BufferedInputStream inFile = new BufferedInputStream(f);

    int ch;
    while ( (ch = inFile.read()) != -1 )
    {
        System.out.print((char)ch);
    }
} catch (IOException e) {
    System.out.println("Error IO file");
}
```

Ghép nối nhiều luồng

- Có thể dùng luồng lọc để ghép nối nhiều luồng với nhau.
- Ví dụ:

```
FileInputStream fStream = new FileInputStream("data.dat");
BufferedInputStream bStream = new BufferedInputStream(fStream);
DataInputStream dStream = new DataInputStream(bStream);
...
dStream.close();
```

System.in và System.out

- System.in
  - Đối tượng nhập chuẩn, gắn với bàn phím.
  - Thuộc lớp InputStream.
- System.out
  - Đổi tượng xuất chuẩn, gắn với màn hình.
  - Thuộc lớp PrintStream.
- Lớp PrintStream
  - Cho phép hiển thị biểu diễn của dữ liệu.
  - PrintStream kế thừa từ FilterOutputStream

Lớp PrintStream

```
// Chương trình này đọc một file các số thực và ghi vào file khác //
dưới dạng văn bản
try {
    FileInputStream f = new FileInputStream("float.dat");
    DataInputStream inStream = new DataInputStream(f);

    FileOutputStream ff = new FileOutputStream("ketqua.txt");
    PrintStream pStream = new PrintStream(ff);
    int count = 0;
    while (true)
    {
        float num = inStream.readFloat();
        count++;
        pStream.println("Float " + count + " = " + num);
    }
    inStream.close();
    pStream.close();
} catch (EOFException e) {
    System.out.println("End of file");
} catch (IOException e) {...}
```

Bài tập tại lớp

- Bài 1: Viết chương trình đếm xem trong một file văn bản cho trước có bao nhiêu câu. Biết rằng các câu kết thúc bởi dấu chấm.
- Bài 2: Viết chương trình tạo file ghi 100 số Fibonacci đầu tiên. Viết chương trình thứ hai để đọc và hiển thị dữ liệu từ file này.

Luồng nhập/xuất đối tượng

- Để lưu lại một đối tượng, ta có thể lưu lần lượt từng thuộc tính của nó. Khi đọc lại đối tượng ta phải tạo đối tượng mới từ các thuộc tính đã ghi.

  => Dài dòng, kém linh hoạt.
- Java hỗ trợ đọc/ghi các đối tượng một cách đơn giản thông qua lớp ObjectInputStream và ObjectOutputStream.
- Một đối tượng muốn có thể được đọc/ghi phải cài đặt giao tiếp java.io.Serializable

Ví dụ: Ghi lại tên và ngày sinh

```
try
{
    FileOutputStream f = new FileOutputStream("birthfile.dat");
    ObjectOutputStream oStream = new ObjectOutputStream(f);

    String babyName = "Briney Spears";
    Date today = new Date();
    oStream.writeObject(babyName);
    oStream.writeObject(today);

    oStream.close();
} catch (IOException e) {
    System.out.println("Error IO file");
}
```

Ví dụ: Đọc tên và ngày sinh

```
try
{
    FileInputStream f = new FileInputStream("birthfile.dat");
    ObjectInputStream inStream = new ObjectInputStream(f);

    String name = (String) inStream.readObject();
    Date birthDate = (Date) inStream.readObject();

    System.out.println("Name of baby: " + name);
    System.out.println("Birth date: " + birthDate);

    inStream.close();
} catch (IOException e) {
    System.out.println("Error IO file");
} catch (ClassNotFoundException e) {
    System.out.println("Class of serialized object not found");
}
```

Đọc/ghi đối tượng tự tạo

```
// file Student.java
public class Student implements Serializable
{
    private String name;
    private int age;

    Student(String name, int age)
    {
        this.name = name;
        this.age = age;
    }

    public String toString()
    {
        String ret = "My name is " + name +
                     "\nI am " + age + " years old";
        return ret;
    }
}
```

Đọc/ghi đối tượng tự tạo

```
// file WriteMyObject.java
import java.io.*;

public class WriteMyObject
{
    public static void main(String[] args)
    {
        try
        {
            FileOutputStream f = new FileOutputStream("student.dat");
            ObjectOutputStream oStream = new ObjectOutputStream(f);
            Student x = new Student("Bill Gates", 18);
            oStream.writeObject(x);
            oStream.close();
        } catch (IOException e) {
            System.out.println("Error IO file");
        }
```

Đọc/ghi đối tượng tự tạo

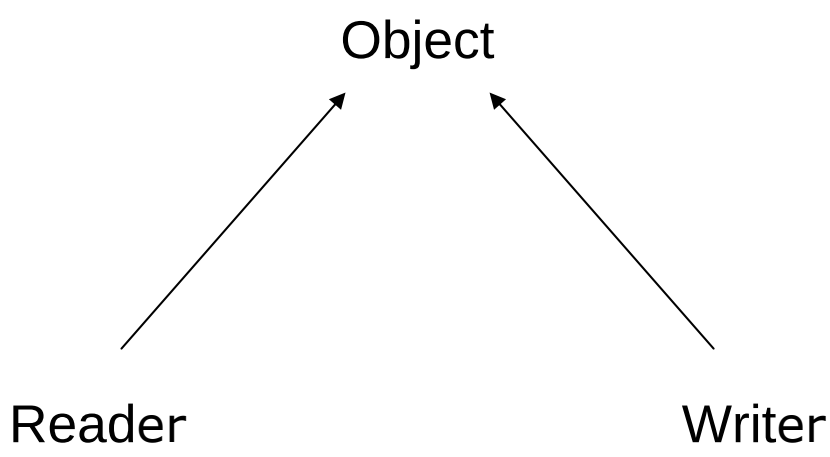
```
    try
    {
        FileInputStream g = new FileInputStream("student.dat");
        ObjectInputStream inStream = new ObjectInputStream(g);
        Student y = (Student) inStream.readObject();
        System.out.println(y.toString());
        inStream.close();
    } catch (ClassNotFoundException e) {
        System.out.println("Class not found");
    } catch (IOException e) {
        System.out.println("Error IO file");
    }
  }
}
```

Đọc/ghi đối tượng tự tạo

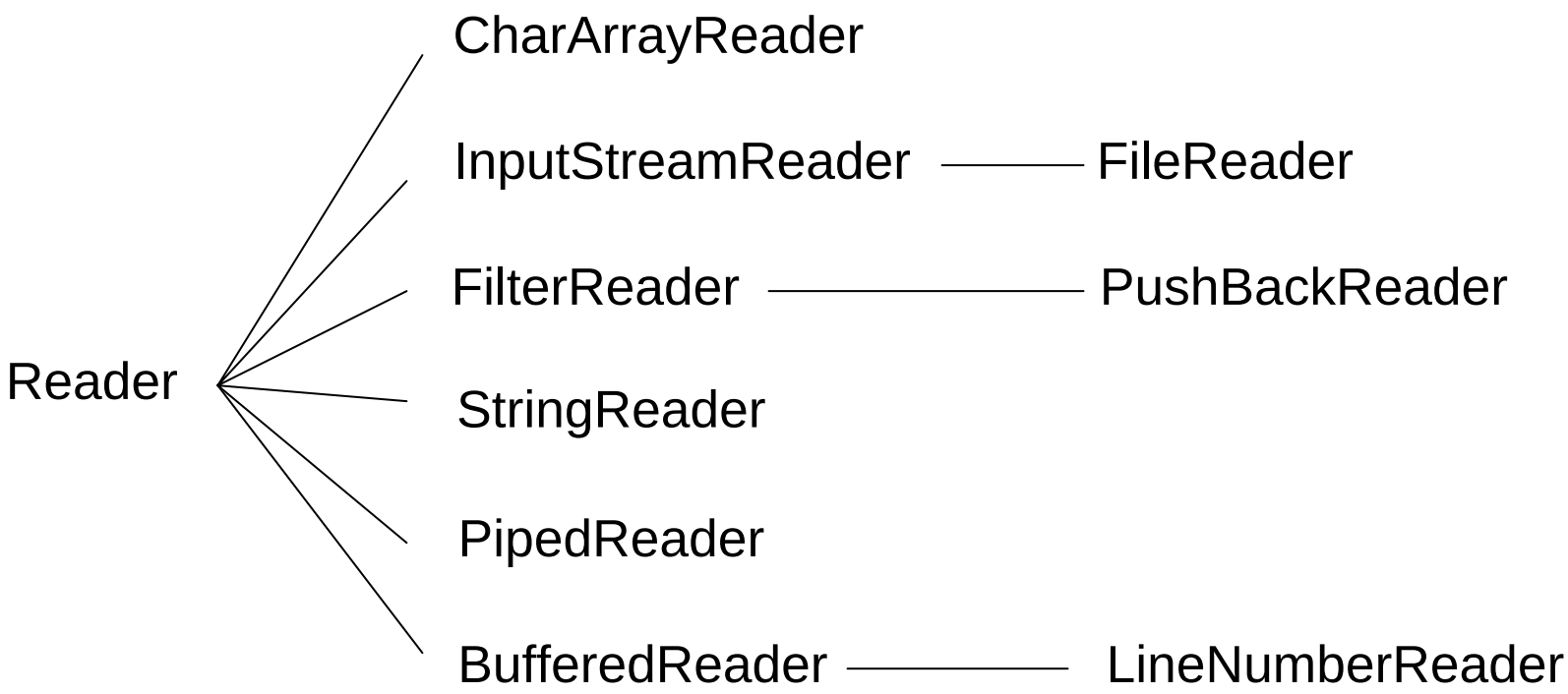
- Đối tượng có thể cài đặt 2 phương thức sau để thực hiện đọc/ghi theo cách riêng của mình.
  - private void writeObject(ObjectOutputStream out) throws IOException
  - private void readObject(ObjectInputStream in) throws IOException, ClassNotFoundException

Luồng ký tự

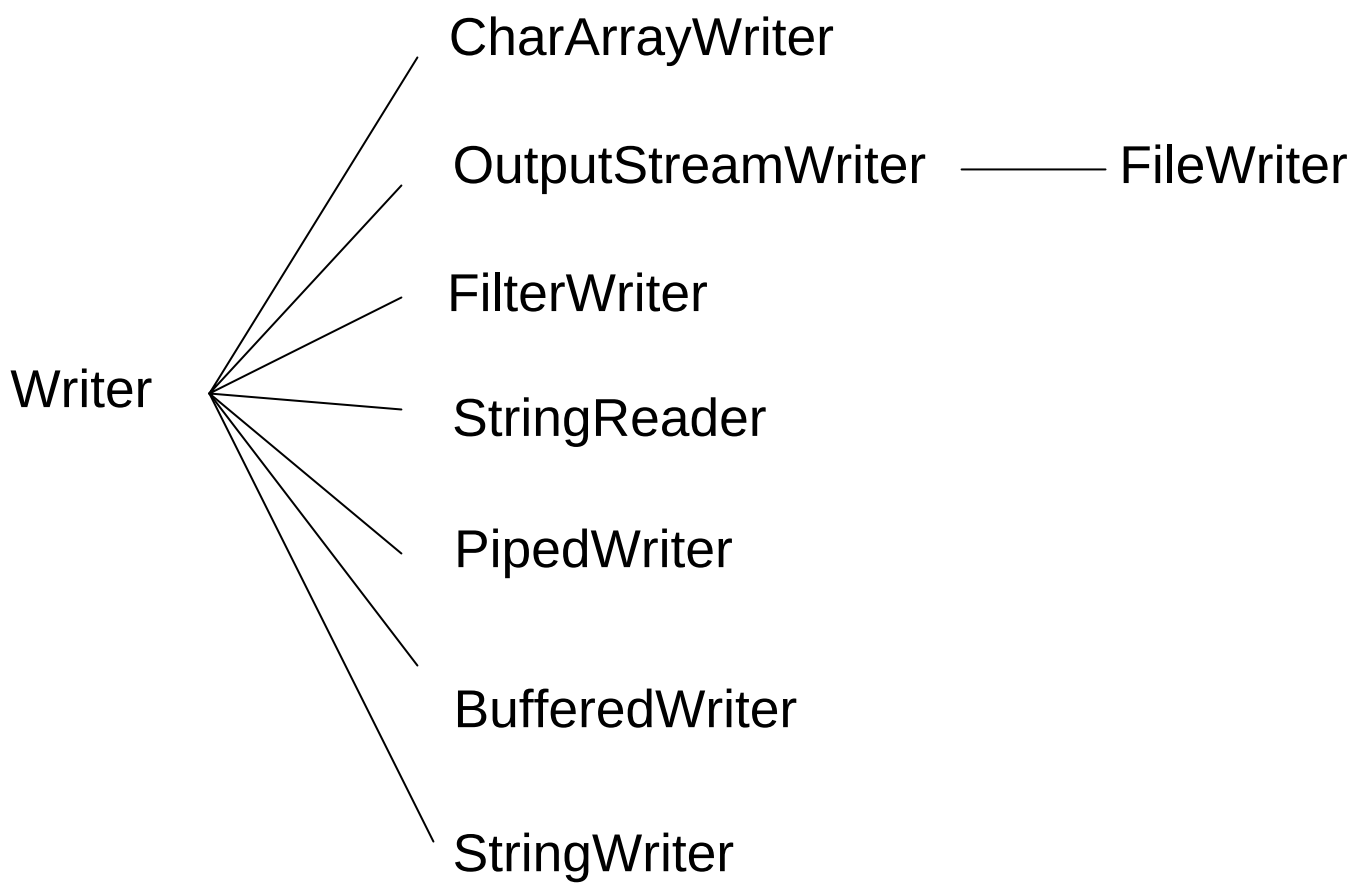
- Từ Jdk 1.1, Java hỗ trợ đọc và thao tác trên luồng đối với các ký tự Unicode (2 byte).
- Luồng ký tự cơ bản
  - Reader (đọc)
  - Writer (ghi)



Luồng ký tự


Reader
CharArrayReader
InputStreamReader
FileReader
FilterReader
PushBackReader
StringReader
PipedReader
BufferedReader
LineNumberReader

Luồng ký tự


Writer
CharArrayWriter
OutputStreamWriter
FileWriter
FilterWriter
StringReader
PipedWriter
BufferedWriter
StringWriter

Kết nối luồng byte và luồng kí tự

- Có thể chuyển từ luồng byte sang luồng ký tự nhờ các lớp
  - InputStreamReader
  - OutputStreamReader
- Ví dụ:
  - BufferedReader buff = new BufferedReader(new InputStreamReader(System.in));
  - String s = buff.readLine();
  - ...

Luồng kí tự và Unicode

- Unicode Encoding
  - UTF-8
  - UTF-16
- Đọc/ghi file với các ký tự Unicode
  - Kết nối từ luồng FileInputStream/FileOutputStream vào InputStreamReader/OutputStreamWriter (có thể vào tiếp BufferedReader/BufferedWriter)
  - Chỉ rõ cách encoding

Ví dụ: Ghi file Unicode

```
try
{
    OutputStreamWriter buff = new OutputStreamWriter(new
                        FileOutputStream("unicode.txt"), "UTF-16");
    buff.write('ồ');
    buff.write('à');
    String s = "\r\nvậy hả";
    buff.write(s, 0, s.length());
    buff.close();
} catch (IOException e) {
    System.out.println("Error IO file");
}
```

Ví dụ: Đọc file Unicode

```
try
{
    InputStreamReader buff = new InputStreamReader(new
                        FileInputStream("unicode.txt"), "UTF-16");
    int ch;
    while ( (ch = buff.read()) != -1)
    {
        System.out.print((char)ch);
        // Ở chế độ console sẽ không hiển thị được ký tự có
        // dấu, nên hiển thị trong TextField hoặc TextArea
    }
    buff.close();
} catch (IOException e) {
    System.out.println("Error IO file");
}
```

Chú ý khi soạn thảo mã

- Muốn đưa trực tiếp tiếng Việt Unicode vào cùng các đoạn mã Java cần phải sử dụng Notepad hoặc các phần mềm hỗ trợ soạn thảo tiếng Việt.
- Các bước cần thực hiện
  - Lưu file source code dưới dạng Unicode
  - Gõ lệnh biên dịch

    javac –encoding unicode filename.java
  - Lệnh thông dịch

    java filename *(như bình thường)*

File truy nhập ngẫu nhiên

- Hai hạn chế của việc xử lý file thông qua luồng
  - Không thể đọc và ghi file cùng một lúc
  - Truy nhập file mang tính tuần tự
- Java hỗ trợ việc truy nhập và xử lý file một cách tự do thông qua lớp RandomAccessFile.

File truy nhập ngẫu nhiên

- Các phương thức cơ bản
  - RandomAccessFile(String name, String mode) // cấu tử, trong đó mode có thể là “r”, “w”, “rw”
  - int readInt(); // đọc số nguyên
  - void writeInt(int v); // ghi số nguyên
  - long readLong(); // đọc số long
  - void writeLong(long v); // ghi số long
  - void seek(long pos); // di chuyển vị trí con trỏ file
  - long getFilePointer(); // lấy vị trí của con trỏ file
  - long length(); // lấy kích cỡ của file
  - void close(); // đóng file
  - ...

Ví dụ với RandomAccessFile

```
try
{
   RandomAccessFile f = new RandomAccessFile("randfile.dat","rw");
   f.writeBoolean(true);
   f.writeInt(123456);
   f.writeChar('j');
   f.writeDouble(1234.56);
   f.seek(1);
   System.out.println(f.readInt());
   System.out.println(f.readChar());
   System.out.println(f.readDouble());
   f.seek(0);
   System.out.println(f.readBoolean());
   f.close();
} catch (IOException e) {
   System.out.println("Error IO file");
}
```

Kết quả

123456
j
1234.56
true

Chú ý khi đóng file

- Nếu để lệnh f.close() trong khối try thì có thể lệnh này sẽ không được thực hiện khi có lỗi ở các lệnh phía trên.
- Có thể viết lại như sau:

Chú ý khi đóng file

```
FileInputStream f = null;
try {
    f = new FileInputStream("somefile.txt");
    // đọc file
} catch (IOException e) {
    // hiển thị lỗi
} finally {
    if (f != null) {
        try {
            f.close(); // đóng file
        } catch (Exception e) {
            // thông báo lỗi khi đóng file
        }
    }
}
```

Lớp File

- Lớp File cho phép lấy thông tin về file và thư mục.
- Một số phương thức của File
  - boolean exists(); // kiểm tra sự tồn tại của file
  - boolean isDirectory(); // kiểm tra xem file có phải là thư mục
  - String getParent(); // lấy thư mục cha
  - long length(); // lấy cỡ file (byte)
  - long lastModified(); // lấy ngày sửa file gần nhất
  - String[] list(); // lấy nội dung của thư mục

Ví dụ: Hiển thị thông tin file

```
import java.io.*;
import java.util.Date;

public class FileInfo
{
    public static void main(String[] args)
    {
        File file = new File("randfile.dat");
        if ( file.exists() )
        {
            System.out.println("Path is: " + file.getAbsolutePath());
            System.out.println("It's size is: " + file.length());
            Date dateModified = new Date(file.lastModified());
            System.out.println("Last update is: " + dateModified);
        }
        else
           System.out.println("The file does not exist");
    }
}
```

Ví dụ: Hiện nội dung thư mục

```
import java.io.*;

public class DirList
{
    public static void main(String[] args)
    {
        File dir = new File(".", "");
        if ( dir.isDirectory() )
        {
            String[] subFiles = dir.list();
            for(int i=0; i < subFiles.length; i++)
                if (new File(subFiles[i]).isDirectory())
                    System.out.println(subFiles[i] + " <DIR>");
                else
                    System.out.println(subFiles[i]);
        }
        else
            System.out.println("The file is not a directory");
    }
}
```

Tóm tắt về xử lý file

- Nên dùng DataInputStream và DataOutputStream để nhập/xuất các dữ liệu kiểu sơ cấp (int, float...)
- Nên dùng ObjectInputStream và ObjectOutputStream để nhập/xuất các đối tượng.
- Nên kết hợp luồng file và luồng đọc/ghi ký tự để nhập xuất các file ký tự Unicode.
- Nên dùng RandomAccessFile nếu muốn đọc/ghi tự do trên file.
- Dùng lớp File để lấy thông tin về file

Một số lớp khác

- java.io.StreamTokenizer
- java.io.FilenameFilter
- java.awt.FileDialog
- javax.swing.JFileChooser
- ...

Bài tập

1. Viết chương trình mycopy sử dụng như sau: *java mycopy filename1 filename2*
   - Nếu filename1 và filename2 là 2 file thì chương trình copy nội dung của filename1 sang filename2
   - Nếu filename2 là thư mục thì copy filename1 sang thư mục filename2
   - Nếu filename1 có tên là **con** thì cho phép tạo filename2 với nội dung gõ từ bàn phím (giống lệnh **copy con**)
2. Viết chương trình mydir sử dụng như sau: *java mydir filename*. Chương trình có chức năng giống lệnh **dir** của DOS.

Bài tập

3. Viết chương trình cho phép người dùng chọn một file văn bản, sau đó hiển thị nội dung của file này trong một đối tượng TextArea. (Dùng lớp JFileChooser để mở hộp thoại chọn file).
4. Viết chương trình đọc cấu trúc của một ảnh bitmap và hiển thị ra màn hình. Tham khảo cấu trúc ảnh bitmap trên Internet.

Bài tập

5. Viết chương trình quản lý một danh sách thí sinh (Candidate). Chương trình cho phép thêm thí sinh, tìm kiếm, cập nhật... Khi bắt đầu, chương trình sẽ lấy dữ liệu từ file *thisinh.dat*. Khi kết thúc, chương trình ghi lại danh sách sinh viên vào file. Có thể dùng RandomAccessFile hoặc dùng ObjectOutputStream và cài đặt Serializable.