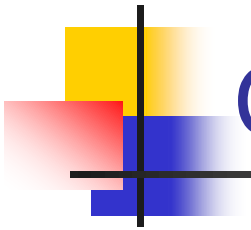
CÔNG NGHỆ JAVA

Trường: Đại học Giao thông vận Tải

Khoa: Công nghệ thông tin



CÔNG NGHỆ JAVA

Chương 6:

Vào ra dữ liệu trong Java

Nội dung

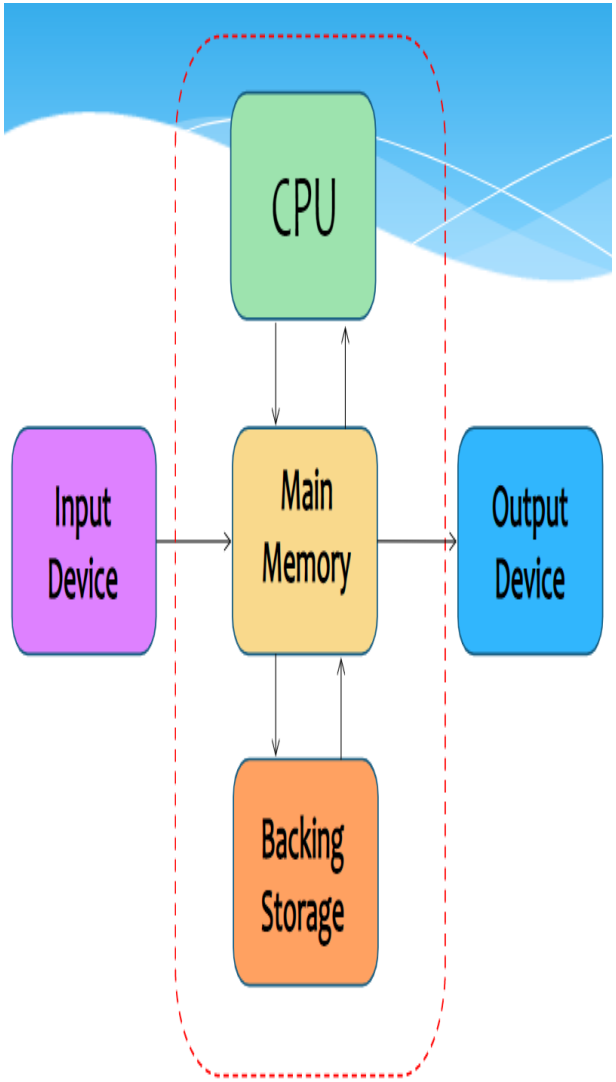
- Nội dung
  - Khái niệm về vào ra dữ liệu
    - Khái niệm
    - Dòng dữ liệu
    - Mô hình tổng thể vào ra dữ liệu
  - Làm việc với tệp tin – File
  - Vào ra dữ liệu trong Java
    - Các dòng dữ liệu trong Java
    - Vào ra dữ liệu với dòng bytes
      - InputStream/OutputStream
      - Object
    - Vào ra dữ liệu với dòng characters
      - Reader/Writer

Chương 6:

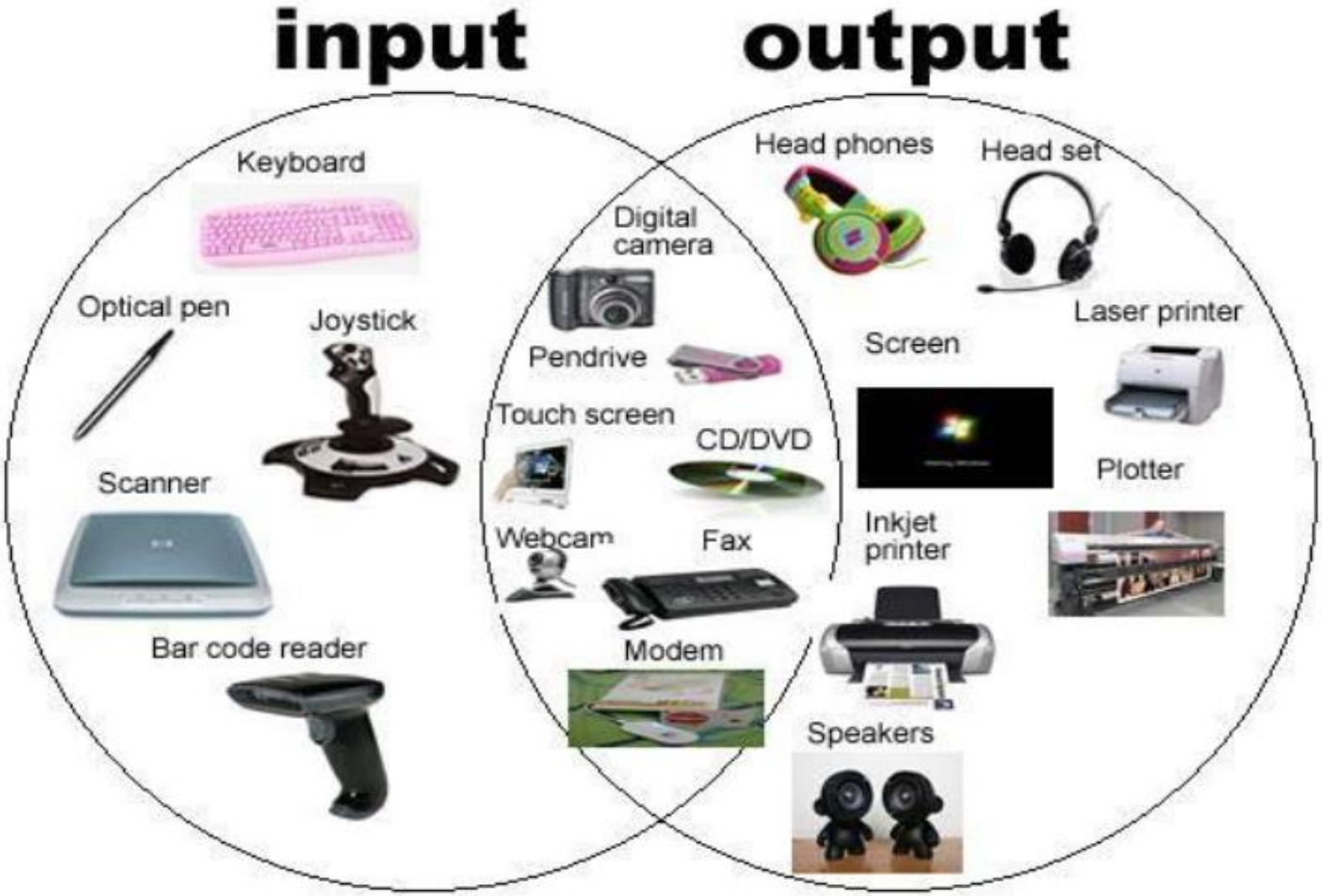
1. Khái niệm về vào ra dữ liệu

Vào ra dữ liệu

- Nhập/xuất dữ liệu là cách thức mà chương trình tương tác với người dùng và không thể thiếu trong đa số các ứng dụng.
- Nhập dữ liệu là tác vụ đưa các dữ liệu cụ thể vào cho biến trong chương trình. Như vậy, phải có một nguồn chứa dữ liệu: thiết bị vào – bàn phím, tệp tin, cơ sở dữ liệu …
- Xuất dữ liệu là tác vụ đưa giá trị cụ thể của biến trong chương trình ra một nơi chứa: thiết bị ra – màn hình, tệp tin, loa, máy in …

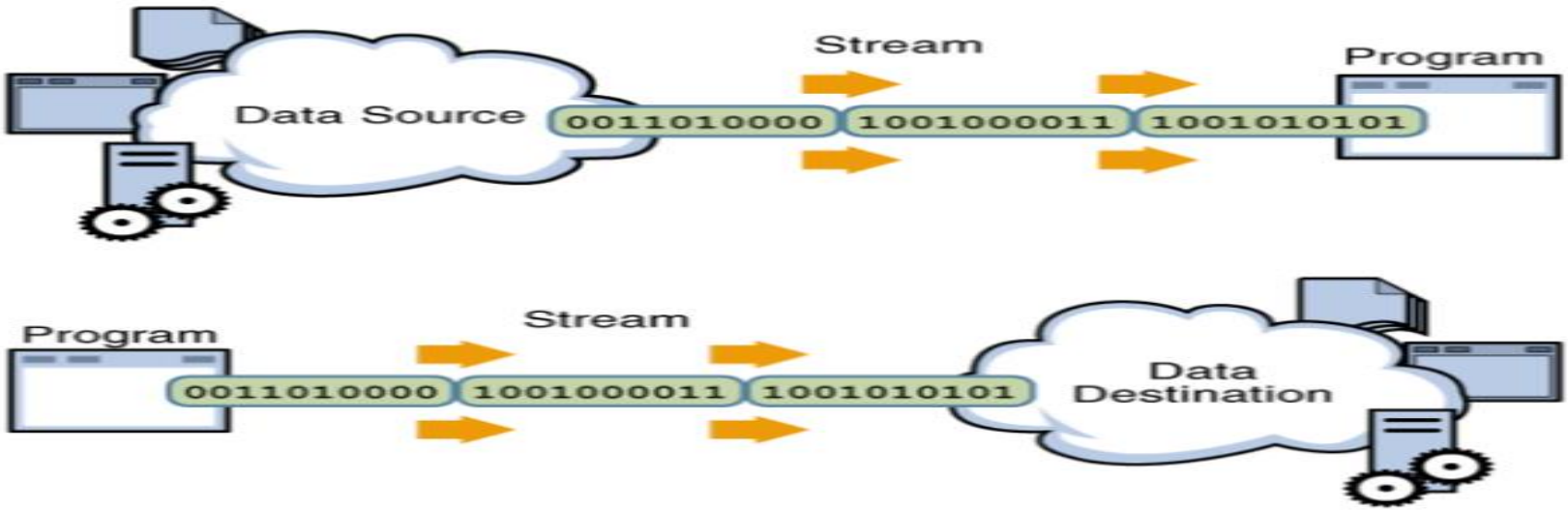


Vào ra dữ liệu

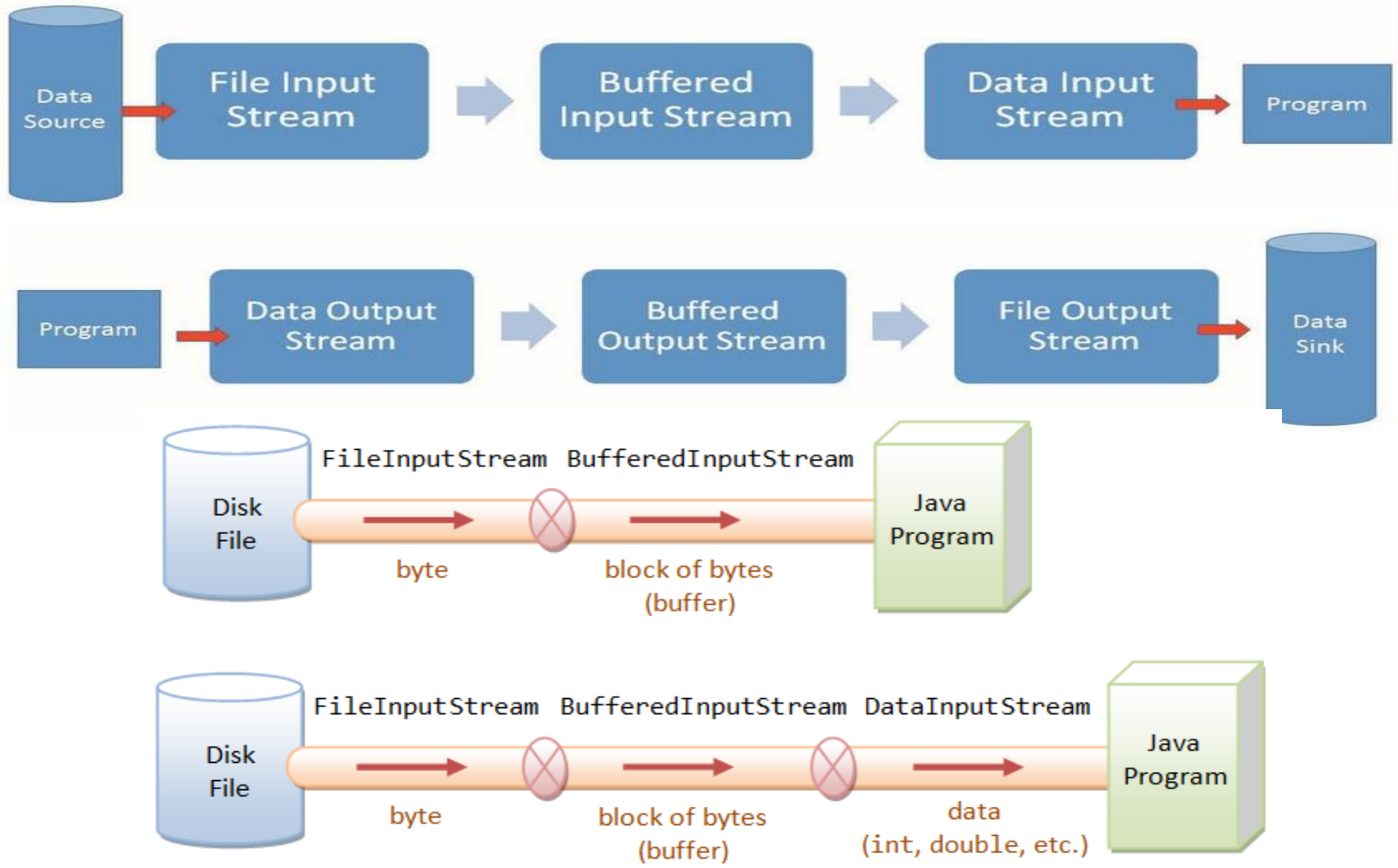


Luồng dữ liệu – stream

- Nhập/xuất dữ liệu trong máy tính được vận hành theo cơ chế dòng dữ liệu: dữ liệu được truyền theo các bytes một cách tuần tự ➔ khái niệm được xuất phát từ hệ điều hành Unix.
- Bàn phím là dòng nhập chuẩn, các ký tự được truyền theo chuỗi bytes để lưu vào biến.
- Màn hình là dòng xuất chuẩn, các ký tự từ biến được truyền theo chuỗi bytes và hiển thị trên màn hình.

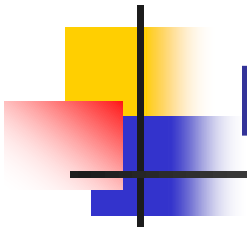


Vào ra dữ liệu – mô hình tổng thể



Chương 6:

2. Làm việc với tệp tin – File



File

- Tập tin – File: là một lớp chứa trong máy tính cho phép chúng ta lưu trữ dữ liệu, thông tin, cấu hình, … được sử dụng trong các phần mềm của máy tính.
- Tập tin có nhiều định dạng khác nhau được phân loại theo chức năng như: file dữ liệu, file text, file thực thi, file âm thanh, file hình ảnh …
- Tập tin được lưu trữ dưới dạng byte và được đọc/ghi thông qua các ứng dụng ➔ cho phép hiển thị thông tin như mong muốn của người dùng.
- Trong Java hỗ trợ:
  - File: cho phép chương trình truy xuất và làm việc với tệp tin hoặc thư mục.
  - FileDescriptor: giúp đồng bộ dữ liệu khi đọc/ghi dữ liệu với tệp tin.

File

File type	File extension
Image	.bmp .eps .gif .jpg .pict .png .psd .tif
Text	.asc .doc .docx .rtf .msg .txt .wpd .wps
Video	.avi .mp4 .mpg .mov .wmv
Compressed	.arc .arj .gz .hqx .rar .sit .tar .z .zip
Program	.bat .com .exe
Sound	aac .au .mid .mp3 .ra .snd .wma .wav

File class trong Java

Field Summary

Fields

Modifier and Type	Field and Description
`static String`	**`pathSeparator`** The system-dependent path-separator character, represented as a string for convenience.
`static char`	**`pathSeparatorChar`** The system-dependent path-separator character.
`static String`	**`separator`** The system-dependent default name-separator character, represented as a string for convenience.
`static char`	**`separatorChar`** The system-dependent default name-separator character.

Constructor Summary

Constructors

Constructor and Description
`File``(File parent, String child)` Creates a new `File` instance from a parent abstract pathname and a child pathname string.
`File``(String pathname)` Creates a new `File` instance by converting the given pathname string into an abstract pathname.
`File``(String parent, String child)` Creates a new `File` instance from a parent pathname string and a child pathname string.
`File``(URI uri)` Creates a new `File` instance by converting the given `file:` URI into an abstract pathname.

File class trong Java

Method Summary

All Methods | Static Methods | Instance Methods | Concrete Methods | Deprecated Methods

Modifier and Type	Method and Description
boolean	**canExecute**() Tests whether the application can execute the file denoted by this abstract pathname.
boolean	**canRead**() Tests whether the application can read the file denoted by this abstract pathname.
boolean	**canWrite**() Tests whether the application can modify the file denoted by this abstract pathname.
int	**compareTo**(File pathname) Compares two abstract pathnames lexicographically.
boolean	**createNewFile**() Atomically creates a new, empty file named by this abstract pathname if and only if a file with this name does not yet exist.
static File	**createTempFile**(String prefix, String suffix) Creates an empty file in the default temporary-file directory, using the given prefix and suffix to generate its name.
static File	**createTempFile**(String prefix, String suffix, File directory) Creates a new empty file in the specified directory, using the given prefix and suffix strings to generate its name.
boolean	**delete**() Deletes the file or directory denoted by this abstract pathname.
void	**deleteOnExit**() Requests that the file or directory denoted by this abstract pathname be deleted when the virtual machine terminates.
boolean	**equals**(Object obj) Tests this abstract pathname for equality with the given object.
boolean	**exists**() Tests whether the file or directory denoted by this abstract pathname exists.
File	**getAbsoluteFile**() Returns the absolute form of this abstract pathname.
String	**getAbsolutePath**() Returns the absolute pathname string of this abstract pathname.
File	**getCanonicalFile**() Returns the canonical form of this abstract pathname.
String	**getCanonicalPath**() Returns the canonical pathname string of this abstract pathname.
long	**getFreeSpace**() Returns the number of unallocated bytes in the partition **named** by this abstract path name.
String	**getName**() Returns the name of the file or directory denoted by this abstract pathname.

Làm việc với File

```
import java.io.File;

public class FileExample {

    public static void main(String[] args) {
        File file = new File("I:\\Java-Workspaces\\ExampleFile.txt");
        /*
         * Get file properties
         */
        System.out.println("File path: " + file.getAbsolutePath());
        System.out.println("File name: " + file.getName());
        System.out.println("Size: " + file.length());
        System.out.println("Is readable: " + file.canRead());
        System.out.println("Is writedable: " + file.canWrite());
        System.out.println("Total space: " + file.getTotalSpace());
        System.out.println("Is directory: " + file.isDirectory());
    }

}
```

Bài tập

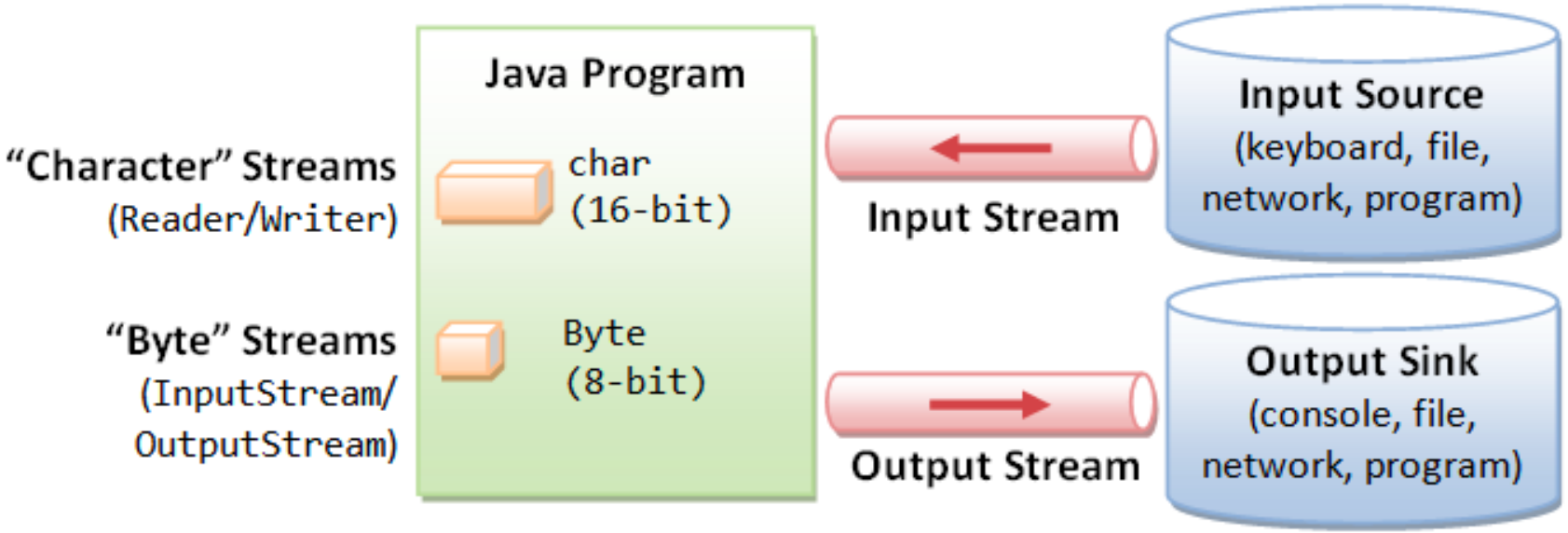
- **Các bài tập với File**

1. Tạo một file có tên là example.txt trong máy tính
   - Viết chương trình kiểm tra các thuộc tính của file example.txt (đọc, ghi, kích thước, đường dẫn, tên file …)
   - Viết đoạn chương trình thay đổi các thuộc tính đọc/ghi và kiểm tra xem file có thay đổi thuộc tính không.
2. Xây dựng chương trình giả lập chương trình diệt virus
   - Xuất phát từ một thư mục hoặc ổ đĩa.
   - Đọc các thư mục và các tệp tin bên trong thư mục (đệ quy)
   - Với các tệp tin, kiểm tra xem tệp tin có đuôi .exe, .bat thì hiển thị trên màn hình.
   - Đếm số file .exe, .bat đã duyệt và thông báo.

Chương 6:

3. Vào ra dữ liệu với dòng dữ liệu

Luồng dữ liệu trong Java – stream



Internal Data Formats:

- Text (char): UCS-2
- int, float, double, etc.

External Data Formats:

- Text in various encodings (US-ASCII, ISO-8859-1, UCS-2, UTF-8, UTF-16, UTF-16BE, UTF16-LE, etc.)
- Binary (raw bytes)

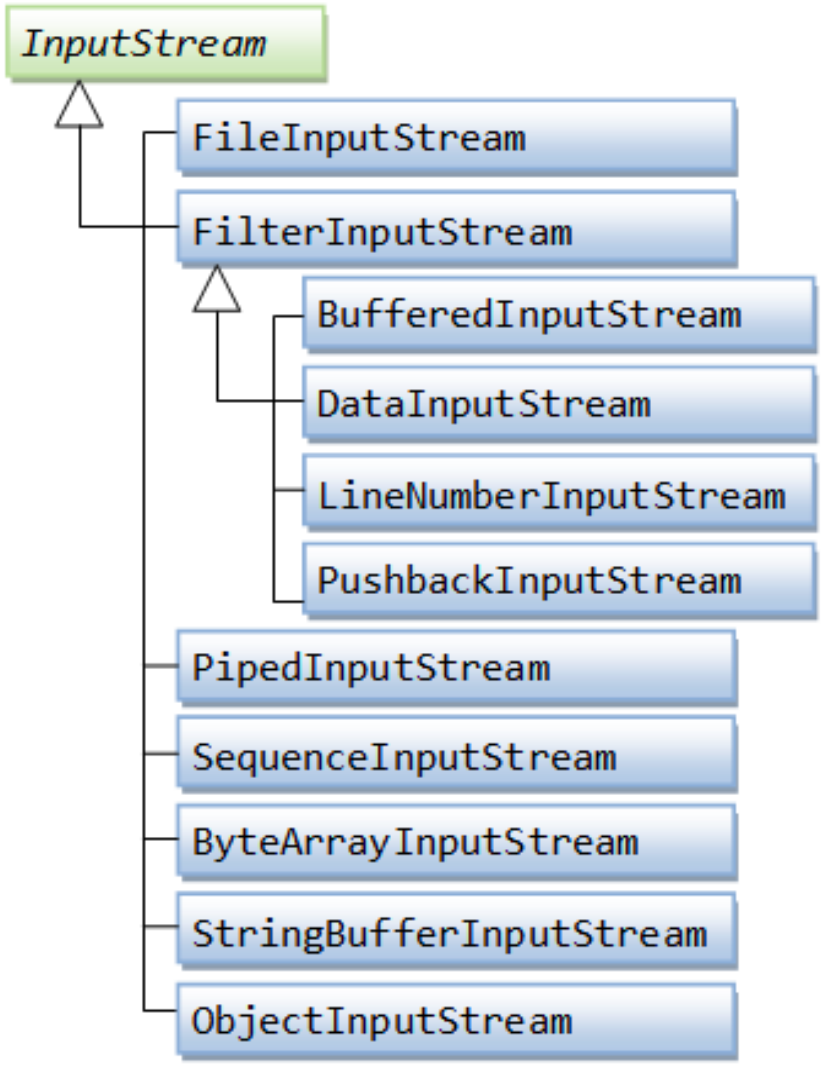
Luồng dữ liệu trong Java – stream

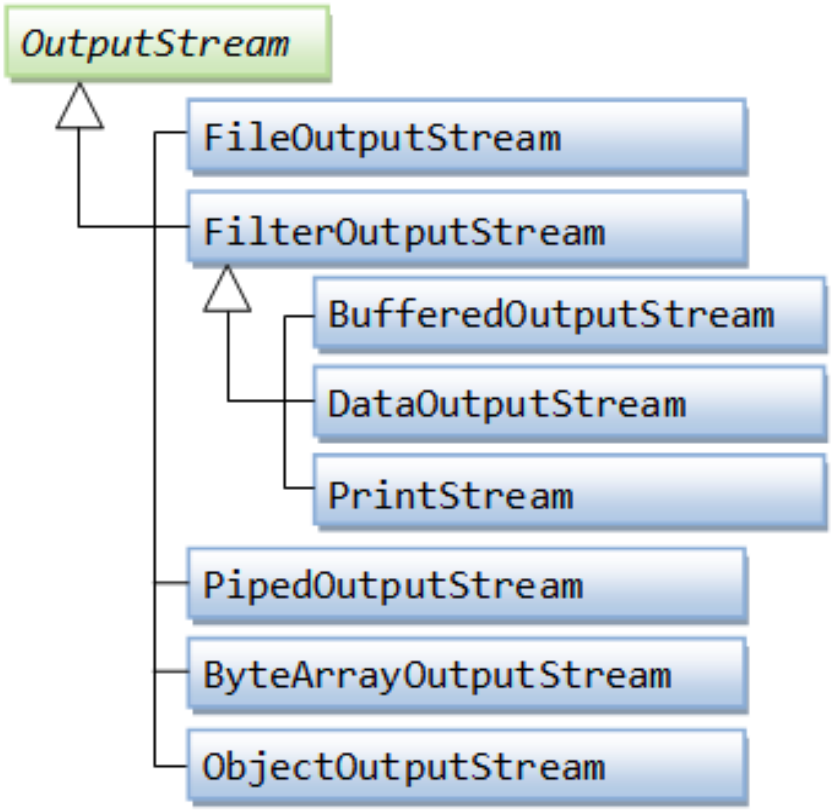
- Tất cả các **InputStream**, **OutputStream**, **Reader**, **Writer** đều implements interface <<***Closeable***>>: có thể sử dụng trong câu lệnh try để tự đóng – ***close()*** sẽ được tự động gọi khi kết thúc ***try_with_resource***.

- Các lớp ***OutputStream***, ***Writer*** đều implements interface << ***Flushable*** >>: tự động xóa bộ đệm sau khi ghi dữ liệu ra bên ngoài như ghi dữ liệu vào bộ đệm, ghi dữ liệu ra file.

- Đối với ***Object***, để đọc ghi dữ liệu cần cung cấp << ***Serializable*** >> để có thể chuyển đổi dữ liệu đối tượng sang dạng bytes trước khi ghi dữ liệu ra file.

Vào ra dữ liệu – bytes





File [InputStream/OutputStream]

java.io

Class InputStream

java.lang.Object
    java.io.InputStream

All Implemented Interfaces:

Closeable, AutoCloseable

Direct Known Subclasses:

AudioInputStream, ByteArrayInputStream, FileInputStream, FilterInputStream, InputStream,

ObjectInputStream, PipedInputStream, SequenceInputStream, StringBufferInputStream

java.io

Class FileInputStream

java.lang.Object
    java.io.InputStream
        java.io.FileInputStream

All Implemented Interfaces:

Closeable, AutoCloseable

java.io

Class OutputStream

java.lang.Object
    java.io.OutputStream

All Implemented Interfaces:

Closeable, Flushable, AutoCloseable

Direct Known Subclasses:

ByteArrayOutputStream, FileOutputStream, FilterOutputStream, ObjectOutputStream,

OutputStream, PipedOutputStream

java.io

Class FileOutputStream

java.lang.Object
    java.io.OutputStream
        java.io.FileOutputStream

All Implemented Interfaces:

Closeable, Flushable, AutoCloseable

File [InputStream/OutputStream]

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
public class FileStreamExample {

    public static String INPUT_FILE_NAME = "I:\\Java-Workspaces\\ExampleFile.zip";
    public static String OUTPUT_FILE_NAME = "I:\\Java-Workspaces\\ExampleFile_1.zip";

    public static void main(String[] args) throws IOException {
        FileInputStream inf = null;
        FileOutputStream outf = null;

        try {
            inf = new FileInputStream(INPUT_FILE_NAME);
            outf = new FileOutputStream(OUTPUT_FILE_NAME);
            int c;
            while ((c = inf.read()) != -1) {
                outf.write(c);
            }
            System.out.println("Finish!");
        } finally {
            if (inf != null) {
                inf.close();
            }
            if (outf != null) {
                outf.close();
            }
        }
    }
}
```

Buffer [InputStream/OutputStream]

```
import java.io.BufferedInputStream;
import java.io.BufferedOutputStream;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;

public class FileStreamWithBufferExample {

    public static String INPUT_FILE_NAME = "I:\\Java-Workspaces\\ExampleFile.zip";
    public static String OUTPUT_FILE_NAME = "I:\\Java-Workspaces\\ExampleFile_1.zip";

    public static void main(String[] args) throws IOException {
        FileInputStream fin = null;
        FileOutputStream fout = null;
        BufferedInputStream bin = null;
        BufferedOutputStream bout = null;

        try {
            fin = new FileInputStream(INPUT_FILE_NAME);
            bin = new BufferedInputStream(fin);
            fout = new FileOutputStream(OUTPUT_FILE_NAME);
            bout = new BufferedOutputStream(fout);
            int c;
            while ((c = bin.read()) != -1) {
                bout.write(c);
            }
            System.out.println("Finish!");
        }
        finally {
            if (bin != null) {
                bin.close();
            }
            if (fin != null) {
                fin.close();
            }
            if (bout != null) {
                bout.close();
            }
            if (fout != null) {
                fout.close();
            }
        }
    }
}
```

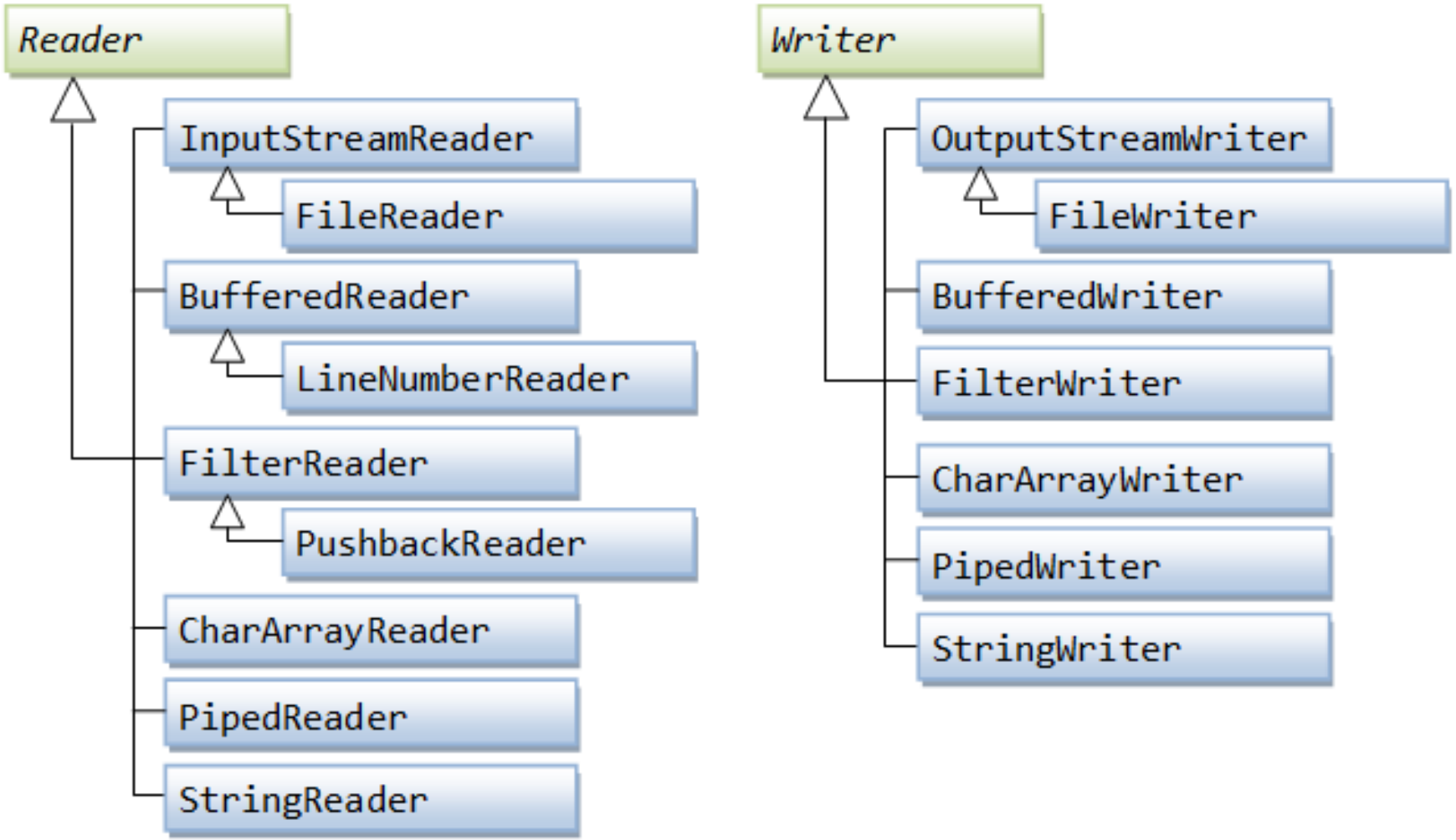
Object [InputStream/OutputStream]

```
public static String FILE_NAME = "I:\\Java-Workspaces\\Person.data";

public static void main(String[] args) throws IOException, ClassNotFoundException {
    Person p1 = new Person(1, "Nguyễn Trọng Phúc", "email_phuc@gmail.com");
    Person p2 = new Person(2, "Nguyễn Vương Phong", "email_phong@gmail.com");
    Person[] p_ins = {p1, p2};
    /*
     * Write objects to file
     */
    FileOutputStream fout = null;
    ObjectOutputStream oout = null;
    try {
        fout = new FileOutputStream(FILE_NAME);
        oout = new ObjectOutputStream(fout);
        oout.writeObject(p_ins);
        System.out.println("Write data: finish.");
    } finally {
        if (oout != null) {
            oout.close();
        }
        if (fout != null) {
            fout.close();
        }
    }
```

Object [InputStream/OutputStream]

```
public static String FILE_NAME = "I:\\Java-Workspaces\\Person.data";

public static void main(String[] args) throws IOException, ClassNotFoundException {
    /*
     * Read objects from file
     */
    FileInputStream fin = new FileInputStream(FILE_NAME);
    ObjectInputStream oin = new ObjectInputStream(fin);
    try {
        fin = new FileInputStream(FILE_NAME);
        oin = new ObjectInputStream(fin);
        Person[] persons = (Person[]) oin.readObject();
        for (Person p : persons) {
            System.out.println(p);
        }
        System.out.println("Read data: finish.");
    } finally {
        if (oin != null) {
            oin.close();
        }
        if (fin != null) {
            fin.close();
        }
    }
```

Vào ra dữ liệu – characters



File [Reader/Writer]

java.io

Class Reader

java.lang.Object
    java.io.Reader

All Implemented Interfaces:

Closeable, AutoCloseable, Readable

Direct Known Subclasses:

BufferedReader, CharArrayReader, FilterReader, InputStreamReader, PipedReader, StringReader

java.io

Class FileReader

java.lang.Object
    java.io.Reader
        java.io.InputStreamReader
            java.io.FileReader

All Implemented Interfaces:

Closeable, AutoCloseable, Readable

java.io

Class Writer

java.lang.Object
    java.io.Writer

All Implemented Interfaces:

Closeable, Flushable, Appendable, AutoCloseable

Direct Known Subclasses:

BufferedWriter, CharArrayWriter, FilterWriter, OutputStreamWriter, PipedWriter, PrintWriter, StringWriter

java.io

Class FileWriter

java.lang.Object
    java.io.Writer
        java.io.OutputStreamWriter
            java.io.FileWriter

All Implemented Interfaces:

Closeable, Flushable, Appendable, AutoCloseable

File [Reader/Writer]

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class FileReaderExample {

    public static String IN_FILE_NAME = "I:\\Java-Workspaces\\Data\\Data.txt";

    public static void main(String[] args) throws IOException{
        FileReader fr = null;
        BufferedReader br = null;
        try{
            fr = new FileReader(IN_FILE_NAME);
            br = new BufferedReader(fr);
            String line;
            while ((line=br.readLine())!=null){
                System.out.println(line);
            }
        }finally{
            if (br != null){
                br.close();
            }
            if (fr != null){
                fr.close();
            }
        }

    }
}
```

File [Reader/Writer]

```
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;

public class FileWriterExample {
    public static String OUT_FILE_NAME = "I:\\Java-Workspaces\\Data\\Data_out.txt";
    public static void main(String[] args) throws IOException {
        FileWriter fw = null;
        BufferedWriter bw = null;
        try {
            fw = new FileWriter(OUT_FILE_NAME);
            bw = new BufferedWriter(fw);
            bw.write("12 23 34.5 56");
            bw.newLine();
            bw.write("175 392 1");
            bw.newLine();
            bw.write("22");
            System.out.println("Write data: finish.");
        } finally {
            if (bw != null) {
                bw.close();
            }
            if (fw != null) {
                fw.close();
            }
        }
    }
}
```

Bài tập

- **Các bài tập với đọc ghi dữ liệu**

1. Cấu trúc các Point [đã học] được lưu trữ vào file dưới dạng như sau:

   [12.3;-12.9]-[23;0.12]-[-15;32.01]…

   - Viết đoạn chương trình thực hiện công việc:
     - Đọc các Point từ file vào trong một danh sách.
     - Tính tổng khoảng cách các Points.
     - Ghi giá trị tổng và số Point ra file kết quả với cấu trúc

       Number of Points: 3

       Total distance: 234.9081

2. Thay đổi chương trình trên với cách lưu trữ là Object với file dữ liệu đầu vào.

Chú ý:

- Kiểm soát lỗi khi đọc ghi dữ liệu thông qua try … catch … finally.
- Sử dụng hàm trong String để tách dữ liệu, kiểm tra dữ liệu.

Bài tập

- **Các bài tập với Sodoku**

1. Từ chương trình Sodoku đã được xây dựng, hãy thiết kế tiếp chương trình với các yêu cầu sau:
   - Thiết kế cấu trúc file dạng txt để lưu trữ dữ liệu các Game.
   - Một tập tin cho phép lưu trữ tất cả các Game, khi người dùng chơi, thực hiện việc lựa chọn ngẫu nhiên một Game.
   - Một tập tin cho phép lưu trữ Game người chơi đang chơi và trạng thái hiện tại cho phép người chơi có thể load lại Game và chơi tiếp.
   - Một tập tin lưu trữ thông tin cấu hình chung.

- **Chú ý**
  - Bố cục chương trình (MVC) sao cho phù hợp.
  - Cấu trúc dữ liệu cần đảm bảo tính phù hợp cho chương trình.
  - Sinh viên có thể lựa chọn cơ chế đọc ghi dữ liệu phù hợp. Có thể dùng InputStream/OutputStream, ObjectInputStream/ObjectOutputStream hoặc Reader/Writer.
  - Kiểm soát lỗi với try … catch … finallly khi làm việc với tệp tin.

CÔNG NGHỆ JAVA