Bài 4
II. I/O theo luồng và thao tác tệp

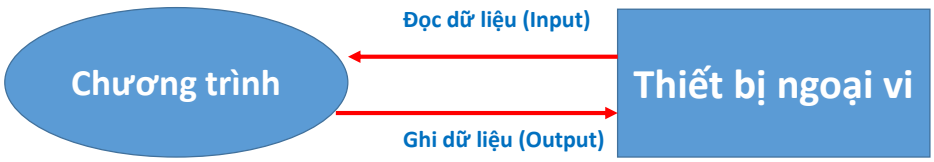
NỘI DUNG

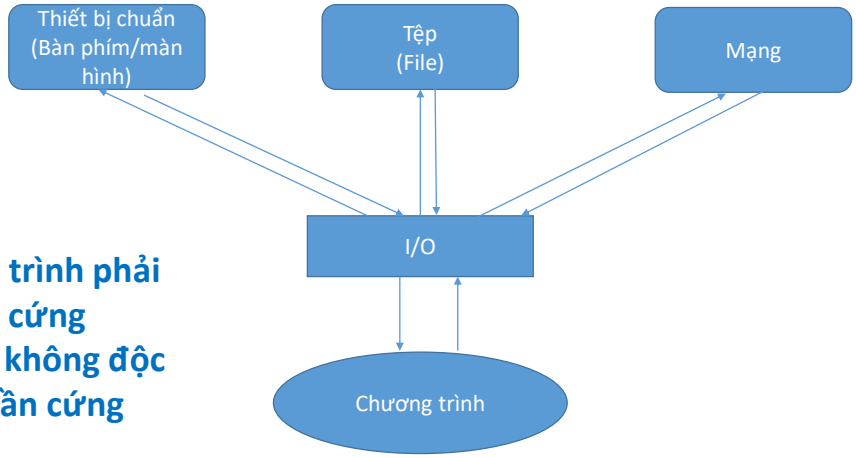
- Khái niệm luồng nhập xuất
- Các kiểu luồng nhập xuất
- Các lớp Java hỗ trợ nhập xuất theo luồng
- Thao tác tệp

1. Khái niệm luồng nhập xuất

- Nhập/xuất: Thực hiện nhập xuất với thiết bị ngoại vi
- 3 hình thức nhập xuất:
  - I/O thiết bị chuẩn
  - I/O tệp
  - I/O mạng
- Có 2 phương pháp thực hiện I/O:
  - Trực tiếp
  - Gán tiếp

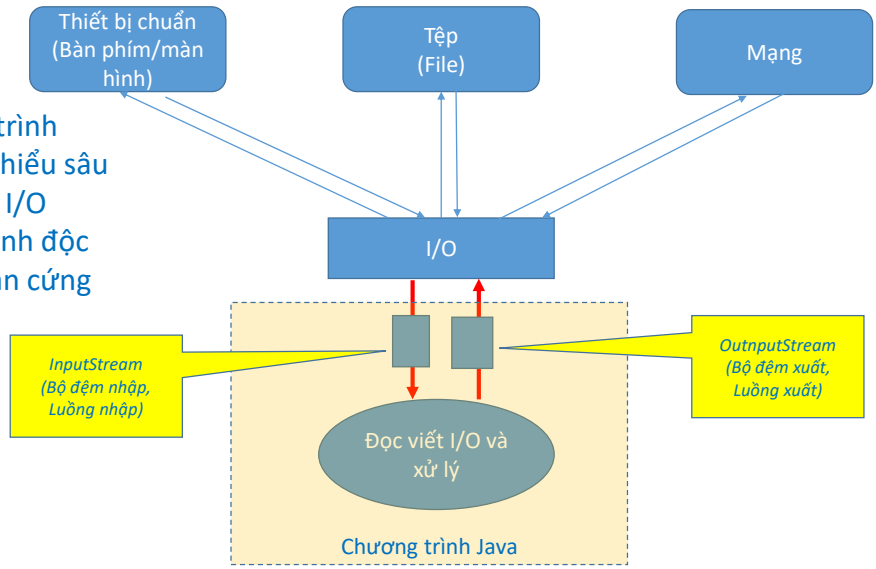
I/O dữ liệu





- **Người lập trình phải hiểu phần cứng**
- **Ứng dụng không độc lập với phần cứng**

I/O trực tiếp



- Người lập trình không cần hiểu sâu phần cứng I/O
- Chương trình độc lập với phần cứng I/O

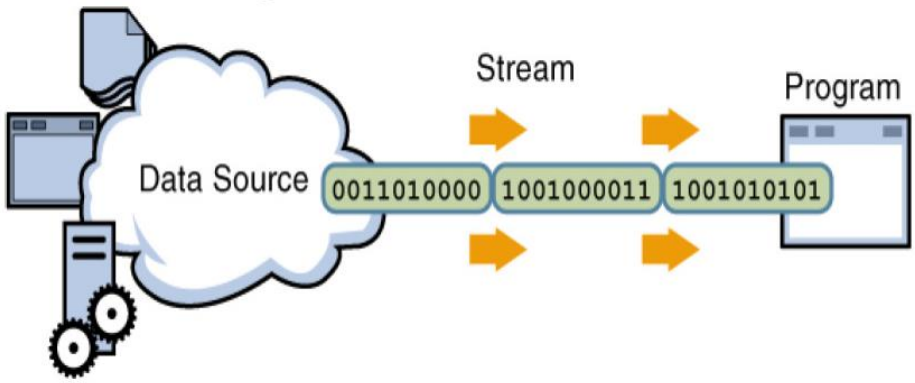
I/O theo luồng (stream)- I/O gián tiếp

2. I/O theo luồng (gián tiếp)

- Java hỗ trợ thực hiện I/O theo luồng (stream)
- Java có một tập các lớp, giao diện hỗ trợ tạo luồng I/O qua các gói thư viện java.io, javax.nio…
- I/O theo luồng, người sử dụng chỉ cần tạo đối tượng luồng I/O và gọi phương thức read()/write() để vào ra dữ liệu với thiết bị ngoại vi (Mọi vấn đề liên quan đến phần cứng đã được Java xử lý và đóng gói trong các lớp, người dùng chỉ cần tạo đối tượng luồng I/O và thực hiện đọc/ghi dữ liệu với các đối tượng này, còn mọi việc I/O với phần cứng các đối tượng tự biết xử lý).
- Java hỗ trợ phong phú các giao diện, các lớp đáp ứng mọi kiểu I/O (thiết bị chuẩn, tệp, mạng…)
- Ưu điểm:
  - Người sử dụng không cần hiểu sâu về phần cứng
  - Chương trình độc lập với phần cứng
  - Đáp ứng đa dạng các hình thức I/O theo cùng một cơ chế.

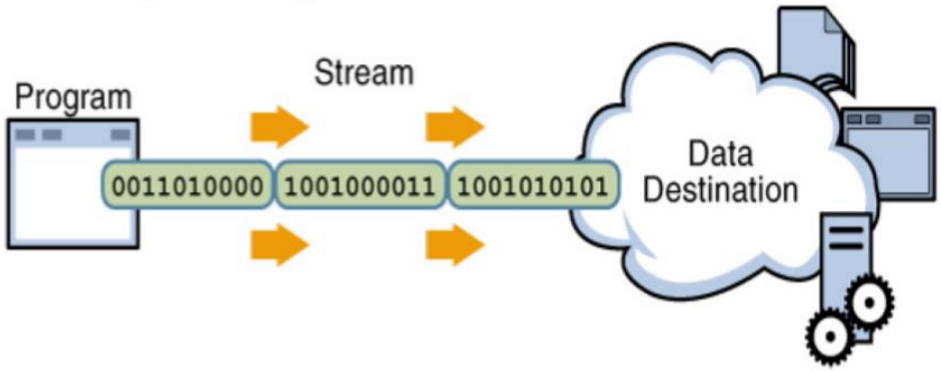
Input Stream

- Chương trình sử dụng input stream để đọc dữ liệu từ nguồn.



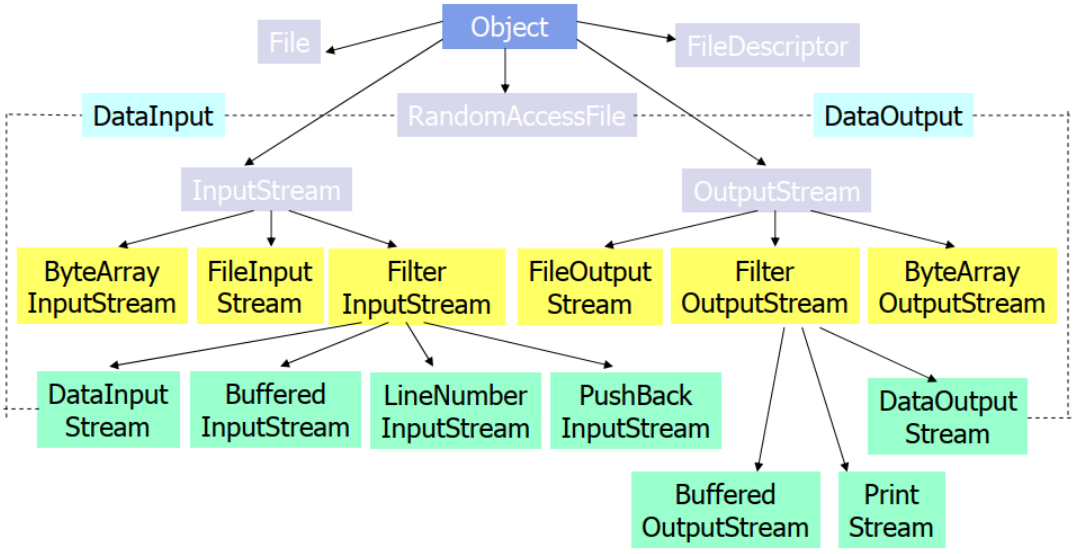
Output Stream

- Chương trình sử dụng output stream để ghi dữ liệu xuống đích.



3. Các kiểu I/O theo luồng: 2 kiểu

- **I/O theo luồng kiểu byte:**
  - Dữ liệu I/O là một dòng các byte liên tục
  - Đơn vị dữ liệu nhỏ nhất là 1 byte
  - Mọi lớp hỗ trợ I/O theo luồng kiểu byte đều có vĩ tố Stream, ví dụ: InputStream, OutputStream, …..
  - Mọi lớp đều kế thừa trực tiếp hoặc gián tiếp từ lớp trừu tượng InputStream/OutputStream với các phương thức cơ bản là read()/write()



Lớp InputStream:

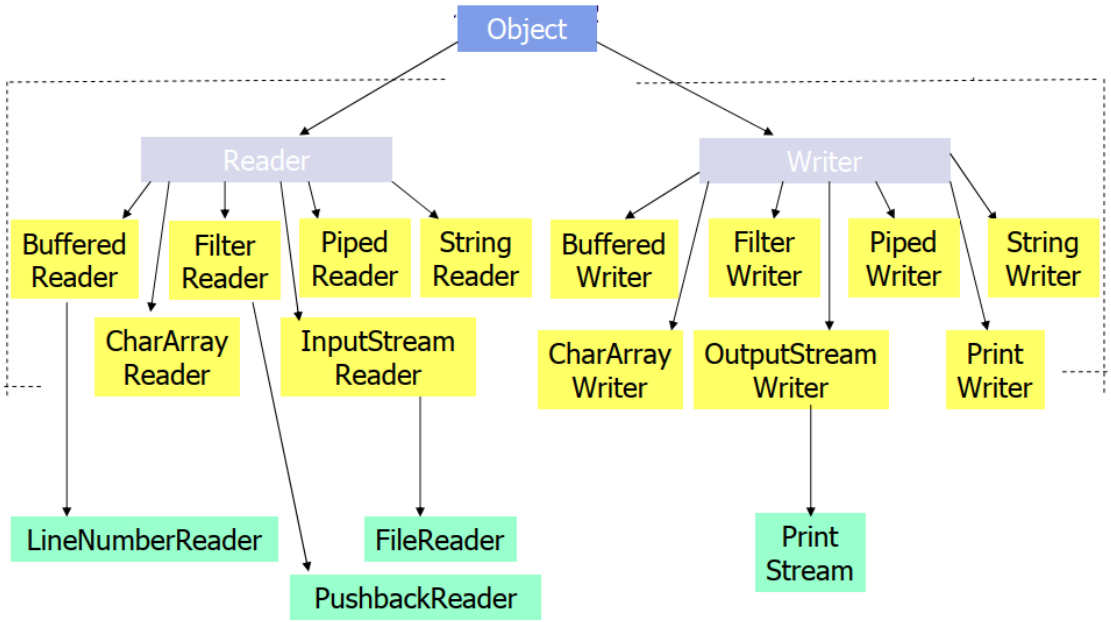
- `public abstract int` **`read`**`() throws IOException`
  Đọc một byte kế tiếp của dữ liệu từ luồng.
- `public int` **`read`**`(byte[] bBuf) throws IOException`
  Đọc một số byte dữ liệu từ luồng và lưu vào mảng byte bBuf.
- `public int` **`read`**`(byte[] cBuf, int offset, int length) throws IOException`
  Đọc `length` byte dữ liệu từ luồng và lưu vào mảng byte cBuf bắt đầu tại vị trí offset.
- `public void` **`close`**`() throws IOException`
  Đóng nguồn. Gọi những phương thức khác sau khi đó nguồn sẽ gây ra lỗi IOException
- `public int` **`mark`**`(int readAheadLimit) throws IOException`
  Đánh dấu vị trí hiện hành của stream. Sau khi đánh dấu, gọi reset() sẽ định lại vị trí của luồng đến điểm này. Không phải tất cả luồng byte –input hỗ trợ cho thao tác này.
- `public int` **`markSupported`**`()`
  Chỉ ra luồng có hỗ trợ thao tác mark và reset hay không

Lớp OutputStream:

- `public void write(int b) throws IOException`
  Ghi giá trị b xác định theo dạng byte xuống output stream
- `public void write(byte[] b) throws IOException`
  Lưu nội dung của mảng byte b xuống luồng
- `public void write(byte[] b, int off, int len) throws IOException`
  Lưu len byte của mảng byte b xuống luồng, bắt đầu từ vị trí off của mảng
- `public void close() throws IOException`
  Đóng nguồn. Gọi những phương thức khác liên quan đến nguồn này sau khi gọi close se gây ra lỗi IOException.
- `public void flush() throws IOException`
  flushes the stream.(ví dụ: Những byte được lưu trong buffer ngay lập tức được ghi xuống đích)

Các kiểu I/O theo luồng: 2 kiểu (tiếp)

- **I/O theo luồng kiểu char:**
  - Dữ liệu I/O là một dòng các ký tự Unicode
  - Đơn vị dữ liệu nhỏ nhất là 2 byte
  - Mọi lớp hỗ trợ I/O theo luồng kiểu char đều có vĩ tố er, ví dụ: Reader, Writer, …..
  - Mọi lớp đều kế thừa trực tiếp hoặc gián tiếp từ lớp trừu tượng Reader/Writer với các phương thức cơ bản là read()/write()



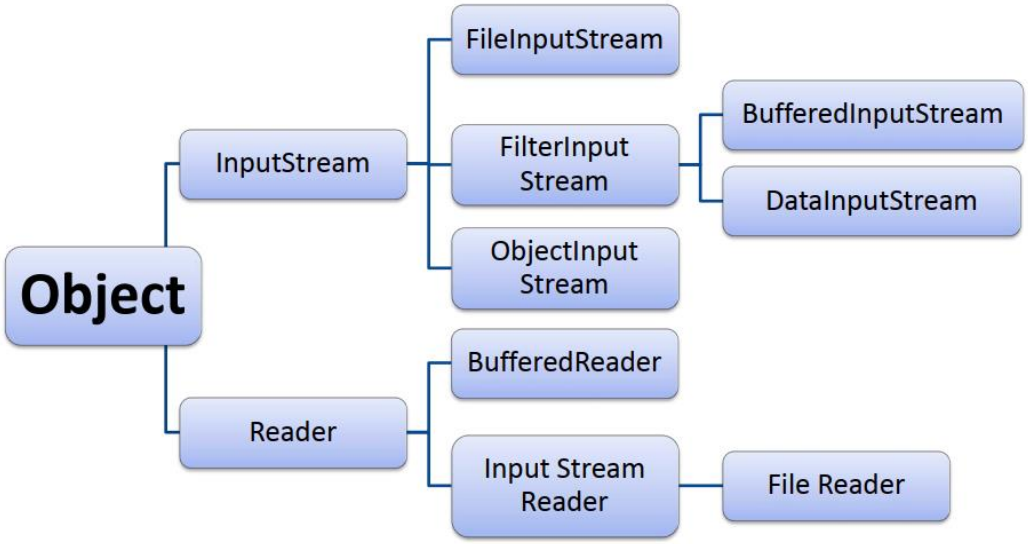
Lớp Reader:

- `public int` **`read`**`() throws IOException`
  - Đọc một ký tự
- `public int` **`read`**`(char[] cbuf) throws IOException`
  - Đọc những ký tự và lưu chúng vào mảng cbuf
- `public abstract int` **`read`**`(char[] cbuf, int off, int len) throws IOException`
  - Đọc len ký tự và lưu chúng vào tron mảng cbuf, bắt đầu tại vị trí off của mảng
- `public abstract void` **`close`**`() throws IOException`
  - Đóng luồng. Gọi những phương thức Reader khác của sau khi gọi close sẽ gây ra lỗi IOException
- `public void` **`mark`**`(int readAheadLimit) throws IOException`
  - Đánh dấu vị trí hiện hành của stream. Sau khi đánh dấu, gọi reset() để thử đặt lại vị trí luồng tới điểm này. Không phải tất cả character-input đều hỗ trợ thao tác này
- `public boolean` **`markSupported`**`()`
  - Chỉ ra luồng có hỗ trợ thao tác này hay không. Mặc định là không hỗ trợ.
- `public void` **`reset`**`() throws IOException`
  - Đặt lại vị trí luồng tới vị trí đánh dấu lần cuối

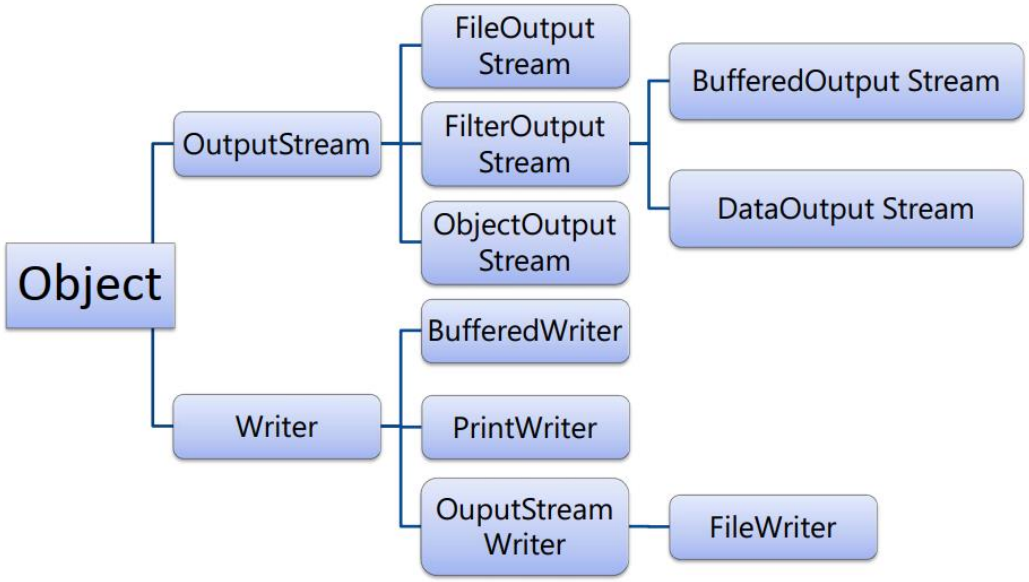
Lớp trừu tượng Writer:

- `public void write(int c) throws IOException`
  Ghi một ký tự đơn được thể hiện bằng số nguyên. Ví dụ: 'A' là được ghi la write(65)
- `public void write(char[] cbuf) throws IOException`
  Ghi nội dung của mảng ký tự cbuf xuống luồng
- `public abstract void write(char[] cbuf, int off, int len) throws IOException`
  Ghi một mảng ký tự cbuf với chiều dài là len, bắt đầu là vị trí off
- `public void write(String str) throws IOException`
  Ghi một chuỗi str
- `public void write(String str, int off, int len) throws IOException`
  Ghi một chuỗi str với chiều dài là len, bắt đầu từ vị trí off
- `public abstract void flush() throws IOException`
  Đẩy dữ liệu xuống đích đến.
- `public abstract void close() throws IOException`
  Đóng luồng.

Tóm lại các lớp hỗ trợ luồng nhập (byte và char):



Tóm lại các lớp hỗ trợ luồng xuất (byte và char):



4. Điều khiển luồng I/O:

- Bước 1: Tạo đối tượng luồng và liên kết nó với dữ liệu nguồn
- Bước 2: Thực hiện đọc hoặc viết dữ liệu ra luồng thông qua các phương thức đọc/ ghi tương ứng.
- Bước 3: Đóng luồng, giải phóng tài nguyên.

5. Các hình thức I/O theo luồng

- I/O thiết bị chuẩn
- I/O tệp
- I/O mạng
- Lưu ý: Ngoài ra còn loại liên quan đến bộ nhớ

5.1. I/O thiết bị chuẩn:

- Thiết bị vào chuẩn mặc định là bàn phím
- Thiết bị xuất chuẩn là màn hình (console)
- Java mặc định khai báo các biến tham chiếu luồng I/O chuẩn kiểu byte trong lớp System:
  - in: biến tham chiếu tới luồng nhập thiết bị chuẩn -> System.in trả về đối tượng InputStream
  - out: biến tham chiếu đến luồng xuất chuẩn -> System.out trả về đối tượng PrintStream
  - err: biến tham chiếu đến luồng lỗi thiết bị chuẩn -> System.err trả về đối tượng PrintStream

  Ví dụ:

  int c=System.in.read() ;//trả về mã Asscii của ký tự nhập

  System.out.println("Hello");//in một chuỗi ký tự ra màn hình

I/O thiết bị chuẩn: (tiếp)

- Một số cách nhập dữ liệu vào từ bàn phím khác:
  - Dùng lớp Scanner
  - Dùng lớp Console
  - Dùng lớp BufferedReader

Ví dụ lớp Scanner:

```
Import java.util.Scanner;
……
Scanner  sc=new Scanner(System.in);
String  s= sc.nextLine();
```

Ví dụ dùng lớp BufferedReader:

```
Import  java.io.*;
….
BufferedReader   inp=
            new BufferedReader(new InputStreamReader(System.in));
char  c= (char)inp.read();
String s=inp.readLine();
```

5.2. I/O tệp:

Các kiểu truy xuất tệp: 3 kiểu

- Kiểu tuần tự (serial)
- Kiểu dẫy (sequence)
- Kiểu ngẫu nhiên (random)

Các lớp tiêu biểu hỗ trợ I/O tệp:

I/O tệp với luồng kiểu byte:

- FileInputStream
- FileOutputStream
- DataInputStream
- DataOutputStream
- ObjectInputStream
- ObjectOutputStream
- ……..

I/O tệp với luồng kiểu char:

- FileReader
- FileWriter
- BufferedReader
- PrintWriter
- ……..

5.3. I/O Mạng:

- I/O mạng cũng sử dụng các lớp tương tự như I/O tệp nhưng nó I/O cho socket.
- Chủ đề này sẽ học trong học phần Lập trình nâng cao hoặc Lập trình mạng với Java.

6. Lập trình với một số lớp I/O và thao tác tệp

- File
- FileInputStream/FileOutputStream
- DataInputStrem/DataOutputStream
- ByteArrayInputStream/ByteArrayOutputStream
- BufferedInoutStream/BufferedOutputStream
- ObjectInputStream/ObjectOutputStream
- RandomAccessFile
- BufferedReader/PrintWriter

6.1. Lớp File

- Được sử dụng truy cập các đối tượng tập tin và thư mục
- Những tập tin có tên được đặt tên theo qui ước của hệ điều hành chủ.
- Lớp này cung cấp phương thức khởi tạo để tạo ra các thư mục và tập tin
- Tất cả các thao tác thư mục và tập tin đều được sử dụng các phương thức truy cập và các phương thức thư mục mà các lớp tập tin cung cấp

Các phương thức:

• **exists()**	• Kiểm tra xem file có tồn tại hay không
• **getName()**	• Lấy tên file (input-file.txt)
• **getParent()**	• Lấy đường dẫn thư mục của file
• **getPath()**	• Đường dẫn đầy đủ
• **isDirectory()**	• Kiểm tra xem là thư mục hay không
• **isFile()**	• Kiểm tra xem là file hay không
• **length()**	• Kích cỡ file (byte)
• **createNewFile()**	• Tạo ra file mới
• **delete()**	• Xóa file
• **list()**	• Lấy tên file, thư mục chứa trong đường dẫn
• **mkdir()**	• Tạo thư mục
• **renameTo(File dest)**	• Đổi tên file

Ví dụ:

```
import java.io.File;
public class MyClass {
  public static void main(String[ ] args) {
    File x = new File("D:\\Data\\test.txt"); //Lấy đối tượng File
    System.out.println("Tên file: " + x.getName());
    System.out.println("Thư mục: " + x.getParent());
    System.out.println("Thư mục: " + x.getPath());
    if(x.exists()) {
     System.out.println(x.getName() +  "exists!");
    }
    else {
     System.out.println("The file does not exist");
    }
  }
}
```

6.2. FileInputStream/FileOutputStream:

a) Lớp FileInputStream:

- Cho phép đầu vào đọc từ một tập tin trong một mẫu của một dòng
- Các đối tượng được tạo ra sử dụng chuỗi tên tập tin, tập tin, đối tượng FileDescriptor như một tham số.
- Các phương thức nạp chồng của lớp InputStream. nó cung cấp phương thức 'finalize( )' và 'getFD( )'

6.2. FileInputStream/FileOutputStream: (tiếp)

b) Lớp FileOutputStream

- Cho phép kết xuất để ghi ra một luồng tập tin
- Các đối tượng cũng tạo ra sử dụng một chuỗi tên tập tin, tập tin, hay đối tượng FileDescriptor như một tham số.
- Lớp này nạp chồng các phương thức của lớp OutputStream và cung cấp phương thức 'finalize( )' và 'getFD( )'

Ví dụ:

```
import java.io.*;
public class CopyBytes {
    public static void main(String[] args) throws IOException {
        FileInputStream in = null;
        FileOutputStream out = null;
        try {
            in = new FileInputStream("d:/src.txt");
            out = new FileOutputStream("d:/dst.txt");
            int c;
            while ((c = in.read()) != -1) { out.write(c); }
        }
        finally {
            if (in != null) { in.close(); }
            if (out != null) { out.close(); }
        }
    }
}
```

6.3. I/O tệp với FileReader/FileWriter

a) FileReader

- FileReader được sử dụng để đọc các luồng ký tự.
- Các phương thức khởi tạo:

  FileReader(File file)

  FileReader(String fileName)
- Các phương thức:

TT	Phương thức và Miêu tả
1	**public int read() throws IOException** Đọc một ký tự đơn và trả về mã của ký tự đọc
2	**public int read(char [] c, int offset, int len):** Đọc các ký tự bên trong một mảng. Trả về số ký tự đã đọc

6.3. I/O tệp với FileReader/FileWriter (tiếp)

b) FileWriter

- Lớp này được sử dụng để ghi các luồng ký tự.
- Phương thức khởi tạo:

  FileWriter(File file)
  FileWriter(String fileName)
- Các phương thức:

STT	Phương thức và Miêu tả
1	**public void write(int c) throws IOException** Ghi một ký tự đơn
2	**public void write(char [] c, int offset, int len)** Ghi một phần của mảng các ký tự bắt đầu từ offset và với độ dài **len**
3	**public void write(String s, int offset, int len)** Ghi một phần của String bắt đầu từ offset và với độ dài **len**

Ví dụ:

```
public class CopyCharacters {
    public static void main(String[] args) throws IOException {
        FileReader inputStream = null;
        FileWriter outputStream = null;
        try {
            inputStream = new FileReader("d:/fileIn.txt");
            outputStream = new FileWriter("d:/fileOut.txt");
            int c;
            while ((c = inputStream.read()) != -1) { outputStream.write(c);}
        }
        finally {
            if (inputStream != null) { inputStream.close(); }
            if (outputStream != null) { outputStream.close(); }
        }
    }
}
```

6.4. DataInputStream/DataOutputStream

- Hỗ trợ nhập xuất nhị phân của các loại dữ liệu nguyên thuỷ (boolean, char, byte, short, int, long, float, and double)
- Là các lớp thực thi của giao diện DataInput/DataOutput

Giao diện DataInput

- Được sử dụng để đọc các byte từ luồng nhị phân, và xây dựng lại dữ liệu trong một số kiểu dữ liệu nguyên thuỷ.
- Cho phép chúng ta chuyển đổi dữ liệu từ từ khuôn dạng UTF-8 được sửa đổi Java đến dạng chuỗi
- Định nghiã số phương thức, bao gồm các phương thức để đọc các kiểu dữ liệu nguyên thuỷ.

Những phương thức giao diện DataInput

- **boolean readBoolean( )**
- **byte readByte( )**
- **char readChar( )**
- **short readShort( )**
- **long readLong( )**
- **float readFloat( )**
- **int readInt( )**
- **double readDouble( )**
- **String readUTF( )**
- **String readLine( )**

Giao diện DataOutput

- Được sử dụng để xây dựng lại dữ liệu một số kiểu dữ liệu nguyên thuỷ vào trong dãy các byte
- Ghi các byte dữ liệu vào luồng nhị phân
- Cho phép chúng ta chuyển đổi một chuỗi vào khuôn dạng UTF-8 được sửa đổi Java và viết nó vào trong một dãy.
- Định nghĩa một số phương thức và tất cả phương thức kích hoạt IOException trong trường hợp lỗi.

Các phương thức giao diện DataOutput

- **void writeBoolean(boolean b)**
- **void writeByte(int value)**
- **void writeChar(int value)**
- **void writeShort(int value)**
- **void writeLong(long value)**
- **void writeFloat(float value)**
- **void writeInt(int value)**
- **void writeDouble(double value)**
- **void writeUTF(String value)**

Ví dụ: Ghi dữ liệu nguyên thủy xuống tệp:

```
public class DataStreamOutput {
    public static void main(String[] args) throws IOException {
        FileOutputStream fos = new FileOutputStream("filedata.dat");
        DataOutputStream dos = new DataOutputStream(fos);
        final int NUMBER = 5;
        dos.writeInt(NUMBER);
        for (int i = 0; i <= NUMBER; i++) {
            dos.writeInt(i);
        }
        dos.writeUTF("Hello !");
        dos.writeDouble(100.75);
        dos.flush();
        dos.close();
    }
}
```

Vi dụ đọc dữ liệu nguyên thủy từ tệp:

```
public class DataStreamInput {

    public static void main(String[] args) throws IOException {
        FileInputStream fis = new FileInputStream("filedata.dat");
        DataInputStream dis = new DataInputStream(fis);
        int items = dis.readInt();
        for (int i = 0; i <= items; i++) {
            int n = dis.readInt();
            System.out.print(n + " ");
        }
        System.out.println(dis.readUTF());
        System.out.println(dis.readDouble());
        dis.close();
    }
}
```

6.5. ByteArrayInputStream/ByteArrayOutputStream
a) ByteArrayInputStream

- Sử dụng các đệm bộ nhớ
- Lớp **ByteArrayInputStream**
  - **ByteArrayInputStream(byte[] buf)**
- Tạo ra một luồng nhập từ đệm bộ nhớ mảng các byte.
  - Không hỗ trợ các phương thức mới
  - Các phương thức nộp chồng của lớp InputStream, giống như 'read()', 'skip()', 'available()' và 'reset()'.

b) ByteArrayOutputStream

- Tạo ra một luồng kết xuất trên mảng byte
- Cung cấp các khả năng bổ sung cho mảng kết xuất tăng trưởng nhằm chừa chổ cho dữ liệu mới ghi vào.
- Cũng cung cấp các phương thức để chuyển đổi luồng tới mảng byte, hay đối tượng String.

- Phương thức của lớp **ByteArrayOutputStream** :
  - **ByteArrayOutputStream()**
  - **void reset( )**
  - **int size( )**
  - **byte[] toByteArray()**
  - **String toString()**

6.6. Bộ lọc

- Lọc:
  - Là kiểu luồng sửa đổi cách điều quản một luồng hiện có.
  - Về cơ bản được sử dụng để thích ứng các luồng theo các nhu cầu của chương trình cụ thể.
  - Bộ lọc nằm giữa luồng cơ sở và CT.
  - Thực hiện một số tiến trình đặt biệt trên các byte được chuyển giao từ đầu vào đến kết xuất.
  - Có thể phối hợp để thực hiện một dãy các tuỳ chọn lọc.

a) FilterInputStream

- Là lớp trừu tượng.
- Là cha của tất cả các lớp luồng nhập đã lọc.
- Cung cấp khả năng tạo ra một luồng từ luồng khác.
- Một luồng có thể đọc và cung cấp cung cấp dưới dạng kết xuất cho luồng khác.
- Duy trì một dãy các đối tượng của lớp 'InputStream'
- Cho phép tạo ra nhiều bộ lọc kết xích (chained filters).

b) Lớp FilterOutputStream

- Là dạng bổ trợ cho lớp FilterInputStream.
- Là cha của tất cả các lớp luồng kết xuất.
- Duy trì đối tượng của lớp OutputStream như là một biến out.
- Dữ liệu ghi ra lớp này có thể sửa đổi để thực hiện các thao tác lọc, và sau đó phản hồi đến đối tượng OutputStream.

6.8. Vùng đệm nhập/xuất

- Vùng đệm:
  - Là kho lưu trữ dữ liệu.
  - Có thể cung cấp dữ liệu thay vì quay trợ lại nguồn dữ liệu gốc ban đầu.
  - Java sử dụng vùng đệm nhập và kết xuất để tạm thời lập cache dữ liệu được đọc hoặc ghi vào một luồng.
- Trong khi thực hiện vùng đệm nhập:
  - Số lượng byte lớn được đọc cùng thời điểm, và lưu trữ trong một vùng đệm nhập.
  - Khi chương trình đọc luồng nhập, các byte nhập được đọc vào vùng đệm nhập.

6.8. Vùng đệm nhập/xuất (tiếp)

- Trong trường hợp vùng đệm kết xuất, một chương trình ghi ra một luồng.
- Dữ liệu kết xuất đượcc lưu trữ trong một vùng đệm kết xuất.
- Dữ liệu được lưu trữ cho đến khi vùng đệm trợ nên đầy, hay luồng kết xuất được xả trống.
- Kết thúc, vùng đệm kết xuất được chuyển gửi đến đích của luồng xuất.

a) BufferedInputStream

- Tự động tạo ra và duy trì vùng đệm để hỗ trợ vùng đệm nhập.
- Lớp BufferedInputStream là một bộ đệm, nó có thể áp dụng cho một số các đối tượng nhất định của lớp InputStream.
- Cũng có thể phối hợp các tập tin đầu vào khác.
- Sử dụng vài biến để triển khai vùng đệm nhập.

b)Lớp BufferedOutputStream

- Thực hiện vùng đệm kết xuất theo cách tương ứng với lớp BufferedInputStream.
- Định nghĩa hai phương thức thiết lập. Nó cho phép chúng ta ấn định kích thước của vùng đệm xuất trong một phương thức thiết lập, cũng giống như cung cấp kích thước vùng đệm mặc định.
- Nạp chồng tất cả phương thức của lớp OutputStream và không đưa vào bất kỳ phương thức nào.

6.9. Lớp RandomAccessFile

- Sử dụng object RandomAccessFile để truy cập ngẫu nhiên nội dung một file.
- RandomAccessFile là class thực thi 2 interface là DataInput và DataOutput trong đó có định nghĩa các phương thức input/output.

Dùng phương thức :

- **seek(vị_trí)**: để di chuyển con trỏ file tới vị_trí mới.
- **seek(0)**: Di chuyển con trỏ tới đầu file
- **seek(file.length())**: Di chuyển con trỏ tới cuối file

Ví dụ:

```
try {
    RandomAccessFile file = new RandomAccessFile("rand.txt", "rw");
    file.writeChar('a');file.writeInt(100);file.writeDouble(8.75);
    file.seek(0);  //Dich chuyen con tro ve dau file va doc du lieu
    System.out.println(file.readChar());
    System.out.println(file.readInt());
    System.out.println(file.readDouble());
    file.seek(2);  //Dich chuyen con tro file vao vi tri thu 2
    System.out.println("Vi tri thu 2: " + file.readInt());
    file.seek(file.length()); //Dich chuyen con tro toi cuoi file
    file.writeBoolean(true);
    file.seek(4);          //Dich chuyen con tro file vao vi tri thu 4
    System.out.println("Vi tri thu 4: " + file.readBoolean());
    file.close();
} catch (Exception e) {
    System.out.println("Co loi: " + e);
}
```

6.10. Lớp ObjectInputStream/ObjectOutputStream

- **Lớp ObjectOutputStream trong java** được sử dụng để ghi các kiểu dữ liệu nguyên thuỷ và các đối tượng Java vào một OutputStream. Chỉ có các đối tượng implements giao tiếp java.io.Serializable mới có thể được ghi vào stream.
- **Lớp ObjectInputStream trong java** được sử dụng để đọc các đối tượng và dữ liệu nguyên thủy mà được ghi bằng việc sử dụng lớp ObjectOutputStream.

Ví dụ:

```
import java.io.Serializable;
public class Student implements Serializable {
   private static final long serialVersionUID = 1L;
   private int id;
   private String name;
   private String address;
   private int age;
   public void Studet() {}
   public Student(int id, String name, String address, int age) {
      super();
      this.id = id;
      this.name = name;
      this.address = address;
      this.age = age;
   }
 public int getId() {
      return id;
   }
   public void setId(int id) {
      this.id = id;
   }
public String getName() {
      return name;
   }
   public void setName(String name) {
      this.name = name;
   }
   public String getAddress() {
      return address;
   }
   public void setAddress(String address) {
      this.address = address;
   }
   public int getAge() {
      return age;
   }
   public void setAge(int age) {
      this.age = age;
   }
   public String toString() {
      return "Student@[id=" + id + " , name=" + name + ", "
            + "address= " + address + ",age =" + age+ "]";
   }
}
```

```
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectOutputStream;

public class ObjectOutputStreamExample {
   public static void main(String[] args) throws IOException {
      ObjectOutputStream oos = null;
      try {
         oos = new ObjectOutputStream(new FileOutputStream("D:\\testout.txt"));
         // create student
         Student student = new Student(1, "Tran Hao Phong", "Ha Noi", 17);
         // write student
         oos.writeObject(student);
         System.out.println("Success...");
      } catch (IOException ex) {
         ex.printStackTrace();
      } finally {
         oos.close();
      }
   }
}
```

```
import java.io.FileInputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
public class ObjectInputStreamExample {
    public static void main(String[] args) throws IOException {
        ObjectInputStream ois = null;
        try {
            ois = new ObjectInputStream(new FileInputStream("D:\\testout.txt"));
            // read student
            Student student = (Student) ois.readObject();
            // show student
            System.out.println(student.toString());
        } catch (ClassNotFoundException ex) {
            ex.printStackTrace();
        } catch (IOException ex) {
            ex.printStackTrace();
        } finally {
            ois.close();
        }
    }
}
```