Byte 입출력 Stream

경복대학교
소프트웨어융합과
배희호 교수
010-2369-4112
031-570-9600
hhbae@kbu.ac.kr

Stream

- File Stream 이란?
  - Data의 입력과 출력 사이에서 중간 역할(표준화 형태)
  - 여러 입출력 장치(Monitor, HardDisk, Printer)들에 상관없이 일관된 작업(동일한 처리 속도)을 위해 사용
  - Stream은 양 방향이 아닌 단 방향
  - 단 방향 통신을 제공하기 때문에 입력, 출력 2개의 Stream이 필요 (입력에는 입력 Stream이, 출력에는 출력 Stream이 필요하다는 뜻)
  - Stream은 연속된 Data 흐름으로 입출력 처리시 다른 작업을 할 수 없는 블로킹(Blocking) 상태
  - Stream은 Byte Stream과 Character Stream으로 구분

Stream

- Byte 입출력 Stream과 Character 입출력 Stream
  - Byte 입출력 Stream
    - 입출력되는 Data를 단순 Byte의 Stream으로 처리
    - 예) Binary File을 읽는 입력 Stream
  - Character 입출력 Stream
    - 문자를 입출력 하는 Stream
    - 문자가 아닌 Binary Data는 Character Stream에서 처리하지 못함
    - 예) Text File을 읽는 입력 Stream
- JDK는 입출력 Stream을 구현한 다양한 클래스 제공

Stream

- Byte Stream(Binary Stream)
  - 1 Byte(8 bits) 단위로 입출력이 이루어 짐
  - Binary Data를 입출력하는 Stream
    - Image, Video 등을 송수신할 때 주로 사용
  - 일반적인 입력 및 출력 Program 작성
  - 그림판에서 그린 File이나 Video을 기록한 File 등은 문자로 내보내고 받을 수 없음
  - Byte 단위의 입출력을 위해 JAVA가 제공하는 클래스
    - InputStream
      - Byte 단위 읽기 위한 최상위 클래스
    - OutputStream
      - Byte 단위 쓰기 위한 최상위 클래스

Stream

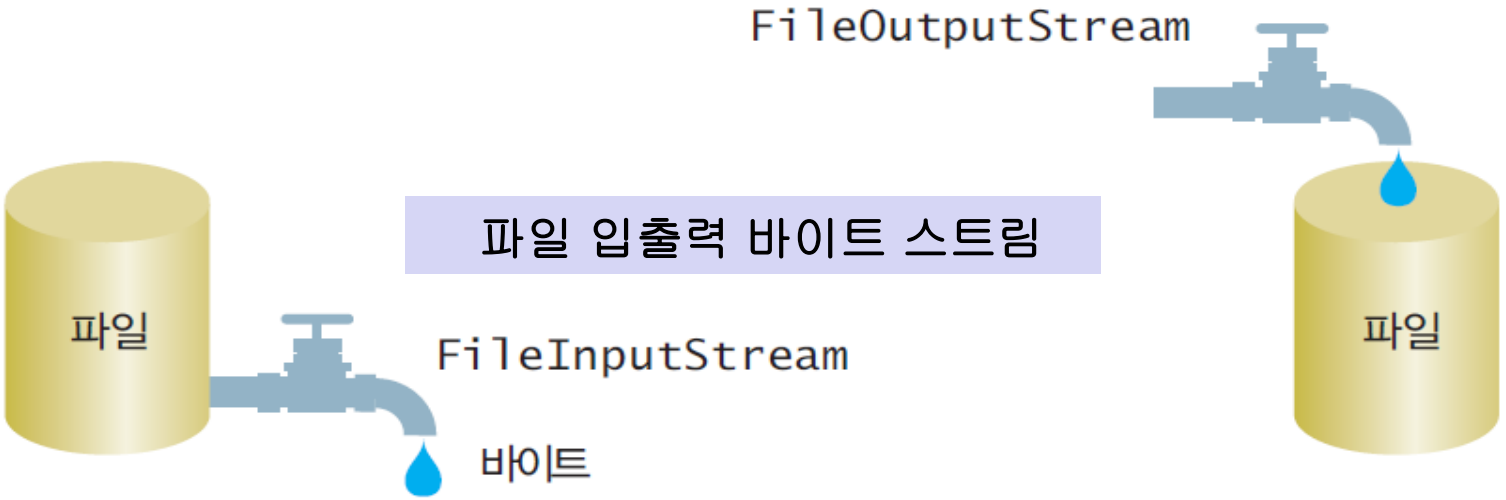
- Character Stream
  - 2 Byte Data(글자(문자)) 단위로 입출력이 이루어 짐
  - 말 그대로 Text Data를 입출력 하는데 사용하는 Stream
  - HTML 문서, Text File을 송수신할 때 주로 사용
  - 기본적으로 Character 단위 Stream은 UTF-8 Encoding 이 되어 있음
  - 한글을 사용하는 입력 또는 출력 Program을 작성할 경우 사용
  - 이것은 메모장 같은 곳에서 바로 확인 할 수 있음
  - 예) Text File을 읽는 입력 Stream
  - 문자 단위의 입출력을 위해 JAVA가 제공하는 클래스
    - Reader
      - 문자 단위 읽기 위한 최상위 클래스
    - Writer
      - 문자 단위 쓰기 위한 최상위 클래스

Stream

- Byte Stream과 Character Stream
  - Text를 8 bits씩 잘라 보내는 것은 개념적으로 불안정, 따라서, Text는 Character Stream
  - Image이나 Sound 등 Text 외의 Data는 Byte Stream으로 처리가 바람직함

Byte Stream

- 8 Bit의 Byte 단위로 입출력을 수행하는 Stream
- Binary(바이너리) Data와 숫자를 읽고, 쓰기 위해 사용
  - 예) 그림 File, 동영상 File, Image의 입출력뿐만 아니라 문자로 구성되어 있는 Text File도 입출력 할 수 있음
- 모든 Byte Stream은 클래스 이름에 InputStream(입력)과 OutputStream(출력)에서 파생

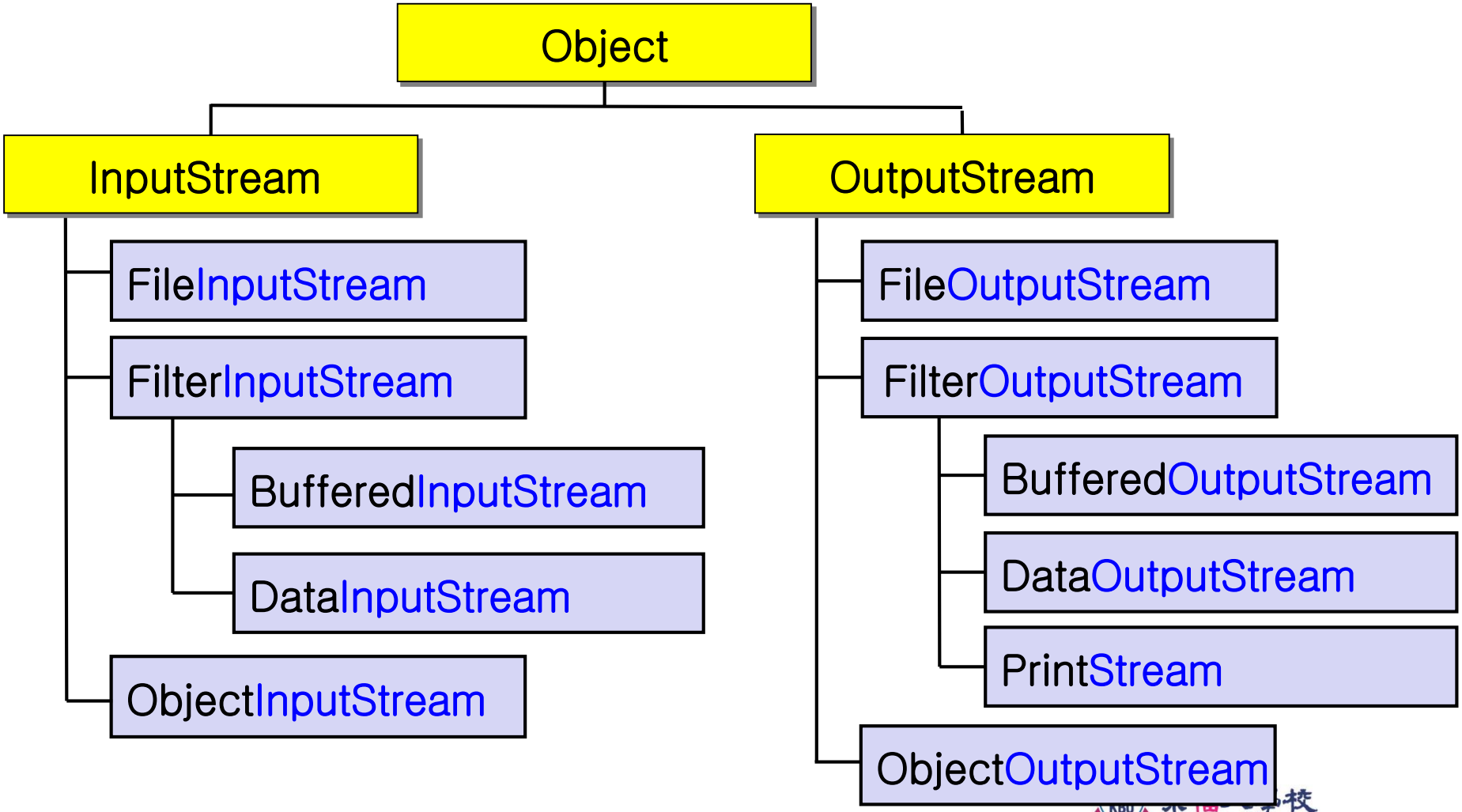


Byte Stream

- Byte Stream
  - Byte 단위의 Binary 값을 읽고 쓰는 Stream
- Byte Stream 클래스
  - java.io 패키지에 포함
  - InputStream/OutputStream
    - 추상 클래스
    - Byte Stream을 다루는 모든 클래스의 Super 클래스
  - FileInputStream/FileOutputStream
    - File로부터 Byte 단위로 읽거나 저장하는 클래스
    - Binary File의 입출력 용도
  - DataInputStream/DataOutputStream
    - JAVA의 기본 Data 타입의 값(변수)을 Binary 값 그대로 입출력
    - 문자열도 Binary 형태로 입출력

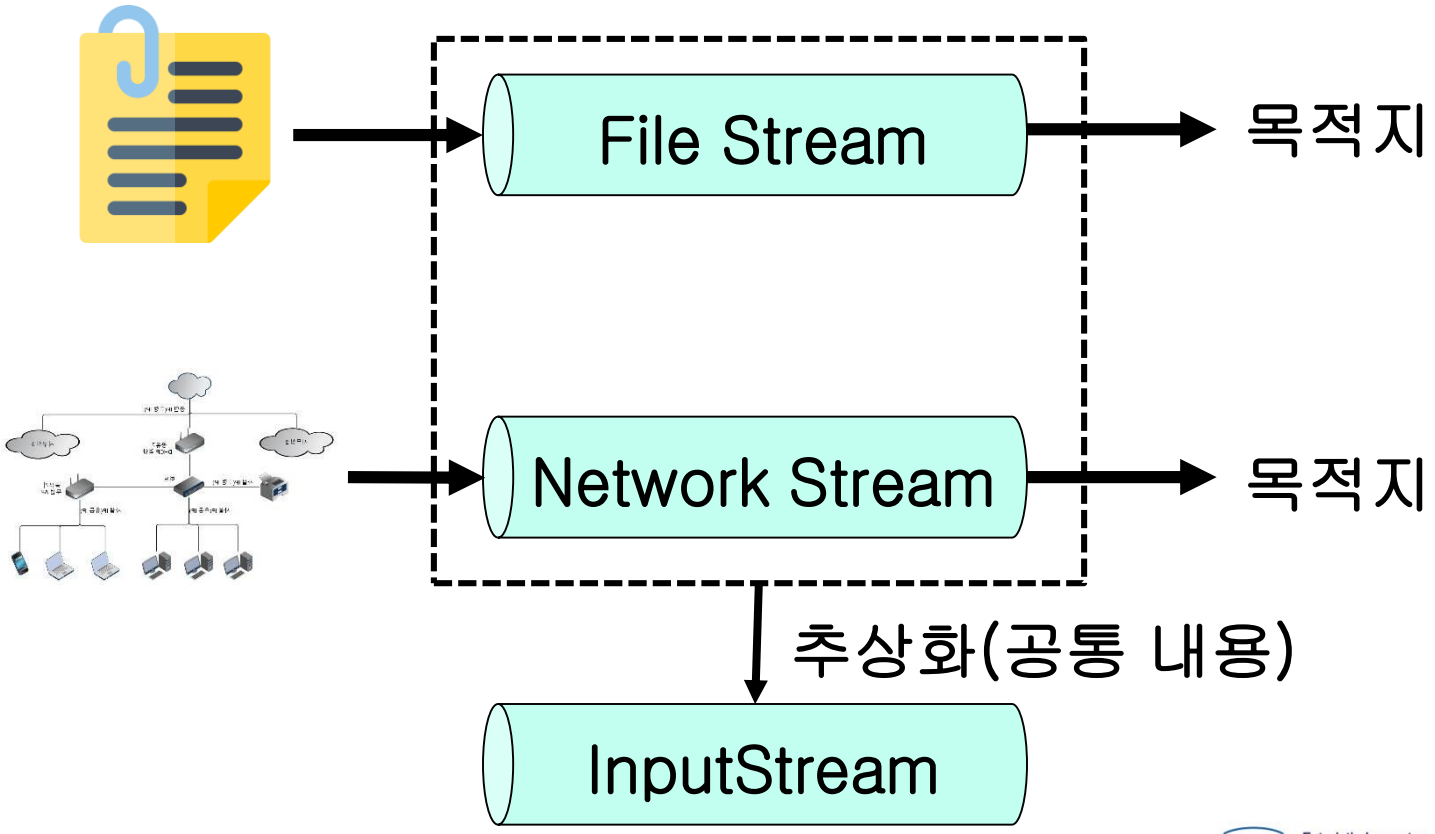
Byte Stream

- Byte Stream을 지원하는 클래스의 구성도



Byte Stream

- InputStream
  - Data가 들어오는 통로의 역할에 관해 규정하고 있는 추상 클래스



Byte Stream

- InputStream
  - InputStream은 Data를 Byte 단위로 읽어 들이는 통로 (읽어 들인 Data를 Byte로 돌려줌)

- InputStream의 메소드
  - Data 읽기 (read())
  - 특정 시점으로 되돌아가기 (mark(), reset(), markSupported())
  - 얼마나 Data가 남았는지 보여주기 (available())
  - 연결 끊기 (close())

Byte Stream

- InputStream

클래스	설 명	Stream
InputStream	바이트 입력 스트림을 위한 추상 클래스	2차
FileInputStream	파일에서 바이트를 읽어 들여 바이트 스트림으로 변환	1차
PipedInputStream	PipedOutputStream에서 읽어 들임	1차
FilterInputStream	필터 적용(filtered) 바이트 입력을 위한 추상 클래스	2차
LineNumberInputStream	바이트 입력 시 라인 번호를 유지(비추천)	2차
DataInputStream	기본 자료형 데이터를 바이트로 입력	2차
BufferedInputStream	바이트 버퍼 입력	2차
PushbackInputStream	읽어들인 바이트를 되돌림(pushback)	2차
ByteArrayInputStream	바이트 배열에서 읽어들임	1차
SequenceInputStream	서로 다른 InputStream을 입력 받은 순서대로 이어 줌	2차
StringBufferInputStream	문자열에서 읽어 들임 (비추천)	1차
ObjectInputStream	객체로 직렬화된 데이터를 역직렬화 하여 읽음	2차

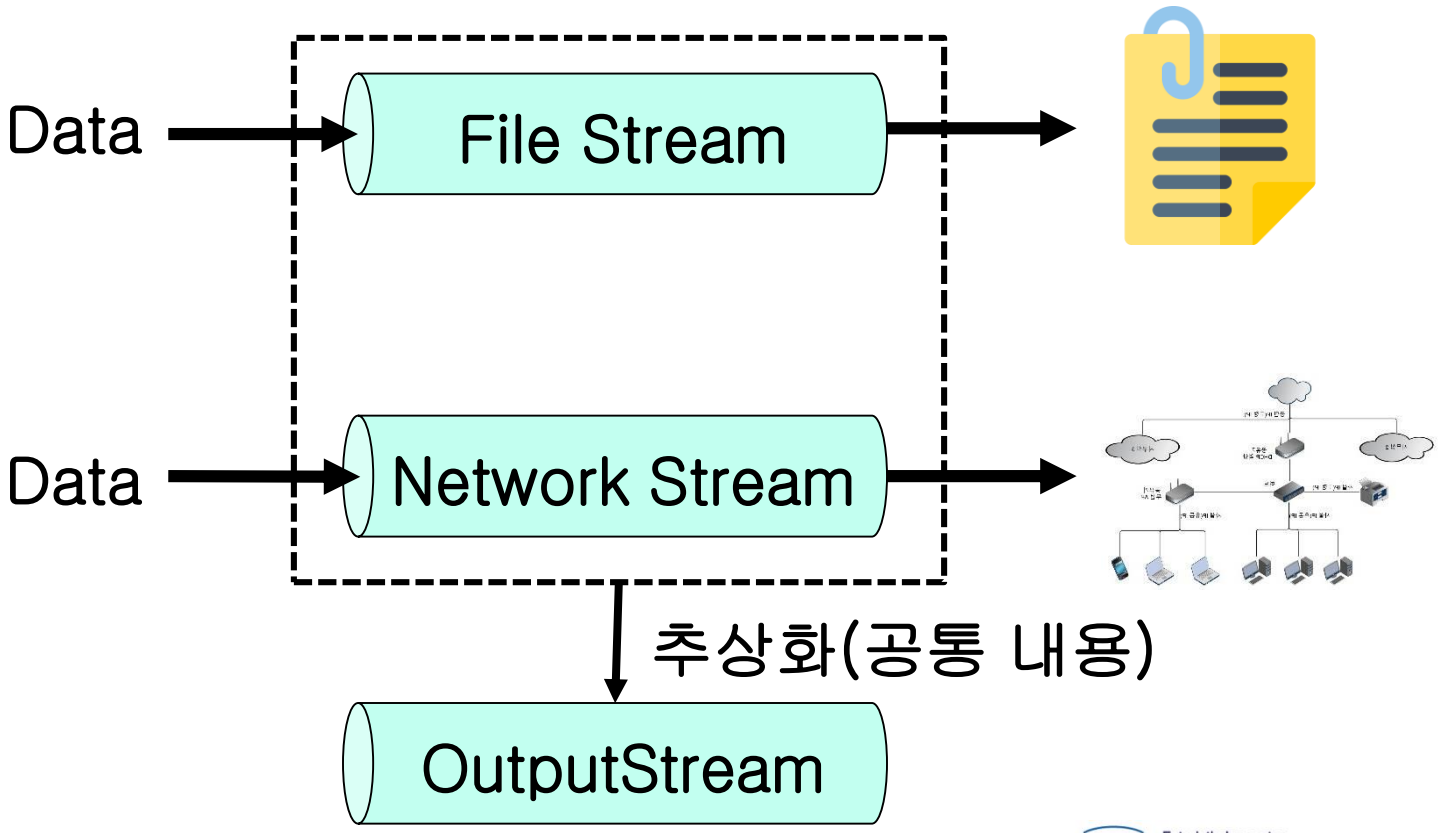
Byte Stream

- InputStream

메소드	설명
read()	입력 스트림으로부터 1 byte를 읽고 읽은 Byte를 반환
read(byte[ ] b)	입력 스트림으로부터 읽은 Byte들을 매개값으로 주어진 Byte 배열 b에 저장하고 실제로 읽은 Byte 수를 반환
read(byte[] b, int off, int len)	입력 스트림으로부터 len개의 바이트만큼 읽고 매개값으로 주어진 바이트 배열 b[off]부터 len개까지 저장. 그리고 실제로 읽은 바이트 수인 len개를 반환 만약 len개를 모두 읽지 못하면 실제로 읽은 바이트 수를 반환
close()	사용한 시스템 자원을 반납하고 입력 스트림을 닫음

Byte Stream

- OutputStream
  - Data가 출력되는 통로의 역할에 관해 규정하고 있는 추상 클래스



Byte Stream

- OutputStream

클래스	설 명	Stream
OutputStream	바이트 출력 스트림을 위한 추상 클래스	2차
FileOutputStream	바이트 스트림을 바이트 파일로 변환	1차
PipedOutputStream	PipedOutputStream에 출력	1차
FilterOutputStream	필터 적용(filiterd) 바이트 출력을 위한 추상 클래스	2차
DataOutputStream	바이트를 기본자료형으로 출력	2차
BufferedOutputStream	바이트 스트림에 버퍼 출력	2차
PrintStream	Stream 값과 객체를 프린트	2차
ByteArrayOutputStream	바이트 스트림에 바이트 배열 출력	1차
ObjectputStream	데이터를 객체로 직렬화 하여 출력	2차

Byte Stream

- OutputStream

메소드	설명
write(int b)	출력 스트림으로부터 1바이트를 보냄
write(byte[ ] b)	출력 스트림으로부터 주어진 바이트 배열 b의 모든 바이트를 보냄
write(byte[ ] b, int off, int len)	출력 스트림으로 주어진 바이트 배열 b[off]부터 len개까지의 바이트를 보냄
flush()	버퍼에 잔류하는 모든 바이트를 출력
close()	사용한 시스템 자원을 반납하고 출력 스트림을 닫음

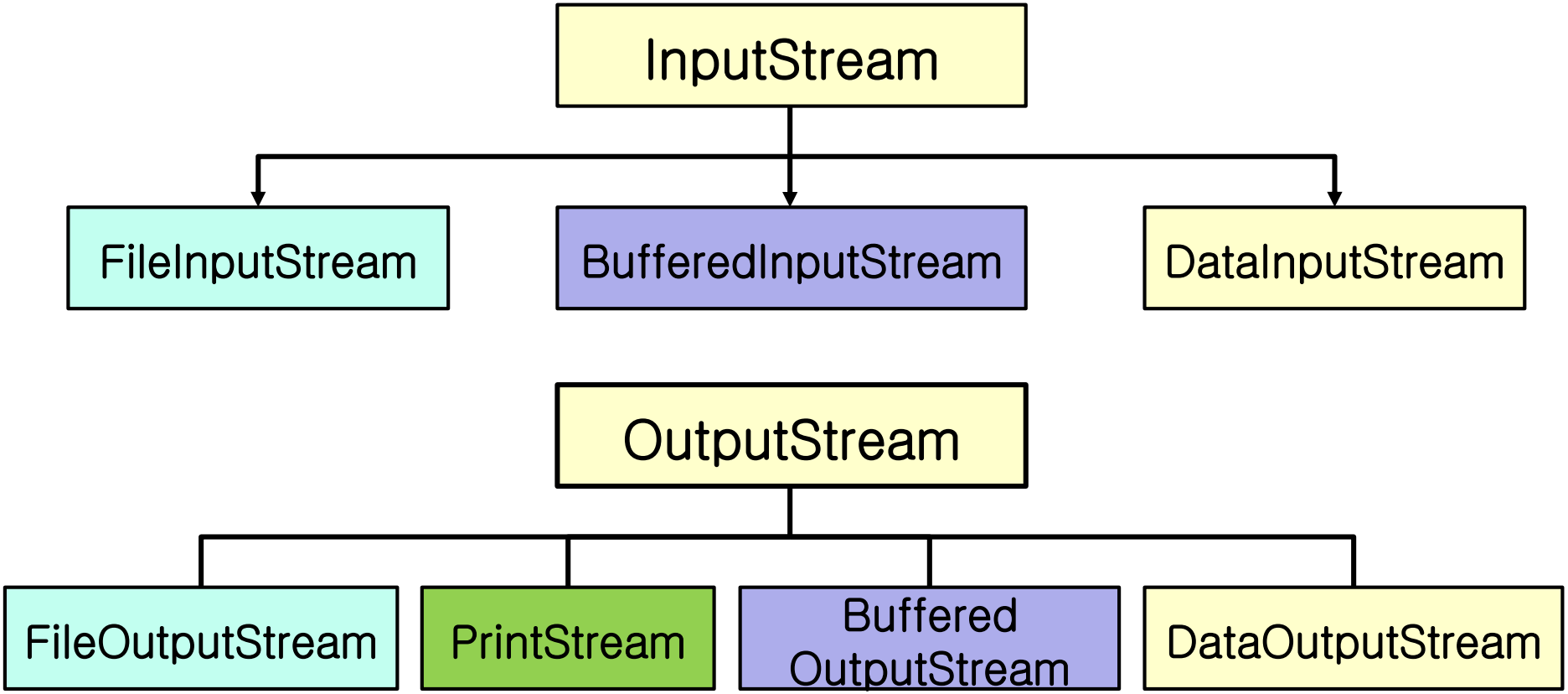
Byte Stream

- Byte Stream
  - InputStream / OutputStream
    - Byte 기반 input/output stream의 최고 조상
  - ByteArrayInputStream / ByteArrayOutputStream
    - byte array(byte[])에 대한 Data를 입출력 하는 클래스
  - FileInputStream / FileOutputStream
    - File에 대한 Data를 입출력 하는 클래스

- Character Stream
  - Reader / Writer
    - Character 기반 input / output stream의 최고 조상
  - FileReader / FileWriter
    - 문자 기반의 File을 입출력 하는 클래스

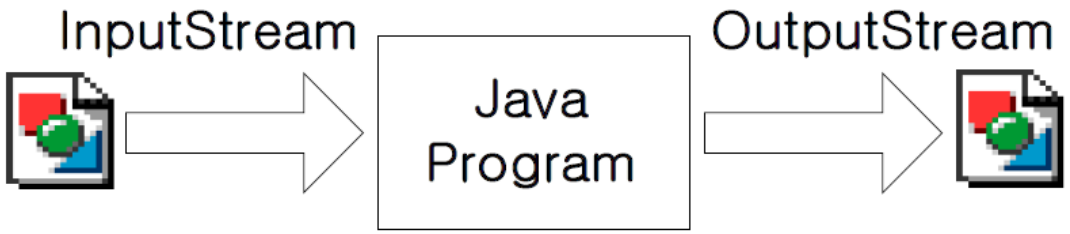
Byte Stream

- InputStream과 OutputStream



Byte Stream

- InputStream / OutputStream 클래스
  - java.io 패키지에 포함
  - 추상 클래스
  - Byte 입출력 처리를 위한 기능을 가진 슈퍼 클래스
  - InputStream 클래스
    - 입력으로(Keyboard, File 등)부터 Data를 읽어오는 메소드들을 제공
  - OutputStream 클래스
    - Byte Stream을 출력하는 메소드들을 제공



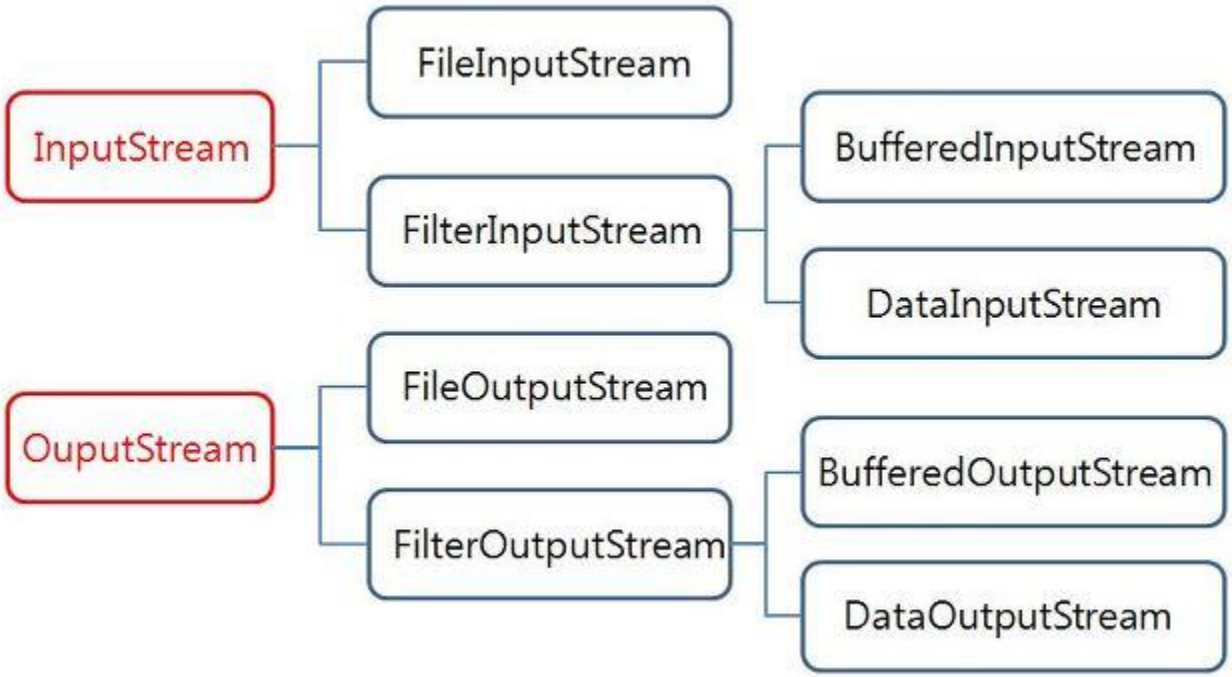
Byte Stream

- FileInputStream / FileOutputStream 클래스
  - File 입출력을 위한 클래스
  - File로부터 Binary Data를 읽거나 File에 Binary Data를 저장할 수 있음

- DataInputStream / DataOutputStream 클래스
  - boolean, char, byte, int, double 등과 같은 값을 Binary 형태로 입출력
  - 문자열도 Binary 형태로 입출력 할 수 있음

InputStream/OutputStream 클래스

- System.in 표준 입력 Stream 객체의 메소드를 호출함으로써 한 문자를 입력하거나 행 단위 문자열을 입력할 수 있음
- java.io Package의 InputStreamReader 클래스와 BufferedReader 클래스를 import 함



InputStream/OutputStream 클래스

- 초기의 JAVA는 Byte 단위로 입출력을 지원하였음
- 이것은 가장 원시적인 형태의 입출력이며 아무런 변환 작업을 수행하지 않음
- Binary 형태의 자원을 다루기 적합하며 InputStream과 OutputStream의 두 클래스가 최상위 클래스
- InputStream과 OutputStream 클래스는 둘 다 abstract 지정자를 가지고 있는 추상 클래스
- InputStream과 OutputStream은 객체를 직접 만들지 못하는 클래스로 대부분 추상 메소드를 정의하여 모든 후손 클래스가 이를 강제로 구현하게 만드는 통일의 의미로 사용

InputStream/OutputStream 클래스

- 방법(처리 순서)
  - read( ) 메소드 호출
    - 한 문자 입력
    - 입력한 한 문자를 반환하고 입력이 끝([Ctrl]+[Z])이면 –1을 반환
    - data 변수의 Data형은 int

```
int data = System.in.read( );      // 한 개의 문자를 입력하여 반환
```

  - readLine( ) 메소드 호출
    - 문자열을 입력
    - 입력이 끝인 경우는 null을 반환

```
String address = rd.readLine( );          // 문자열 입력
```

InputStream 클래스

- 주요 메소드

메소드	설명
int available() throws IOException	현재 읽기 가능한 바이트의 수를 반환
void close() throws IOException	입력 스트림을 닫음
abstract int read() throws IOException	입력 스트림으로부터 한 바이트를 읽어 int 값으로 반환 더 이상 읽을 값이 없을 경우 -1을 반환
int read(byte buffer[]) throws IOException	입력 스트림으로부터 buffer[] 크기만큼을 읽어 buffer 배열에 저장하고 읽은 바이트 수를 반환
int read(byte buffer[], int offset, int numBytes) throws IOException	입력 스트림으로부터 numBytes만큼을 읽어 buffer[]의 offset 위치에 저장하고 읽은 바이트 수를 반환
int skip(long numBytes) throws IOException	numBytes로 지정된 바이트를 스킵(skip)하고 스킵된 바이트 수를 반환

InputStream 클래스

- 주요 메소드

메소드	설명
void mark(int numBytes)	입력 Stream의 현재의 위치에 mark 함
boolean markSupported()	현재의 입력 Stream이 mark()와 reset()을 지원하면 true를 반환
void reset()	입력 Stream의 입력 시작 부분을 현재의 위치에서 가장 가까운 이전의 mark 위치로 설정

OutputStream 클래스

- 주요 메소드

메소드	설명
void close() throws IOException	출력 스트림을 닫음
void flush() throws IOException	버퍼에 남아 있는 출력 스트림을 모두 출력
void write(int i) throws IOException	정수 i의 하위 8비트를 출력
void write(byte buffer[]) throws IOException	buffer의 내용을 출력
void write(byte buffer[], int index, int size) throws IOException	buffer의 index위치부터 size만큼의 바이트를 출력

InputStream 예제

```
public static void main(String[] args) throws IOException {
    InputStream keyboard = System.in;

    System.out.print("입력 : ");
    try {
        int code = keyboard.read();
        System.out.println((char) code + " : " + code);
    } catch (IOException e) {
        System.err.println(e.getMessage());
    }
    keyboard.close();
}
```

```
입력 : a
a : 97
```

- ✓ InputStream은 입력 받을 수 있는 객체로 1 byte 단위 처리 스트림
- ✓ System.in은 InputStream 타입의 System 클래스의 static final 상수
- ✓ 키보드와 연결된 InputStream 타입의 참조값을 keyboard라는 지역변수에 담기
- ✓ 따라서 Scanner 객체에 참조값을 전달한 것
- ✓ InputStream은 입력받을 수 있는 객체로, 1byte단위 처리 스트림
- ✓ 영문자, 숫자, 특수문자만 처리 가능, 한글 처리 불가

OutputStream 예제

```
public static void main(String[] args) {
    OutputStream output = System.out;

    try {
        output.write(97);
        output.write('A');
        output.write('1');
        output.write('가');
        output.flush();   //방출하기
        output.close();
    } catch (IOException e) {
        System.err.println(e.getMessage());
    }
}
```

- ✓ System 클래스의 out이라는 static 필드에는 콘솔창에 출력할 수 있는 PrintStream 객체의 참조값이 들어있음
- ✓ OutputStream도 1byte 처리 스트림
- ✓ OutputStream은 한글 출력 불가
- ✓ 끝 내려면 <Ctrl> + <D>

InputStream/OutputStream 예제

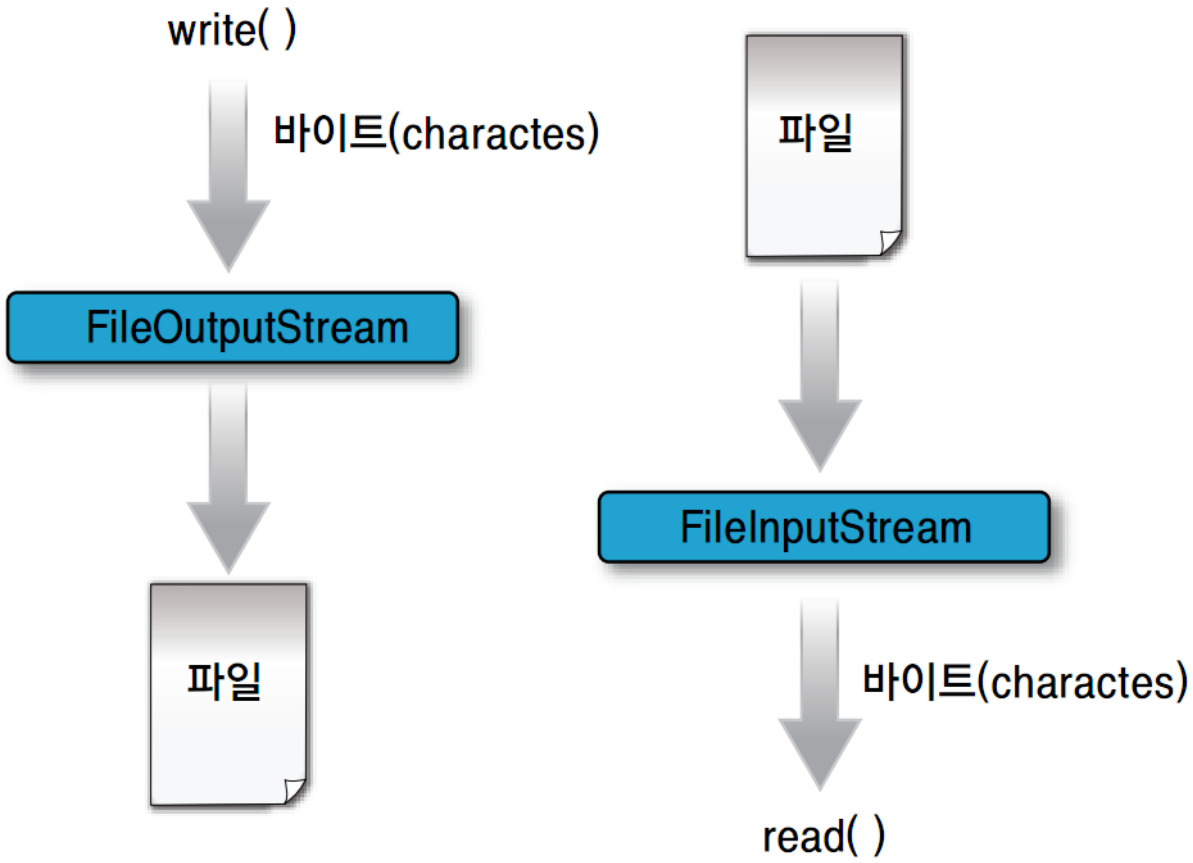
```
public static void main(String[] args) throws IOException {
    InputStream keyboard = System.in;
    PrintStream monitor = System.out;
    OutputStream ouput = monitor;
    int data;
    int cnt = 0;
    try {
        while ((data = keyboard.read()) != -1) {
            cnt++;
            ouput.write((char) data);   // 입력 받은 내용을 표준 출력
        }
    } catch(IOException e) {
        System.err.println(e.getMessage());  // 에러 발생시 에러를 출력
    }
    monitor.println();
    monitor.println(" 총 bytes : " + cnt);
    keyboard.close();
    ouput.close();
    monitor.close();
}
```

FileInputStream/FileOutputStream 클래스

- 직접 Keyboard를 통하여 입력하는 Data는 대개 임시 자료인 경우가 많음
- 중요한 Data는 대부분 Database에 저장되어 있거나 File System에 저장 됨
- FileInputStream과 FileOutputStream은 Byte 단위로 File을 통한 입출력을 처리

FileInputStream/FileOutputStream 클래스

- FileInputStream 클래스와 FileOutputStream 클래스 역할



FileInputStream 클래스

- InputStream 클래스로부터 상속된 클래스
- File로부터 입력을 받을 수 있는 기능을 제공

- 생성자

생성자	설명
FileInputStream(File file)	file이 지정하는 파일로부터 입력 받는 FileInputStream 생성
FileInputStream(String name)	name이 지정하는 파일로부터 입력 받는 FileInputStream 생성

FileInputStream 클래스

- 주요 메소드

메소드	설명
int read()	입력 스트림에서 한 바이트를 읽어 int형으로 반환
int read(byte[] b)	최대 배열 b의 크기만큼 바이트를 읽음 실제 읽은 바이트 수 반환
int read(byte[] b, int off, int len)	최대 len개의 바이트를 읽어 b 배열의 off 위치에 저장, 실제 읽은 바이트 수 반환
int available()	입력 스트림에서 현재 읽을 수 있는 바이트 수 반환
void close()	입력 스트림을 닫고 관련된 시스템 자원 해제
void reset()	스트림 리셋, 스트림이 마킹되어 있으면 그 위치부터 시작

FileInputStream 클래스

- File로부터 Byte 단위로 Data를 읽는 클래스
- 사용 방법
  - File을 연다
    - FileReader 클래스와 동일 방법
  - File로부터 Data를 읽음

```
while (true) {
    int data = inputStream.read();
    if (data < 0)
        break;
    byte b = (byte) data;
    ...
}
```

데이터를 읽어서

-1이면 반복을 중단하고

아니면 byte 타입으로 캐스트

  - File을 닫음
    - FileReader 클래스와 동일 방법

FileInputStream 클래스

- 한꺼번에 여러 Byte를 읽는 read() 메소드

FileInputStream 예제 1

- File의 내용을 읽어서 16진수로 출력하는 Program

```
public static void main(String[] args) {
    String path = ".₩₩data₩₩test.txt";

    try {
        FileInputStream inputStream = new FileInputStream(path);
        byte[] arr = new byte[16];
        while (true) {
            int num = inputStream.read(arr);
            if (num < 0)
                break;
            for (int cnt = 0; cnt < num; cnt ++)
                System.out.printf("%02X ", arr[cnt]);
            System.out.println();
        }
        inputStream.close();
```

FileInputStream 예제 1

```
    } catch (FileNotFoundException e) {
        System.out.println(" 파일이 존재하지 않습니다.");
    } catch (IOException e) {
        System.out.println(" 파일을 읽을 수 없습니다.");
    }
}
```

FileInputStream 예제 2

- File 전체를 읽어 화면에 출력하는 Program

```
public static void main(String[] args) throws IOException {
    String path = ".₩₩data₩₩test.txt";

    FileInputStream inputStream = new FileInputStream(path);
    int ch;

    while ((ch = inputStream.read()) != -1) {
        System.out.print((char) ch);
    }
    System.out.printf("₩n지정한 파일로부터 바이트 스트림을 입력받아
                                                화면에 출력하였습니다.");

    inputStream.close();
}
```

./data/test.txt 파일을 열고 파일과 입력 바이트 스트림 객체 inputStream을 연결

파일 끝까지 바이트씩 ch에 읽어 들임.
파일의 끝을 만나면 read()는 -1(EOF) 반환

바이트 ch를 문자로 변환하여 화면에 출력

스트림을 닫음. 파일도 닫힘.
스트림과 파일의 연결을 끊음.
더 이상 스트림으로부터 읽을 수 없음

FileInputStream 예제 2

- File로부터 1 Byte를 읽어 들여 int형의 변수에 저장
- int형 변수의 값을 출력
- 8 Bit 단위의 입출력이고 입출력된 내용이 int형에 저장되었기 때문에 $2^8 - 1 = 255$까지 출력됨을 볼 수 있음
- 일반적인 Byte 단위의 수 표현에서는 가장 왼쪽 Bit를 부호 Bit로 취급하므로 $2^7 - 1 = 127$까지만 표현이 가능

FileOutputStream 클래스

- OutputStream 클래스로부터 상속된 클래스
- File에 Byte Stream을 출력할 수 있는 기능을 제공

FileOutputStream 클래스

생성자

생성자	설명
FileOutputStream(File file)	file이 지정하는 파일에 출력하는 FileOutputStream 생성
FileOutputStream(String name)	name이 지정하는 파일에 출력하는 FileOutputStream 생성
FileOutputStream (File file, boolean append)	FileOutputStream을 생성하며 append가 true이면 file이 지정하는 파일의 마지막부터 데이터 저장
FileOutputStream (String name, boolean append)	FileOutputStream을 생성하며 append가 true이면 name 이라는 이름을 가진 파일의 마지막부터 데이터 저장

FileOutputStream 클래스

- 주요 메소드

메소드	설명
void write(int b)	int 형으로 넘겨진 한 바이트를 출력 스트림으로 출력
void write(byte[] b)	배열 b의 바이트를 모두 출력 스트림으로 출력
void write (byte[] b, int off, int len)	len 크기만큼 off부터 배열 b를 출력 스트림으로 출력
void flush()	출력 스트림에서 남아 있는 데이터 모두 출력
void close()	출력 스트림을 닫고 관련된 시스템 자원 해제

FileOutputStream 클래스

- Byte Data를 File에 쓰는 클래스
  - 사용 방법
    - 1단계 : File을 연다
      - 방법은 FileWriter 클래스와 동일
    - 2단계 : File에 Data를 쓴다

```
outputStream.write(71);
```

71의 비트 패턴인 01000111을 갖는 하나의 바이트를 파일로 출력

    - 3단계 : File을 닫음
      - 방법은 FileWriter 클래스와 동일

FileOutputStream 예제 0

- 바이너리 값을 파일에 저장하는 바이트 스트림 코드

```
FileOutputStream fout = new FileOutputStream("c:₩₩test.out");

int num[]={1,4,-1,88,50};
byte b[]={7,51,3,4,1,24};

for(int i=0; i<num.length; i++)
   fout.write(num[i]);

fout.write(b);

fout.close();
```

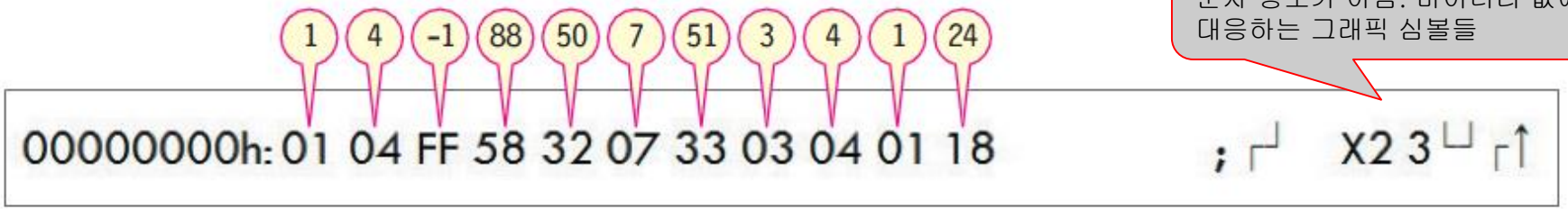
c:₩test.out 파일을 열고, 출력 바이트 스트림인 객체와 연결

파일에 배열 num[i]의 정수 값(바이너리)을 그대로 기록

파일에 바이트 배열(바이너리) 값을 그대로 기록

스트림을 닫음. 파일도 닫힘. 더 이상 스트림으로부터 읽을 수 없음

파일에 있는 각 바이너리 값들은 문자 정보가 아님. 바이너리 값에 대응하는 그래픽 심볼들



test.out 파일의 내부

FileOutputStream 예제 1

```
public static void main(String[] args) {
    String path = ".₩₩data₩₩output.txt";

    try {
        byte[] arr = {72, 65, 80, 80, 89, 5, 6, 7, 8, 9, 10,
                11, 12, 13, 14, 15, 16, 17, 18, 19, 33, 45, 97};
        FileOutputStream output = new FileOutputStream(path);
        for (int cnt = 0; cnt < arr.length; cnt++)
            output.write(arr[cnt]);
        output.close();
    } catch (IOException e) {
        System.out.println("파일로 출력할 수 없습니다.");
    }
}
```

- ✓ 기존의 파일이 없으면 만들어지고 있으면 덮어쓰게 되어 기존 파일 내용이 지워짐 (파일이 존재하면 삭제 후 생성되기 때문에 기존의 파일이 지워 짐)
- ✓ Byte에 저장된 데이터는 ASCII 코드면 확인 가능함

FileOutputStream 예제 2

```
public static void main(String[] args) throws IOException {
    String path = ".₩₩data₩₩test.txt";

    String message = " 파일에 저장될 문자열 입니다.₩n Hellow world !!";
    FileOutputStream outputStream = new FileOutputStream(path, true);
    outputStream.write(message.getBytes());

    outputStream.close();
}
```

- ✓ 기존 파일에 내용을 추가 하려면 두번째 인자로 true를 적어 줌. true를 추가해도 파일이 없으면 만듦
- ✓ 문자열에서 getBytes() 메소드로 byte 단위로 읽어서 사용

FileOutputStream 예제 2

```
File file = new File(path);
if (file.exists()) {
    FileInputStream inputStream = new FileInputStream(path);
    byte[] buffer = new byte[inputStream.available()];

    int size = inputStream.read(buffer);
    message = new String(buffer, 0, size);

    System.out.printf("읽은 바이트 수[%,d] :
                                    ₩n읽은 내용 :  ₩n%s ₩n", size, message);
    inputStream.close();
} else {
    System.err.println("입력 파일이 존재하지 않아요");
}
```

- ✓ 앞의 Program 밑에 추가하여 보자
- ✓ 하나의 파일을 두 용도로 사용하기 때문에 사용하기 전에 close() 해주어야 함
- ✓ FileInputStream에서 파일이 존재하지 않으면 오류가 발생하므로 반드시 파일 존재 여부를 체크해 주는 것이 필요함

FileInput(Output)Stream 예제 1

```
public static void main(String[] args) throws IOException {
    String source = ".//data//kbu1.jpg";
    String target = ".//data//target.jpg";
    int data;
    int tot = 0; // 읽어온 바이트 수
    File file = new File(source);
    if (file.exists()) {
        FileInputStream fis = new FileInputStream(file);
        FileOutputStream fos = new FileOutputStream(target);
        byte[] buf = new byte[1024]; // 메모리를 일시적 저장위한 buffer
        while ((data = fis.read(buf)) != -1) {
            fos.write(buf, 0, data); // 쓰는게 아니라 스트림에 넣는느낌
            fos.flush();
            tot += data;
        }
        System.out.println(String.format("%,d 바이트가 복사 되었습니다", tot));
        fis.close(); fos.close();
    } else {
        System.err.println("입력 파일이 존재하지 않아요");
    }
}
```

FileInput(Output)Stream 예제 2

```
public static void main(String[] args) throws IOException {
    String path = ".₩₩data₩₩";

    File file = new File(path + "test.txt");
    if (file.exists()) {
        FileInputStream input = new FileInputStream(file);
        FileOutputStream output = new FileOutputStream(path + "copyFile.txt", true);
        int size = input.available();
        byte[] buffer = new byte[size];
        Date date = new Date();
        long start = date.getTime();
        output.write(buffer, 0, input.read(buffer));
        date = new Date();
        long end = date.getTime();
        System.out.println("복사 시간 : " + (end - start));
        input.close();
        output.close();
    } else {
        System.err.println("입력 파일이 존재하지 않아요");
    }
}
```

FileInput(Output)Stream 예제 3

```
public static void main(String[] args) throws IOException {
    String path = ".₩₩data₩₩";
    File file = new File(path + "test.txt");
    if (file.exists()) {
        FileInputStream input = new FileInputStream(file);
        FileOutputStream output = new FileOutputStream(path + "copyFile.txt", true);
        int data;
        Date date = new Date();
        long start = date.getTime();
        while ((data = input.read()) != EOF) {
            output.write(data);
        }
        date = new Date();
         long end = date.getTime();
        System.out.println("복사 시간 : " + (end - start));
        input.close();
        output.close();
    } else {
        System.err.println("입력 파일이 존재하지 않아요");
    }
}
```

시간이 소요됨

FileInput(Output)Stream 예제 4

```
public static void main(String[] args) throws IOException {
    String path = ".₩₩data₩₩";
    Scanner scan = new Scanner(System.in);
    System.out.print("원본 파일 이름 입력 : ");
    String inputName = scan.next();
    System.out.print("복사 파일 이름 입력 : ");
    String outputName = scan.next();
    int ch;
    File file = new File(path + inputName);
    if (file.exists()) {
        InputStream inputStream = new FileInputStream(file);
        OutputStream outputStream = new FileOutputStream(path + outputName);
        while ((ch = inputStream.read()) != EOF) {
            outputStream.write(ch);
        }
        inputStream.close();
        outputStream.close();
        System.out.println(inputName + "을 " + outputName + "로 복사하였다.");
    } else {
        System.err.println("입력 파일이 존재하자 않아요");
    }
}
```

추상 클래스

FileInput(Output)Stream 예제 5

```
public static void main(String[] args) throws IOException {
    String path = ".₩₩data₩₩log.txt";
    OutputStream outputStream = new FileOutputStream(path);
    PrintStream printStream = new PrintStream(outputStream);
    PrintStream sysout = System.out;

    sysout.println("Hello World!");
    System.err.println("ERROR!(1)");
    System.setOut(printStream);
    System.setErr(printStream);
    printStream.println("안녕하세요");
    System.err.println("ERROR!(2)");
    System.setOut(sysout);
    sysout.println("Hello World!(안녕하세요)");
    System.err.println("ERROR!(3)");

    outputStream.close();
    printStream.close();
}
```

FileInput(Output)Stream 예제 6

```
public static void main(String[] args) throws IOException {
    String source = ".₩₩data₩₩kbu1.jpg";
    String target = ".₩₩data₩₩target.jpg";
    int data;
    int tot = 0; // 읽어온 바이트 수
    File file = new File(source);
    if (file.exists()) {
        FileInputStream fis = new FileInputStream(file);
        FileOutputStream fos = new FileOutputStream(target);
        byte[] buf = new byte[1024]; // 메모리를 일시적 저장위한 buffer
        while ((data = fis.read(buf)) != -1) {
            fos.write(buf, 0, data); // 쓰는게 아니라 스트림에 넣는느낌
            fos.flush();
            tot += data;
        }
        System.out.println(String.format("%,d 바이트가 복사 되었습니다", tot));
        fis.close(); fos.close();
    } else {
        System.err.println("입력 파일이 존재하지 않아요");
    }
}
```