



## 11. 입출력 스트림과 파일 입출력

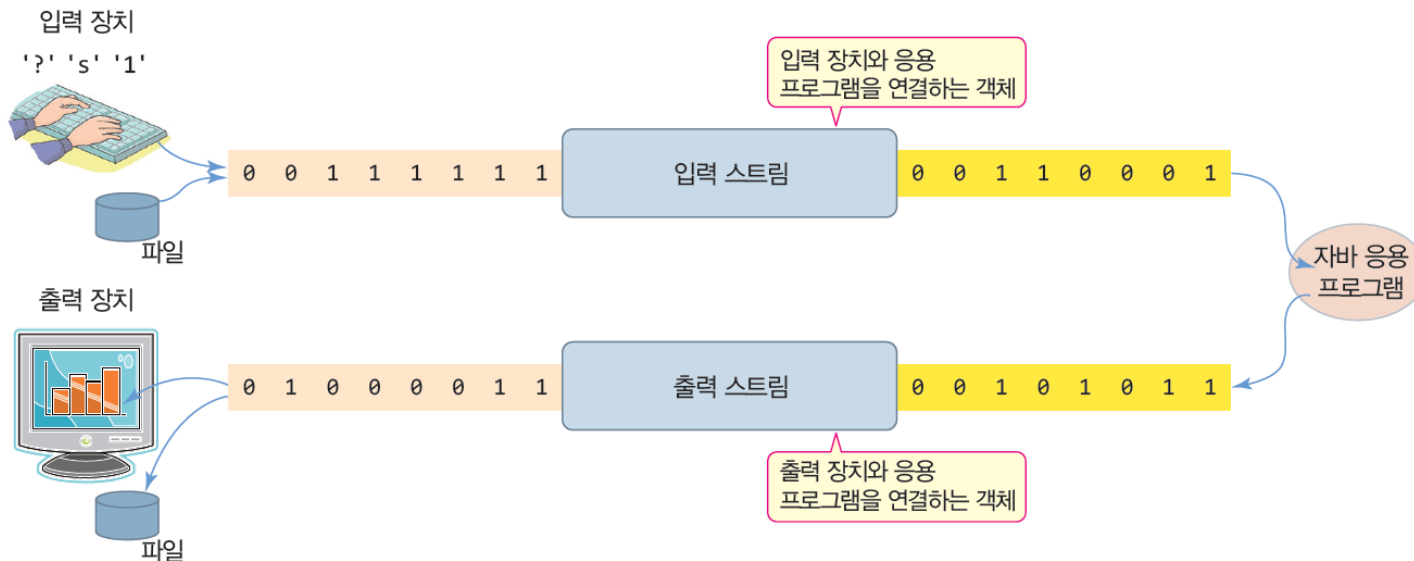
1. 자바의 입출력 스트림
2. 텍스트 파일 입출력
3. 바이너리 파일 입출력
4. **FILE** 클래스로 파일 속성 알아내기
5. 파일 복사 응용 사례

# 1. 자바의 입출력 스트림

## ❖ 자바의 입출력 스트림

- 입출력 장치와 자바 응용 프로그램 연결
  - 입력 스트림 : 입력 장치로부터 자바 프로그램으로 데이터를 전달하는 객체
  - 출력 스트림 : 자바 프로그램에서 출력 장치로 데이터를 보내는 객체
- 특징
  - 입출력 스트림 기본 단위 : 바이트
  - 단방향 스트림, 선입선출 구조

자바 프로그램 개발자는 직접 입력 장치에서 읽지 않고 입력 스트림을 통해 읽으며, 스크린 등 출력 장치에 직접 출력하지 않고 출력 스트림에 출력하면 된다.



# 자바의 입출력 스트림 종류

- 문자 스트림

- 문자만 입출력하는 스트림
- 문자가 아닌 바이너리 데이터는 스트림에서 처리하지 못함
- 문자가 아닌 데이터를 문자 스트림으로 출력하면 깨진 기호가 출력
- 바이너리 파일을 문자 스트림으로 읽으면 읽을 수 없는 바이트가 생겨서 오류 발생  
예) 텍스트 파일을 읽는 입력 스트림

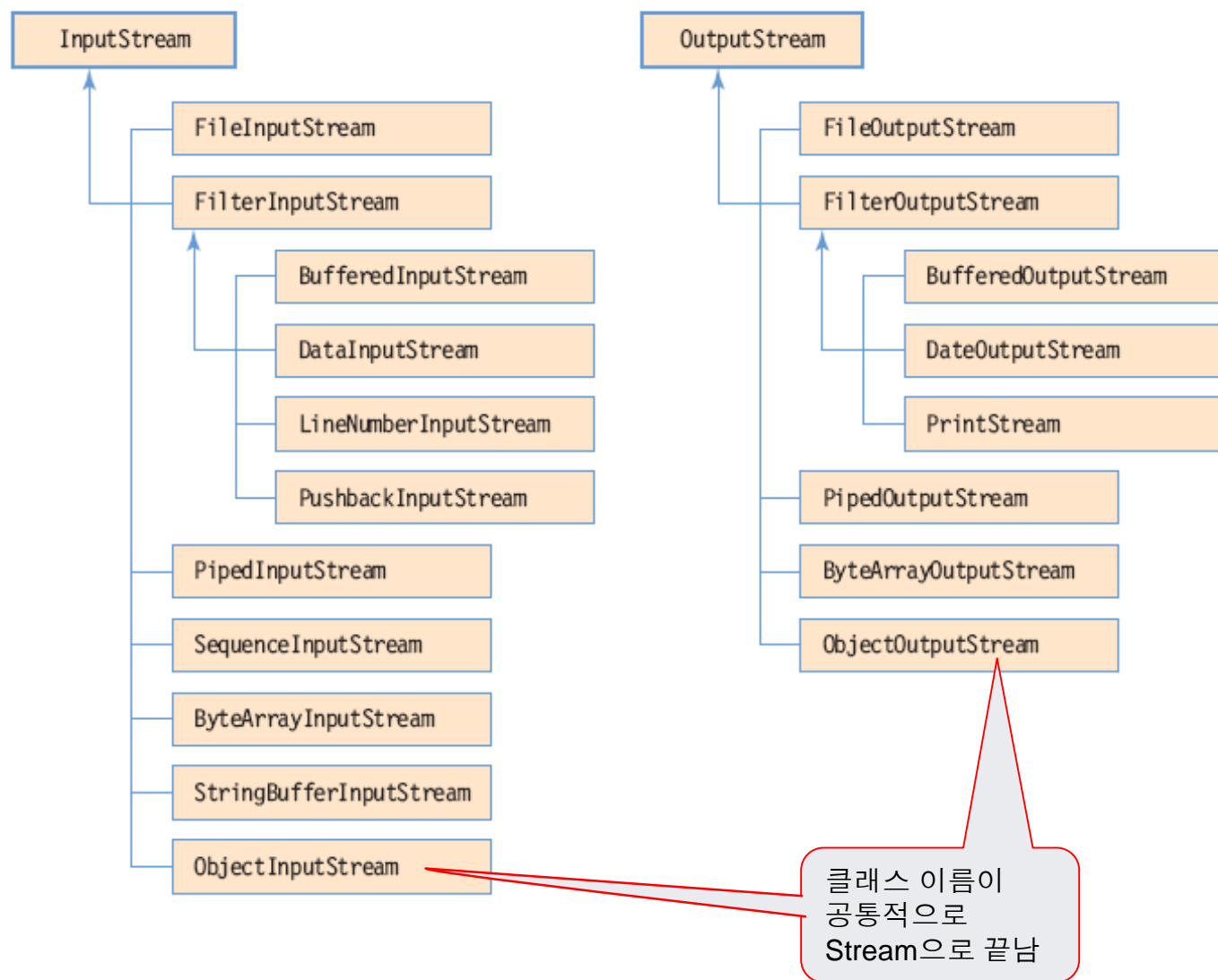
- 바이트 스트림

- 입출력 데이터를 단순 바이트의 흐름으로 처리
- 문자 데이터든 바이너리 데이터든 상관없이 처리 가능  
예) 바이너리 파일을 읽는 입력 스트림

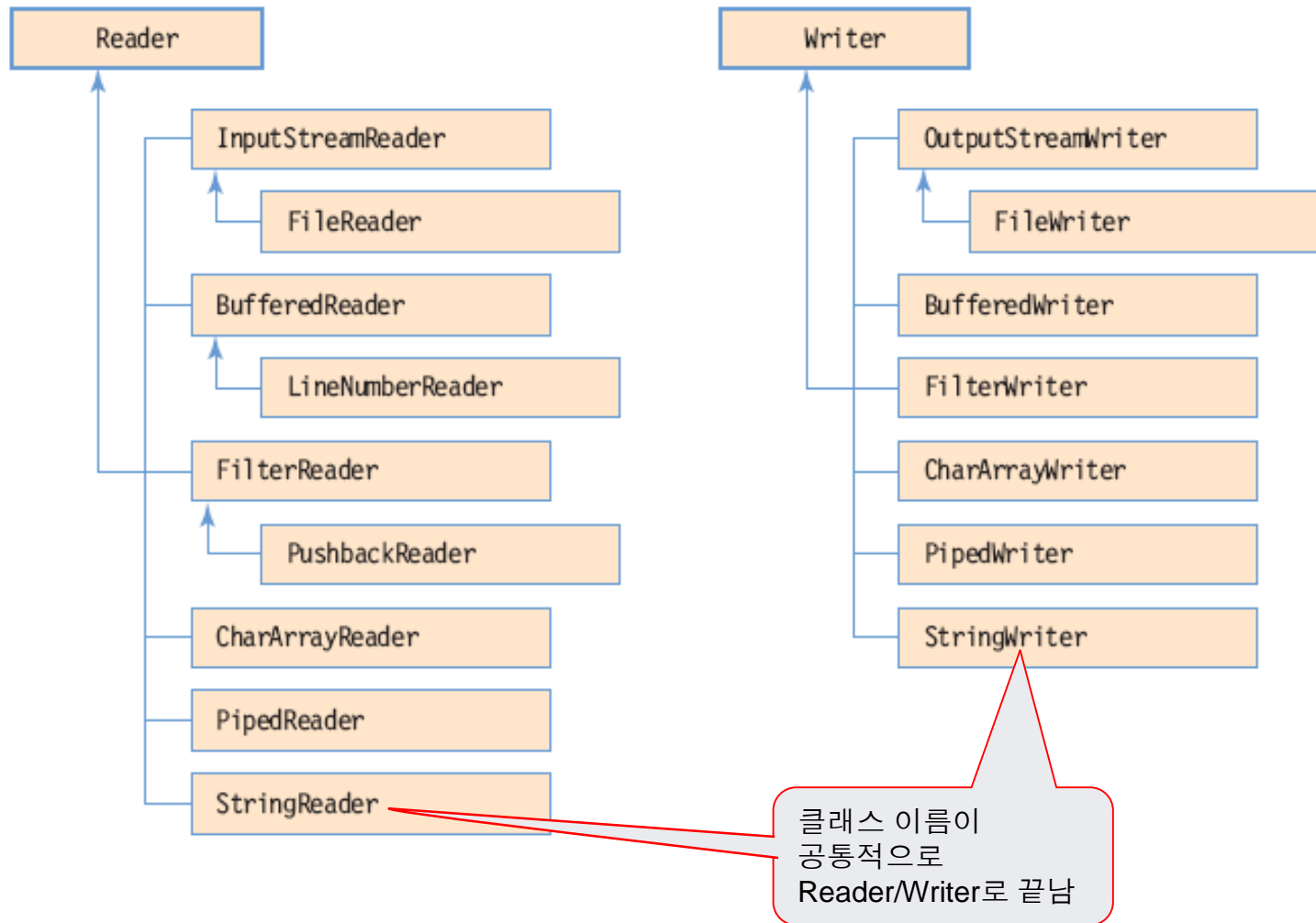
# 문자 스트림과 바이트 스트림의 흐름 비교



## JDK의 바이트 스트림 클래스 계층 구조



# JDK의 문자 스트림 클래스 계층 구조



# 스트림 연결

- 여러 개의 스트림을 연결하여 사용할 수 있음
  - 예) 키보드에서 문자를 입력받기 위해 System.in과 InputStreamReader를 연결한 코드

```
InputStreamReader rd = new InputStreamReader(System.in);
```

```
while(true) {  
    int c = rd.read(); // 입력 스트림으로부터 키 입력. c는 입력된 키 문자 값  
    if(c == -1) // 입력 스트림의 끝을 만나는 경우  
        break; // 입력 종료  
}
```



## 2. 문자 스트림으로 텍스트 파일 읽기

### ❖ 텍스트 파일을 읽기 위해 문자 스트림 `FileReader` 클래스 이용

#### 1. 파일 입력 스트림 생성(파일 열기)

- 스트림을 생성하고 파일을 열어 스트림과 연결

```
FileReader fin = new FileReader("c:\\test.txt");
```

#### 2. 파일 읽기

- `read()`로 문자 하나 씩 파일에서 읽음

```
int c;  
while((c = fin.read()) != -1) { // 문자를 c에 읽음. 파일 끝까지 반복  
    System.out.print((char)c); // 문자 c 화면에 출력  
}
```

#### 3. 스트림 닫기

- 스트림이 더 이상 필요 없으면 닫아야 함. 닫힌 스트림에서는 읽을 수 없음
- `close()`로 스트림 닫기

```
fin.close();
```



# 파일 입출력과 예외 처리

## ❖ 파일 입출력 동안 예외 발생 가능

- 스트림 생성 동안 : **FileNotFoundException** 발생 가능
  - 파일의 경로명이 틀리거나, 디스크의 고장 등으로 파일을 열 수 없음

```
FileReader fin = new FileReader("c:\\test.txt"); // FileNotFoundException 발생가능
```

- 파일 읽기, 쓰기, 닫기를 하는 동안 : **IOException** 발생 가능
  - 디스크 오동작, 파일이 중간에 깨진 경우, 디스크 공간이 모자라서 파일 입출력 불가

```
int c = fin.read(); // IOException 발생 가능
```

## ❖ try-catch 블록 반드시 필요

- 자바 컴파일러의 강제 사항

생략 가능. FileNotFoundException은 IOException을 상속받기 때문에 아래의 catch 블록 하나만 있으면 됨

```
try {
    FileReader fin = new FileReader("c:\\test.txt");
    ..
    int c = fin.read();
    ...
    fin.close();
} catch (FileNotFoundException e) {
    System.out.println("파일을 열 수 없음");
} catch (IOException e) {
    System.out.println("입출력 오류");
}
```

# FILEREADER의 생성자와 주요 메소드

생성자	설명
<code>FileReader(File file)</code>	file에 지정된 파일로부터 읽는 <code>FileReader</code> 생성
<code>FileReader(String name)</code>	name 이름의 파일로부터 읽는 <code>FileReader</code> 생성

메소드	설명
<code>int read()</code>	한 개의 문자를 읽어 정수형으로 리턴
<code>int read(char[] cbuf)</code>	최대 cbuf 배열의 크기만큼 문자들을 읽어 cbuf 배열에 저장. 만일 읽는 도중 EOF를 만나면 실제 읽은 문자 개수 리턴
<code>int read(char[] cbuf, int off, int len)</code>	최대 len 크기만큼 읽어 cbuf 배열의 off부터 저장. 읽는 도중 EOF를 만나면 실제 읽은 문자 개수 리턴
<code>String getEncoding()</code>	스트림이 사용하는 문자 집합의 이름 리턴
<code>void close()</code>	입력 스트림을 닫고 관련된 시스템 자원 해제

# 예제 1 : FILEREADER로 텍스트 파일 읽기

FileReader를 이용하여 c:\windows\system.ini 파일을 읽어 화면에 출력하는 프로그램을 작성하라.  
system.ini는 텍스트 파일이다..

```
import java.io.*;

public class FileReaderEx {
    public static void main(String[] args) {
        FileReader in = null;
        try {
            in = new FileReader("c:\\windows\\system.ini");
            int c;
            while ((c = in.read()) != -1) { // 한 문자씩 파일 끝까지 읽는다.
                System.out.print((char)c);
            }
            in.close();
        }
        catch (IOException e) {
            System.out.println("입출력 오류");
        }
    }
}
```

파일 끝을 만나면 -1 리턴

```
; for 16-bit app support
[386Enh]
woafont=dosapp.fon
EGA80WOA.FON=EGA80WOA.FON
EGA40WOA.FON=EGA40WOA.FON
CGA80WOA.FON=CGA80WOA.FON
CGA40WOA.FON=CGA40WOA.FON
```

```
[drivers]
wave=mmdrv.dll
timer=timer.drv
```

```
[mci]
```

# 문자 스트림으로 텍스트 파일 쓰기

## ❖ 텍스트 파일에 쓰기 위해 문자 스트림 FileWriter 클래스 이용

### 1. 파일 출력 스트림 생성(파일 열기)

- 스트림을 생성하고 파일을 열어 스트림과 연결

```
FileWriter fout = new FileWriter("c:\\Temp\\test.txt");
```

### 2. 파일 쓰기

- write()로 문자 하나 씩 파일에 기록

```
fout.write('A'); // 문자 'A'를 파일에 기록
```

- 블록 단위로 쓰기 가능

```
char [] buf = new char [1024];  
fout.write(buf, 0, buf.length); // buf[0]부터 버퍼 크기만큼 쓰기
```

### 3. 스트림 닫기

- close()로 스트림 닫기

```
fout.close(); // 스트림 닫기. 더 이상 스트림에 기록할 수 없다.
```

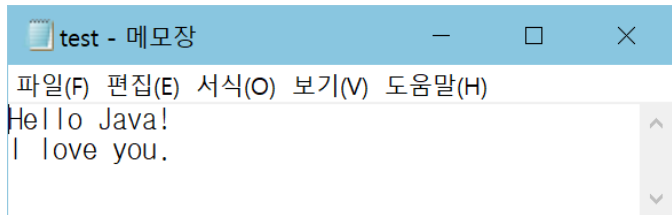
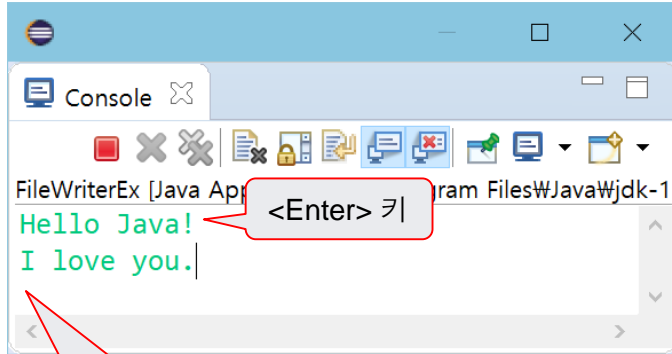
# FILEWRITER의 생성자와 주요 메소드

생성자	설명
<code>FileWriter(File file)</code>	file에 데이터를 저장할 <code>FileWriter</code> 생성
<code>FileWriter(String name)</code>	name 파일에 데이터를 저장할 <code>FileWriter</code> 생성
<code>FileWriter(File file, boolean append)</code>	<code>FileWriter</code> 를 생성하며, <code>append</code> 가 <code>true</code> 이면 파일의 마지막부터 데이터 저장
<code>FileWriter(String name, boolean append)</code>	<code>FileWriter</code> 를 생성하며, <code>append</code> 가 <code>true</code> 이면 파일의 마지막부터 데이터 저장

메소드	설명
<code>void write(int c)</code>	c를 char로 변환하여 한 개의 문자 출력
<code>void write(String str, int off, int len)</code>	인덱스 off부터 len개의 문자를 str 문자열에서 출력
<code>void write(char[] cbuf, int off, int len)</code>	인덱스 off부터 len개의 문자를 배열 cbuf에서 출력
<code>void flush()</code>	스트림에 남아 있는 데이터 모두 출력
<code>String getEncoding()</code>	스트림이 사용하는 문자 집합의 이름 리턴
<code>void close()</code>	출력 스트림을 닫고 관련된 시스템 자원 해제

## 예제 2 : FILEWRITER를 이용하여 텍스트 파일 쓰기

사용자로부터 입력받은 텍스트를 c:\Temp\Wtest.txt 파일에 저장하는 프로그램을 작성하라. 사용자는 키 입력 후 라인 첫 위치에 ctrl-z 키(EOF)를 입력하라.



실행 결과 test.txt 파일 생성

```
import java.io.*;

public class FileWriterEx {
    public static void main(String[] args) {
        InputStreamReader in = new InputStreamReader(System.in);

        FileWriter fout = null;
        int c;
        try {
            fout = new FileWriter("c:\\Temp\\test.txt");
            while ((c = in.read()) != -1) {
                fout.write(c); // 키보드로부터 받은 문자를 파일에 저장
            }
            in.close();
            fout.close();
        }
        catch (IOException e) {
            System.out.println("입출력 오류");
        }
    }
}
```

# 3. 바이트 스트림으로 바이너리 파일 쓰기

## ❖ 바이너리 값을 파일에 저장하기

- 프로그램 내의 변수, 배열, 버퍼에 든 바이너리 값을 파일에 그대로 기록
  - FileOutputStream 클래스 이용

### 1. 파일 출력 스트림 생성(파일 열기)

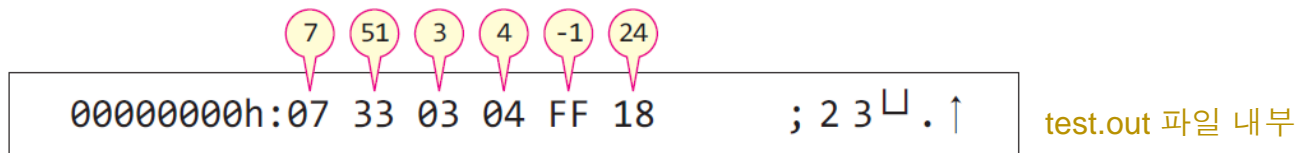
- 스트림을 생성하고 파일을 열어 스트림과 연결

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

### 2. 파일 쓰기

- write()로 문자 하나 씩 파일에 기록

```
byte b[] = {7,51,3,4,-1,24};  
for(int i=0; i<b.length; i++) fout.write(b[i]); // 배열 b를 바이너리 그대로 기록
```



### 3. 스트림 닫기

- close()로 스트림 닫기

# FILEOUTPUTSTREAM의 생성자와 주요 메소드

생성자	설명
<code>FileOutputStream(File file)</code>	<code>file</code> 이 지정하는 파일에 출력하는 <code>FileOutputStream</code> 생성
<code>FileOutputStream(String name)</code>	<code>name</code> 이 지정하는 파일에 출력하는 <code>FileOutputStream</code> 생성
<code>FileOutputStream</code> <code>(File file, boolean append)</code>	<code>FileOutputStream</code> 을 생성하며 <code>append</code> 가 <code>true</code> 이면 <code>file</code> 이 지정하는 파일의 마지막부터 데이터 저장
<code>FileOutputStream</code> <code>(String name, boolean append)</code>	<code>FileOutputStream</code> 을 생성하며 <code>append</code> 가 <code>true</code> 이면 <code>name</code> 이름의 파일의 마지막부터 데이터 저장

메소드	설명
<code>void write(int b)</code>	<code>int</code> 형으로 넘겨진 한 바이트를 출력 스트림으로 출력
<code>void write(byte[] b)</code>	배열 <code>b</code> 의 바이트를 모두 출력 스트림으로 출력
<code>void write(byte[] b, int off, int len)</code>	<code>len</code> 크기만큼 <code>off</code> 부터 배열 <code>b</code> 를 출력 스트림으로 출력
<code>void flush()</code>	출력 스트림에서 남아 있는 데이터 모두 출력
<code>void close()</code>	출력 스트림을 닫고 관련된 시스템 자원 해제



## 예제 3 : FILEOUTPUTSTREAM으로 바이너리 파일 쓰기

FileOutputStream을 이용하여 byte [] 배열 속에 들어 있는 바이너리 값을 c:\Temp\test.out 파일에 저장하라.  
이 파일은 바이너리 파일이 된다. 이 파일은 예제 13-4에서 읽어 출력할 것이다.

```
import java.io.*;

public class FileOutputStreamEx {
    public static void main(String[] args) {
        byte b[] = {7,51,3,4,-1,24};

        try {
            FileOutputStream fout = new FileOutputStream("c:\\Temp\\test.out");
            for(int i=0; i<b.length; i++)
                fout.write(b[i]); // 배열 b의 바이너리를 그대로 기록

            fout.close();
        } catch(IOException e) {}
        System.out.println("c:\\Temp\\test.out을 저장하였습니다.");
    }
}
```

fout.write(b); 한 줄로 코딩 가능

c:\Temp\test.out을 저장하였습니다.

7 51 3 4 -1 24  
00000000h:07 33 03 04 FF 18

; 2 3 4 . ↑

test.out 파일 내부

# 바이트 스트림으로 바이너리 파일 읽기

## ❖ 바이너리 파일에서 읽기 위해 FileInputStream 클래스 이용

### 1. 파일 입력 스트림 생성(파일 열기)

- 스트림을 생성하고 파일을 열어 스트림과 연결

```
FileInputStream fin = new FileInputStream("c:\\Temp\\test.out");
```

### 2. 파일 읽기

- read()로 문자 하나 씩 파일에서 읽기

```
int n=0, c;  
while((c = fin.read()) != -1) {  
    b[n] = (byte)c; // 읽은 바이트를 배열에 저장  
    n++;  
}
```

- 블록 단위로 읽기 가능

```
fin.read(b); // 배열 b의 바이트 크기만큼 바이너리 그대로 읽기
```

### 3. 스트림 닫기

- close()로 스트림 닫기

# FILEINPUTSTREAM의 생성자와 주요 메소드

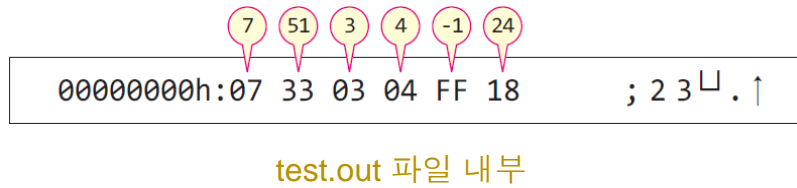
생성자	설명
<code>FileInputStream(File file)</code>	<code>file</code> 이 지정하는 파일로부터 읽는 <code>FileInputStream</code> 생성
<code>FileInputStream(String name)</code>	<code>name</code> 이 지정하는 파일로부터 읽는 <code>FileInputStream</code> 생성

메소드	설명
<code>int read()</code>	입력 스트림에서 한 바이트를 읽어 <code>int</code> 형으로 리턴
<code>int read(byte[] b)</code>	최대 배열 <code>b</code> 의 크기만큼 바이트를 읽음. 읽는 도중 EOF를 만나면 실제 읽은 바이트 수 리턴
<code>int read(byte[] b, int off, int len)</code>	최대 <code>len</code> 개의 바이트를 읽어 <code>b</code> 배열의 <code>off</code> 위치에 저장. 읽는 도중 EOF를 만나면 실제 읽은 바이트 수 리턴
<code>int available()</code>	입력 스트림에서 현재 읽을 수 있는 바이트 수 리턴
<code>void close()</code>	입력 스트림을 닫고 관련된 시스템 자원 해제

## 예제 4 : FILEINPUTSTREAM으로 바이너리 파일 읽기

FileInputStream을 이용하여 c:\Temp\test.out 파일(예제 13-3에서 저장한 파일)을 읽어 바이너리 값들을 byte [] 배열 속에 저장하고 화면에 출력하라.



```
import java.io.*;
public class FileInputStreamEx {
    public static void main(String[] args) {
        byte b[] = new byte [6]; // 비어 있는 byte 배열
        try {
            FileInputStream fin =
                new FileInputStream("c:\\Temp\\test.out");
            int n=0, c;
            while((c = fin.read())!= -1) {
                b[n] = (byte)c; // 읽은 바이트를 배열에 저장
                n++;
            }

            System.out.println(
                "c:\\Temp\\test.out에서 읽은 배열을 출력합니다.");
            for(int i=0; i<b.length; i++)
                System.out.print(b[i]+" ");
            System.out.println();

            fin.close();
        } catch(IOException e) { }
    }
}
```

c:\Temp\test.out에서 읽은 배열을 출력합니다.  
7 51 3 4 -1 24

## 4. FILE 클래스

### ❖ File 클래스

- 파일의 경로명 및 속성을 다루는 클래스
  - java.io.File
  - 파일과 디렉터리 경로명의 추상적 표현
- 파일 이름 변경, 삭제, 디렉터리 생성, 크기 등 파일 관리
- File 객체에는 파일 읽기/쓰기 기능 없음
  - 파일 입출력은 파일 입출력 스트림 이용

### ❖ File 객체 생성

- 생성자에 파일 경로명을 주어 File 객체 생성

```
File f = new File("c:\\Temp\\test.txt");
```

- 디렉터리와 파일명을 나누어 생성자 호출

```
File f = new File("c:\\Temp", "test.txt");
```

# FILE 클래스 생성자와 주요 메소드

메소드	설명
<code>File(File parent, String child)</code>	parent 디렉터리에 child 이름의 디렉터리나 파일을 나타내는 File 객체 생성
<code>File(String pathname)</code>	pathname의 완전 경로명이 나타내는 File 객체 생성
<code>File(String parent, String child)</code>	parent 디렉터리에 child 이름의 디렉터리나 파일을 나타내는 File 객체 생성
<code>File(URI uri)</code>	file:URI를 추상 경로명으로 변환하여 File 객체 생성

메소드	설명
<code>boolean mkdir()</code>	새로운 디렉터리 생성
<code>String[] list()</code>	디렉터리 내의 파일과 서브 디렉터리 리스트를 문자열 배열로 리턴
<code>File [] listFiles()</code>	디렉터리 내의 파일과 서브 디렉터리 리스트를 File [] 배열로 리턴
<code>boolean renameTo(File dest)</code>	dest가 지정하는 경로명으로 파일 이름 변경
<code>boolean delete()</code>	파일 또는 디렉터리 삭제
<code>long length()</code>	파일의 크기 리턴
<code>String getPath()</code>	경로명 전체를 문자열로 변환하여 리턴
<code>String getParent()</code>	파일이나 디렉터리의 부모 디렉터리 이름 리턴
<code>String getName()</code>	파일 또는 디렉터리 이름을 문자열로 리턴
<code>boolean isFile()</code>	일반 파일이면 true 리턴
<code>boolean isDirectory()</code>	디렉터리이면 true 리턴
<code>long lastModified()</code>	파일이 마지막으로 변경된 시간 리턴
<code>boolean exists()</code>	파일 또는 디렉터리가 존재하면 true 리턴

# FILE 클래스 활용

- 파일 크기

```
long size = f.length();
```

- 파일 경로명

```
File f = new File("c:\\windows\\system.ini");  
String filename = f.getName();    // "system.ini"  
String path = f.getPath();        // "c:\\windows\\system.ini"  
String parent = f.getParent();    // "c:\\windows"
```

- 파일 타입

```
if(f.isFile())  
    System.out.println(f.getPath() + "는 파일입니다."); // 파일  
else if(f.isDirectory())  
    System.out.println(f.getPath() + "는 디렉터리입니다."); // 디렉터리
```

c:\windows\system.ini은 파일입니다.

- 디렉터리 파일  
리스트 얻기

```
File f = new File("c:\\Temp");  
File[] subfiles = f.listFiles(); // c:\Temp의 파일 및 서브 디렉터리 리스트 얻기  
  
for(int i=0; i<subfiles.length; i++) {  
    System.out.print(subfiles[i].getName()); // 서브 파일명 출력  
    System.out.println("\t파일 크기: " + subfiles[i].length()); //서브파일크기출력  
}
```

## 예제 5 : FILE 클래스를 활용한 파일 관리

File 클래스를 이용하여, 파일 타입 및 경로명 알아내기, 디렉터리 생성, 파일 이름 변경, 디렉터리의 파일 리스트 출력 등 다양한 파일 관리 사례를 보여준다..

```
import java.io.File;

public class FileClassExample {
    public static void listDirectory(File dir) {
        System.out.println("-----" + dir.getPath() + "의 서브 리스트 입니다.-----");
        File[] subFiles = dir.listFiles();

        for(int i=0; i<subFiles.length; i++) {
            File f = subFiles[i];
            long t = f.lastModified(); // 마지막으로 수정된 시간
            System.out.print(f.getName());
            System.out.print("\t파일 크기: " + f.length()); // 파일 크기
            System.out.printf("\t수정 한 시간: %tb %td %ta %t\n",t, t, t, t);
        }
    }

    public static void main(String[] args) {
        File f1 = new File("c:\\windows\\system.ini");
        System.out.println( f1.getPath() + ", " + f1.getParent() + ", " + f1.getName());
        String res="";
        if(f1.isFile()) res = "파일";
        else if(f1.isDirectory()) res = "디렉토리";
        System.out.println(f1.getPath() + "은 " + res + "입니다.");
    }
}
```

```
File f2 = new File("c:\\Temp\\java_sample");
if(!f2.exists()) {
    f2.mkdir();
}
listDirectory(new File("c:\\Temp"));
f2.renameTo(new
    File("c:\\Temp\\javasample"));
listDirectory(new File("c:\\Temp"));
}
```

```
c:\windows\system.ini, c:\windows, system.ini
c:\windows\system.ini은 파일입니다.
-----c:\Temp의 서브 리스트 입니다.-----
HncDownload 파일 크기: 0   수정한 시간: 1월 18 목 20:43:33
java_sample  파일 크기: 0   수정한 시간: 6월 06 수 12:34:02
jlinktest    파일 크기: 4096   수정한 시간: 4월 26 목
17:32:07
test.out      파일 크기: 6   수정한 시간: 6월 06 수 12:28:05
test.txt      파일 크기: 26   수정한 시간: 6월 06 수 12:41:19
-----c:\Temp의 서브 리스트 입니다.-----
HncDownload 파일 크기: 0   수정한 시간: 1월 18 목 20:43:33
javasample   파일 크기: 0   수정한 시간: 6월 06 수 12:34:02
jlinktest    파일 크기: 4096   수정한 시간: 4월 26 목
17:32:07
test.out      파일 크기: 6   수정한 시간: 6월 06 수 12:28:05
test.txt      파일 크기: 26   수정한 시간: 6월 06 수 12:41:19
```



## 예제 6 : 텍스트 파일 복사

문자 스트림 `FileReader`와 `FileWriter`를 이용하여 `c:\windows\system.ini`를 `c:\Temp\system.txt` 파일로 복사하는 프로그램을 작성하라.

```
import java.io.*;

public class TextCopy {
    public static void main(String[] args){
        File src = new File("c:\\windows\\system.ini"); // 원본 파일 경로명
        File dest = new File("c:\\Temp\\system.txt"); // 복사 파일 경로명
        int c;
        try {
            FileReader fr = new FileReader(src); // 파일 입력 문자 스트림 생성
            FileWriter fw = new FileWriter(dest); // 파일 출력 문자 스트림 생성
            while((c = fr.read()) != -1) { // 문자 하나 읽고
                fw.write((char)c); // 문자 하나 쓰고
            }
            fr.close();
            fw.close();
            System.out.println( src.getPath()+ "를 " + dest.getPath()+ "로 복사하였습니다.");
        } catch (IOException e) {
            System.out.println("파일 복사 오류");
        }
    }
}
```

c:\windows\system.ini를 c:\Temp\system.txt로 복사하였습니다.

# 예제 7 : 바이너리 파일 복사

바이트 스트림 FileInputStream과 FileOutputStream을 이용하여 이미지 파일을 복사하라.  
실행 전에 미리 c:\Temp 디렉터리에 img.jpg를 준비하라.

```
import java.io.*;

public class BinaryCopy {
    public static void main(String[] args) {
        File src = new File( "c:\\Temp\\img.jpg");
        File dest = new File("c:\\Temp\\back.jpg");
        int c;
        try {
            FileInputStream fi = new FileInputStream(src);
            FileOutputStream fo = new FileOutputStream(dest);
            while((c = fi.read()) != -1) {
                fo.write((byte)c);
            }
            fi.close();
            fo.close();
            System.out.println( src.getPath()+ "를 " + dest.getPath()+
                "로 복사하였습니다.");
        } catch (IOException e) {
            System.out.println("파일 복사 오류");
        }
    }
}
```

c:\Temp\img.jpg를 c:\Temp\back.jpg로 복사하였습니다.



한 바이트씩 복사하므로 실행 시간이 많이 걸리는 것을 느낄 수 있다. 고속복사는 예제 8을 보라.

## 예제 8 : 고속 복사를 위한 블록 단위 바이너리 파일 복사

예제 7을 10KB씩 읽고 쓰도록 수정하여 고속으로 파일을 복사하라.

```
import java.io.*;

public class BlockBinaryCopy {
    public static void main(String[] args) {
        File src = new File("c:\\Temp\\img.jpg"); // 원본 파일
        File dest = new File("c:\\Temp\\back.jpg"); // 복사 파일
        try {
            FileInputStream fi = new FileInputStream(src); // 파일 입력 바이트 스트림 생성
            FileOutputStream fo = new FileOutputStream(dest); // 파일 출력 바이트 스트림 생성

            byte [] buf = new byte [1024*10]; // 10KB 버퍼
            while(true) {
                int n = fi.read(buf); // 버퍼 크기만큼 읽기. n은 실제 읽은 바이트
                fo.write(buf, 0, n); // buf[0]부터 n 바이트 쓰기
                if(n < buf.length)
                    break; // 버퍼 크기보다 작게 읽었기 때문에 파일 끝에 도달. 복사 종료
            }
            fi.close();
            fo.close();
            System.out.println(src.getPath() + "를 " + dest.getPath() + "로 복사하였습니다.");
        } catch (IOException e) { System.out.println("파일 복사 오류"); }
    }
}
```

c:\Temp\img.jpg를 c:\Temp\back.jpg로 복사하였습니다.