

Chapter 04. 바이트 스트림

Originally made by Prof. Hanku Lee

Modified by Mingyu Lim

**Collaborative Computing Systems Lab.
School of Internet & Multimedia Engineering
Konkuk University, Seoul, Korea**

1. 스트림이란?(계속)

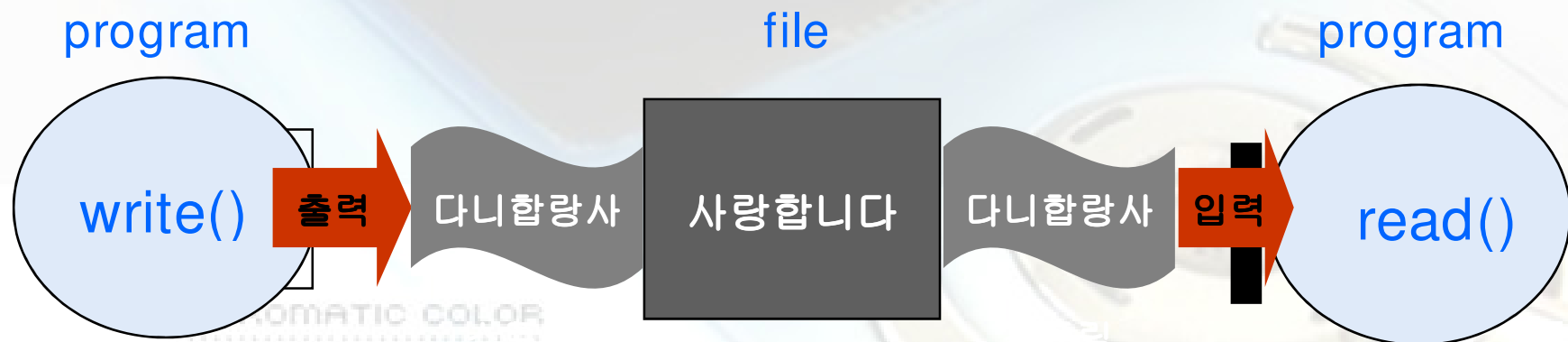
□ 스트림이란?

- 데이터를 목적지로 입출력하기 위한 방법이다
- 스트림에 데이터를 쓸 수 있고, 스트림에서 데이터를 읽을 수 있다.
- 스트림에 데이터를 쓸 경우, 이러한 스트림을 출력 스트림 (output stream)이라고 한다.
- 스트림에서 데이터를 읽을 경우, 이러한 스트림을 입력 스트림(input stream)이라고 한다.

ACHROMATIC COLOR
100% COLOUR FIDELITY

1. 스트림이란?(계속)

□ 스트림이란?



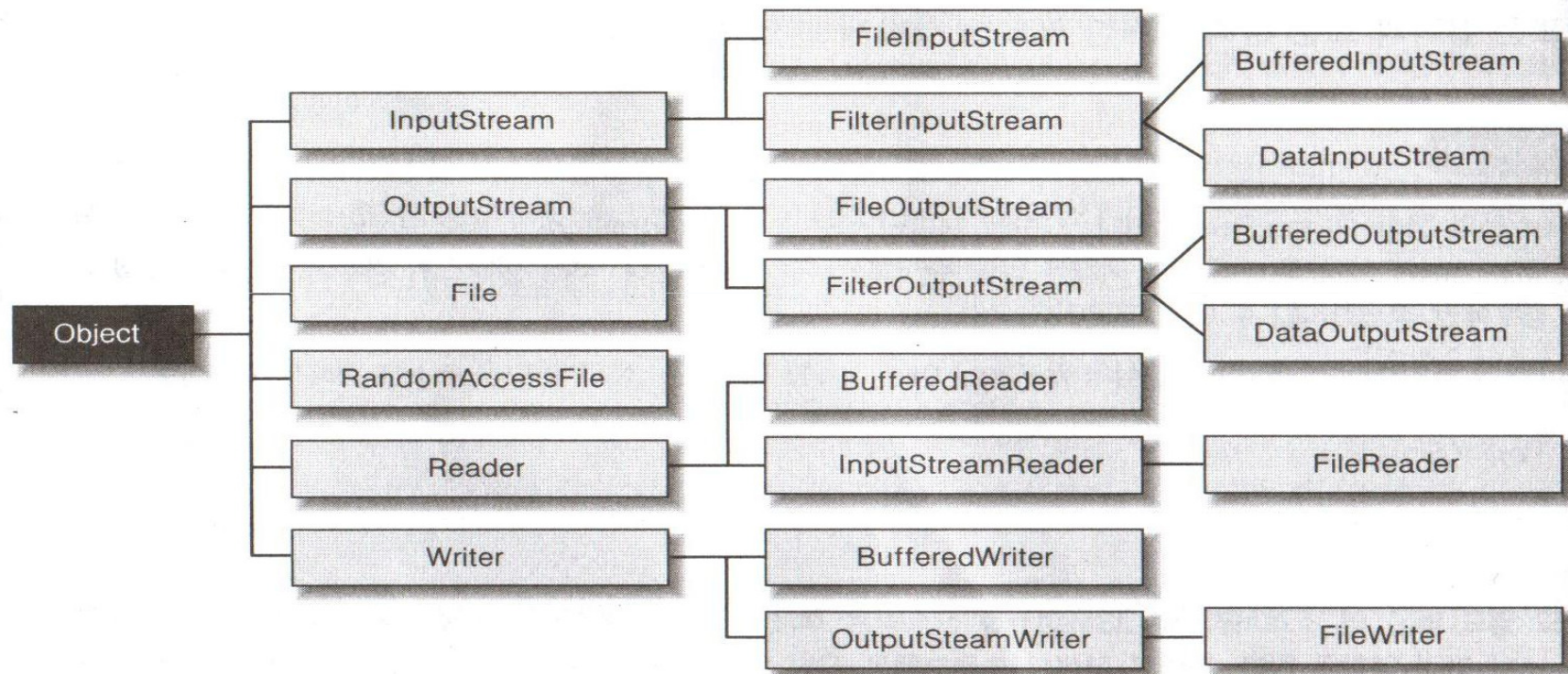
1. 스트림이란?(계속)

□ 스트림의 특징

- 스트림은 FIFO 구조 – FIFO구조란 먼저 들어간 것이 먼저 나오는 형태로서 데이터의 순서가 바뀌지 않는다는 특징이 있다.
- 스트림은 단방향 – 자바에서 스트림은 읽기, 쓰기가 동시에 되지 않는다. 따라서 읽기, 쓰기가 필요하다면 읽는 스트림과 쓰는 스트림을 하나씩 열어 사용해야 한다.
- 스트림은 지연될 수 있다 – 스트림은 넣어진 데이터가 처리되기 전까지는 스트림에 사용되는 스레드는 지연상태에 빠진다. 따라서 네트워크 내에서는 데이터가 모두 전송되기 전까지 네트워크 스레드는 지연상태가 된다.

1. 스트림이란?(계속)

□ 스트림의 종류



[그림 3-2] 자바 IO 클래스의 상속도

2. File 클래스

□ File 클래스

- 시스템에 있는 파일이나 디렉토리를 추상화한 클래스이다.
- File 클래스를 이용하면 파일의 크기, 생성, 삭제, 변경 및 마지막 수정날짜 등 다양한 정보를 알 수 있는 메소드를 제공하고 있다.

2바이트 문화권

- 영어권의 경우 1바이트로 모든 글자를 표현할 수 있다. 그러나 한국, 중국, 일본등은 1바이트로 모든 글자를 표현할 수 없다.
- 따라서 한국, 일본, 중국 등에서 사용되는 글자를 표현하려면 2바이트가 필요하다. 이러한 지역을 2바이트 문화권이라고 한다.

2. File 클래스(계속)

□ File 클래스 생성자

생성자	설 명
File(File parent, String child)	Parent 디렉토리에 child라는 파일에 대한 File 객체 생성
File(String child)	Child라는 파일에 대한 File 객체 생성
File(String parent, String child)	Parent 디렉토리에 child라는 파일에 대한 File 객체 생성

File 클래스 생성자

- File 클래스 생성자는 파일 클래스에 대한 객체를 자비 힙 메모리에 생성하는것이지 실제의 파일 시스템에 파일을 생성하지는 않는다.
- File 클래스의 인자로 지정되는 파일명은 실제로 존재하지 않을수도 있다.
(File 클래스의 `exists()` 메소드를 이용해서 파일 존재 유무 판단)

2. File 클래스(계속)

□ File 클래스 중요 메소드

메 소 드	설 명
boolean canRead()	파일이 읽기 가능하면 true, 아니면 false를 반환
boolean canWrite()	파일이 쓰기 가능하면 true, 아니면 false를 반환
boolean delete()	파일을 삭제하고 true를 반환, 파일을 삭제할 수 없으면 false를 반환
boolean equals(Object <i>obj</i>)	현재의 객체와 <i>obj</i> 로 지정된 객체가 같은 파일을 가지고 있으면 true, 아니면 false를 반환
boolean exists()	파일이 존재하면 true, 아니면 false를 반환
String getAbsolutePath()	파일에 대한 절대 경로를 반환
String getCanonicalPath()	파일에 대한 정규 경로를 반환
String getParent()	부모 디렉토리 이름을 반환
String getName()	파일의 이름을 반환
String getPath()	파일의 경로를 반환
boolean isAbsolute()	경로가 절대 경로이면 true, 아니면 false를 반환
boolean isDirectory()	현재의 객체가 디렉토리이면 true, 아니면 false를 반환

2. File 클래스(계속)

□ File 클래스 중요 메소드

메 소 드	설 명
<code>boolean createNewFile()</code>	지정하지 않는 파일이 존재하지 않을 경우에 파일을 생성한다.
<code>void deleteOnExit()</code>	JVM이 종료될 때 파일을 삭제한다.
<code>boolean isFile()</code>	현재의 객체가 파일이면 <code>true</code> , 아니면 <code>false</code> 를 반환
<code>long lastModified()</code>	1970년 1월 1일(GMT)부터 파일이 마지막으로 수정된 날짜까지의 시간을 밀리 초로 반환
<code>long length()</code>	파일의 바이트 수를 반환
<code>String[] list()</code>	디렉토리에서 파일의 이름을 반환
<code>boolean mkdir()</code>	디렉토리를 생성. 경로로 지정된 모든 부모 디렉토리가 존재하여야 한다. 지정한 디렉토리가 생성되면 <code>true</code> 를 반환하고, 아니면 <code>false</code> 를 반환
<code>boolean mkdirs()</code>	디렉토리를 생성. 경로로 지정된 디렉토리가 존재하지 않으면 생성한 다음 지정된 디렉토리를 생성. 디렉토리가 생성되면 <code>true</code> 를 아니면 <code>false</code> 를 반환
<code>boolean renameTo(File newName)</code>	파일이나 디렉토리의 이름을 <code>newName</code> 으로 변경한 다음 <code>true</code> 를 반환. 이름을 변경하지 못하면 <code>false</code> 를 반환

2. File 클래스(계속)

□ File 클래스를 이용한 파일의 정보 구하기(예제)

```
FileExam.java FileExam2.java FileInfo.java X
import java.io.File;

public class FileInfo {

    public static void main(String[] args) {
        if(args.length != 1){
            System.out.println("사용법 : java FileInfo 파일이름");
            System.exit(0);
        } // if end

        File f = new File(args[0]);
        if(f.exists()){ // 파일이 존재할 경우
            System.out.println("length : " + f.length());
            System.out.println("canRead : " + f.canRead());
            System.out.println("canWrite : " + f.canWrite());
            System.out.println("getAbsolutePath : " + f.getAbsolutePath());
            try{
                System.out.println("getCanonicalPath : " + f.getCanonicalPath());
            } catch (IOException e){
                System.out.println(e);
            }
            System.out.println("getName : " + f.getName());
            System.out.println("getParent : " + f.getParent());
            System.out.println("getPath : " + f.getPath());
        }
    } // main end
}
```

2. File 클래스(계속)

□ File 클래스를 이용한 파일의 정보 구하기(결과)

```
C:\java\weclipse-SDK-3.3-win32\weclipse\workspace\자바 io nio\4장\bin>dir/w
c 드라이브의 볼륨에는 이름이 없습니다.
볼륨 일련 번호: 747A-FF42

C:\java\weclipse-SDK-3.3-win32\weclipse\workspace\자바 io nio\4장\bin 디렉터리

[.]                [..]                FileInfo.class    FileExam.class    FileExam2.class
FileInfo.class
3개 파일                5,266 바이트
2개 디렉터리  86,668,062,720 바이트 남음

C:\java\weclipse-SDK-3.3-win32\weclipse\workspace\자바 io nio\4장\bin>java FileInfo.class
Exception in thread "main" java.lang.NoClassDefFoundError: FileInfo.class

C:\java\weclipse-SDK-3.3-win32\weclipse\workspace\자바 io nio\4장\bin>java FileInfo.class
length : 1662
canRead : true
canWrite : true
getAbsolutePath : C:\java\weclipse-SDK-3.3-win32\weclipse\workspace\자바 io nio\4장\bin\FileInfo.class
getCanonicalPath : C:\java\weclipse-SDK-3.3-win32\weclipse\workspace\자바 io nio\4장\bin\FileInfo.class
getName : FileInfo.class
getParent : null
getPath : FileInfo.class

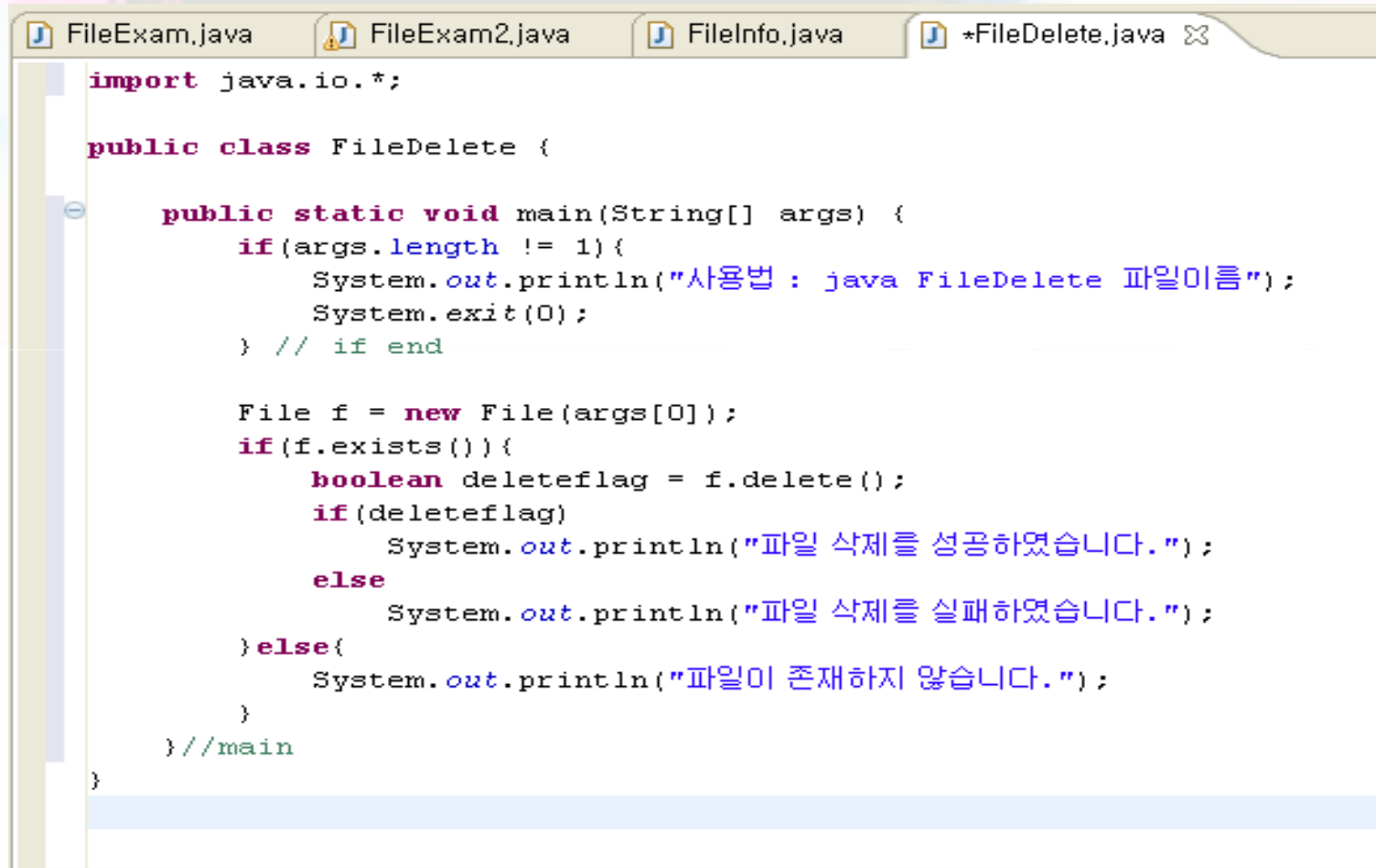
C:\java\weclipse-SDK-3.3-win32\weclipse\workspace\자바 io nio\4장\bin>java FileInfo.class
length : 1662
canRead : true
canWrite : true
getAbsolutePath : C:\java\weclipse-SDK-3.3-win32\weclipse\workspace\자바 io nio\4장\bin\FileInfo.class
getCanonicalPath : C:\java\weclipse-SDK-3.3-win32\weclipse\workspace\자바 io nio\4장\bin\FileInfo.class
getName : FileInfo.class
getParent : .
getPath : .\FileInfo.class

C:\java\weclipse-SDK-3.3-win32\weclipse\workspace\자바 io nio\4장\bin>
```

결과 비교
하기

2. File 클래스(계속)

□ File 클래스를 이용한 파일 삭제(예제)

A screenshot of a Java IDE window showing a file named FileDelete.java. The window has four tabs: FileExam.java, FileExam2.java, FileInfo.java, and *FileDelete.java. The code in FileDelete.java is as follows:

```
import java.io.*;

public class FileDelete {

    public static void main(String[] args) {
        if(args.length != 1){
            System.out.println("사용법 : java FileDelete 파일이름");
            System.exit(0);
        } // if end

        File f = new File(args[0]);
        if(f.exists()){
            boolean deleteflag = f.delete();
            if(deleteflag)
                System.out.println("파일 삭제를 성공하였습니다.");
            else
                System.out.println("파일 삭제를 실패하였습니다.");
        } else{
            System.out.println("파일이 존재하지 않습니다.");
        }
    } //main
}
```

2. File 클래스(계속)

□ File 클래스를 이용한 파일 삭제(결과)

```
C:\>명령 프롬프트

C:\Wjava\weclipse-SDK-3.3-win32\weclipse\workspace\자바 io nio\4장\bin>dir/w
C 드라이브의 볼륨에는 이름이 없습니다.
볼륨 일련 번호: 747A-FF42

C:\Wjava\weclipse-SDK-3.3-win32\weclipse\workspace\자바 io nio\4장\bin 디렉터리

[.]          [...]          FileDelete.class  FileExam2.class
FileInfo.class  java.txt          5,107 바이트
                4개 파일
                2개 디렉터리 86,667,395,072 바이트 남음

C:\Wjava\weclipse-SDK-3.3-win32\weclipse\workspace\자바 io nio\4장\bin>
java FileDelete java.txt 실행명령
파일 삭제를 성공하였습니다.

C:\Wjava\weclipse-SDK-3.3-win32\weclipse\workspace\자바 io nio\4장\bin>dir/w
C 드라이브의 볼륨에는 이름이 없습니다.
볼륨 일련 번호: 747A-FF42

C:\Wjava\weclipse-SDK-3.3-win32\weclipse\workspace\자바 io nio\4장\bin 디렉터리

[.]          [...]          FileDelete.class  FileExam2.class
FileInfo.class
                3개 파일          5,091 바이트
                2개 디렉터리 86,667,395,072 바이트 남음
파일 삭제후 목록

C:\Wjava\weclipse-SDK-3.3-win32\weclipse\workspace\자바 io nio\4장\bin>
```

2. File 클래스(계속)

□ File 클래스를 이용한 디렉토리의 파일 목록 출력(예제)



```
FileDelete.java  FileExam2.java  FileList.java X
import java.io.*;

public class FileList {
    public static void main(String [] args){

        File dir = new File("C:\\java\\eclipse-SDK-3.3-win32\\eclipse\\workspace\\자바 io nio\\4장\\bin ")

        File[] list = dir.listFiles();

        for(int i=0; i<list.length; i++)
            System.out.println("파일"+i+"번은 : "+list[i].getName());

    }
}
```

2. File 클래스(계속)

□ File 클래스를 이용한 디렉토리의 파일 목록 출력(결과)

```
C:\명령 프롬프트

C:\wjava\weclipse-SDK-3.3-win32\weclipse\workspace\자바 io nio\4장\bin>dir/w
C 드라이브의 볼륨에는 이름이 없습니다.
볼륨 일련 번호: 747A-FF42

C:\wjava\weclipse-SDK-3.3-win32\weclipse\workspace\자바 io nio\4장\bin 디렉터리

[.]          [...]          FileDelete.class  FileExam2.class
FileInfo.class  FileList.class
              4개 파일          6,279 바이트
              2개 디렉터리 86,667,329,536 바이트 남음

C:\wjava\weclipse-SDK-3.3-win32\weclipse\workspace\자바 io nio\4장\bin>
java FileList
0 : FileDelete.class
1 : FileExam2.class
2 : FileInfo.class
3 : FileList.class

C:\wjava\weclipse-SDK-3.3-win32\weclipse\workspace\자바 io nio\4장\bin>
```


2. File 클래스(계속)

□ File 클래스를 이용한 임시 파일의 생성과 삭제

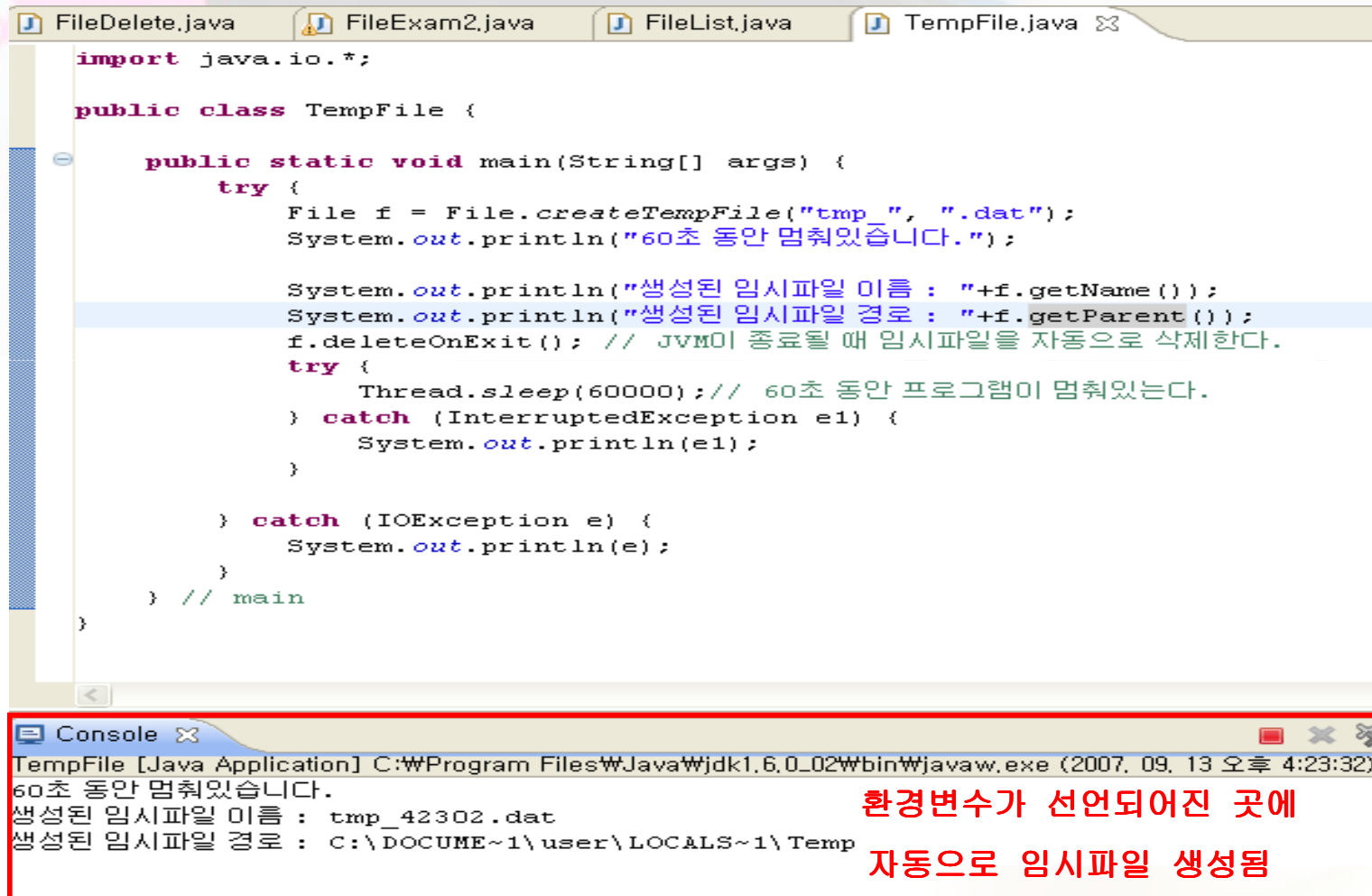
- 프로그램 작성시 임시로 파일을 작성할 필요가 생긴다.
(ex : 외부 파일을 이용해서 정렬할 경우가 대표적인 예)
 - 임시파일의 경우 이름이 동일하면 심각한 문제가 발생
- 이름의 중복성을 해결하기 위해서 File 클래스는 임시 파일을 제공하는 메소드 제공[`creatempFile()`]

static File	<code>createTempFile(String prefix, String suffix)</code> Creates an empty file in the default temporary-file directory, using the given prefix and suffix to generate its name.
static File	<code>createTempFile(String prefix, String suffix, File directory)</code> Creates a new empty file in the specified directory, using the given prefix and suffix strings to generate its name.

- prefix, suffix 만 사용자가 정의하면 중간 문자열은 임의 글자로 자동지정
- 윈도우의 경우 임시 파일은 환경변수 'TMP'로 지정된 디렉토리에 생성
- 임시파일의 경우 프로그램이 종료된후 삭제하는 것이 일반적임
: `deleteOnExit()` 메소드를 통해 JVM이 종료되면 자동으로 삭제됨

2. File 클래스(계속)

□ File 클래스를 이용한 임시 파일의 생성과 삭제(예제)



```
import java.io.*;

public class TempFile {

    public static void main(String[] args) {
        try {
            File f = File.createTempFile("tmp_", ".dat");
            System.out.println("60초 동안 멈춰있습니다.");

            System.out.println("생성된 임시파일 이름 : "+f.getName());
            System.out.println("생성된 임시파일 경로 : "+f.getParent());
            f.deleteOnExit(); // JVM이 종료될 때 임시파일을 자동으로 삭제한다.
        } catch (InterruptedException e1) {
            Thread.sleep(60000); // 60초 동안 프로그램이 멈춰있다.
            System.out.println(e1);
        } catch (IOException e) {
            System.out.println(e);
        }
    } // main
}
```

Console

TempFile [Java Application] C:\Program Files\Java\jdk1.6.0_02\bin\javaw.exe (2007. 09. 13 오후 4:23:32)

60초 동안 멈춰있습니다.

생성된 임시파일 이름 : tmp_42302.dat

생성된 임시파일 경로 : C:\DOCUME~1\user\LOCALS~1\Temp

환경변수가 선언되어진 곳에
자동으로 임시파일 생성됨

2. File 클래스(계속)

- File 클래스를 이용한 임시 파일의 생성과 삭제
(사용자가 원하는 폴더에 임시파일 생성하는 방법 예제)

The screenshot displays a Java IDE with the `TempFile1.java` file open. The code defines a `TempFile1` class with a `main` method that creates a temporary directory and file. The console output shows the execution results, including the path of the generated file.

```
import java.io.*;

public class TempFile1 {
    public static void main(String [] args){

        File dir = new File ("c:\\java\\tmp");           //임시파일 저장할 경로 파일 생성

        try{
            dir.mkdir();                                //디렉토리를 생성하는 함수 메소드
            File f =File.createTempFile("tmp_", "dat", dir); //dir 지정된 폴더에 임시파일 생성
            System.out.println("60초 동안 멈춰있습니다.");
            System.out.println("생성된 임시파일 이름 : "+f.getName());
            System.out.println("생성된 임시파일 경로 : "+f.getParent());
            f.deleteOnExit();                            // JVM이 종료될 때 임시파일을 자동으로 삭제한다.

        } catch (IOException e) {
            System.out.println(e);
        }

        try {
            Thread.sleep(60000);                          // 60초 동안 프로그램이 멈춰있다.
        } catch (InterruptedException e1) {
            System.out.println(e1);
        }

    } // main
}
```

Console Output:

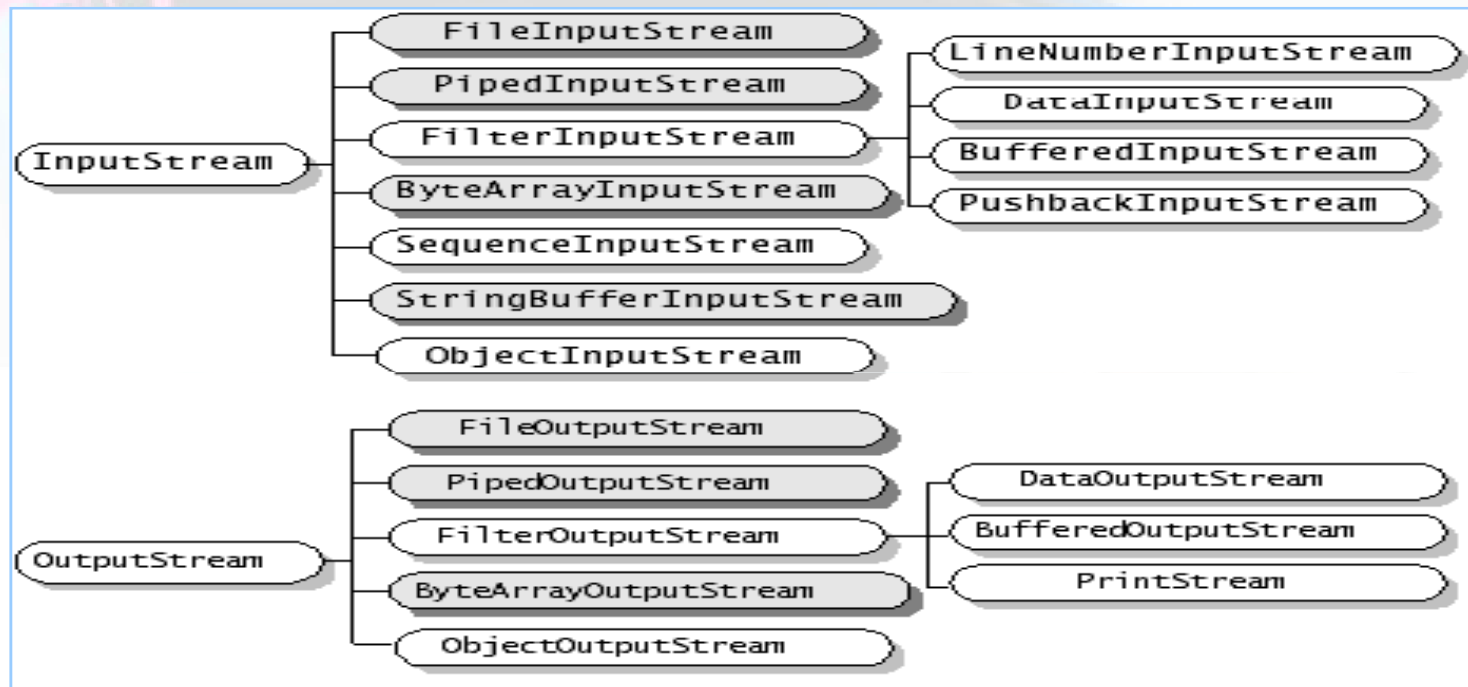
```
TempFile1 [Java Application] C:\Program Files\Java\jdk1.6.0_02\bin\javaw.exe (2007. 09. 13 오후 4:39:09)
60초 동안 멈춰있습니다.
생성된 임시파일 이름 : tmp_20031dat
생성된 임시파일 경로 : c:\java\tmp
```

The Windows Explorer window on the right shows the `C:\java\tmp` directory. A file named `tmp_10411dat` is visible in the file list. The address bar shows the path `C:\java\tmp`.

파일 탐색기에
생성된 모습

3. 바이트 단위 IO 클래스

□ 바이트 스트림 구성도



- InputStream은 **입력용 바이트 스트림**이다.
- OutputStream은 **출력용 바이트 스트림**이다.
- '**InputStream**'나 '**OutputStream**'이라는 단어가 붙어 있다면 **바이트 스트림**이다.

3. 바이트 단위 IO 클래스(계속)

□ InputStream 과 OutputStream

- 모든 바이트 스트림의 최상위 클래스
- 추상클래스로서, 바이트 단위로 입출력하기 위한 가장 기본적인 클래스 이다.

[표 14-4] InputStream의 주요 메서드

반환형	메서드	설명
abstract int	read()	스트림 데이터 1바이트를 읽어온다. 반환값은 0~255의 아스키코드값이기 때문에 문자로 나타내려면 char로 캐스팅해야 한다. 더 이상 읽을 수 없을 때는 -1을 반환한다.
int	read(byte b[])	스트림 데이터 1바이트를 읽어 바이트 배열에 저장하고, 읽은 수만큼 반환한다.
	read(byte b[], int start, int length)	스트림 데이터를 length만큼 읽어 바이트 배열 b의 start 위치에 저장하고, 읽을 수만큼 반환한다.
	available()	읽을 수 있는 바이트 수를 반환한다.
long	skip(long n)	읽을 수 있는 바이트에서 n만큼 건너뛴다.
void	close()	입력 스트림을 닫는다.

3. 바이트 단위 IO 클래스(계속)

□ InputStream 과 OutputStream

여기서 잠깐

- `InputStream`은 바이트 단위로 입력받기 위한 클래스이다. 그러나 `read()` 메소드는 `byte` 형으로 반환하는 것이 아니라 `int` 형으로 반환한다. 그 이유는 **파일끝을 나타내는 기호로 사용할 마땅할 값이 없기 때문이다.**
- `-1`을 반환할 수 있는 `int` 형을 사용한다. 따라서 `read()` 메소드는 정수 하위 8비트에 실제로 읽어 들인 값을 채워서 반환한다.

[표 14-8] OutputStream의 주요 메서드

반환형	메서드	설명
abstract void	<code>write(int b)</code>	출력 스트림으로 b의 값을 바이트로 변환하여 쓰기한다.
void	<code>write(byte[] b)</code>	출력 스트림으로 바이트 배열 b를 쓰기한다.
	<code>write(byte[] b, int start, int length)</code>	출력 스트림으로 바이트 배열 b를 start부터 length만큼 쓰기한다.
	<code>flush()</code>	출력 스트림을 통하여 쓰기를 할 때 일반적으로 버퍼에 가득차게 되면 한꺼번에 보내게 되는데, 이 메서드를 사용하게 되면 버퍼에 가득 차 있지 않더라도 버퍼의 내용을 바로 보내게 된다.
	<code>close()</code>	모든 자원을 반납한다.

3. 바이트 단위 IO 클래스(계속)

□ InputStream 과 OutputStream

여기서 잠깐

- InputStream 과 OutputStream은 추상클래스 이다. 이는 new 연산자를 이용해서 객체화 시킬수가 없다.

`InputStream in = new InputStream();` [불가능]

- 따라서 다음과 같은 형태로 객체화 하여야 한다.

`InputStream in = new InputStream 클래스를 상속받은 클래스의 생성자();`

ACHROMATIC COLOR

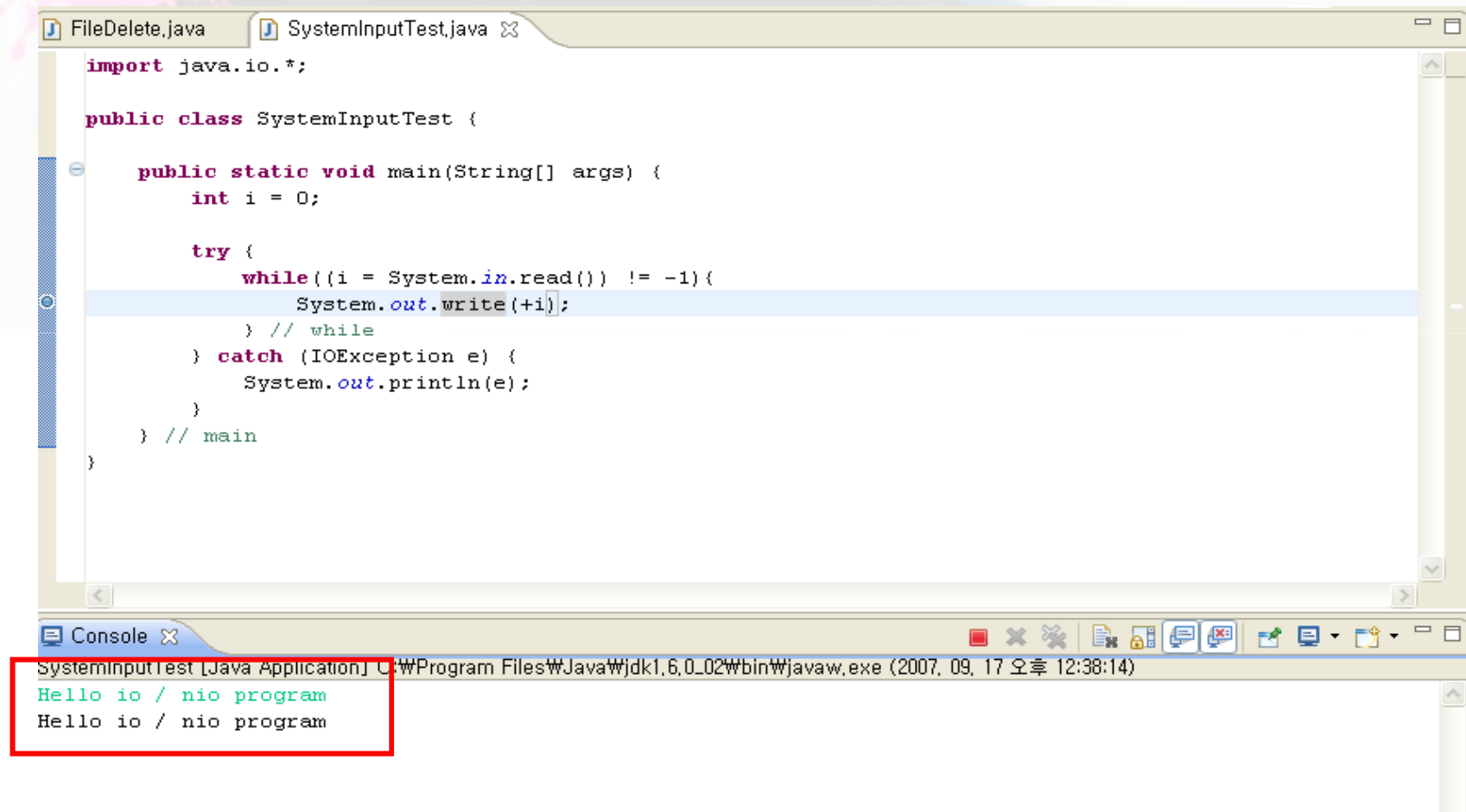
3. 바이트 단위 IO 클래스(계속)

□ System.in 을 이용해서 키보드로부터 입력받기

- 표준 입력 : 보통 키보드로부터의 입력을 의미
- 키보드에서 문자를 누를경우의 전달 경로
 - 키보드를 누를 경우 눌러진 값은 OS가 관리하는 키보드 버퍼에 쌓인다. (JVM 에게는 값이 전달 안된다)
 - 키보드의 엔터키를 눌러야만 JVM에게 값이 전달된다.
 - 엔터키를 입력하면 키보드 버퍼에 있던값과 엔터키 값이 동시에 JVM에게 전달된다.
 - 전달 되어진 문자열은 `read()` 통해서 차례대로 한바이트씩 읽어 들임
- `read()` 를 통해 읽어들이어진 문자열은 `PrintStream` 형식의 `System.out` 에 의해서 출력되어진다.

3. 바이트 단위 IO 클래스(계속)

□ System.in 을 이용해서 키보드로부터 입력받기(예제)



The screenshot shows a Java IDE with two tabs: 'FileDelete.java' and 'SystemInputTest.java'. The 'SystemInputTest.java' tab is active, displaying the following code:

```
import java.io.*;

public class SystemInputTest {

    public static void main(String[] args) {
        int i = 0;

        try {
            while((i = System.in.read()) != -1){
                System.out.write(+i);
            } // while
        } catch (IOException e) {
            System.out.println(e);
        }
    } // main
}
```

Below the code editor is a 'Console' window. It shows the output of the program: 'Hello io / nio program' followed by 'Hello io / nio program' again. The console window title is 'SystemInputTest [Java Application] C:\Program Files\Java\jdk1.6.0_02\bin\javaw.exe (2007. 09. 17 오후 12:38:14)'. The console output is highlighted with a red rectangle.

3. 바이트 단위 IO 클래스(계속)

□ FileInputStream 과 FileOutputStream

- InputStream과 OutputStream 을 상속받는다.
 - 파일로부터 바이트 단위로 입력 받고, 대상 파일에 바이트 단위로 출력할수 있는 클래스
 - 생성자는 파일명이나 파일정보가 있는 File 클래스에 대한 객체이다.
 - FileInputStream 의 경우 생성자에 전달한 파일명이 존재하지 않으면 `java.io.FileNotFoundException` 을 발생시킨다.
-
- FileInputStream 생성자

생성자	설 명
<code>FileInputStream(String filepath)</code> throws <code>FileNotFoundException</code>	<code>filepath</code> 로 지정한 파일로 부터 바이트 단위로 읽어 들이는 스트림 객체를 생성한다.
<code>FileInputStream(File fileobj)</code> throws <code>FileNotFoundException</code>	<code>fileobj</code> 로 지정한 파일로 부터 바이트 단위로 읽어 들이는 스트림 객체를 생성한다.

3. 바이트 단위 IO 클래스(계속)

□ FileInputStream 과 FileOutputStream

- FileOutputStream 생성자

생성자	설 명
FileOutputStream(String filepath) throws IOException	filepath로 지정한 파일에 대한 출력 스트림을 생성
FileOutputStream(String filepath, boolean append) throws IOException	지정한 파일로 출력 스트림을 생성 append 변수값이 true로 설정되면 기존 파일에 이어서 쓰게된다.
FileOutputStream(File fileobj) throws IOException	fileobj로 지정한 파일에 대한 출력 스트림을 생성한다.

3. 바이트 단위 IO 클래스(계속)

□ FileInputStream 과 FileOutputStream

- 파일의 내용을 읽어 들여 화면에 출력하는 프로그램(예제)

```
FileDelete.java  SystemInputTest.java  FileView.java ✕

import java.io.*;

public class FileView {

    public static void main(String[] args) {
        if(args.length != 1){
            System.out.println("사용법 : java FileView 파일이름");
            System.exit(0);
        } // if end

        FileInputStream fis = null;
        try{
            fis = new FileInputStream(args[0]);
            int i = 0;
            while((i = fis.read()) != -1){
                System.out.write(i);
            }
        }catch(Exception ex){
            System.out.println(ex);
        }finally{
            try {
                fis.close();
            } catch (IOException e) {}
        }
    } // main
}
```

표준 스트림을 제외하고 IO 클래스를 사용할때
다음 내용을 반드시 지킨다.

1. try 문에서 사용할 IO 클래스를 선언 (보통 null 값)
2. try 블록안에서 IO 클래스 객체를 생성한다.
3. Finally 블록안에서 IO 클래스의 close() 메소드를 호출한다.

3. 바이트 단위 IO 클래스(계속)

□ FileInputStream 과 FileOutputStream

- 파일의 내용을 읽어 들여 화면에 출력하는 프로그램(결과)

```
C:\ 명령 프롬프트

C:\w\java\weclipse-SDK-3.3-win32\weclipse\workspace\자바 io nio\4장\bin>dir/w
C 드라이브의 볼륨에는 이름이 없습니다.
볼륨 일련 번호: 747A-FF42

C:\w\java\weclipse-SDK-3.3-win32\weclipse\workspace\자바 io nio\4장\bin 디렉터리

[.]
FileExam2.class      FileInfo.class      FileDelete.class
FileView.class       SystemInputTest.class FileList.class
TempFile1.class      8개 파일           11,249 바이트
                     2개 디렉터리      86,640,263,168 바이트 남음

C:\w\java\weclipse-SDK-3.3-win32\weclipse\workspace\자바 io nio\4장\bin>
java FileView ..\src\FileView.java
import java.io.*;

public class FileView {

    public static void main(String[] args) {
        if(args.length != 1){
            System.out.println("사용법 : java FileView 파일이름");
            System.exit(0);
        } // if end

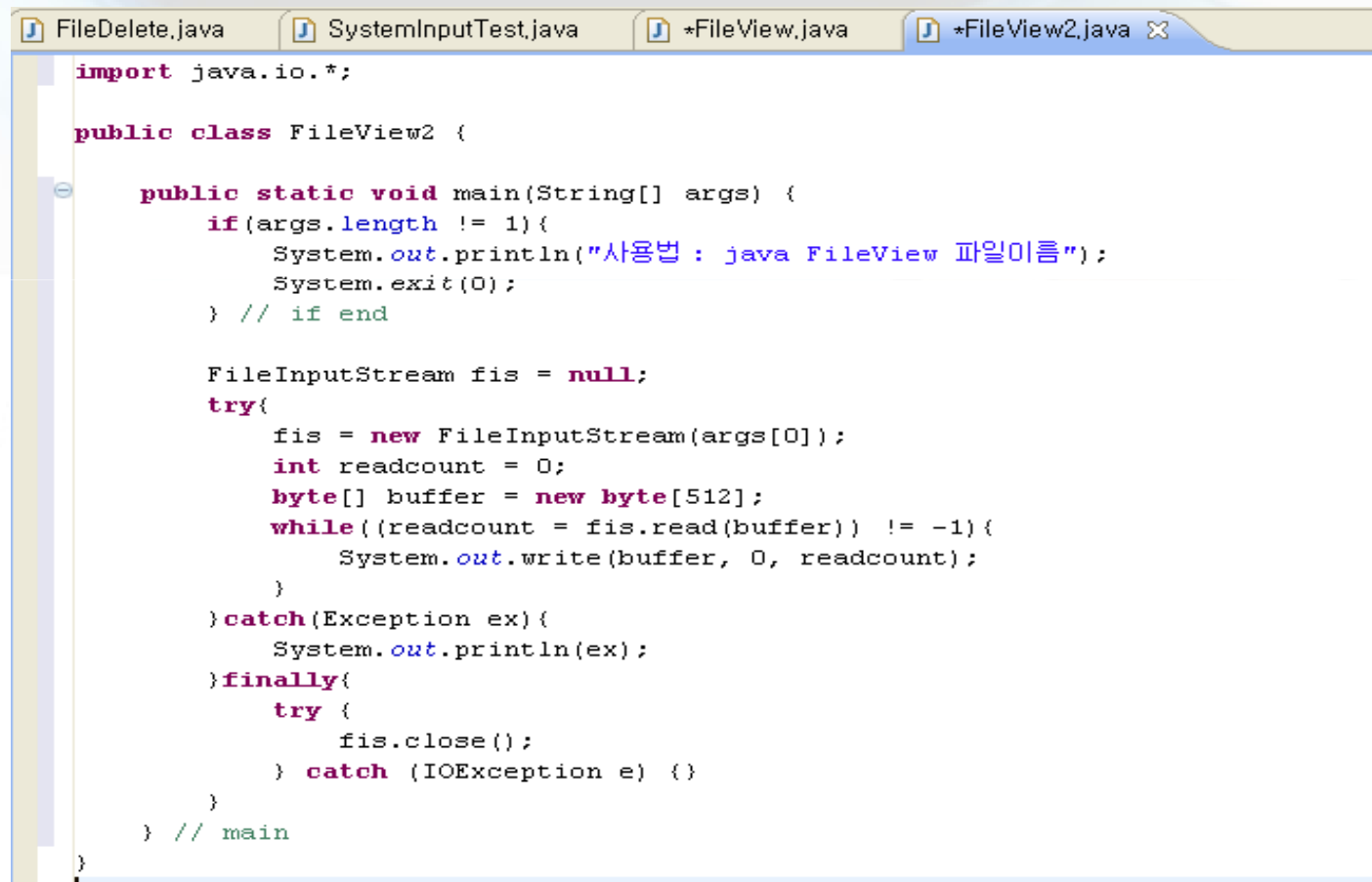
        FileInputStream fis = null;
        try{
            fis = new FileInputStream(args[0]);
            int i = 0;
            while((i = fis.read()) != -1){
                System.out.write(i);
            }
        }catch(Exception ex){
            System.out.println(ex);
        }finally{
            try {
                fis.close();
            } catch (IOException e) {}
        }
    } // main
}

C:\w\java\weclipse-SDK-3.3-win32\weclipse\workspace\자바 io nio\4장\bin>
```

3. 바이트 단위 IO 클래스(계속)

□ FileInputStream 과 FileOutputStream

- 파일을 읽어 출력하는 프로그램의 개선(예제)



```
import java.io.*;

public class FileView2 {

    public static void main(String[] args) {
        if(args.length != 1){
            System.out.println("사용법 : java FileView 파일이름");
            System.exit(0);
        } // if end

        FileInputStream fis = null;
        try{
            fis = new FileInputStream(args[0]);
            int readcount = 0;
            byte[] buffer = new byte[512];
            while((readcount = fis.read(buffer)) != -1){
                System.out.write(buffer, 0, readcount);
            }
        }catch(Exception ex){
            System.out.println(ex);
        }finally{
            try {
                fis.close();
            } catch (IOException e) {}
        }
    } // main
}
```


3. 바이트 단위 IO 클래스(계속)

□ FileInputStream 과 FileOutputStream

- 파일을 읽어 출력하는 프로그램의 개선(예제)
 - 자바 프로그래밍에서 1byte 읽으라고 실행하면
: OS는 1byte를 읽지 않고 보통 인접한 256byte 나 512byte를 읽는다.
(1000byte 파일을 1바이트씩 읽어오라고 실행하면 내부적으로 512byte씩 1000번 읽어온다는 것을 의미)
 - 이문제를 해결하기 위해서는 1byte 단위가 아니라 512byte씩 읽어오라고 OS 에게 명령을 내려야 한다
(1000byte 파일이라면 2번만 디스크를 읽어 들이면 되기 때문에 상당히 효율적이다.)

3. 바이트 단위 IO 클래스(계속)

□ FileInputStream 과 FileOutputStream

- 파일을 읽어 출력하는 프로그램의 개선(예제)

```
int readcount = 0;
byte[] buffer = new byte[512];
while((readcount = fis.read(buffer)) != -1){
    System.out.write(buffer, 0, readcount);
}
```

- 이전예제와 달라진 코드 부분
- read() 메소드에 512byte 배열을 전달함으로써 읽어들이는 내용을 바이트 배열인 buffer에 저장된다.
- 저장된 바이트 배열은 write() 메소드에 의해 화면에 출력
- write() 메소드의 경우, 바이트 배열의 0번째부터 읽어 들인 바이트의 수 만큼만 화면에 출력
(파일로부터 읽어 들여야 할 내용이 없을 경우에는 -1을 반환)

3. 바이트 단위 IO 클래스(계속)

□ FileInputStream 과 FileOutputStream

- 파일을 읽어 출력하는 프로그램의 개선(결과)

```
C:\java\weclipse-SDK-3.3-win32\weclipse\workspace\자바 io nio\4장\bin>dir/w
C 드라이브의 볼륨에는 이름이 없습니다.
볼륨 일련 번호: 747A-FF42

C:\java\weclipse-SDK-3.3-win32\weclipse\workspace\자바 io nio\4장\bin 디렉터리

[.]
FileExam2.class      [..]      FileDelete.class
FileInfo.class       FileInfo2.class  FileList.class
FileView.class       TempFile1.class SystemInputTest.class
TempFile.class
          9개 파일              12,505 바이트
          2개 디렉터리 86,640,209,920 바이트 남음

C:\java\weclipse-SDK-3.3-win32\weclipse\workspace\자바 io nio\4장\bin>
java FileView2 ..\src\FileView2.java
import java.io.*;

public class FileView2 {

    public static void main(String[] args) {
        if(args.length != 1){
            System.out.println("사용법 : java FileView 파일이름");
            System.exit(0);
        } // if end

        FileInputStream fis = null;
        try{
            fis = new FileInputStream(args[0]);
            int readcount = 0;
            byte[] buffer = new byte[512];
            while((readcount = fis.read(buffer)) != -1){
                System.out.write(buffer, 0, readcount);
            }
        }catch(Exception ex){
            System.out.println(ex);
        }finally{
            try {
                fis.close();
            } catch (IOException e) {}
        }
    }

    > // main
}

C:\java\weclipse-SDK-3.3-win32\weclipse\workspace\자바 io nio\4장\bin>
```

3. 바이트 단위 IO 클래스(계속)

□ FileInputStream 과 FileOutputStream

- 파일 복사(예제)

```
*FileStreamCopy.java X
import java.io.*;

public class FileStreamCopy {

    public static void main(String[] args) {
        if(args.length != 2){
            System.out.println("사용법 : java FileStreamCopy 파일1 파일2");
            System.exit(0);
        } // if end

        FileInputStream fis = null;
        FileOutputStream fos = null;
        try{
            fis = new FileInputStream(args[0]);
            fos = new FileOutputStream(args[1]);
            byte[] buffer = new byte[512];
            int readcount = 0;
            while((readcount = fis.read(buffer)) != -1){
                fos.write(buffer,0,readcount);
            }
            System.out.println("복사가 완료되었습니다.");
        } catch (Exception ex){
            System.out.println(ex);
        } finally{
            try {
                fis.close();
            } catch (IOException ex) {}
            try {
                fos.close();
            } catch (IOException ex) {}
        }
    } // main
}
```

3. 바이트 단위 IO 클래스(계속)

□ FileInputStream 과 FileOutputStream

- 파일 복사(결과)

```
C:\명령 프롬프트
C:\WjavaWeclipse-SDK-3.3-win32Weclipse\workspace\자바 io nio\4장\bin>dir/w
C 드라이브의 볼륨에는 이름이 없습니다.
볼륨 일련 번호: 747A-FF42

C:\WjavaWeclipse-SDK-3.3-win32Weclipse\workspace\자바 io nio\4장\bin 디렉터리

[.]
FileExam2.class      [..]      FileDelete.class
FileStreamCopy.class  FileInfo.class  FileList.class
SystemInputTest.class  FileView.class  FileView2.class
TempFile1.class       TempFile.class  TempFile1.class
10개 파일              14,051 바이트
2개 디렉터리          86,639,788,032 바이트 남음

C:\WjavaWeclipse-SDK-3.3-win32Weclipse\workspace\자바 io nio\4장\bin>java FileStr
eamCopy FileStreamCopy.class 2.class
복사가 완료되었습니다.

C:\WjavaWeclipse-SDK-3.3-win32Weclipse\workspace\자바 io nio\4장\bin>dir/w
C 드라이브의 볼륨에는 이름이 없습니다.
볼륨 일련 번호: 747A-FF42

C:\WjavaWeclipse-SDK-3.3-win32Weclipse\workspace\자바 io nio\4장\bin 디렉터리

[.]
FileDelete.class     [..]      2.class
FileExam2.class      FileInfo.class
FileList.class        FileStreamCopy.class  FileView.class
FileView2.class       SystemInputTest.class  TempFile.class
TempFile1.class
11개 파일              15,597 바이트
2개 디렉터리          86,639,779,840 바이트 남음

C:\WjavaWeclipse-SDK-3.3-win32Weclipse\workspace\자바 io nio\4장\bin>
```

명령실행, 원본파일명, 복사파일명

복사완료

3. 바이트 단위 IO 클래스(계속)

□ DataInputStream 과 DataOutputStream

- 자바의 기본형 데이터인 int, float, double, boolean, short, byte 등의 정보를 입력하고 출력하는 클래스
- 생성자에서 InputStream, OutputStream 을 받아들인다.
(다른 바이트 스트림을 통해서 읽어 들이거나 쓴다는 의미 내포)

• DataInputStream 과 DataOutputStream 이 각각 InputStream과 OutputStream 을 받아들인다는것은 InputStream의 자식 클래스, OutputStream의 자식 클래스나 자손클래스를 받아들인다는 것을 의미한다.

• DataInputStream 생성자

생성자	설 명
DataInputStream (InputStream inputStream)	inputStream 을 인자로 DataInputStream을 생성한다.

3. 바이트 단위 IO 클래스(계속)

□ DataInputStream 과 DataOutputStream

• DataInputStream 생성자의 추가 메소드

[표 4-8] DataInputStream 생성자의 추가 메소드

메소드	설명
boolean readBoolean() throws IOException	스트림으로부터 읽은 boolean을 반환한다.
byte readByte() throws IOException	스트림으로부터 읽은 byte를 반환한다.
char readChar() throws IOException	스트림으로부터 읽은 char를 반환한다.
double readDouble() throws IOException	스트림으로부터 읽은 double을 반환한다.
float readFloat() throws IOException	스트림으로부터 읽은 float를 반환한다.
long readLong() throws IOException	스트림으로부터 읽은 long을 반환한다.
short readShort() throws IOException	스트림으로부터 읽은 short를 반환한다.
int readInt() throws IOException	스트림으로부터 읽은 int를 반환한다.
void readFully(byte buf[]) throws IOException	스트림으로부터 buf 크기만큼의 바이트를 읽어 buf에 저장한다.
String readUTF() throws IOException	UTF 인코딩 값을 얻어 문자열로 반환한다. 네트워크 프로그래밍을 할 때, 네트워크로부터 문자열을 읽어 들일 때 많이 사용한다.

• DataOutputStream 생성자

생성자	설 명
DataOutputStream (OutputStream outputStream)	outputStream을 인자로 DataOutputStream을 생성한다.

3. 바이트 단위 IO 클래스(계속)

□ DataInputStream 과 DataOutputStream

• DataOutputStream 생성자의 추가 메소드

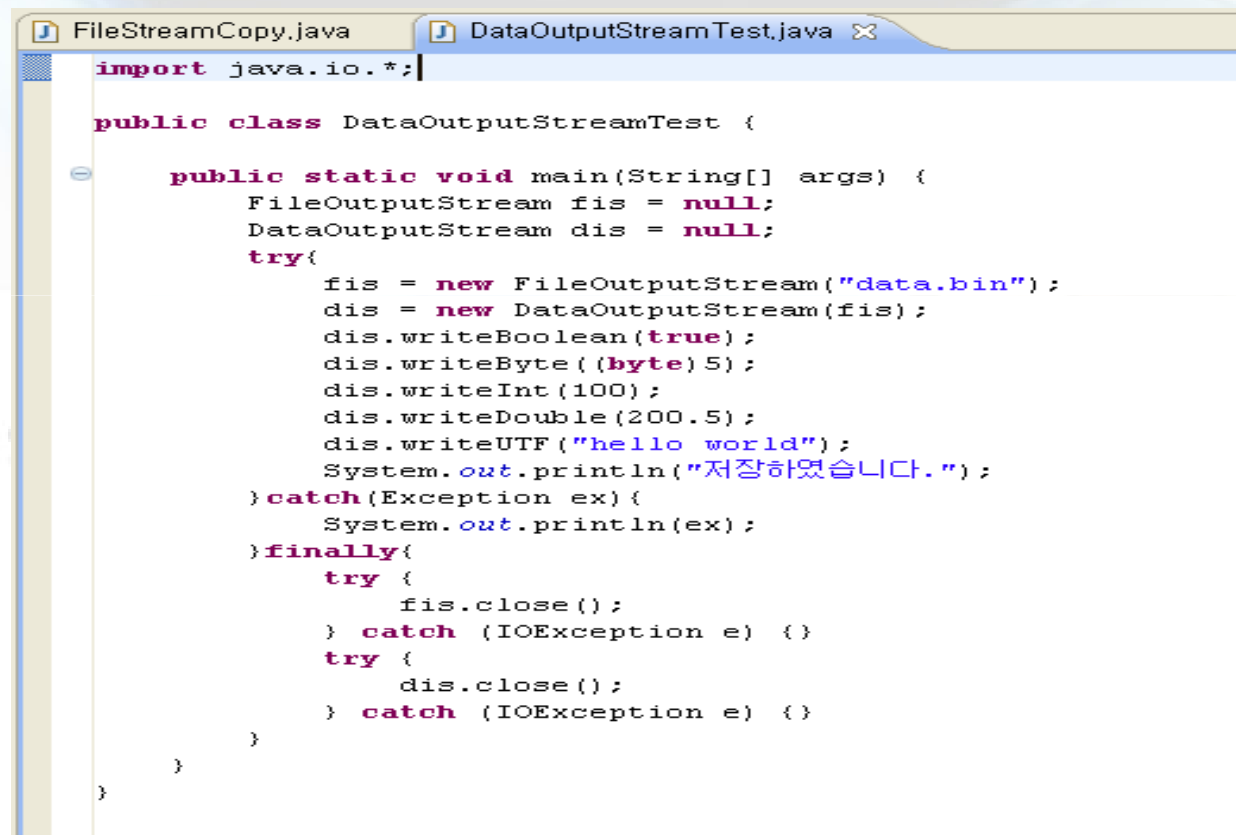
[표 4-10] DataOutputStream 생성자의 추가 메소드

메소드	설명
void write(int i) throws IOException	int형 i 값이 갖는 1바이트를 출력한다.
void write(byte buf[], int index, int size) throws IOException	바이트 배열 buf의 주어진 위치 index에서 size만큼 출력한다.
void writeBoolean(boolean b) throws IOException	boolean을 1바이트 값으로 출력한다.
void writeByte(int i) throws IOException	int를 4바이트 값으로 상위 바이트 먼저 출력한다.
void writeBytes(String s) throws IOException	문자열을 바이트순으로 출력한다.
void writeChar(int i) throws IOException	char를 2바이트 값으로 상위 바이트를 먼저 출력한다.
void writeDouble(double d) throws IOException	Double 클래스의 doubleToBits()를 사용해서 long으로 변환한 다음 long 값을 8바이트 수량으로 상위 바이트 먼저 출력한다.
void writeFloat(float f) throws IOException	Float를 floatToBits()를 사용해서 변환한 후, int 값을 4바이트 수량으로 상위 바이트 먼저 출력한다.
void writeInt(int i) throws IOException	int의 상위 바이트를 먼저 출력한다.
void writeLong(long l) throws IOException	long형의 인자 값을 출력한다.
void writeShort(short s) throws IOException	short형의 인자 값을 출력한다.
void writeUTF(String s) throws IOException	UTF-8 인코딩을 사용해서 문자열을 출력한다. 네트워크 프로그래밍을 할 때, 문자열 전송 시에 자주 사용한다.

3. 바이트 단위 IO 클래스(계속)

□ DataInputStream 과 DataOutputStream

- 예제) 4- 9



```
import java.io.*;

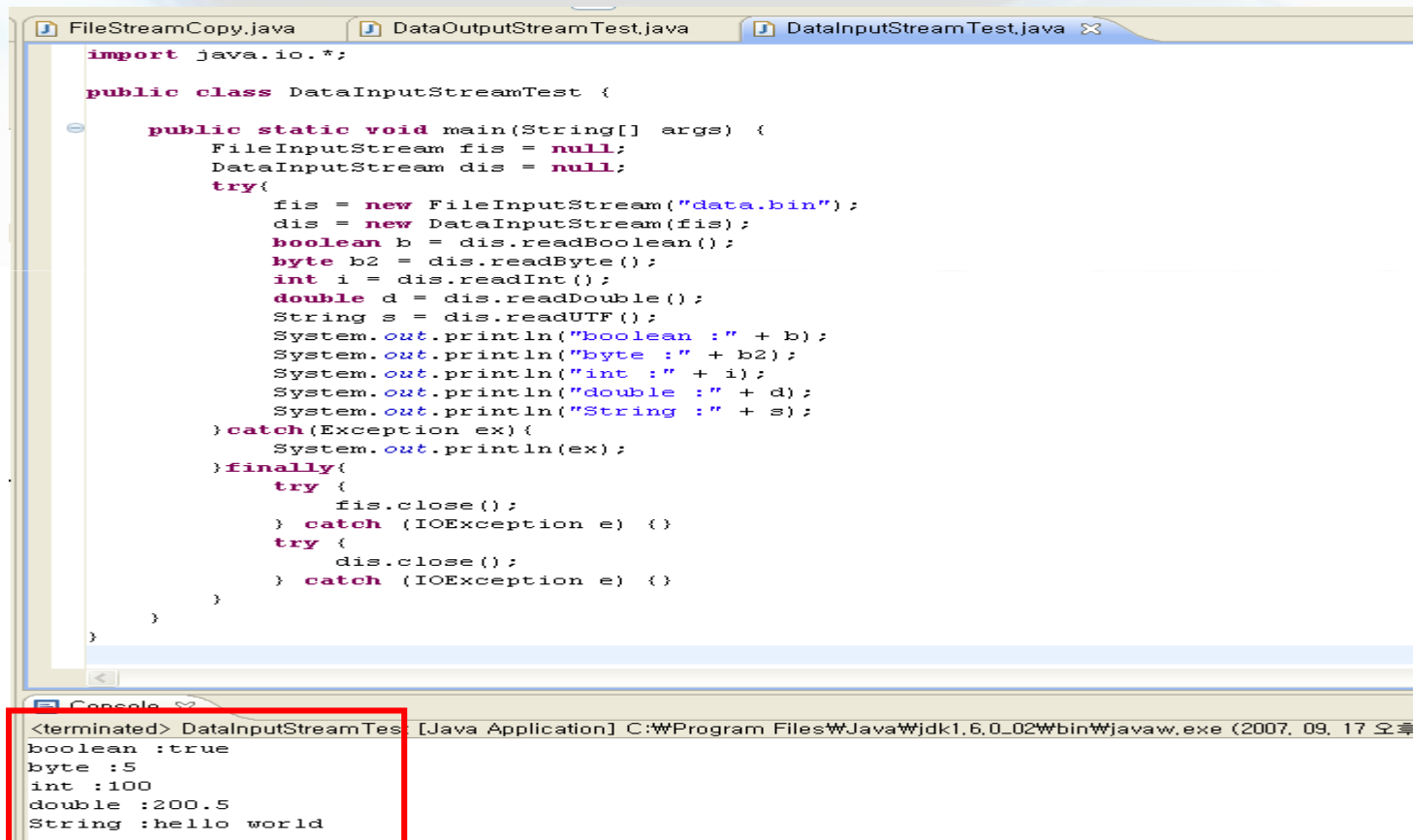
public class DataOutputStreamTest {

    public static void main(String[] args) {
        FileOutputStream fis = null;
        DataOutputStream dis = null;
        try{
            fis = new FileOutputStream("data.bin");
            dis = new DataOutputStream(fis);
            dis.writeBoolean(true);
            dis.writeByte((byte)5);
            dis.writeInt(100);
            dis.writeDouble(200.5);
            dis.writeUTF("hello world");
            System.out.println("저장하였습니다.");
        } catch(Exception ex){
            System.out.println(ex);
        } finally{
            try {
                fis.close();
            } catch (IOException e) {}
            try {
                dis.close();
            } catch (IOException e) {}
        }
    }
}
```

3. 바이트 단위 IO 클래스(계속)

□ DataInputStream 과 DataOutputStream

- 예제) 4- 10



```
import java.io.*;

public class DataInputStreamTest {

    public static void main(String[] args) {
        FileInputStream fis = null;
        DataInputStream dis = null;
        try{
            fis = new FileInputStream("data.bin");
            dis = new DataInputStream(fis);
            boolean b = dis.readBoolean();
            byte b2 = dis.readByte();
            int i = dis.readInt();
            double d = dis.readDouble();
            String s = dis.readUTF();
            System.out.println("boolean : " + b);
            System.out.println("byte : " + b2);
            System.out.println("int : " + i);
            System.out.println("double : " + d);
            System.out.println("String : " + s);
        } catch (Exception ex) {
            System.out.println(ex);
        } finally {
            try {
                fis.close();
            } catch (IOException e) {}
            try {
                dis.close();
            } catch (IOException e) {}
        }
    }
}
```

<terminated> DataInputStreamTest [Java Application] C:\Program Files\Java\jdk1.6.0_02\bin\javaw.exe (2007. 09. 17 오후
boolean :true
byte :5
int :100
double :200.5
String :hello world

3. 바이트 단위 IO 클래스(계속)

□ ByteArrayInputStream 과 ByteArrayOutputStream

- 바이트 배열을 읽어 들이기 위한 클래스
 - ByteArrayOutputStream에 있는 메소드를 이용해서 출력하면 출력되는 모든 내용이 내부적인 저장 공간에 쌓인다.
 - ByteArrayOutputStream에 있는 toByteArray() 를 실행하면 저장된 모든 내용이 바이트 배열로 반환
-
- ByteArrayInputStream 클래스 생성자

생성자	설 명
ByteArrayInputStream(byte[] buf)	바이트 배열 buf로부터 읽어들이는 ByteArrayInputStream을 생성
ByteArrayInputStream(byte[] buf, int offset, int length)	바이트 배열 buf의 offset부터 length 길이까지 읽어 들이는 ByteArrayInputStream을 생성한다.

3. 바이트 단위 IO 클래스(계속)

□ ByteArrayInputStream 과 ByteArrayOutputStream

- ByteArrayOutputStream 클래스 생성자

생성자	설 명
ByteArrayOutputStream()	바이트 배열을 저장할 수 있는 공간을 내부적으로 가지고 있는 ByteArrayOutputStream 을 생성한다.

- toByteArray() 메소드

생성자	설 명
byte[]toByteArray()	ByteArrayOutputStream의 내부 저장 공간에 저장되어 있는 바이트 배열을 반환한다.

3. 바이트 단위 IO 클래스(계속)

□파일 내용을 바이트 배열에 저장한 후 화면에 출력 (예제 4-11)

- FileInputStream으로 파일 내용 읽어 ByteArrayOutputStream으로 내부저장 공간에 출력

```
fis = new FileInputStream(args[0]);
baos = new ByteArrayOutputStream();
byte[] buffer = new byte[512];
int readcount = 0;
// 파일로 부터 읽어들이는 byte배열을 ByteArrayOutputStream에게 쓴다.
while((readcount = fis.read(buffer)) != -1){
    baos.write(buffer, 0, readcount);
}
```

- toByteArray()이용하여 내부공간의 내용을 byte array로 반환

```
byte[] fileArray = baos.toByteArray();
```

- ByteArrayInputStream의 read()로 바이트배열 읽어 출력

```
bais = new ByteArrayInputStream(fileArray);
// ByteArrayInputStream을 통하여 읽어들이는 byte배열을 표준 출력 장치(모니터)
// 에 출력한다.
while((readcount = bais.read(buffer)) != -1){
    System.out.write(buffer, 0, readcount);
}
```

3. 바이트 단위 IO 클래스(계속)

□파일이나 배열의 내용을 읽어 출력하는 클래스(예제 4-12)

– InputStream을 인자로 받는 print()

```
public static void print(InputStream in){
    byte[] buffer = new byte[512];
    int readcount = 0;
    try {
        while((readcount = in.read(buffer)) != -1){
            System.out.write(buffer, 0, readcount);
        }
    } catch (IOException e) {
        System.out.println(e);
    }
} // print
```

– FileInputStream이나 ByteArrayInputStream으로 읽어 print()로 출력

◆ File

```
fis = new FileInputStream("file.dat");
print(fis);
```

◆ array

```
bais = new ByteArrayInputStream(abc);
print(bais);
```

3. 바이트 단위 IO 클래스(계속)

□PipedInputStream과 PipedOutputStream

- Unix에서의 Pipe
 - ◆ 파이프 기호로 앞 명령의 결과를 뒤 명령의 입력으로 바꾸는 역할
 - ◆ `Ls | wc`
- PipedOutputStream의 결과를 PipedInputStream을 통해 읽을 수 있음
- 주로 멀티스레드에서 종종 사용
 - ◆ 하나의 스레드가 읽은 내용을 다른 스레드에 전달하고자 할때
- PipedInputStream 생성자/메소드
 - ◆ `PipedInputStream()`: 연결되지 않은 `PipedInputStream` 생성
 - ◆ `PipedInputStream(PipedOutputStream src)`: `src`에 연결된 `PipedInputStream` 생성
 - ◆ `Void connect(PipedOutputStream src)`: 연결할 `src` 지정
- PipedOutputStream 생성자/메소드
 - ◆ `PipedOutputStream()`: 연결되지 않은 `PipedOutputStream` 생성
 - ◆ `PipedOutputStream(PipedInputStream snk)`: `snk`에 연결된 `PipedOutputStream` 생성
 - ◆ `Void connect(PipedInputStream snk)`: 연결할 `snk` 지정

3. 바이트 단위 IO 클래스(계속)

□예제 4-13 SystemStream

- ReadThread는 InputStream, OutputStream 인자로 받음
- PipedInputStream을 생성하여 PipedOutputStream에 지정

```
PipedInputStream writeIn = new PipedInputStream();
PipedOutputStream readOut = new PipedOutputStream( writeIn );

ReadThread rt = new ReadThread( System.in, readOut );
ReadThread wt = new ReadThread( writeIn, System.out );
```

- InputStream에서 읽어들이어 OutputStream으로 출력하는 스레드

```
public void run() {
    int ch;
    int i;
    try {
        for(;;) {
            i = pi.read();
            if (i == -1) { return; }
            po.write(i);
        }
    } catch (Exception e) { }
}
```

