

제 14 장

입출력(I/O)

1. 입출력(I/O)

1.1 입출력(I/O)과 스트림(stream)

1.2 바이트기반 스트림 – InputStream, OutputStream

1.3 보조스트림

1.4 문자기반 스트림 – Reader, Writer

2. 바이트기반 스트림

2.1 InputStream과 OutputStream

2.2 ByteArrayInputStream과 ByteArrayOutputStream

2.3 FileInputStream과 FileOutputStream

3. 바이트기반 보조스트림

3.1 FilterInputStream과 FilterOutputStream

3.2 BufferedInputStream과 BufferedOutputStream

3.3 DataInputStream과 DataOutputStream

3.4 SequenceInputStream

3.5 PrintStream

4. 문자기반 스트림

4.1 Reader와 Writer

4.2 FileReader와 FileWriter

4.3 PipedReader와 PipedWriter

4.4 StringReader와 StringWriter

5. 문자기반 보조스트림

5.1 BufferedReader와 BufferedWriter

5.2 InputStreamReader와 OutputStreamWriter

6. 표준입출력과 File

6.1 표준입출력

6.2 RandomAccessFile

6.3 File

7. 직렬화(serialization)

7.1 직렬화(serialization)란?

7.2 ObjectInputStream, ObjectOutputStream

7.3 직렬화 가능한 클래스 만들기

7.4 직렬화 가능한 클래스의 버전 관리

1. 입출력(I/O)

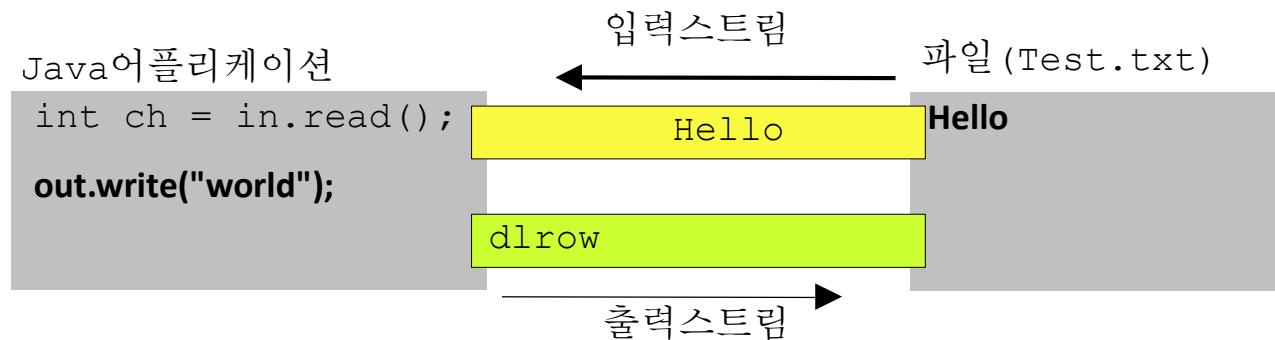
1.1 입출력(I/O)과 스트림(stream)

▶ 입출력(I/O)이란?

- 입력(Input)과 출력(Output)을 줄여 부르는 말
- 두 대상 간의 데이터를 주고 받는 것

▶ 스트림(stream)이란?

- 데이터를 운반(입출력)하는데 사용되는 연결통로
- 연속적인 데이터의 흐름을 물(stream)에 비유해서 붙여진 이름
- 하나의 스트림으로 입출력을 동시에 수행할 수 없다.(단방향 통신)
- 입출력을 동시에 수행하려면, 2개의 스트림이 필요하다.



1.2 바이트기반 스트림 – InputStream, OutputStream

- 데이터를 바이트(byte)단위로 주고 받는다.

| InputStream | OutputStream |
|--------------------------------------|--|
| abstract int read() | abstract void write(int b) |
| int read(byte[] b) | void write(byte[] b) |
| int read(byte[] b, int off, int len) | void write(byte[] b, int off, int len) |

```

public abstract class InputStream {
    ...
    // 입력스트림으로부터 1 byte를 읽어서 반환한다. 읽을 수 없으면 -1을 반환한다.
    abstract int read();

    // 입력스트림으로부터 len개의 byte를 읽어서 byte배열 b의 off위치부터 저장한다.
    int read(byte[] b, int off, int len) {
        ...
        for(int i=off; i < off + len; i++) {
            // read()를 호출해서 데이터를 읽어서 배열
            b[i] = (byte)read();
        }
        ...
    }
    // 입력스트림으로부터 byte배열 b의 크기만큼 데이터를 읽어서 배열 b에 저장한다.
    int read(byte[] b) {
        return read(b, 0, b.length);
    }
    ...
}

```

| 입력스트림 | 출력스트림 | 대상 |
|----------------------|-----------------------|-------|
| FileInputStream | FileOutputStream | 파일 |
| ByteArrayInputStream | ByteArrayOutputStream | 메모리 |
| PipedInputStream | PipedOutputStream | 프로세스 |
| AudioInputStream | AudioOutputStream | 오디오장치 |

1.3 보조스트림

- 스트림의 기능을 향상시키거나 새로운 기능을 추가하기 위해 사용
- 독립적으로 입출력을 수행할 수 없다.

```
// 먼저 기반스트림을 생성한다.
FileInputStream fis = new FileInputStream("test.txt");
// 기반스트림을 이용해서 보조스트림을 생성한다.
BufferedInputStream bis = new BufferedInputStream(fis);

bis.read(); // 보조스트림인 BufferedInputStream으로부터 데이터를 읽는다.
```

| 입력 | 출력 | 설명 |
|-----------------------|----------------------|--|
| FilterInputStream | FilterOutputStream | 필터를 이용한 입출력 처리 |
| BufferedInputStream | BufferedOutputStream | 버퍼를 이용한 입출력 성능향상 |
| DataInputStream | DataOutputStream | int, float와 같은 기본형 단위(primitive type)로 데이터를 처리하는 가능 |
| SequenceInputStream | SequenceOutputStream | 두 개의 스트림을 하나로 연결 |
| LineNumberInputStream | 없음 | 읽어 온 데이터의 라인 번호를 카운트 (JDK1.1부터 LineNumberReader로 대체) |
| ObjectInputStream | ObjectOutputStream | 데이터를 객체단위로 읽고 쓰는데 사용. 주로 파일을 이용하여 객체 직렬화와 관련있음 |
| 없음 | PrintStream | 버퍼를 이용하며, 추가적인 print관련 기능(print, printf, println메서드) |
| PushbackInputStream | 없음 | 버퍼를 이용해서 읽어 온 데이터를 다시 되돌리는 기능 (unread, push back to buffer) |

1.4 문자기반 스트림 – Reader, Writer

- 입출력 단위가 문자(char, 2 byte)인 스트림. 문자기반 스트림의 최고조상

| 바이트기반 스트림 | 문자기반 스트림 | 대상 | 바이트기반 보조스트림 | 문자기반 보조스트림 |
|---|--|------|---|---|
| <code>FileInputStream</code> <code> FileOutputStream</code> | <code>FileReader</code> <code> FileWriter</code> | 파일 | <code>BufferedInputStream</code> <code>BufferedOutputStream</code> | <code>BufferedReader</code> <code> BufferedWriter</code> |
| <code>ByteArrayInputStream</code> <code>ByteArrayOutputStream</code> | <code>CharArrayReader</code> <code>CharArrayWriter</code> | 메모리 | <code>FilterInputStream</code> <code>FilterOutputStream</code> | <code>FilterReader</code> <code> FilterWriter</code> |
| <code>PipedInputStream</code> <code>PipedOutputStream</code> | <code>PipedReader</code> <code>Pipewriter</code> | 프로세스 | <code>LineNumberInputStream</code> | <code>LineNumberReader</code> |
| <code>StringBufferInputStream</code> <code>StringBufferOutputStream</code> | <code>StringReader</code> <code>StringWriter</code> | 메모리 | <code>PrintStream</code> | <code>PrintWriter</code> |
| | | | <code>PushbackInputStream</code> | <code>PushbackReader</code> |



| InputStream | Reader |
|---|---|
| abstract int read() int read(byte [] b) int read(byte [] b, int off, int len) | int read() int read(char [] cbuf) abstract int read(char [] cbuf, int off, int len) |
| OutputStream | Writer |
| abstract void write(int b) void write(byte [] b) void write(byte [] b, int off, int len) | void write(int c) void write(char [] cbuf) abstract void write(char [] cbuf, int off, int len) void write(String str) void write(String str, int off, int len) |

2. 바이트기반 스트림

2.1 InputStream과 OutputStream

▶ InputStream(바이트기반 입력스트림의 최고 조상)의 메서드

| 메서드명 | 설 명 |
|--------------------------------------|---|
| int available() | 스트림으로부터 읽어 올 수 있는 데이터의 크기를 반환한다. |
| void close() | 스트림을 닫음으로써 사용하고 있던 자원을 반환한다. |
| void mark(int readlimit) | 현재위치를 표시해 놓는다. 후에 reset()에 의해서 표시해 놓은 위치로 다시 돌아갈 수 있다. readlimit은 되돌아갈 수 있는 byte의 수이다. |
| boolean markSupported() | mark()와 reset()을 지원하는지를 알려 준다. mark()와 reset() 기능을 지원하는 것은 선택적이므로, mark()와 reset()을 사용하기 전에 markSupported()를 호출해서 지원 여부를 확인해야 한다. |
| abstract int read() | 1 byte를 읽어 온다(0~255사이의 값). 더 이상 읽어 올 데이터가 없으면 -1을 반환한다. abstract메서드라서 InputStream의 자손들은 자신의 상황에 알맞게 구현해야 한다. |
| int read(byte[] b) | 배열 b의 크기만큼 읽어서 배열을 채우고 읽어 온 데이터의 수를 반환한다. 반환하는 값은 항상 배열의 크기보다 작거나 같다. |
| int read(byte[] b, int off, int len) | 최대 len개의 byte를 읽어서, 배열 b의 지정된 위치(off)부터 저장한다. 실제로 읽어 올 수 있는 데이터가 len개보다 적을 수 있다. |
| void reset() | 스트림에서의 위치를 마지막으로 mark()이 호출되었던 위치로 되돌린다. |
| long skip(long n) | 스트림에서 주어진 길이(n)만큼을 건너뛴다. |

▶ OutputStream(바이트기반 출력스트림의 최고 조상)의 메서드

| 메서드명 | 설 명 |
|--|---|
| void close() | 입력소스를 닫음으로써 사용하고 있던 자원을 반환한다. |
| void flush() | 스트림의 버퍼에 있는 모든 내용을 출력소스에 쓴다. |
| abstract void write(int b) | 주어진 값을 출력소스에 쓴다. |
| void write(byte[] b) | 주어진 배열 b에 저장된 모든 내용을 출력소스에 쓴다. |
| void write(byte[] b, int off, int len) | 주어진 배열 b에 저장된 내용 중에서 off번째부터 len개 만큼만 읽어서 출력소스에 쓴다. |

2.2 ByteArrayInputStream과 ByteArrayOutputStream

- 바이트배열(byte[])에 데이터를 입출력하는 바이트기반 스트림

```

import java.io.*;
import java.util.Arrays;

class IOEx1 {
    public static void main(String[] args) {
        byte[] inSrc = {0,1,2,3,4,5,6,7,8,9};
        byte[] outSrc = null;

        ByteArrayInputStream input = null;
        ByteArrayOutputStream output = null;

        input = new ByteArrayInputStream(inSrc);
        output = new ByteArrayOutputStream();

        int data = 0;

        while((data = input.read()) != -1) {
            output.write(data); // void write(int b)
        }

        outSrc = output.toByteArray(); // 스트림의 내용을 byte배열로 반환한다.

        System.out.println("Input Source :" + Arrays.toString(inSrc));
        System.out.println("Output Source :" + Arrays.toString(outSrc));
    }
}

```

(data = input.read()) != -1
 ① data = input.read() // read()를 호출한 반환값을 변수 data에 저장한다.
 ② data != -1 // data에 저장된 값이 -1이 아닌지 비교한다.

| | |
|---------------------|--|
| abstract int read() | 1 byte를 읽어 온다(0~255사이의 값). 더 이상 읽어 올 데이터가 없으면 -1을 반환한다. |
|---------------------|--|

[실행결과]

| |
|---|
| Input Source :[0, 1, 2, 3, 4, 5, 6, 7, 8, 9] |
| Output Source :[0, 1, 2, 3, 4, 5, 6, 7, 8, 9] |

2.2 ByteArrayInputStream과 ByteArrayOutputStream

```

import java.io.*;
import java.util.Arrays;

class IOEx3 {
    public static void main(String[] args) {
        byte[] inSrc = {0,1,2,3,4,5,6,7,8,9};
        byte[] outSrc = null;

        byte[] temp = new byte[4]; // 이전 예제와 배열의 크기가 다르다.

        ByteArrayInputStream input = null;
        ByteArrayOutputStream output = null;

        input = new ByteArrayInputStream(inSrc);
        output = new ByteArrayOutputStream();

        try {
            while(input.available() > 0) {
                input.read(temp);
                output.write(temp);
            }
        } catch(IOException e) {}

        outSrc = output.toByteArray();

        System.out.println("Input Source : " + Arrays.toString(inSrc));
        System.out.println("temp          :" + Arrays.toString(temp));
        System.out.println("Output Source :" + Arrays.toString(outSrc));
    }
}

```

【실행결과】

| | |
|-----------------|--------------------------------------|
| Input Source : | [0, 1, 2, 3, 4, 5, 6, 7, 8, 9] |
| temp | :[8, 9, 6, 7] |
| Output Source : | [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 6, 7] |

【실행결과】

| | |
|-----------------|--------------------------------|
| Input Source : | [0, 1, 2, 3, 4, 5, 6, 7, 8, 9] |
| temp | :[8, 9, 6, 7] |
| Output Source : | [0, 1, 2, 3, 4, 5, 6, 7, 8, 9] |

2.3 FileInputStream과 FileOutputStream

- 파일(file)에 데이터를 입출력하는 바이트기반 스트림

| 생성자 | 설명 |
|---|---|
| FileInputStream(String name) | 지정된 파일이름(name)을 가진 실제 파일과 연결된 FileInput Stream을 생성한다. |
| FileInputStream(File file) | 파일의 이름이 String이 아닌 File인스턴스로 지정해주어야 하는 점을 제외하고 FileInputStream(String name)과 같다. |
| FileOutputStream(String name) | 지정된 파일이름(name)을 가진 실제 파일과의 연결된 File OutputStream을 생성한다. |
| FileOutputStream(String name, boolean append) | 지정된 파일이름(name)을 가진 실제 파일과 연결된 File OutputStream을 생성한다. 두번째 인자인 append를 true로 하면, 출력 시 기존의 파일내용의 마지막에 덧붙인다. false면, 기존의 파일내용을 덮어쓰게 된다. |
| FileOutputStream(File file) | 파일의 이름을 String이 아닌 File인스턴스로 지정해주어야 하는 점을 제외하고 FileOutputStream(String name)과 같다. |

```

import java.io.*;

class FileCopy {
    public static void main(String args[]) {
        try {
            FileInputStream fis = new FileInputStream(args[0]);
            FileOutputStream fos = new FileOutputStream(args[1]);

            int data =0;
            while((data=fis.read())!=-1) {
                fos.write(data);      // void write(int b)
            }

            fis.close();
            fos.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

```

[실행결과]

```
C:\jdk1.5\work\ch14>java FileCopy FileCopy.java FileCopy.bak
C:\jdk1.5\work\ch14>
```

3. 바이트기반 보조스트림

3.1 FilterInputStream과 FilterOutputStream

- 모든 바이트기반 보조스트림의 최고조상
- 보조스트림은 자체적으로 입출력을 수행할 수 없다.

```
protected FilterInputStream(InputStream in)
public FilterOutputStream(OutputStream out)
```

- 상속을 통해 FilterInputStream/FilterOutputStream의 read()와 write()를 원하는 기능대로 오버라이딩해야 한다.

```
public class FilterInputStream extends InputStream {
    protected volatile InputStream in;
    protected FilterInputStream(InputStream in) {
        this.in = in;
    }

    public int read() throws IOException {
        return in.read();
    }
    ...
}
```

FilterInputStream의 자손 - BufferedInputStream, DataInputStream, PushbackInputStream 등
FilterOutputStream의 자손 - BufferedOutputStream, DataOutputStream, PrintStream 등

3.2 BufferedInputStream과 BufferedOutputStream

- 입출력 효율을 높이기 위해 버퍼(byte[])를 사용하는 보조스트림

| 메서드 / 생성자 | 설명 |
|---|---|
| BufferedInputStream(InputStream in, int size) | 주어진 InputStream인스턴스를 입력소스(input source)로하며 지정된 크기(byte 단위)의 버퍼를 갖는 BufferedInputStream인스턴스를 생성한다. |
| BufferedInputStream(InputStream in) | 주어진 InputStream인스턴스를 입력소스(input source)로하며 버퍼의 크기를 지정해주지 않으므로 기본적으로 8192 byte 크기의 버퍼를 갖게 된다. |

| 메서드 / 생성자 | 설명 |
|--|---|
| BufferedOutputStream(OutputStream out, int size) | 주어진 OutputStream인스턴스를 출력소스(output source)로하며 지정된 크기(단위byte)의 버퍼를 갖는 BufferedOutputStream인스턴스를 생성한다. |
| BufferedOutputStream(OutputStream out) | 주어진 OutputStream인스턴스를 출력소스(output source)로하며 버퍼의 크기를 지정해주지 않으므로 기본적으로 8192 byte 크기의 버퍼를 갖게 된다. |
| flush() | 버퍼의 모든 내용을 출력소스에 출력한 다음, 버퍼를 비운다. |
| close() | flush()를 호출해서 버퍼의 모든 내용을 출력소스에 출력하고, BufferedOutputStream인스턴스가 사용하던 모든 자원을 반환한다. |

- 보조스트림을 닫으면 기반스트림도 닫힌다.

```
public class FilterOutputStream extends OutputStream {
    protected OutputStream out;
    public FilterOutputStream(OutputStream out) {
        this.out = out;
    }
    ...
    public void close() throws IOException {
        try { flush(); } catch (IOException ignored) {}
        out.close(); // 기반 스트림의 close()를 호출한다.
    }
}
```

3.2 BufferedInputStream과 BufferedOutputStream

```

import java.io.*;

class BufferedOutputStreamEx1 {
    public static void main(String args[]) {
        try {
            FileOutputStream fos = new FileOutputStream("123.txt");
            // BufferedOutputStream의 버퍼 크기를 5로 한다.
            BufferedOutputStream bos = new BufferedOutputStream(fos, 5);
            // 파일 123.txt에 1부터 9까지 출력한다.
            for(int i='1'; i <= '9'; i++) {
                bos.write(i);
            }
            fos.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

```

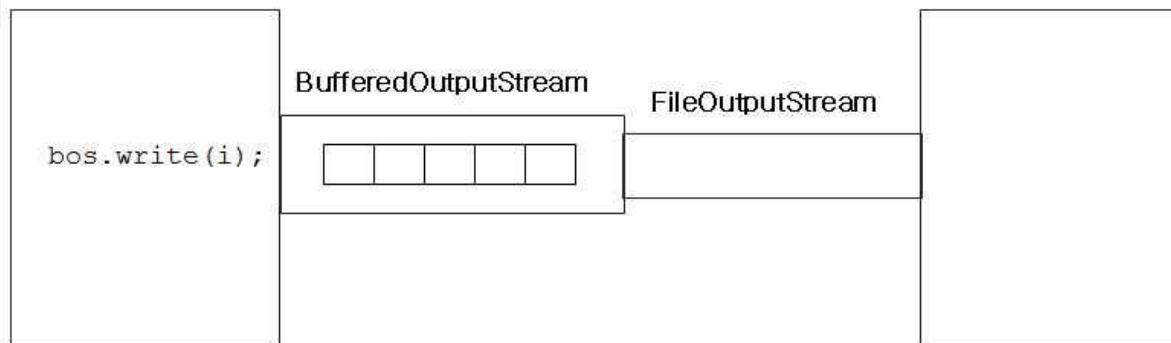
자바 프로그램
BufferedOutputStreamEx1

[실행결과]

C:\jdk1.5\work\ch14>java BufferedOutputStreamEx1

C:\jdk1.5\work\ch14>type 123.txt
12345

123.txt



3.3 DataInputStream과 DataOutputStream

- 기본형 단위로 읽고 쓰는 보조스트림
- 각 자료형의 크기가 다르므로 출력할 때와 입력할 때 순서에 주의

| 메서드 / 생성자 | 설명 | 메서드 / 생성자 | 설명 |
|--|--|--|--|
| DataInputStream(InputStream in) | 주어진 InputStream인스턴스를 기반스트림으로 하는 DataInputStream인스턴스를 생성한다. | DataOutputStream(OutputStream out) | 주어진 OutputStream인스턴스를 기반스트림으로 하는 DataOutputStream인스턴스를 생성한다. |
| boolean readBoolean() byte readByte() char readChar() short readShort() int readInt() long readLong() float readFloat() double readDouble() | 각 자료형에 알맞은 값을 읽어온다. 더 이상 읽을 값이 없으면 EOFException을 발생시킨다. | void writeBoolean(boolean b) void writeByte(int b) void writeChar(int c) void writeShort(int s) void writeInt(int l) void writeLong(long l) void writeFloat(float f) void writeDouble(double d) | 각 자료형에 알맞은 값을 출력한다. |
| String readUTF() | UTF형식으로 쓰여진 문자를 읽는다. 더 이상 읽을 값이 없으면 EOFException을 발생시킨다. | void writeUTF(String s) | UTF형식으로 문자를 출력한다. |
| int skipBytes(int n) | 현재 읽고 있는 위치에서 지정된 숫자(n) 만큼을 건너뛴다. | void writeChars(String s) | 주어진 문자열을 출력한다. writeChar(char c)메서드를 여러 번 호출한 결과와 같다. |
| | | int size() | 지금까지 DataOutputStream에 쓰여진 byte의 수를 알려준다. |

3.4 SequenceInputStream

- 여러 입력스트림을 연결해서 하나의 스트림처럼 다룰 수 있게 해준다.

| 메서드 / 생성자 | 설명 |
|---|--|
| SequenceInputStream(Enumeration e) | Enumeration에 저장된 순서대로 입력스트림을 하나의 스트림으로 연결한다. |
| SequenceInputStream(InputStream s1, InputStream s2) | 두 개의 입력스트림을 하나로 연결한다. |

【사용예1】

```
Vector files = new Vector();
files.add(new FileInputStream("file.001"));
files.add(new FileInputStream("file.002"));
SequenceInputStream in = new SequenceInputStream(files.elements());
```

【사용예2】

```
FileInputStream file1 = new FileInputStream("file.001");
FileInputStream file2 = new FileInputStream("file.002");
SequenceInputStream in = new SequenceInputStream(file1, file2);
```

3.5 PrintStream (1/2)

- 데이터를 다양한 형식의 문자로 출력하는 기능을 제공하는 보조스트림
- System.out과 System.err이 PrintStream이다.
- PrintStream보다 PrintWriter를 사용할 것을 권장한다.

| 생성자 / 메서드 | 설명 | |
|--|---|--|
| PrintStream(File file) PrintStream(File file, String csn) PrintStream(OutputStream out) PrintStream(OutputStream out, boolean autoFlush) PrintStream(OutputStream out, boolean autoFlush, String encoding) PrintStream(String fileName) PrintStream(String fileName, String csn) | 지정된 출력스트림을 기반으로 하는 PrintStream 인스턴스를 생성한다. autoFlush의 값을 true로 하면 println메서드가 호출되거나 개행문자가 출력될 때 자동으로 flush된다. 기본값은 false이다. | |
| boolean checkError() | 스트림을 flush하고 에러가 발생했는지를 알려 준다. | |
| void print(boolean b) void print(char c) void print(char[] c) void print(double d) void print(float f) void print(int i) void print(long l) void print(Object o) void print(String s) | 인자로 주어진 값을 출력소스에 문자로 출력한다. println메서드는 출력 후 줄바꿈을 하고, print메서드는 줄을 바꾸지 않는다. | |
| void println() | 줄바꿈 문자(line separator)를 출력함으로써 줄을 바꾼다. | |
| PrintStream printf(String format, Object... args) | 정형화된(formatted) 출력을 가능하게 한다. | |
| protected void setError() | 작업 중에 오류가 발생했음을 알린다.(setError()를 호출한 후에, checkError()를 호출하면 true를 반환한다.) | |

3.5 PrintStream (2/2)

| format | 설명 | 결과(int i=65) |
|--------|-------------------------------|--------------|
| %d | 10진수(decimal integer) | 65 |
| %o | 8진수(octal integer) | 101 |
| %x | 16진수(hexadecimal integer) | 41 |
| %c | 문자 | A |
| %s | 문자열 | 65 |
| %5d | 5자리 숫자. 빈자리는 공백으로 채운다. | 65 |
| %-5d | 5자리 숫자. 빈자리는 공백으로 채운다.(왼쪽 정렬) | 65 |
| %05d | 5자리 숫자. 빈자리는 0으로 채운다. | 00065 |

| format | 설명 | 결과 |
|--------|--------------------------------|-----|
| %s | 문자열(string) | ABC |
| %5s | 5자리 문자열. 빈자리는 공백으로 채운다. | ABC |
| %-5s | 5자리 문자열. 빈자리는 공백으로 채운다.(왼쪽 정렬) | ABC |

| format | 설명 | 결과 |
|--------|---|--------------|
| %e | 지수형태표현(exponent) | 1.234568e+03 |
| %f | 10진수(decimal float) | 1234.56789 |
| %3.1f | 출력될 자리수를 최소 3자리(소수점포함), 소수점 이하 1자리(2번째 자리에서 반올림) | 1234.6 |
| %8.1f | 소수점이상 최소 6자리, 소수점 이하 1자리. 출력될 자리수를 최소 8자리(소수점포함)를 확보한다. 빈자리는 공백으로 채워진다.(오른쪽 정렬) | 1234.6 |
| %08.1f | 소수점이상 최소 6자리, 소수점 이하 1자리. 출력될 자리수를 최소 8자리(소수점포함)를 확보한다. 빈자리는 0으로 채워진다. | 001234.6 |
| %-8.1f | 소수점이상 최소 6자리, 소수점 이하 1자리. 출력될 자리수를 최소 8자리(소수점포함)를 확보한다. 빈자리는 공백으로 채워진다.(왼쪽 정렬) | 1234.6 |

| format | 설명 |
|--------|------------------|
| \t | 탭(tab) |
| \n | 줄바꿈 문자(new line) |
| % % | % |

| format | 설명 | 결과 |
|--------------------|-----------|--------------------------|
| %tR %tH:%tM | 시분(24시간) | 21:05 21:05 |
| %tT %tH:%tM:%tS | 시분초(24시간) | 21:05:33 21:05:33 |
| %tD %tm/%td/%ty | 연월일 | 02/16/07 02/16/07 |
| %tF %tY-%tm-%td | 연월일 | 2007-02-16 2007-02-16 |

4. 문자기반 스트림

4.1 Reader와 Writer

▶ Reader(문자기반 입력스트림의 최고 조상)의 메서드

| 메서드 | 설명 |
|---|---|
| abstract void close() | 입력스트림을 닫음으로써 사용하고 있던 자원을 반환한다. |
| void mark(int readlimit) | 현재위치를 표시해놓는다. 후에 reset()에 의해서 표시해 놓은 위치로 다시 돌아갈 수 있다. |
| boolean markSupported() | mark()와 reset()을 지원하는지를 알려 준다. |
| int read() | 입력소스로부터 하나의 문자를 읽어 온다. char의 범위인 0~65535범위의 정수를 반환하며, 입력스트림의 마지막 데이터에 도달하면, -1을 반환한다. |
| int read(char[] c); | 입력소스로부터 매개변수로 주어진 배열 c의 크기만큼 읽어서 배열 c에 저장한다. 읽어 온 데이터의 개수 또는 -1을 반환한다. |
| abstract int read(char[] c, int off, int len) | 입력소스로부터 최대 len개의 문자를 읽어서, 배열 c의 지정된 위치(off)부터 읽은 만큼 저장한다. 읽어 온 데이터의 개수 또는 -1을 반환한다. |
| boolean ready() | 입력소스로부터 데이터를 읽을 준비가 되어있는지 알려 준다. |
| void reset() | 입력소스에서의 위치를 마지막으로 mark()가 호출되었던 위치로 되돌린다. |
| long skip(long n) | 현재 위치에서 주어진 문자 수(n)만큼을 건너뛴다. |

▶ Writer(문자기반 출력스트림의 최고 조상)의 메서드

| 메서드 | 설명 |
|---|--|
| abstract void close() | 출력스트림을 닫음으로써 사용하고 있던 자원을 반환한다. |
| abstract void flush() | 스트림의 버퍼에 있는 모든 내용을 출력소스에 쓴다.(버퍼가 있는 스트림에만 해당됨) |
| void write(int b) | 주어진 값을 출력소스에 쓴다. |
| void write(char[] c) | 주어진 배열 c에 저장된 모든 내용을 출력소스에 쓴다. |
| abstract void write(char[] c, int off, int len) | 주어진 배열 c에 저장된 내용 중에서 off번째부터 len길이 만큼만 출력소스에 쓴다. |
| void write(String str) | 주어진 문자열(str)을 출력소스에 쓴다. |
| void write(String str, int off, int len) | 주어진 문자열(str)의 일부를 출력소스에 쓴다.(off번째 문자부터 len개 만큼의 문자열) |

4.2 FileReader와 FileWriter

- 문자기반의 파일 입출력. 텍스트 파일의 입출력에 사용한다.

```
import java.io.*;

class FileReaderEx1 {
    public static void main(String args[]) {
        try {
            String fileName = "test.txt";
            FileInputStream fis = new FileInputStream(fileName);
            FileReader fr = new FileReader(fileName);

            int data =0;
            // FileInputStream을 이용해서 파일내용을 읽어 화면에 출력한다.
            while((data=fis.read())!=-1) {
                System.out.print((char)data);
            }
            System.out.println();
            fis.close();

            // FileReader를 이용해서 파일내용을 읽어 화면에 출력한다.
            while((data=fr.read())!=-1) {
                System.out.print((char)data);
            }
            System.out.println();
            fr.close();

        } catch (IOException e) {
            e.printStackTrace();
        }
    } // main
}
```

[실행결과]

```
C:\jdk1.5\work\ch14>type test.txt
Hello, 안녕하세요?
```

```
C:\jdk1.5\work\ch14>java FileReaderEx1
Hello, ¾?°????½???
Hello, 안녕하세요?
```

4.3 PipedReader와 PipedWriter

- 프로세스(쓰레드)간의 통신(데이터를 주고 받음)에 사용한다.

```

class InputThread extends Thread {
    PipedReader input = new PipedReader();
    StringWriter sw = new StringWriter();

    InputThread(String name) { super(name); }

    public void run() {
        try {
            int data = 0;

            while((data=input.read()) != -1) {
                sw.write(data);
            }
            System.out.println(getName()
                + " received : " + sw.toString());
        } catch(IOException e) {}
    } // run

    public PipedReader getInput() { return input; }

    public void connect(PipedWriter output) {
        try {
            input.connect(output);
        } catch(IOException e) {}
    } // connect
}

```

[실행결과]

```

OutputThread sent : Hello
InputThread received : Hello

```

```

class OutputThread extends Thread {
    PipedWriter output = new PipedWriter();

    OutputThread(String name) { super(name); }

    public void run() {
        try {
            String msg = "Hello";
            System.out.println(getName()
                + " sent : " + msg);
            output.write(msg);
            output.close();
        } catch(IOException e) {}
    } // run

    public PipedWriter getOutput() { return output; }

    public void connect(PipedReader input) {
        try {
            output.connect(input);
        } catch(IOException e) {}
    } // connect
}

```

```

public static void main(String args[]) {
    InputThread inThread = new InputThread("InputThread");
    OutputThread outThread = new OutputThread("OutputThread");
    //PipedReader와 PipedWriter를 연결한다.
    inThread.connect(outThread.getOutput());
    inThread.start(); outThread.start();
} // main

```

4.4 StringReader와 StringWriter

- CharArrayReader, CharArrayWriter처럼 메모리의 입출력에 사용한다.
- StringWriter에 출력되는 데이터는 내부의 StringBuffer에 저장된다.

```
StringBuffer getBuffer() : StringWriter에 출력한 데이터가 저장된 StringBuffer를 반환한다.  

String toString() : StringWriter에 출력된 (StringBuffer에 저장된) 문자열을 반환한다.
```

```
import java.io.*;

class StringReaderWriterEx {
    public static void main(String[] args) {
        String inputData = "ABCD";
        StringReader input = new StringReader(inputData);
        StringWriter output = new StringWriter();

        int data = 0;

        try {
            while((data = input.read()) != -1) {
                output.write(data); // void write(int b)
            }
        } catch(IOException e) {}

        System.out.println("Input Data :" + inputData);
        System.out.println("Output Data :" + output.toString());
        System.out.println("Output Data :" + output.getBuffer().toString());
    }
}
```

【실행결과】

```
Input Data :ABCD
Output Data :ABCD
```

5. 문자기반 보조스트림

5.1 BufferedReader와 BufferedWriter

- 입출력 효율을 높이기 위해 버퍼(char[])를 사용하는 보조스트림
- 라인(line)단위의 입출력이 편리하다.

String readLine() - 한 라인을 읽어온다. (**BufferedReader**의 메서드)

void newLine() - '라인 구분자(개행문자)'를 출력한다. (**BufferedWriter**의 메서드)

```
import java.io.*;

class BufferedReaderEx1 {
    public static void main(String[] args) {
        try {
            FileReader fr = new FileReader("BufferedReaderEx1.java");
            BufferedReader br = new BufferedReader(fr);

            String line = "";
            for(int i=1;(line = br.readLine())!=null;i++) {
                // ";"를 포함한 라인을 출력한다.
                if(line.indexOf(";")!=-1)
                    System.out.println(i+":"+line);
            }
            br.close();
        } catch(IOException e) {}
    } // main
}
```

[실행결과]

```
1: import java.io.*;
6:             FileReader fr = new FileReader("BufferedReaderEx1.java");
7:             BufferedReader br = new BufferedReader(fr);
9:             String line = "";
10:            for(int i=1;(line = br.readLine())!=null;i++) {
11:                // ";"를 포함한 라인을 출력한다.
12:                if(line.indexOf(";")!=-1)
13:                    System.out.println(i+":"+line);
```

5.2 InputStreamReader와 OutputStreamWriter

- 바이트기반스트림을 문자기반스트림처럼 쓸 수 있게 해준다.
- 인코딩(encoding)을 변환하여 입출력할 수 있게 해준다.

| 생성자 / 메서드 | 설명 |
|--|---|
| InputStreamReader(InputStream in) | OS에서 사용하는 기본 인코딩의 문자로 변환하는 InputStreamReader를 생성한다. |
| InputStreamReader(InputStream in, String encoding) | 지정된 인코딩을 사용하는 InputStreamReader를 생성한다. |
| String getEncoding() | InputStreamReader의 인코딩을 알려 준다. |

| 생성자 / 메서드 | 설명 |
|--|--|
| OutputStreamWriter(OutputStream in) | OS에서 사용하는 기본 인코딩의 문자로 변환하는 OutputStreamWriter를 생성한다. |
| OutputStreamWriter(OutputStream in, String encoding) | 지정된 인코딩을 사용하는 OutputStreamWriter를 생성한다. |
| String getEncoding() | OutputStreamWriter의 인코딩을 알려 준다. |

- 콘솔(console, 화면)로부터 라인단위로 입력받기

```
InputStreamReader isr = new InputStreamReader(System.in);
BufferedReader br = new BufferedReader(isr);
String line = br.readLine();
```

- 인코딩 변환하기

```
Properties prop = System.getProperties();
System.out.println(prop.get("sun.jnu.encoding"));
```

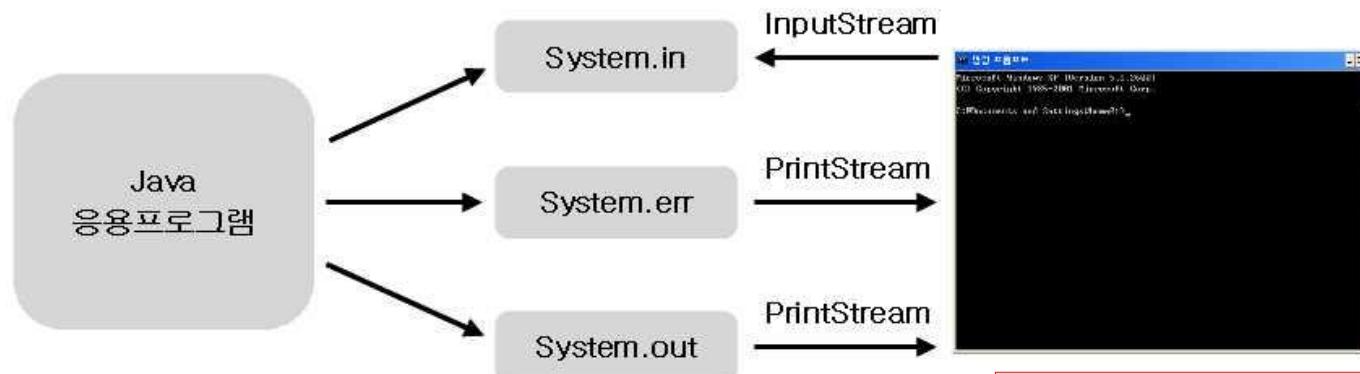
```
FileInputStream fis = new FileInputStream("korean.txt");
InputStreamReader isr = new InputStreamReader(fis, "KSC5601");
```

6. 표준입출력과 File

6.1 표준입출력 – System.in, System.out, System.err

- 콘솔(console, 화면)을 통한 데이터의 입출력을 ‘표준 입출력’이라 한다.
- JVM이 시작되면서 자동적으로 생성되는 스트림이다.

```
System.in - 콘솔로부터 데이터를 입력받는데 사용  
System.out - 콘솔로 데이터를 출력하는데 사용  
System.err - 콘솔로 데이터를 출력하는데 사용
```



```
public final class System {  
    public final static InputStream in = nullInputStream();  
    public final static PrintStream out = nullPrintStream();  
    public final static PrintStream err = nullPrintStream();  
    ...  
}
```

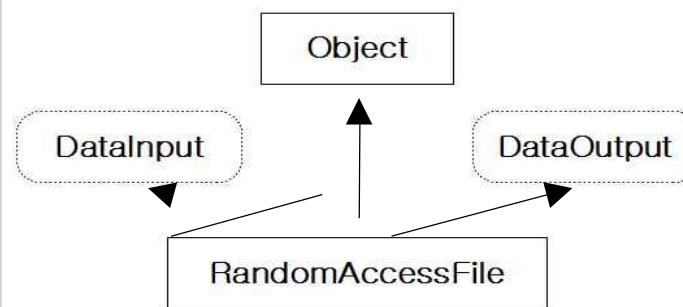
```
static void setOut(PrintStream out)  
static void setErr(PrintStream err)  
(InputStream in)
```

6.2 RandomAccessFile

- 하나의 스트림으로 파일에 입력과 출력을 모두 수행할 수 있는 스트림
- 다른 스트림들과 달리 Object의 자손이다.

* **DataInput** 인터페이스의 메서드

```
boolean readBoolean()
byte    readByte()
int     readInt()
void   readFully(byte[] b)
String  readLine()
...
...
```



* **DataOutput** 인터페이스의 메서드

```
void write(byte[] b)
void write(int b)
void writeBoolean(boolean b)
void writeInt(int v)
void writeBytes(String s)
...
...
```

| 생성자 / 메서드 | 설명 |
|--|--|
| RandomAccessFile(File file, String mode) RandomAccessFile(String fileName, String mode) | 주어진 file에 읽기 또는 읽기와 쓰기를 하기 위한 RandomAccessFile 인스턴스를 생성한다. mode에는 "r"과 "rw" 두 가지 값이 지정 가능하다. "r" - 파일로부터 읽기(r)만을 수행할 때 "rw" - 파일에 읽기(r)와 쓰기(w) |
| long getFilePointer() | 파일 포인터의 위치를 알려 준다. |
| long length() | 파일의 크기를 얻을 수 있다.(단위 byte) |
| void seek(long pos) | 파일 포인터의 위치를 변경한다. 위치는 파일의 첫 부분부터 pos크기만큼 떨어진 곳이다.(단위 byte) |
| void setLength(long newLength) | 파일의 크기를 지정된 길이로 변경한다.(byte 단위) |
| int skipBytes(int n) | 지정된 수만큼의 byte를 건너뛴다. |

6.3 File (1/2) – 생성자와 경로관련 메서드

- 파일과 디렉토리를 다루는데 사용되는 클래스

| 생성자 / 메서드 | 설명 |
|--|--|
| File(String fileName) | 주어진 문자열(fileName)을 이름으로 갖는 파일을 위한 File인스턴스를 생성한다. 파일 뿐만 아니라 디렉토리도 같은 방법으로 다룬다. 여기서 fileName은 주로 경로(path)를 포함해서 지정해주지만, 파일 이름만 사용해도 되는 데 이 경우 프로그램이 실행되는 위치가 경로(path)로 간주된다. |
| File(String pathName, String fileName) File(File pathName, String fileName) | 파일의 경로와 이름을 따로 분리해서 지정할 수 있도록 한 생성자. 이 중 두 번째 것은 경로를 문자열이 아닌 File인스턴스인 경우를 위해서 제공된 것이다. |
| String getName() | 파일이름을 String으로 반환한다. |
| String getPath() | 파일의 경로(path)를 String으로 반환한다. |
| String getAbsolutePath() File getAbsoluteFile() | 파일의 절대경로를 String으로 반환한다. 파일의 절대경로를 File로 반환한다. |
| String getParent() File getParentFile() | 파일의 조상 디렉토리를 String으로 반환한다. 파일의 조상 디렉토리를 File로 반환한다. |
| String getCanonicalPath() File getCanonicalFile() | 파일의 정규경로를 String으로 반환한다. 파일의 정규경로를 File로 반환한다. |

| 멤버변수 | 설명 |
|-------------------------------|--|
| static String pathSeparator | OS에서 사용하는 경로(path) 구분자. 윈도우 ";", 유닉스 ":" |
| static char pathSeparatorChar | OS에서 사용하는 경로(path) 구분자. 윈도우에서는 ';', 유닉스 ':' |
| static String separator | OS에서 사용하는 이름 구분자. 윈도우 "₩", 유닉스 "/" |
| static char separatorChar | OS에서 사용하는 이름 구분자. 윈도우 '₩', 유닉스 '/' |

6.3 File (1/2) – 생성자와 경로관련 메서드(예제)

```
File f = new File("c:\\jdk1.5\\work\\ch14\\FileEx1.java");
String fileName = f.getName();
int pos = fileName.lastIndexOf(".") ;
```

경로를 제외한 파일이름 - FileEx1.java
 확장자를 제외한 파일이름 - FileEx1
 확장자 - java

```
System.out.println("경로를 제외한 파일이름 - " + f.getName());
System.out.println("확장자를 제외한 파일이름 - " + fileName.substring(0,pos));
System.out.println("확장자 - " + fileName.substring(pos+1));
```

```
System.out.println("경로를 포함한 파일이름 - " + f.getPath());
System.out.println("파일의 절대경로 - " + f.getAbsolutePath());
System.out.println("파일이 속해 있는 디렉토리 - " + f.getParent());
```

경로를 포함한 파일이름 - c:\\jdk1.5\\work\\ch14\\FileEx1.java
 파일의 절대경로 - c:\\jdk1.5\\work\\ch14\\FileEx1.java
 파일의 정규경로 - C:\\jdk1.5\\work\\ch14\\FileEx1.java
 파일이 속해 있는 디렉토리 - c:\\jdk1.5\\work\\ch14

```
File.separator - ;
File.separatorChar - ;
File.separator - \
File.separatorChar - \
        ("File.separator - " + File.separator);
System.out.println("File.separatorChar - " + File.separatorChar);
System.out.println("File.separator - " + File.separator);
System.out.println("File.separatorChar - " + File.separatorChar);
```

```
System.out.println("user.dir="+System.getProperty("user.dir"));
System.out.println("sun.boot.class.path="+ System.getProperty("sun.boot.class.path"));
```

user.dir=C:\\jdk1.5\\work\\ch14
 sun.boot.class.path=C:\\jdk1.5\\jre\\lib\\rt.jar;C:\\jdk1.5\\jre\\lib\\i18n.jar;C:\\jdk1.5\\jre\\lib\\sunrsasign.jar;C:\\jdk1.5\\jre\\lib\\jsse.jar;C:\\jdk1.5\\jre\\lib\\jce.jar;C:\\jdk1.5\\jre\\lib\\charsets.jar;C:\\jdk1.5\\jre\\classes

6.3 File (2/2) – 파일의 속성, 생성, 삭제, 목록

| 메서드 | 설명 |
|--|---|
| boolean canRead() | 읽을 수 있는 파일인지 검사한다. |
| boolean canWrite() | 쓸 수 있는 파일인지 검사한다. |
| boolean exists() | 파일이 존재하는지 검사한다. |
| boolean isAbsolute() | 파일 또는 디렉토리가 절대경로명으로 지정되었는지 확인한다. |
| boolean isDirectory() | 디렉토리인지 확인한다. |
| boolean isFile() | 파일인지 확인한다. |
| boolean isHidden() | 파일의 속성이 '숨김(Hidden)'인지 확인한다. 또한 파일이 존재하지 않으면 false를 반환한다. |
| int compareTo(File pathname) | 주어진 파일 또는 디렉토리를 비교한다. 같으면 0을 반환하며, 다르면 1 또는 -1을 반환한다. (Unix시스템에서는 대소문자를 구별하며, Windows에서는 구별하지 않는다.) |
| boolean createNewFile() | 아무런 내용이 없는 새로운 파일을 생성한다.(파일이 이미 존재하면 생성되지 않는다.) File f = new File("c:\jdk1.5\work\test3.java"); f.createNewFile(); |
| static File createTempFile(String prefix, String suffix) | 임시파일을 시스템의 임시 디렉토리에 생성한다. System.out.println(File.createTempFile("work", ".tmp")); 결과 : c:\windows\TEMP\work14247.tmp |
| static File createTempFile(String prefix, String suffix, File directory) | 임시파일을 시스템의 지정된 디렉토리에 생성한다. |
| boolean delete() | 파일을 삭제한다. |
| void deleteOnExit() | 응용 프로그램 종료시 파일을 삭제한다. 주로 임시파일을 삭제하는데 사용된다. |
| boolean equals(Object obj) | 주어진 객체(주로 File인스턴스)가 같은 파일인지 비교한다. (Unix시스템에서는 대소문자를 구별하며, Windows에서는 구별하지 않는다.) |
| long length() | 파일의 크기를 반환한다. |
| String [] list() | 디렉토리의 파일목록(디렉토리 포함)을 String 배열로 반환한다. |
| String [] list(FilenameFilter filter) | FilenameFilter인스턴스에 구현된 조건에 맞는 파일을 String 배열로 반환한다. |
| File [] listFiles() | 디렉토리의 파일목록(디렉토리 포함)을 File 배열로 반환한다. |
| static File [] listRoots() | 컴퓨터의 파일시스템의 root의 목록(floppy, CD-ROM, HDD drive)을 반환한다. (예: A:\, C:\, D:\) |

6.3 File (2/2) – 파일의 속성, 생성, 삭제, 목록(예제1)

```

import java.io.*;

class FileEx2 {
    public static void main(String[] args)
    {
        if(args.length != 1) {
            System.out.println("USAGE : java FileEx2 DIRECTORY");
            System.exit(0);
        }

        File f = new File(args[0]);

        if(!f.exists() || !f.isDirectory()) {
            System.out.println("유효하지 않은 디렉토리입니다.");
            System.exit(0);
        }

        File[] files = f.listFiles();

        for(int i=0; i < files.length; i++) {
            String fileName = files[i].getName();
            System.out.println(
                files[i].isDirectory() ? "["+fileName+"]" : fileName);
        }
    } // main
}

```

【실행결과】

```

C:\jdk1.5\work\ch14>java FileEx2
USAGE : java FileEx2 DIRECTORY

C:\jdk1.5\work\ch14>java FileEx2 work
유효하지 않은 디렉토리입니다.

C:\jdk1.5\work\ch14>java FileEx2 c:\jdk1.5
[bin]
COPYRIGHT
[demo]
[docs]
[include]
jdk-1_5_0-doc.zip
[jre]
[lib]
LICENSE
... 중간생략 ...

```

C:\jdk1.5\work\ch14>

6.3 File (2/2) – 파일의 속성, 생성, 삭제, 목록(예제2)

```

public static void printFileList(File dir) {
    System.out.println(dir.getAbsolutePath()+" 디렉토리");
    File[] files = dir.listFiles();
    ArrayList subDir = new ArrayList();
    for(int i=0; i < files.length; i++) {
        String filename = files[i].getName();
        if(files[i].isDirectory()) {
            filename = "[" + filename + "]";
            subDir.add(i+"");
        }
        System.out.println(filename);
    }
    int dirNum = subDir.size();
    int fileNum = files.length - dirNum;
    totalFiles += fileNum;
    totalDirs += dirNum;

    System.out.println(fileNum + "개의 파일, " + dirNum + "개의 디렉토리");
    System.out.println();

    for(int i=0; i < subDir.size(); i++) {
        int index = Integer.parseInt((String)subDir.get(i));
        printFileList(files[index]);
    }
} // printFileList

```

```

C:\jdk1.5\work\ch14>java FileEx3 c:\jdk1.5\work\ch14
c:\jdk1.5\work\ch14 디렉토리
FileEx1.java
FileEx2.class
FileEx2.java
...
VectorEx2.java
20개의 파일, 2개의 디렉토리

c:\jdk1.5\work\ch14\temp 디렉토리
FileEx9.class
FileEx9.java
FileEx9.java.bak
result.txt
[temptemp]
[temptemp2]
4개의 파일, 2개의 디렉토리

```

6.3 File (2/2) – 파일의 속성, 생성, 삭제, 목록(예제3)

```

public static void main(String[] args) {
    String currDir = System.getProperty("user.dir");
    File dir = new File(currDir);

    File[] files = dir.listFiles();

    for(int i=0; i < files.length; i++) {
        File f = files[i];
        String name = f.getName();
        SimpleDateFormat df =new SimpleDateFormat ("yyyy-MM-dd HH:mma");
        String attribute = "";
        String size = "";

        if(files[i].isDirectory()) {
            attribute = "DIR";
        } else {
            size = f.length() + "";
            attribute = f.canRead() ? "R" : " ";
            attribute += f.canWrite() ? "W" : " ";
            attribute += f.isHidden() ? "H" : " ";
        }

        System.out.printf("%s %3s %6s %s\n", df.format(new Date(f.lastModified()))
                         , attribute, size, name );
    }
}

```

【실행결과】

```

C:\jdk1.5\work\ch14>java FileEx4
2006-11-20 16:39오후 RW      1484 FileEx4.class
2006-11-20 16:39오후 RW      2171 FileEx4.java
2006-11-20 16:38오후 RW      2170 FileEx4.java.bak
...
2006-07-27 17:10오후 DIR      Temp
C:\jdk1.5\work\ch14>

```

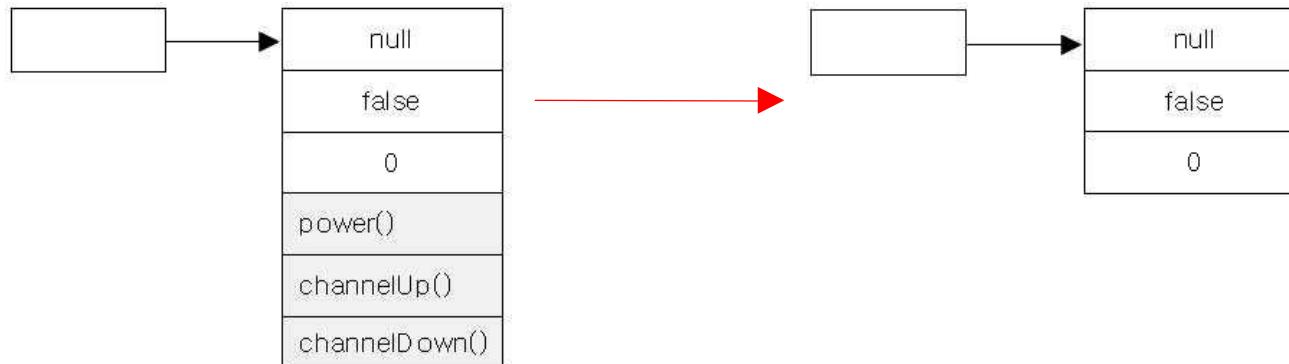
7. 직렬화(Serialization)

7.1 직렬화(serialization)란?

- 객체를 ‘연속적인 데이터’로 변환하는 것. 반대과정은 ‘역직렬화’라고 한다.
- 객체의 인스턴스변수들의 값을 일렬로 나열하는 것



- 객체를 저장하기 위해서는 객체를 직렬화해야 한다.
- 객체를 저장한다는 것은 객체의 모든 인스턴스변수의 값을 저장하는 것



7.2 ObjectInputStream, ObjectOutputStream

- 객체를 직렬화하여 입출력할 수 있게 해주는 보조스트림

```
ObjectInputStream(InputStream in)
ObjectOutputStream(OutputStream out)
```

- 객체를 파일에 저장하는 방법

```
FileOutputStream fos = new FileOutputStream("objectfile.ser");
ObjectOutputStream out = new ObjectOutputStream(fos);

out.writeObject(new UserInfo());
```

- 파일에 저장된 객체를 다시 읽어오는 방법

```
FileInputStream fis = new FileInputStream("objectfile.ser");
ObjectInputStream in= new ObjectInputStream(fis);

UserInfo info = (UserInfo)in.readObject();
```

| ObjectOutputStream |
|--|
| void defaultWriteObject() |
| void write(byte[] buf) |
| void write(byte[] buf, int off, int len) |
| void write(int val) |
| void writeBoolean(boolean val) |
| void writeByte(int val) |
| void writeBytes(String str) |
| void writeChar(int val) |
| void writeChars(String str) |
| void writeDouble(double val) |
| void writeFloat(float val) |
| void writeInt(int val) |
| void writeLong(long val) |
| void writeObject(Object obj) |
| void writeShort(int val) |
| void writeUTF(String str) |

| ObjectInputStream |
|--|
| readObject() |
| readObject(byte[] buf, int off, int len) |
| readBoolean() |
| readByte() |
| readBytes() |
| readChar() |
| readChars() |
| readDouble() |
| readFloat() |
| readInt() |
| readLong() |
| readShort() |
| readObject(Object obj) |
| readUTF() |

7.3 직렬화 가능한 클래스 만들기(1/2)

- java.io.Serializable을 구현해야만 직렬화가 가능하다.

```
public class UserInfo {
    String name;
    String password;
    int age;
}
```

```
public class UserInfo
    implements java.io.Serializable {
    String name;
    String password;
    int age;
}
```

```
public interface Serializable { }
```

- 제어자 transient가 붙은 인스턴스변수는 직렬화 대상에서 제외된다.

```
public class UserInfo implements Serializable {
    String name;
    transient String password; // 직렬화 대상에서 제외된다.
    int age;
}
```

- Serializable을 구현하지 않은 클래스의 인스턴스도 직렬화 대상에서 제외

```
public class UserInfo implements Serializable {
    String name;
    transient String password;
    int age;

    Object obj = new Object(); // Object 객체는 직렬화할 수 없다.
}
```

7.3 직렬화 가능한 클래스 만들기(2/2)

- Serializable을 구현하지 않은 조상의 멤버들은 직렬화 대상에서 제외된다.

```
public class SuperUserInfo {
    String name;      // 직렬화되지 않는다.
    String password; // 직렬화되지 않는다.
}

public class UserInfo extends SuperUserInfo implements Serializable {
    int age;
}
```

- readObject()와 writeObject()를 오버라이딩하면 직렬화를 마음대로...

```
private void writeObject(ObjectOutputStream out)
    throws IOException {
    out.writeUTF(name);
    out.writeUTF(password);
    out.defaultWriteObject();
}

private void readObject(ObjectInputStream in)
    throws IOException, ClassNotFoundException {
    name = in.readUTF();
    password = in.readUTF();
    in.defaultReadObject();
}
```

7.4 직렬화 가능한 클래스의 버전관리

- 직렬화했을 때와 역직렬화했을 때의 클래스가 같은지 확인할 필요가 있다.

```
java.io.InvalidClassException: UserInfo; local class incompatible: stream  
classdesc    serialVersionUID      =      6953673583338942489,      local      class  
serialVersionUID = -6256164443556992367  
...
```

- 직렬화할 때, 클래스의 버전(serialVersionUID)을 자동계산해서 저장한다.
- 클래스의 버전을 수동으로 관리하려면, 클래스 내에 정의해야 한다.

```
class MyData implements java.io.Serializable {  
    static final long serialVersionUID = 3518731767529258119L;  
    int value1;  
}
```

- serialver.exe는 클래스의 serialVersionUID를 자동생성해준다.

```
C:\jdk1.5\work\ch14>serialver MyData  
MyData:    static final long serialVersionUID = 3518731767529258119L;
```