*Power Java*

# 제24장 입출력



## 이번 장에서 학습할 내용

- 스트림이란?
- 스트림의 분류
- 바이트 스트림
- 문자 스트림
- 형식 입출력
- 명령어행에서 입출력
- 파일 입출력

# 스트림(stream)

- 스트림(stream)은 "순서가 있는 데이터의 연속적인 흐름"이다.
- 스트림은 입출력을 물의 흐름처럼 간주하는 것이다.

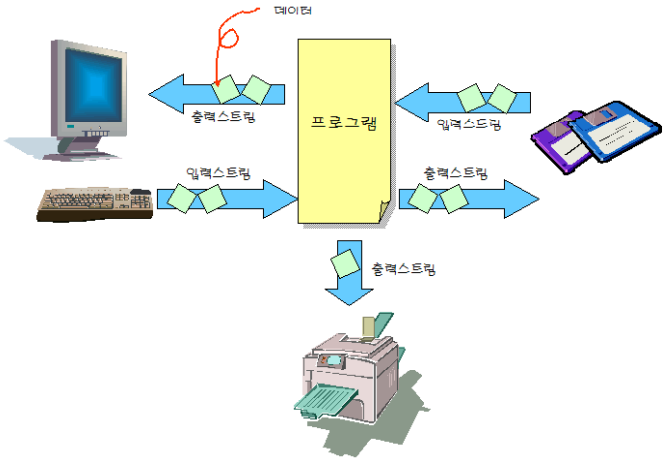



그림 24-1 스트림의 개념



# 스트림들은 연결될 수 있다.

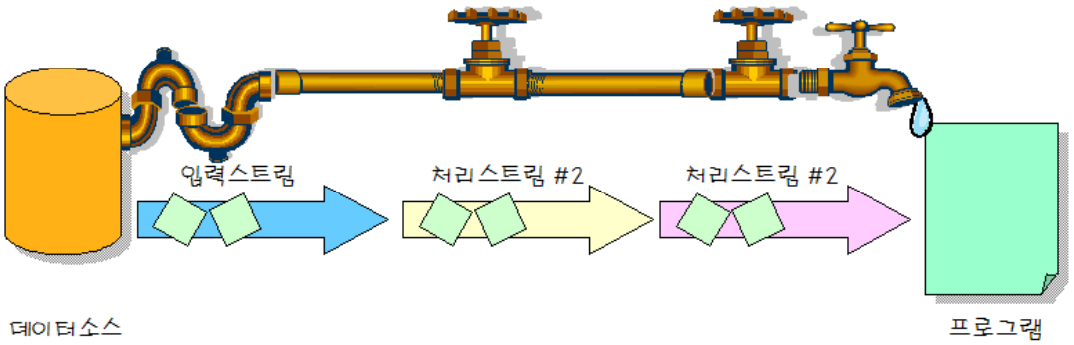



그림 24-2 스트림은 연결될 수 있다.

# 중간 점검 문제

1. 자바에서는 입출력을 무엇이라고 추상화하는가?

2. 스트림은 _____가 있는 데이터의 _____ 인 흐름이다.



# 스트림의 분류 #1

- 입출력의 단위에 따라서 분류

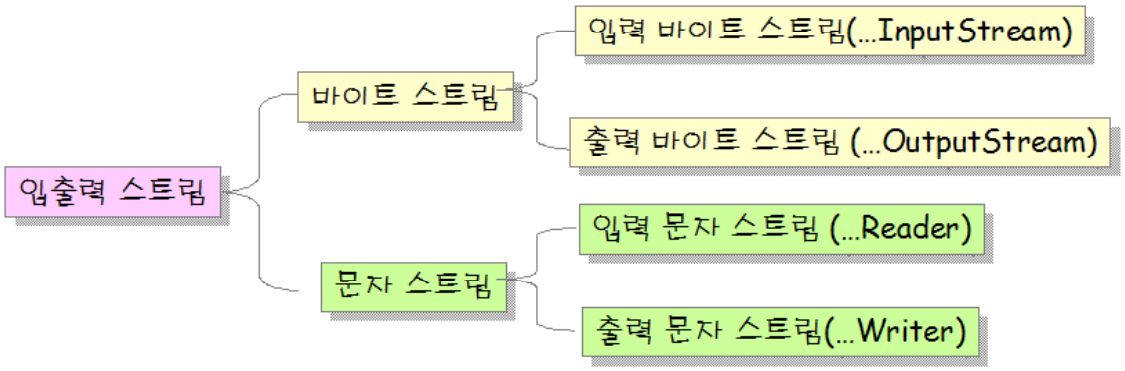



그림 24-3 스트림의 분류

# 스트림의 분류 #2

- 데이터 씽크 클래스(Data Sink Class)와 데이터 처리 클래스(Data Processing Class)로 분류할 수도 있다.

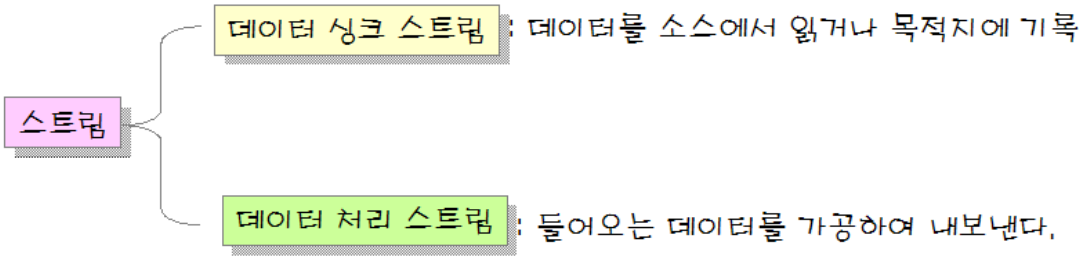



그림 24-4 스트림의 또 다른 분류



# 데이터 씽크 스트림

| 소스/목적지 | 문자 스트림 | 바이트 스트림 |
|---|---|---|
| 메모리 | CharArrayReader<br>CharArrayWriter | ByteArrayInputStream<br>ByteArrayOutputStream |
| | StringReader<br>StringWriter | StringBufferInputStream |
| 파이프 | PipedReader<br>PipedWriter | PipedInputStream<br>PipedOutputStream |
| 파일 | FileReader<br>FileWriter | FileInputStream<br>FileOutputStream |

# 데이터 처리 스트림

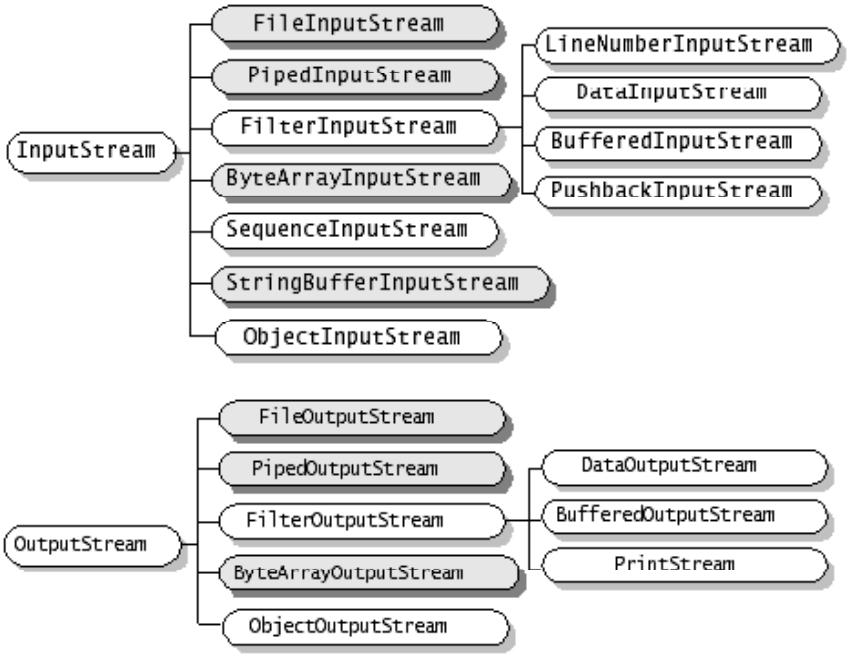
| 처리의 내용 | 문자 스트림 | 바이트 스트림 | 설명 |
|---|---|---|---|
| 버퍼링 | BufferedReader<br>BufferedWriter | BufferedInputStream<br>BufferedOutputStream | 효율성을 위하여 입출력시에 버퍼링을 한다. |
| 필터링 | FilterReader<br>FilterWriter | FilterInputStream<br>FilterOuputStream | 필터링을 위한 추상 클래스들 |
| 문자와 바이트 변환 | InputStreamReader<br>InputStreamWriter | | 문자 스트림과 바이트 스트림간의 연결 역할을 한다. |
| 결합 | | SequenceInputStream | 여러 개의 입력 스트림을 하나의 입력 스트림으로 결합 |
| 객체 직렬화 | | ObjectInputStream<br>ObjectOutputStream | 객체를 직렬화하는데 사용 |
| 데이터 변환 | | DataInputStream<br>DataOutputStream | 기초자료형 데이터를 읽거나 쓴다. |
| 문장 번호 세기 | LineNumberReader | LineNumberInputStream | 입력시 줄수를 센다. |
| 입력 되돌림 | PushbackReader | PushbackInputStream | 한 글자나 한 바이트의 되돌림 버퍼를 갖는다. |
| 출력 | PrintWriter | PrintStream | 사용이 편리한 출력 메소드 제공 |



# 중간 점검 문제

1. 문자 스트림과 바이트 스트림의 차이점은 무엇인가?
2. 데이터 씽크 스트림과 데이터 처리 스트림의 차이점은 무엇인가?
3. LineNumberReader는 데이터 씽크 스트림인가, 아니면 데이터 처리 스트림인가?

# 바이트 스트림






# InputStream과 OutputStream

- 추상 클래스로서 모든 바이트 스트림의 조상 클래스

| 메소드 | 설명 |
|---|---|
| int read() | 한 바이트를 읽어서 int 타입으로 반환, 읽을 값이 없으면 -1을 반환 |
| int read(byte[] buf) | buf의 크기만큼 데이터를 읽어서 buf에 저장하고 읽은 바이트 수를 반환 |
| int read(byte[] buf, int offset, int length) | length만큼의 데이터를 읽어서 buf의 offset 위치에 저장하고 읽은 바이트 수를 반환 |
| void write(int data) | data의 하위 8비트를 출력 |
| void write(byte[] buf) | buf[]에 저장된 바이트들을 출력 |
| void write(byte[] buf, int offset, int length) | buf의 offset 위치에 length만큼의 바이트를 출력 |

# FileInputStream과 FileOutputStream

- 파일이 입출력 대상이 된다.


그림 24-7 파일 스트림



# 예제

```
import java.io.*;

public class FileStreamTest {
    public static void main(String[] args) throws IOException {
        FileInputStream in = null;
        FileOutputStream out = null;
        try {
         int c;
            out = new FileOutputStream("data.txt", false);
            for(int i=0;i <10; i++) {
                out.write(i);
            }
            in = new FileInputStream("data.txt");
            while ((c = in.read()) != -1) {
                System.out.print(c+" ");
            }
```

# 예제

```
    } finally {
        if (in != null) {
            in.close();
        }
        if (out != null) {
            out.close();
        }
      }
   }
}
```

```
0 1 2 3 4 5 6 7 8 9
```



# BufferedInputStream과 BufferedOutputStream

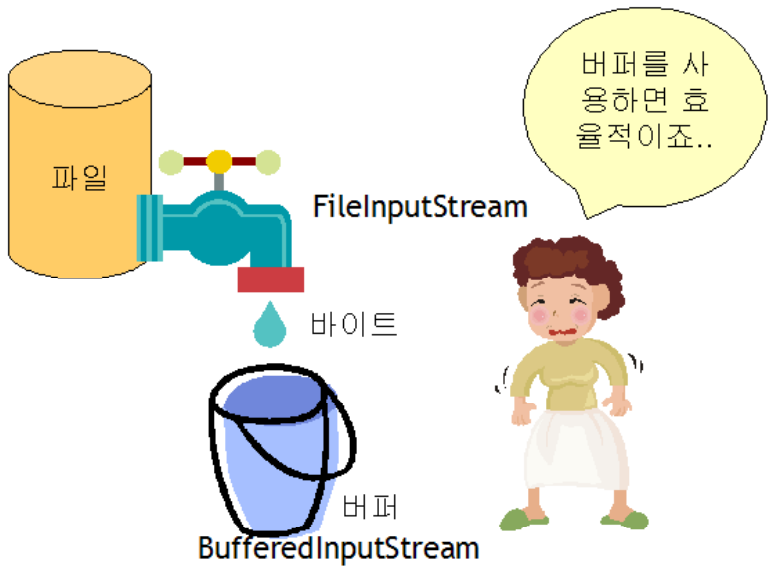



그림 24-8 버퍼 스트림의 개념

# 예제

```
import java.io.*;
public class BufferedStreamTest {
	public static void main(String[] args) throws IOException {
		BufferedInputStream in = null;
		BufferedOutputStream out = null;
		try {
			int c;
			out = new BufferedOutputStream(new FileOutputStream("data.txt"));
			for (int i = 0; i < 10; i++) {
				out.write(i);
			}
			out.flush(); // 버퍼의 내용을 파일에 쓴다.
			in = new BufferedInputStream(new FileInputStream("data.txt"));
			while ((c = in.read()) != -1) {
				System.out.print(c + " ");
			}
```



# 예제

```
		} finally {
			if (in != null) {
				in.close();
			}
			if (out != null) {
				out.close();
			}
		}
	}
}
```

```
0 1 2 3 4 5 6 7 8 9
```

# DataInputStream 과 DataOutputStream

- DataInputStream 과 DataOutputStream 클래스는 기초 자료형 단위로 데이터를 읽고 쓸 수 있다.

| 생성자 또는 메소드 | 설명 |
|---|---|
| DataInputStream(InputStream in) | 주어진 InputStream과 연결된 객체를 생성한다. |
| booelan readBoolean(boolean b) | 스트림으로 booelan 타입을 읽어서 반환한다. |
| byte readByte() | 해당되는 자료형을 읽어서 반환한다. |
| char readChar() | |
| double readDouble() | |
| float readFloat() | |
| int readInt() | |
| long readLong() | |
| short readShort() | |
| int readUnsignedByte() | |
| int readUnsignedShort() | |
| String readUTF() | UTF-8 형식으로 코딩된 문자열을 읽는다. |
| void readFully(byte[] b, int off, int len) | 입력 스트림에서 len 바이트를 읽어서 b[]의 off 위치에 저장한다. |
| void readFully(byte[] b) | 입력 스트림에서 바이트를 읽어서 b[]에 저장한다. |
| int skipBytes(int n) | 입력 스트림에서 n 바이트를 건너뛴다. |



# 예제

```
import java.io.*;
public class DataStreamTest {
    public static void main(String[] args) throws IOException {
        DataInputStream in = null;
        DataOutputStream out = null;
        try {
            int c;
            out = new DataOutputStream(new BufferedOutputStream(
                    new FileOutputStream("data.bin")));
            out.writeDouble(3.14);
            out.writeInt(100);
            out.writeUTF("자신의 생각을 바꾸지 못하는 사람은 결코 현실을 바꿀 수 없다.");
            out.flush();
            in = new DataInputStream(new BufferedInputStream(
                    new FileInputStream("data.bin")));
```

# 예제

```
            System.out.println(in.readDouble());
                System.out.println(in.readInt());
                System.out.println(in.readUTF());

            } finally {
                if (in != null) {
                    in.close();
                }
                if (out != null) {
                    out.close();
                }
            }
        }
}
```

```
3.14
100
자신의 생각을 바꾸지 못하는 사람은 결코 현실을 바꿀 수 없다.
```



# ObjectInputStream과 ObjectOutputStream

- 직렬화(serialization):
  - 객체가 가진 데이터들을 순차적인 데이터로 변환하는 것

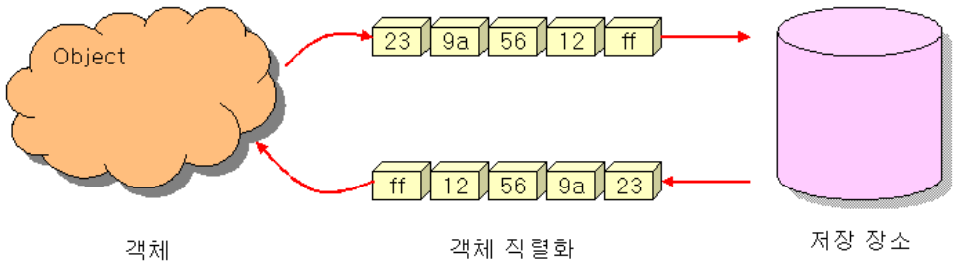



그림 24-9 객체 직렬화의 개념

# 예제

```
import java.io.*;
import java.util.Date;
public class ObjectStreamTest {
      public static void main(String[] args) throws IOException {
            ObjectInputStream in = null;
            ObjectOutputStream out = null;
            try {
                  int c;
                  out = new ObjectOutputStream(new FileOutputStream("object.dat"));
                  out.writeObject(new Date());

                  out.flush();
                  in = new ObjectInputStream(new FileInputStream("object.dat"));
                  Date d = (Date) in.readObject();
                  System.out.println(d);
```

객체를 직렬화하여서 쓴다.



# 예제

```
      } catch (ClassNotFoundException e) {

      } finally {
                  if (in != null) {
                        in.close();
                  }
                  if (out != null) {
                        out.close();
                  }
            }
      }
}
```

```
Fri May 01 15:46:56 KST 2009
```

# 중간 점검 문제

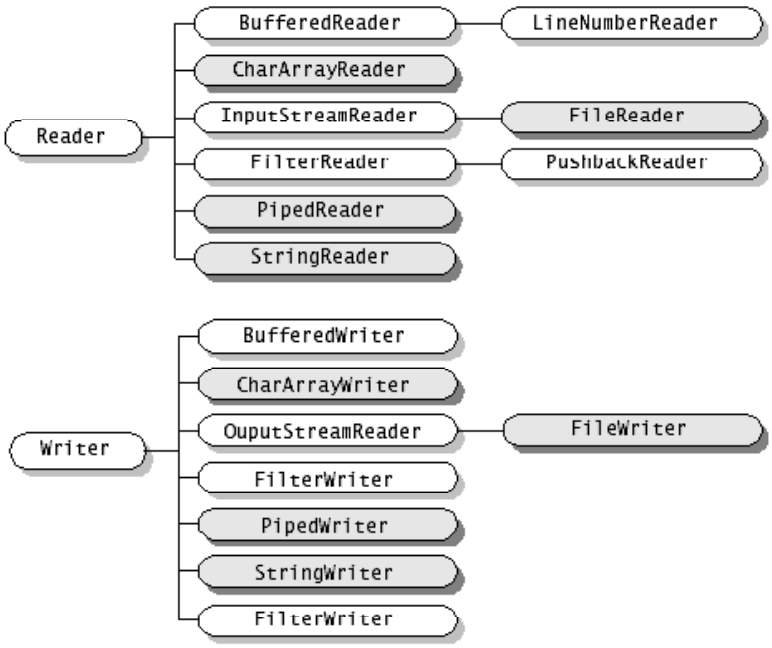
1. 파일 data.bin에서 바이트 형태로 버퍼를 사용하여서 데이터를 읽는 스트림을 생성하여 보라.

2. 객체를 네트워크를 통하여 보냈다가 다시 받으려면 어떤 클래스들을 이용하여야 하는가?

3. double형의 데이터를 저장하였다가 다시 읽으려면 어떤 스트림 클래스가 적합한가?



# 문자 스트림



# Reader와 Writer 클래스

- 추상 클래스로서 모든 문자 스트림의 조상 클래스

| 메소드 | 설명 |
|---|---|
| int read() | 한 바이트를 읽어서 int 타입으로 반환, 읽을 값이 없으면 -1을 반환 |
| int read(byte[] buf) | buf의 크기만큼 데이터를 읽어서 buf에 저장하고 읽은 바이트 수를 반환 |
| int read(byte[] buf, int offset, int length) | length만큼의 데이터를 읽어서 buf의 offset 위치에 저장하고 읽은 바이트 수를 반환 |
| void mark(int readAheadLimit) | 스트림의 현재 위치를 표시해 놓는다. |
| boolean markSupported() | 마크 기능이 지원되는지 여부를 반환한다. |
| abstract void close() | 스트림을 닫고 모든 자원을 반납한다. |
| boolean ready() | 스트림이 읽을 준비가 되었는 지 여부를 반환한다. |
| void reset() | 스트림을 리셋한다. |
| long skip(long n) | n개의 문자를 건너뛴다. |



# FileReader와 FileWriter

- 입출력의 대상이 파일이고 단위는 문자


그림 24-11 문자 스크림의 개념

# 예 제

```
import java.io.*;

public class FileReaderTest {
	public static void main(String[] args) throws IOException {
		FileReader in = null;
		FileWriter out = null;
		String s = "꿈에 미치면 신화가 된다";

		out = new FileWriter("test.txt");
		out.write(s);		// 문자열은 write()로 출력 가능
		out.append('.');	// 문자 추가
		out.flush();
```



# 예 제

```
		in = new FileReader("test.txt");
		int c;
		while ((c = in.read()) != -1) {
			System.out.print((char) c);
		}

		if(in != null) in.close();
		if(out != null) out.close();
	}
}
```

```
꿈에 미치면 신화가 된다.
```

# InputStreamReader와 OutputStreamWriter

- 바이트 스트림과 문자 스트림을 연결하는 클래스

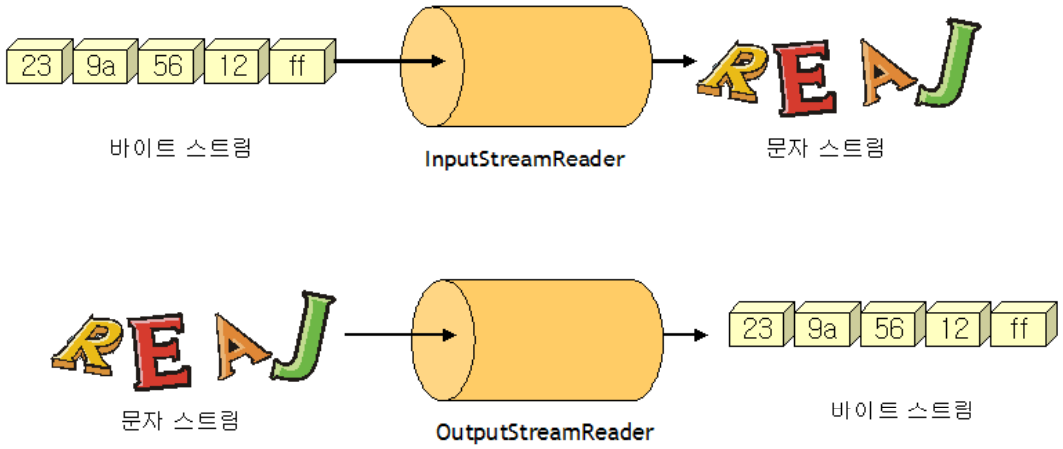
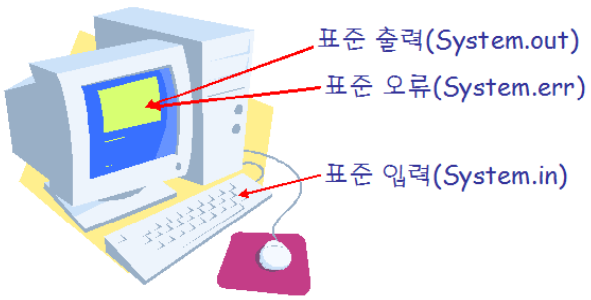



그림 24-12 브릿지 스트림



# 명령어 행에서 입출력

- System.in, System.out, System.err
- 표준 스트림은 모두 역사적인 이유로 바이트 스트림으로 정의



# File 클래스

- File 클래스는 파일을 나타낸다.

```
File  file = new File("data.txt");
```

- 파일에 대한 여러 가지 메소드를 제공

| 반환형 | 메소드 | 설명 |
|---|---|---|
| boolean | canExecute() | 파일을 실행할 수 있는지의 여부 |
| boolean | canRead() | 파일을 읽을 수 있는지의 여부 |
| boolean | canWrite() | 파일을 변경할 수 있는지의 여부 |
| static File | createTempFile(String prefix, String suffix) | 임시 파일을 생성한다. |
| boolean | delete() | 파일을 삭제한다. |
| void | deleteOnExit() | 가상 기계가 종료되면 파일을 삭제한다. |
| boolean | exists() | 파일의 존재 여부 |
| String | getAbsolutePath() | 절대 경로를 반환 |



# 예제

```
import java.io.File;
import java.io.IOException;
public class FileTest {
	public static void main(String[] args) throws IOException {
		String name = "c:/eclipse";
		File dir = new File(name);
		String[] fileNames = dir.list();  // 현재 디렉토리의 전체 파일 리스트
		for (String s : fileNames) {
			File f = new File(name + "/" + s);  // 절대 경로로 이름을 주어야 함
			System.out.println("================================");
			System.out.println("이름: " + f.getName());
			System.out.println("경로: " + f.getPath());
			System.out.println("부모: " + f.getParent());
			System.out.println("절대경로: " + f.getAbsolutePath());
			System.out.println("정규경로: " + f.getCanonicalPath());
			System.out.println("디렉토리 여부:" + f.isDirectory());
			System.out.println("파일 여부:" + f.isFile());
			System.out.println("================================");
		}
	}
```

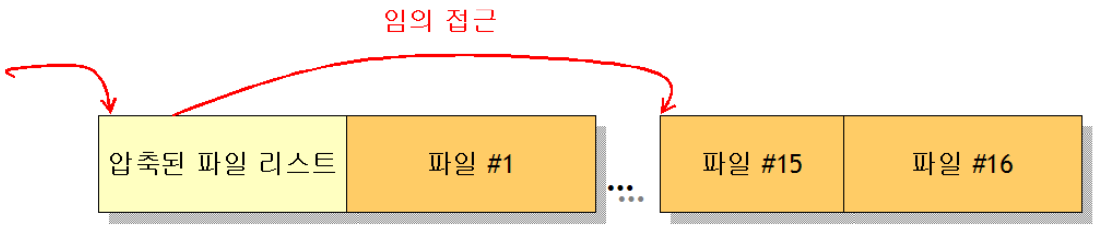
# 예제

```
==============================
이름: .eclipseproduct
경로: c:\eclipse\.eclipseproduct
부모: c:\eclipse
절대경로: c:\eclipse\.eclipseproduct
정규경로: C:\eclipse\.eclipseproduct
디렉토리 여부:false
파일 여부:true
==============================
...
```



# 임의 접근 파일

- 임의 접근 파일은 파일에 비순차적인 접근을 가능하게 한다.
- **new** RandomAccessFile("all.zip", "r");




| 메소드 | 설명 |
|---|---|
| int skipBytes(int) | 지정된 바이트만큼 파일 포인터를 앞쪽으로 이동한다. |
| void seek(long) | 지정된 바이트 위치로 파일 포인터를 설정한다. |
| long getFilePointer() | 파일 포인터의 현재 위치를 반환한다. |

# Q & A