



# 고급자바 : 입출력 처리

## 스트림(Stream)

순서가 있는 데이터의 연속적인 흐름이다.  
스트림은 입출력을 물의 흐름처럼 간주하는 것이다.

## 입출력 스트림

### 스트림 개념

연속된 데이터의 단방향 흐름을 추상화한다.  
데이터 소스와 상관없이 적용할 수 있어 매우 효과적이다.

- 스트림의 예

키보드 및 모니터의 입출력

프로그램과 외부장치, 파일의 입출력에서 데이터 흐름

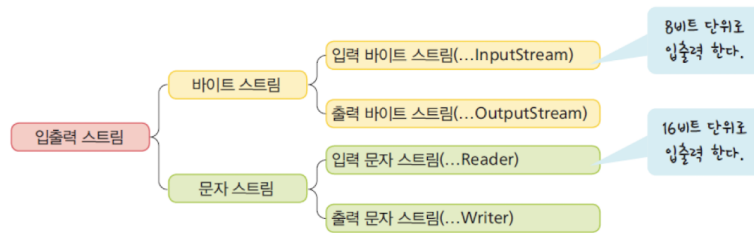
네트워크와 통신하는 데이터의 흐름

데이터 집합체의 각 원소를 순회하면서 람다식으로 반복 처리되는 데이터 흐름

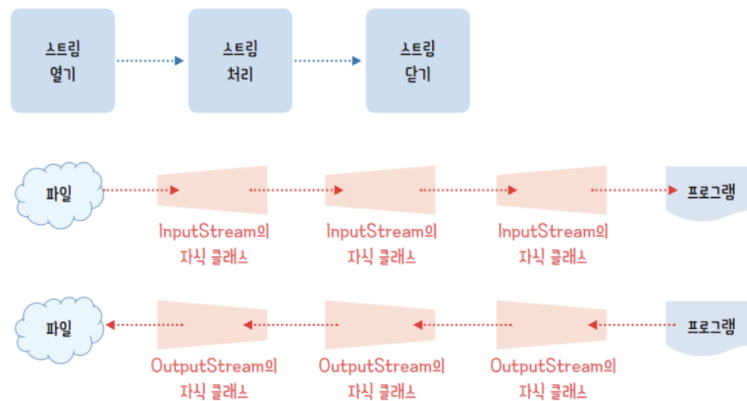
### 입출력 스트림의 특징

- 선입선출 구조라서 순차적으로 흘러가고 순차적으로 접근한다. (FIFO 구조)
- 임의 접근 파일 스트림을 제외한 모든 스트림은 단방향이다.
- 입출력 스트림은 객체이다.
- 출력 스트림과 입력 스트림을 연결해서 파이프라인 구성이 가능하다.
- 지연이 가능하다. 프로그램에 연결한 출력 스트림에 데이터가 가득 차면 프로그램을 더 이상 출력할 수 없어 빈 공간이 생길 때까지 지연되며, 데이터 소스에 연결한 입력 스트림도 가득 차면 프로그래밍 데이터를 처리해서 빈 공간이 생길 때까지 스트림이 지연된다.

### 입출력 스트림의 분류



## 입출력 스트림의 사용

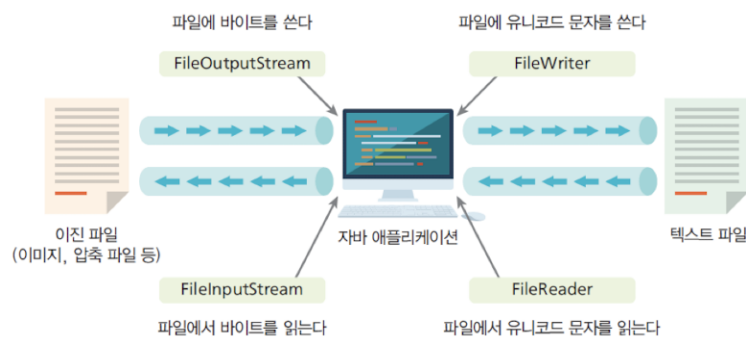


## 바이트 스트림(byte stream)

바이트 단위로 입출력하는 클래스이다.

바이트 스트림 클래스들은 추상 클래스인 InputStream와 OutputStream에서 파생된다.

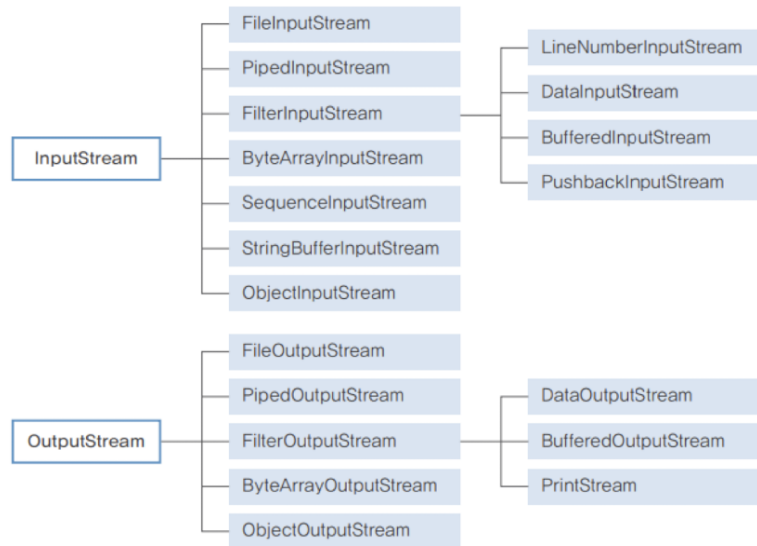
바이트 스트림 클래스 이름에는 InputStream(입력)과 OutputStream(출력)이 붙는다.



## 기초

바이트 단위의 이진 데이터를 다루므로 이미지나 동영상 파일을 처리할 때 유용하다.

## 종류



## InputStream과 OutputStream

모든 자식 바이트 스트림에서 공통으로 사용하는 메소드를 포함한 바이트 스트림의 최상위 클래스이다.

각각 read()와 write()라는 추상 메소드를 포함한다.

- 주요 메소드

클래스	메서드	설명
InputStream	int available()	읽을 수 있는 바이트의 개수를 반환한다.
	void close()	입력 스트림을 닫는다.
	abstract int read()	1바이트를 읽는다.
	int read(byte b[])	1바이트씩 읽어 b[]에 저장한 후 읽은 개수를 반환한다.
	int read(byte b[], int off, int len)	len만큼 읽어 b[]의 off 위치에 저장한 후 읽은 개수를 반환한다.
	long skip(long n)	입력 스트림을 n바이트만큼 건너뛴다.
OutputStream	void close()	출력 스트림을 닫는다.
	void flush()	출력하려고 버퍼의 내용을 비운다. <i>필수적이다.</i>
	abstract void write(int b)	b 값을 바이트로 변환해서 쓴다.
	void write(byte b[])	b[] 값을 바이트로 변환해서 쓴다.
	void write(byte b[], int off, int len)	b[] 값을 바이트로 변환해서 off 위치부터 len 만큼 쓴다.

- read() 메소드의 반환 값은 0~255의 ASCII값이며, 더 이상 읽을 데이터가 없을 때는 -1을 반환한다.
- read() 메소드는 int 타입을 반환한다. (문자로 나타내려면 char 타입으로 변경해줘야 한다.)
- write() 메소드는 인수가 배열일 때는 byte[], 배열이 아닐 때는 int 타입이다.
- 대부분의 운영체제나 JVM 표준 출력 장치를 효율적으로 관리하려고 버퍼를 사용한다. (버퍼를 사용하지 않으면 지연된다.)

- BufferedStream이 아니지만 System.out은 표준 출력이므로 버퍼를 사용한다.
- System.out을 이용해 출력할 때는 버퍼를 비우기 위해 flush()호출이 필요하다.

## InputStream과 OutputStream예제

```

import java.io.IOException;

public class IOStreamDemo {
    public static void main(String[] args) throws IOException {
        (int) b, len= 0;
        (int) ba[] = new int[100];
        System.out.println("--- 입력 스트림 ---");
        while((b= System.in.read()) != '\n'){
            System.out.printf("%c(%d)", (char) b, b);
            ba[len++] = b;
        }
        System.out.println("\n\n--- 출력 스트림 ---");
        for(int i= 0; i< len; i++)
            System.out.write(ba[i]);
        System.out.flush(); // System.out.close();
    }
}

public abstract int read() throws IOException
public void write(byte[] b) throws IOException
  
```

--- 입력 스트림 ---  
 hello  
 키보드로 입력한 hello와 Enter이다.  
 h(104)e(101)l(108)l(108)o(111)  
 --- 출력 스트림 ---  
 hello

*Handwritten notes in Korean:*  
 3바이트가 저장된다.  
 배열을 채워야 할지 모름.  
 아니면 클라이언트 이용해서 저장과 flush는 일괄로 가능.

## FileInputStream과 FileOutputStream

시스템에 있는 모든 파일을 읽거나 쓸 수 있는 기능을 제공한다.

생성자로 스트림 객체를 생성할 때는 FileNotFoundException 예외 가능성이 있기 때문에 반드시 예외 처리가 필요하다.

```

FileInputStream(String name)           // 파일 시스템의 경로를 나타내는 문자열
FileInputStream(File file)

FileOutputStream(String name)
FileOutputStream(File file)
FileOutputStream(String name, boolean append) // true: 이어쓴다.(append) false: 덮어쓴다.(overwrite)
FileOutputStream(File file, boolean append)
  
```

## FileInputStream과 FileOutputStream예제

```
import java.io.*;

public class CopyFileDemo {
    public static void main(String[] args) {
        String input = "D:\\Temp\\org.txt";
        String output = "D:\\Temp\\dup.txt";

        try (FileInputStream fis = new FileInputStream(input);
             FileOutputStream fos = new FileOutputStream(output)) {
            int c;

            while ((c = fis.read()) != -1)
                fos.write(c);
        } catch (IOException e) {
        }
    }
}
```

## BufferedInputStream과 BufferedOutputStream

버퍼는 스트림과 프로그램 간에 데이터를 효율적으로 전송하려고 사용하는 메모리이다.  
입출력 장치와 프로그램 간 동작 속도가 크게 차이 날 때 버퍼를 사용하면 매우 효율적이다.

```
BufferedInputStream(InputStream in)
BufferedInputStream(InputStream in, int size)

BufferedOutputStream(OutputStream out)
BufferedOutputStream(OutputStream out, int size)
```

## BufferedFileInputStream과 BufferedFileOutputStream예제

```
import java.io.*;

public class BufferedStreamDemo {
    public static void main(String[] args) {
        long start, end, duration;
        String org = "C:\\Program Files (x86)\\Internet Explorer\\iexplore.exe";
        String dst = "D:\\Temp\\iexplore.exe";

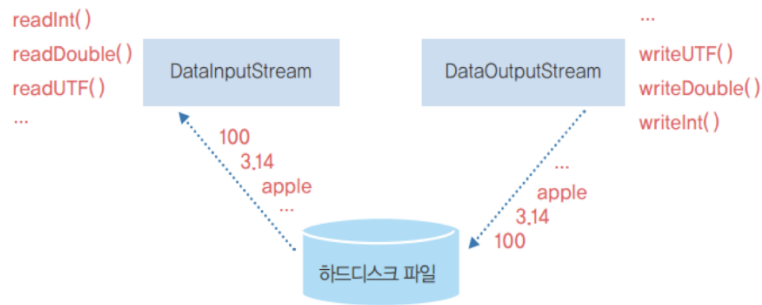
        start = System.nanoTime();
        try (BufferedInputStream bis = new BufferedInputStream(new FileInputStream(org));
             BufferedOutputStream bos = new BufferedOutputStream(new FileOutputStream(dst))) {
            while (bis.available() > 0) {
                int b = bis.read();
                bos.write(b);
            }
            bos.flush();
        } catch (IOException e) {
        }
        end = System.nanoTime();
        duration = end - start;
        System.out.println("버퍼를 사용한 경우 : " + duration);

        start = System.nanoTime();
        try (FileInputStream fis = new FileInputStream(org); FileOutputStream fos = new FileOutputStream(dst)) {
            while (fis.available() > 0) {
                int b = fis.read();
                fos.write(b);
            }
            fos.flush();
        } catch (IOException e) {
        }
        end = System.nanoTime();
        duration = end - start;
        System.out.println("버퍼를 사용하지 않은 경우 : " + duration);
    }
}
```

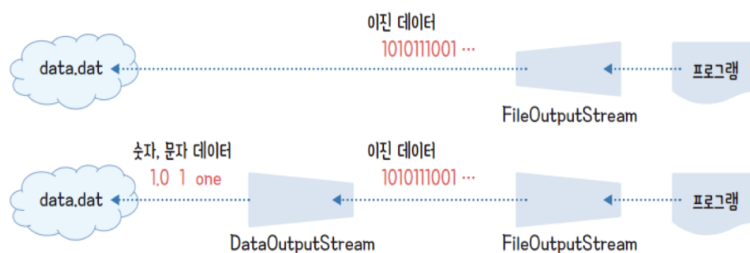
```
버퍼를 사용한 경우 : 1369231621
버퍼를 사용하지 않은 경우 : 3691319106
```

## DataInputStream과 DataOutputStream

각각 기초 타입 데이터를 읽는 메소드와 기초 타입 데이터를 기록하는 메소드를 사용할 수 있는 스트림이다.



직접 키보드에서 데이터를 입력 받고나 콘솔 뷰에 데이터를 출력하기에는 부적합하다. `FileInputStream` 이나 `FileOutputStream` 등 다른 스트림과 연결해서 파이프 라인을 구성해 사용한다.



## DataInputStream과 DataOutputStream예제

```
import java.io.*;

public class DataStreamDemo {
    public static void main(String[] args) {
        try (DataOutputStream dos
            = new DataOutputStream(new FileOutputStream("D:\\Temp\\data.dat"));
            DataInputStream dis
            = new DataInputStream(new FileInputStream("D:\\Temp\\data.dat"))) {

            dos.writeDouble(1.0);
            dos.writeInt(1);
            dos.writeUTF("one");

            dos.flush();

            System.out.println(dis.readDouble());
            System.out.println(dis.readInt());
            System.out.println(dis.readUTF());
        } catch (IOException e) {
        }
    }
}
```

1.0  
1  
one

## PrintStream

다양한 데이터 값을 편리하게 표현할 수 있도록 출력 스트림에 기능을 추가한 스트림이다.

- 특징

`IOException`을 발생하지 않는다.

자동 플러시 기능을 제공해 `flush()` 메소드를 호출하지 않고도 버퍼를 비울 수 있다.

```
PrintStream(File file)
PrintStream(String filename)
PrintStream(OutputStream out)
PrintStream(OutputStream out, boolean autoFlush)
```

System.out 객체의 println(), print(), printf() 메소드는 PrintStream 으로 출력한다.

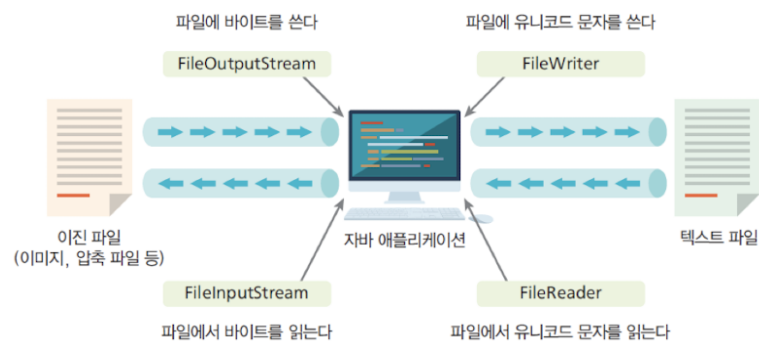
```
System.out.println("안녕!");
```

## 문자 스트림(character stream)

문자 단위로 입출력하는 클래스이다.

이들은 모두 기본 추상 클래스인 Reader와 Write 클래스에서 파생된다.

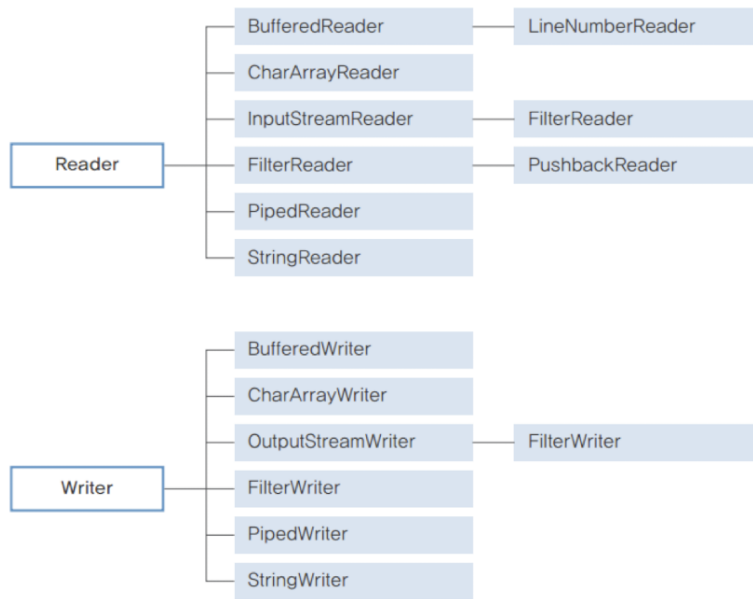
문자 스트림 클래스 이름에는 Reader(입력)와 Writer(출력)가 붙는다.



## 기초

데이터를 2바이트 단위인 유니코드로 전송하거나 수신하는데, 이진 데이터로 된 이미지나 동영상 파일보다는 한글처럼 언어로 된 파일을 처리할 때 유용하다.

## 종류



## Reader와 Writer

Reader와 Writer는 객체를 생성할 수 없는 추상 클래스이기 때문에 Reader와 Writer의 자식인 구현 클래스를 사용한다.

추상 메서드인 read(), close(), write(), flush(), close()를 각각 포함하는 추상클래스이다.

- 문자 스트림 클래스가 제공하는 주요 클래스

클래스	메서드	설명
Reader	abstract void close()	입력 스트림을 닫는다.
	int read()	1개의 문자를 읽는다.
	int read(char[] cbuf)	문자 단위로 읽어 cbuf[]에 저장한 후 읽은 개수를 반환한다.
	abstract int read(char cbuf[], int off, int len)	len만큼 읽어 cbuf[]의 off 위치에 저장한 후 읽은 개수를 반환한다.
	long skip(long n)	입력 스트림을 n 문자만큼 건너뛴다.
Writer	abstract void close()	스트림을 닫고 관련된 모든 자원을 반납한다.
	abstract void flush()	버퍼의 내용을 비운다.
	void write(int c)	c 값을 char로 변환해 출력 스트림에 쓴다.
	void write(char cbuf[])	cbuf[] 값을 char로 변환해 출력 스트림에 쓴다.
	abstract void write(char cbuf[], int off, int len)	cbuf[] 값을 char로 변환해 off 위치부터 len만큼 출력 스트림에 쓴다.
	void write(String str)	문자열 str을 출력 스트림에 쓴다.

## FileReader와 FileWriter

FileReader와 FileWriter는 파일 입출력 클래스로, 파일에서 문자 데이터를 읽거나 파일에 문자 데이터를 저장할 때 사용한다.

시스템에 있는 모든 파일을 읽거나 쓸 수 있는 기능을 제공한다.

생성자로 스트림 객체를 생성할 때는 FileNotFoundException 예외 가능성이 있기 때문에 반드시 예외 처리가 필요하다.



- 생성자

```
FileReader(String name)          // 파일 시스템의 경로를 나타내는 문자열
FileReader(File file)

FileWriter(String name)
FileWriter(File file)
FileWriter(String name, boolean append) // true: 이어쓴다.(append) false: 덮어쓴다.(over
write)
FileWriter(File file, boolean append)
```

## FileReader와 FileWriter 예제

```
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;

public class CopyFileDemo {
    public static void main(String[] args) {
        String input = "D:\\Temp\\org.txt";
        String output = "D:\\Temp\\dup.txt";

        try (FileReader fr = new FileReader(input);
            FileWriter fw = new FileWriter(output)) {
            int c;

            while ((c = fr.read()) != -1)
                fw.write(c);
        } catch (IOException e) {
        }
    }
}
```

## BufferedReader와 BufferedWriter

BufferedReader와 BufferedWriter는 데이터를 효율적으로 전송하려고 버퍼로 처리할 때 사용한다.

스트림의 효율을 높이려고 버퍼를 사용한다.

- 생성자

```
BufferedInputStream(InputStream in)
BufferedInputStream(InputStream in, int size)

BufferedOutputStream(OutputStream out)
BufferedOutputStream(OutputStream out, int size)
```

- BufferedReader 클래스에 추가된 주요 메소드

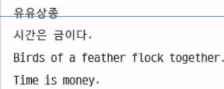
`Stream<String> lines()` : 읽은 행을 스트림으로 반환한다.

`String readLine()` : 한 행을 읽어 문자열로 반환한다.

## BufferedReader와 BufferedWriter 예제

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

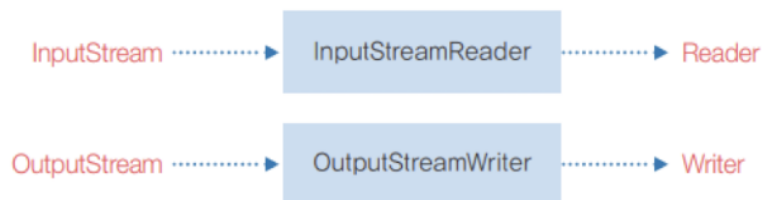
public class BufferedReaderDemo {
    public static void main(String[] args) {
        try (BufferedReader br
            = new BufferedReader(new FileReader("D:\\Temp\\org.txt"))); {
            br.lines().forEach(s -> System.out.println(s));
        } catch (IOException e) {
        }
    }
}
```



## InputStreamReader와 OutputStreamReader

InputStreamReader와 OutputStreamReader는 바이트 스트림과 문자 스트림을 연결하는 브리지 스트림으로 사용한다.

바이트 기반의 InputStream과 OutputStream을 포장해 문자 기반의 Reader와 Writer로 변환하는 클래스이다.

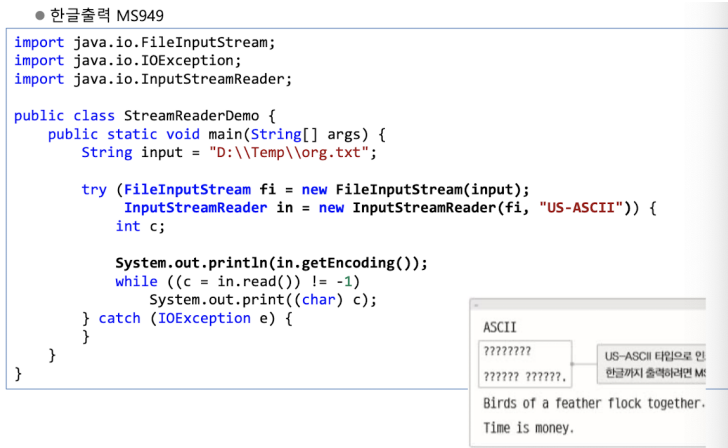


- 생성자

```
InputStreamReader(InputStream in)
InputStreamReader(InputStream in, Charset cs)

OutputStreamWriter(OutputStream out)
OutputStreamWriter(OutputStream out, Charset cs)
```

## InputStreamReader와 OutputStreamReader 예제



## PrintWriter

PrintStream처럼 다양한 데이터 값을 편리하게 표현할 수 있도록 스트림에 기능을 추가한 Writer의 자식 클래스이다.

JAVA 5부터 PrintStream이 문자 인코딩을 허용하면서 PrintWriter와 거의 차이가 없다. 특히, 두 클래스가 제공하는 println(), print(), printf() 메소드는 같으므로 표준 출력에서 사용할 때는 차이가 없다.

- 생성자

```
PrintWriter(File file)
PrintWriter(String filename)
PrintWriter(OutputStream out)
PrintWriter(OutputStream out, boolean autoFlush)
PrintWriter(Writer out)
PrintWriter(Writer out, boolean autoFlush)
```

## PrintWriter 예제

```
import java.io.*;

public class PrintWriterDemo {
    public static void main(String[] args) {
        try (BufferedReader br = new BufferedReader(new FileReader("D:\\Temp\\org.txt"));
            PrintWriter pr = new PrintWriter(new FileWriter("D:\\Temp\\dup.txt"))) {
            br.lines().forEach(x -> pr.println(x));
        } catch (IOException e) {
        }
    }
}
```

# 파일 관리

## 기초

입출력 스트림은 파일이나 장치를 읽거나 쓰기 위해 사용한다.

입출력 스트림에 파일 생성 혹은 삭제, 파일 속성 변경 등의 관리 기능은 없다.

Java는 파일을 관리하기 위해 File 클래스를 제공한다.

여전히 File 클래스를 많이 사용하지만, Java 4부터 도입되고 Java 7에서 기능을 보완한 NIO(java.nio) 및 NIO2(java.nio2) 기반의 Path 인터페이스, Files 클래스 및 FileChannel도 유용하다.

## File 클래스

파일이나 폴더의 경로를 추상화한 클래스로 java.io 패키지에 포함한다.

파일 유무, 삭제, 접근 권한 조사 등을 수행한다.

- File 클래스 생성자

생성자	설명
File(File parent, String child)	parent 객체 폴더의 child라는 File 객체를 생성한다.
File(String pathname)	pathname에 해당하는 File 객체를 생성한다.
File(String parent, String child)	parent 폴더에 child라는 File 객체를 생성한다.
File(URI uri)	uri 경로에서 File 객체를 생성한다.

- 주요 메소드

메서드	설명
boolean canExecute()	실행 가능한 파일인지 여부를 반환한다.
boolean canRead()	읽을 수 있는 파일인지 여부를 반환한다.
boolean canWrite()	쓸 수 있는 파일인지 여부를 반환한다.
boolean createNewFile()	파일을 새로 생성하면 true, 아니면 false를 반환한다.
boolean delete()	파일을 삭제하면 true, 아니면 false를 반환한다.
boolean exists()	파일의 존재 유무를 반환한다.
String getAbsolutePath()	파일의 절대 경로를 반환한다.
String getName()	파일의 이름을 반환한다.

String getPath( )	파일의 경로를 반환한다.
boolean isDirectory( )	폴더 존재 여부를 반환한다.
boolean isFile( )	파일 존재 여부를 반환한다.
long lastModified( )	파일의 마지막 수정 시간을 반환한다.
long length( )	파일의 크기를 반환한다.
String[ ] list( )	모든 자식 파일과 폴더를 문자열 배열로 반환한다.
File[ ] listFiles( )	모든 자식 파일과 폴더를 File 배열로 반환한다.
boolean mkdir( )	폴더를 생성하면 true, 아니면 false를 반환한다.
Path toPath( )	파일 경로에서 구성한 Path 객체를 반환한다.

## • 예제

```
import java.io.File;
import java.io.IOException;

public class FileDemo {
    public static void main(String[] args) throws IOException {
        File file = new File("C:\\Windows");
        File[] fs = file.listFiles();

        for (File f : fs)
            if (f.isDirectory())
                System.out.printf("dir : %s\n", f);
            else
                System.out.printf("file: %s(%d bytes)\n", f, f.length());
    }
}
```

```
dir : C:\Windows\addins
dir : C:\Windows\appcompat
dir : C:\Windows\AppPatch
dir : C:\Windows\AppReadiness
dir : C:\Windows\assembly
dir : C:\Windows\bcastdvr
file: C:\Windows\bfsvc.exe(61440 bytes)
...
```

## Path 인터페이스

운영체제에 따라 인관성 없이 동작하는 File 클래스를 대체하는 것이다.

기존 File 객체도 File 클래스의 toPath() 메소드를 이용해 Path 타입으로 변환 가능하다.

Path 인터페이스의 구현 객체는 파일 시스템에서 경로를 나타낸다. ("C:\homework"와 같은 경로를 받아 객체를 반환한다.)

java.nio.file 패키지에 포함된다.

java.io.File 클래스와 마찬가지로 파일의 유무, 삭제, 접근 권한 조사 등이 주요 기능이다.

메서드	설명
Path getFileName( )	객체가 가리키는 파일(폴더) 이름을 반환한다.
FileSystem getFileSystem( )	객체를 생성한 파일 시스템을 반환한다.
int getNameCount( )	객체가 가리키는 경로의 구성 요소 개수를 반환한다.
Path getParent( )	부모 경로를 반환하며, 없으면 null을 반환한다.
Path getRoot( )	루트를 반환하며, 없으면 null을 반환한다.
boolean isAbsolute( )	절대 경로 여부를 반환한다.
Path toAbsolutePath( )	절대 경로를 나타내는 객체를 반환한다.
URI toUri( )	객체가 가리키는 경로에서 URI를 반환한다.

## Files 클래스

파일 연산을 수행하는 정적 메소드로 구성된 클래스이다.  
java.nio.file 패키지에 포함된다.

- 주요 메소드

메서드	설명
long copy( )	파일을 복사한 후 복사된 바이트 개수를 반환한다.
Path copy( )	파일을 복사한 후 복사된 경로를 반환한다.
Path createDirectory( )	폴더를 생성한다.
Path createFile( )	파일을 생성한다.
void delete( )	파일을 삭제한다.
boolean deleteIfExists( )	파일이 있으면 삭제한다.
boolean exists( )	파일의 존재 여부를 조사한다.
boolean isDirectory( )	폴더인지 조사한다.
boolean isExecutable( )	실행 가능한 파일인지 조사한다.
boolean isHidden( )	숨김 파일인지 조사한다.
boolean isReadable( )	읽기 가능한 파일인지 조사한다.
boolean isWritable( )	쓰기 가능한 파일인지 조사한다.
Path move( )	파일을 이동한다.
boolean notExists( )	파일(폴더)의 부재를 조사한다.
byte[ ] readAllBytes( )	파일의 모든 바이트를 읽어 배열로 반환한다.
List<String> readAllLines( )	파일의 모든 행을 읽어 리스트로 반환한다.
long size( )	파일의 크기를 반환한다.
Path write( )	파일에 데이터를 쓴다.

- 예제

```
import java.io.File;
import java.nio.file.Files;

public class Files1Demo {
    public static void main(String[] args) throws Exception {
        File f1 = new File("D:\\Temp\\org.txt");
        File f2 = new File("D:\\Temp");

        System.out.println("org.txt는 폴더? " + Files.isDirectory(f1.toPath()));

        System.out.println("Temp는 폴더? " + Files.isDirectory(f2.toPath()));

        System.out.println("org.txt는 읽을 수 있는 파일? " + Files.isReadable(f1.toPath()));

        System.out.println("org.txt의 크기?" + Files.size(f1.toPath()));
    }
}
```

