Chapter 12

입출력 처리

Contents

스트림의 개념

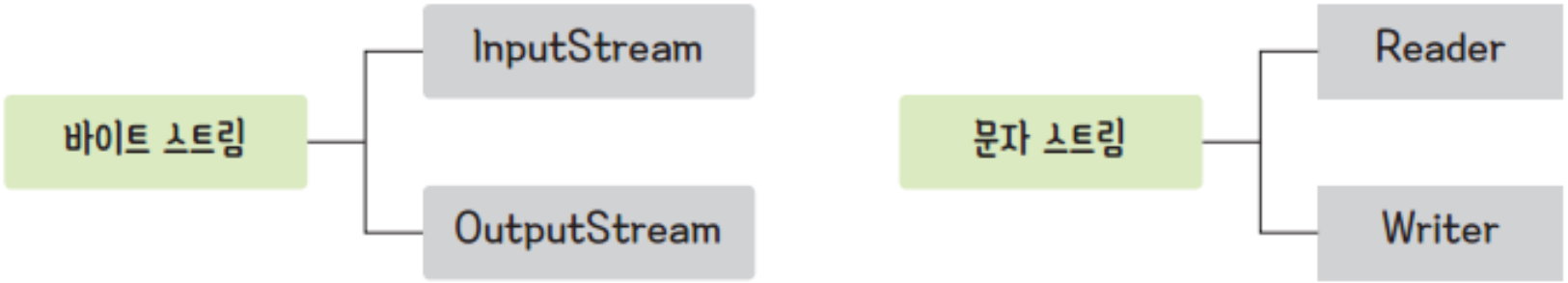
- 연속된 데이터의 단방향 흐름을 추상화
- 데이터 소스와 상관없이 적용할 수 있어 매우 효과적
- 스트림의 예
  - 키보드 및 모니터의 입출력
  - 프로그램과 외부 장치, 파일의 입출력에서 데이터 흐름도 스트림
  - 네트워크와 통신하는 데이터의 흐름

입출력 스트림의 특징

- 선입선출 구조라서 순차적으로 흘러가고 순차적으로 접근
- 임의 접근 파일 스트림을 제외한 모든 스트림은 단방향
- 입출력 스트림은 객체
- 출력 스트림과 입력 스트림을 연결해서 파이프라인 구성 가능
- 지연 가능. 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 프로그램이 데이터를 처리해서 빈 공간이 생길 때까지 스트림이 지연

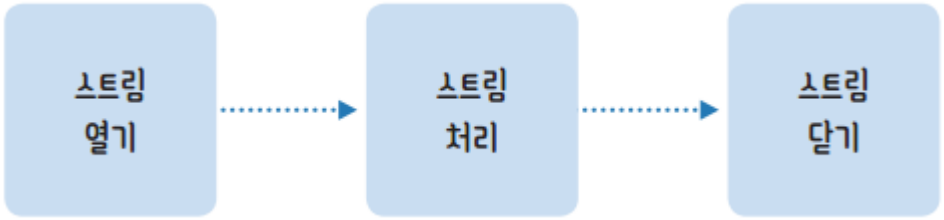
바이트 스트림과 문자 스트림

- 바이트 스트림(Byte Stream / Binary Stream)
  - 1 바이트 단위 전송
  - 이진 데이터 처리
- 문자 스트림(Character Stream)
  - 2 바이트 단위 전송

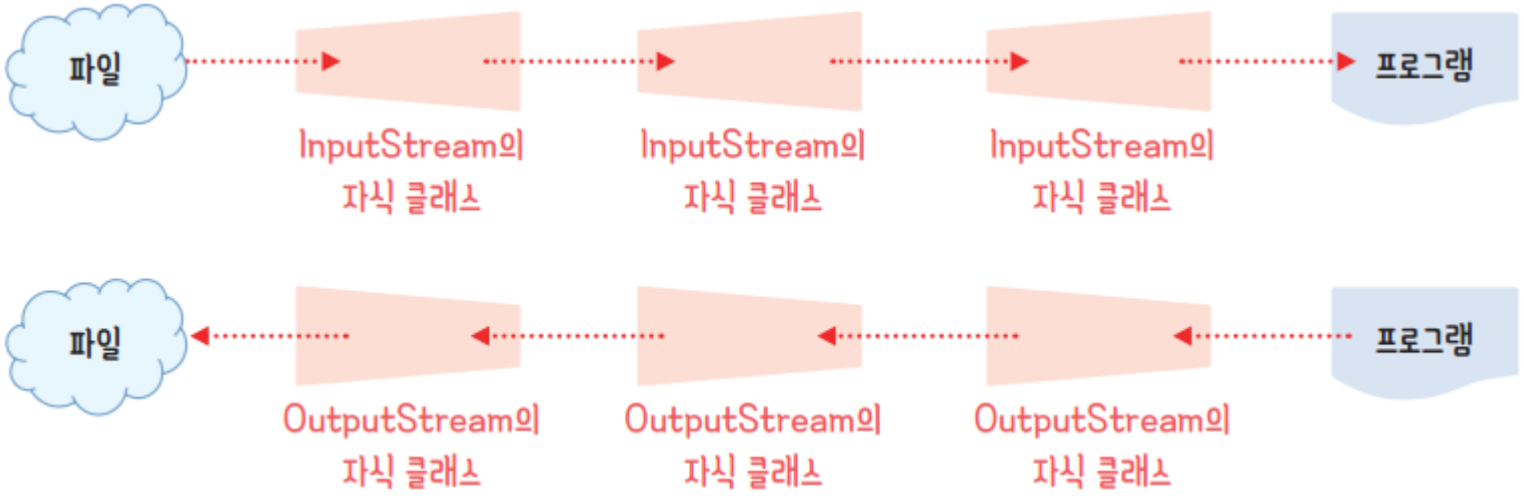


입출력 스트림의 사용

- 스트림은 사용 전에 먼저 열어야 하며, 사용을 마친 후에는 닫아야 함



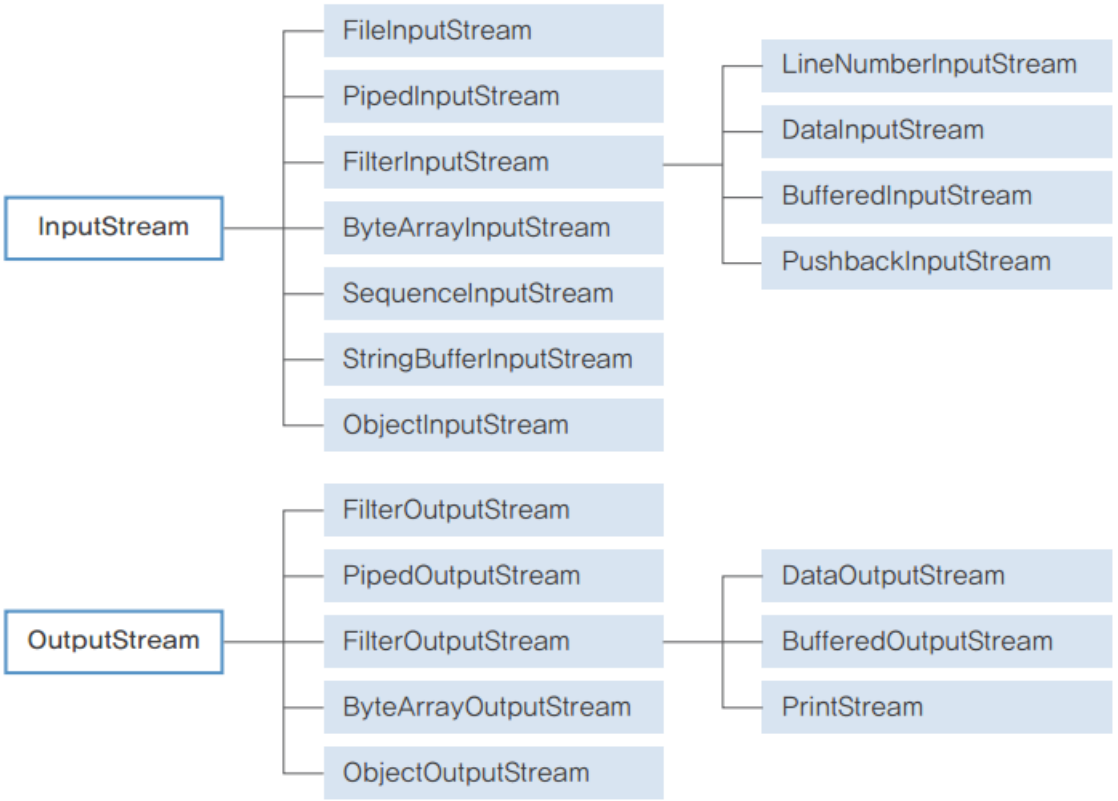
- 다수의 스트림을 연결해 파이프라인으로 구성하면 연결된 모든 기능을 사용할 수 있음



바이트 스트림의 개념과 종류

- 개념
  - 바이트 단위의 이진 데이터를 다루므로 이미지나 동영상 파일을 처리할 때 유용

- 종류



InputStream과 OutputStream(1)

- 모든 자식 바이트 스트림에서 공통으로 사용하는 메서드를 포함한 바이트 스트림의 최상위 클래스
- 각각 `read( )`와 `write( )`라는 추상 메서드를 포함
- 바이트 스트림 클래스가 제공하는 디폴트 메서드

InputStream과 OutputStream(2)

- 바이트 스트림 클래스가 제공하는 디폴트 메서드

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.
	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.

InputStream과 OutputStream(3)

- read( ) 메서드의 반환 값은 0~255의 ASCII 값이며, 더 이상 읽을 데이터가 없을 때는 -1을 반환
- read( ) 메서드는 int 타입을 반환
- write( ) 메서드에서 인수가 배열일 때는 byte[ ], 배열이 아닐 때는 int 타입

- 대부분의 운영체제나 JVM은 표준 출력 장치를 효율적으로 관리하려고 버퍼를 사용
- BufferedStream이 아니지만 System.out은 표준 출력이므로 버퍼를 사용
- System.out을 사용해 출력할 때는 버퍼를 비우기 위하여 flush( ) 호출 필요

- 예제 : [예제 12-1]

```java
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();
    }
}
```

```
--- 입력 스트림 ---
hello
h(104)e(101)l(108)l(108)o(111)
(13)

--- 출력 스트림 ---
hello
```

FileInputStream및 FileOutputStream

- 시스템에 있는 모든 파일을 읽거나 쓸 수 있는 기능을 제공
- 생성자로 스트림 객체를 생성할 때는 FileNotFoundException 예외 가능성이 있기 때문에 반드시 예외 처리 필요

```
FileInputStream(String name)
FileInputStream(File file)
FileOutputStream(String name)
FileOutputStream(File file)
FileOutputStream(String name, boolean append)
FileOutputStream(File file, boolean append)
```

파일 시스템의 경로를 나타내는 문자열이다.

true면 이어쓰고(append), false면 덮어쓴다(overwrite).

- 예제 : [예제 12-2]

```
3  import java.io.FileInputStream;
4  import java.io.FileOutputStream;
5  import java.io.IOException;
6
7  public class CopyFileDemo {
8      public static void main(String[] args) {
9          String input = "D:\\Temp\\org.txt";
10         String output = "D:\\Temp\\dup.txt";
11
12         try (FileInputStream fis = new FileInputStream(input);
13                 FileOutputStream fos = new FileOutputStream(output)) {
14             int c;
15
16             while ((c = fis.read()) != -1)
17                 fos.write(c);
18         } catch (IOException e) {
19         }
20     }
21 }
```

BufferedInputStream및 BufferedOutputStream

- 버퍼는 스트림과 프로그램 간에 데이터를 효율적으로 전송하려고 사용하는 메모리
- 입출력 장치와 프로그램 간 동작 속도가 크게 차이가 날 때 버퍼를 사용하면 매우 효율적

- 생성자

```
BufferedInputStream(InputStream in)
BufferedInputStream(InputStream in, int size)
BufferedOutputStream(OutputStream out)
BufferedOutputStream(OutputStream out, int size)
```

버퍼의 크기를 나타낸다.

- 예제 : [예제 12-3]

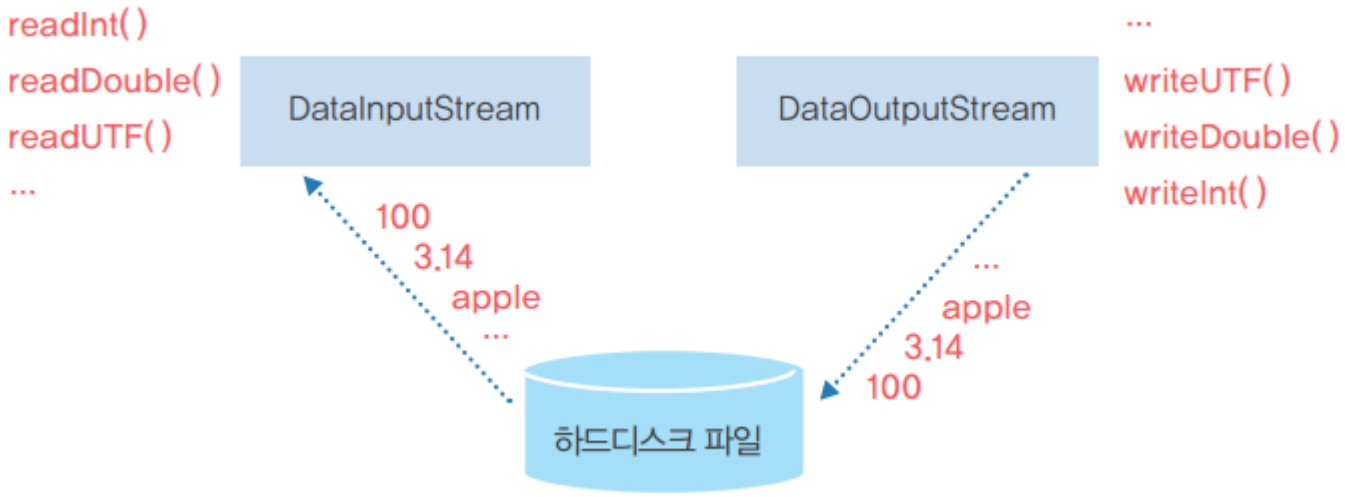
```java
import java.io.BufferedInputStream;
import java.io.BufferedOutputStream;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;

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\\iexplore1.exe";

        start = System.nanoTime();
        try (BufferedInputStream bis = new BufferedInputStream(new FileInputStream(org));
                BufferedOutputStream bos = new BufferedOutputStream(new FileOutputStream(dst));) {
            while (bis.available() > 1) {
                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);
```

```
29          start = System.nanoTime();
30          try (FileInputStream fis = new FileInputStream(org);
31                  FileOutputStream fos = new FileOutputStream(dst);) {
32              while (fis.available() > 1) {
33                  int b = fis.read();
34                  fos.write(b);
35              }
36              fos.flush();
37          } catch (IOException e) {
38          }
39          end = System.nanoTime();
40          duration = end - start;
41          System.out.println("버퍼를 사용하지 않은 경우  : " + duration);
42      }
43  }
```

```
버퍼를 사용한 경우 : 1219480
버퍼를 사용하지 않은 경우  : 388611
```

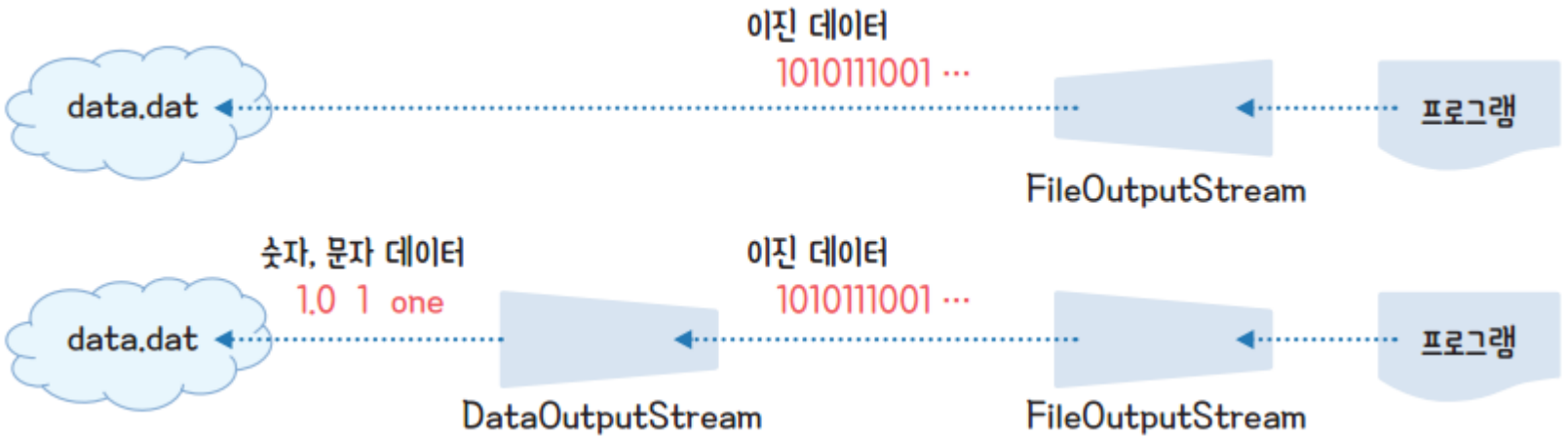
DataInputStream및 DataOutputStream(1)

- 각각 기초 타입 데이터를 읽는 메서드와 기초 타입 데이터를 기록하는 메서드를 사용할 수 있는 스트림



DataInputStream및 DataOutputStream(2)

- 직접 키보드에서 데이터를 입력받거나 콘솔 뷰에 데이터를 출력하기에는 부적합
- FileInputStream및 FileOutputStream등 다른 스트림과 연결해서 파이프라인을 구성해 사용



- 예제 : [예제 12-4]

```java
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;

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
```

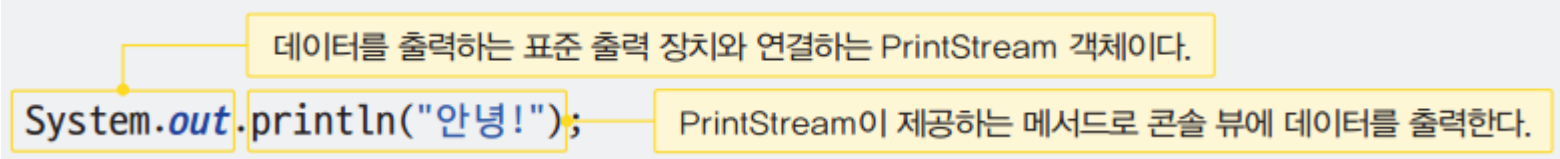
PrintStream

- 다양한 데이터 값을 편리하게 표현할 수 있도록 출력 스트림에 기능을 추가한 스트림
- 특징
  - IOException을 발생하지 않음
  - 자동 플러시 기능을 제공해 flush( ) 메서드를 호출하지 않고도 버퍼를 비울 수 있음

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

자동 플러시 기능을 선택할 수도 있다.

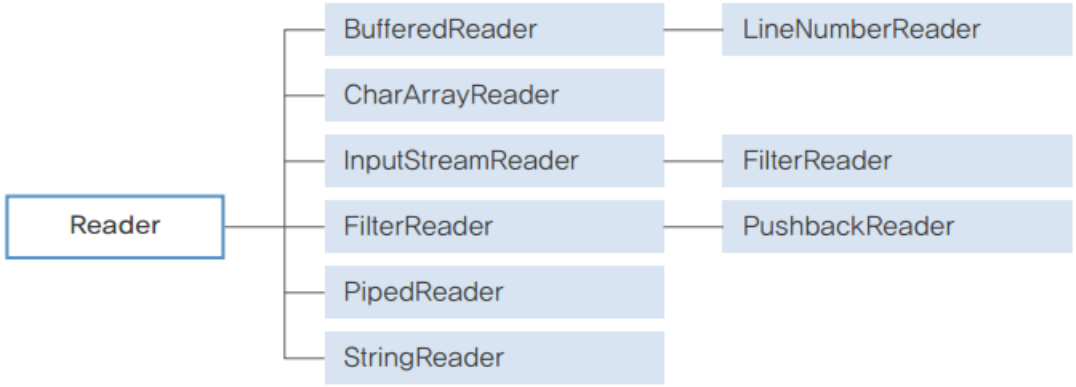
- System.out객체의 println( ), print( ), printf( ) 메서드는 PrintStream으로 출력

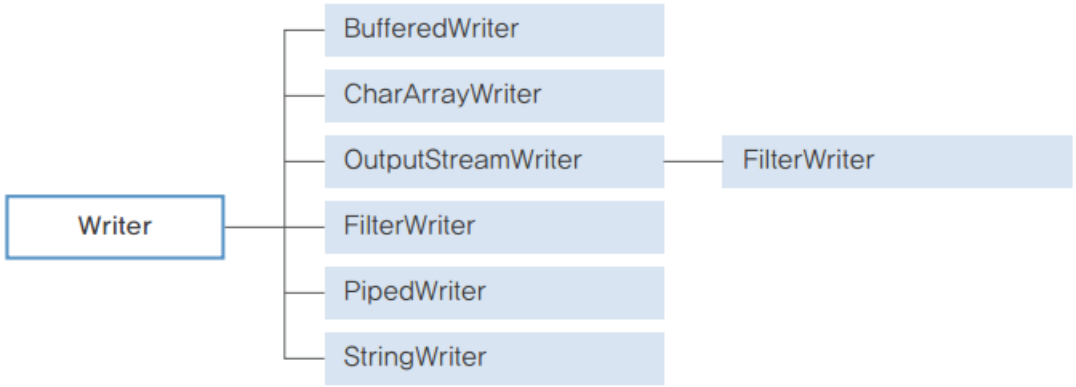


문자 스트림의 개념과 종류(1)

- 개념
  - 데이터를 2바이트 단위인 유니코드로 전송하거나 수신하는데, 이진 데이터로 된 이미지나 동영상 파일보다는 한글처럼 언어로 된 파일을 처리할 때 유용

- 종류





문자 스트림의 개념과 종류(2)

- 기초
  - Reader와 Writer는 객체를 생성할 수 없는 추상 클래스이기 때문에 Reader와 Writer의 자식인 구현 클래스를 사용
  - FileReader와 FileWriter는 파일 입출력 클래스로, 파일에서 문자 데이터를 읽거나 파일에 문자 데이터를 저장할 때 사용
  - InputStreamReader및 OutputStreamWriter는 바이트 스트림과 문자 스트림을 연결하는 브리지 스트림으로 사용
  - BufferedReader와 BufferedWriter는 데이터를 효율적으로 전송하려고 버퍼로 처리할 때 사용

Reader와 Writer

- 추상 메서드인 read( ), close( )와 write( ), flush( ), close( )를 각각 포함하는 추상 클래스
- 문자 스트림 클래스가 제공하는 디폴트 메서드

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

- 시스템에 있는 모든 문자 파일을 읽거나 파일에 쓸 수 있는 기능을 제공
- 생성자로 스트림 객체를 생성할 때는 FileNotFoundException 예외 처리 필요
- 생성자

```
FileReader(String name)
FileReader(File file)
FileWriter(String name)
FileWriter(File file)
FileWriter(String name, boolean append)
FileWriter(File file, boolean append)
```

파일 시스템의 경로를 나타내는 문자열이다.

true면 이어쓰고(append),
false면 덮어쓴다(overwrite).

- 예제 : [예제 12-5]

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

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

BufferedReader및 BufferedWriter

- 스트림의 효율을 높이려고 버퍼를 사용
- 생성자

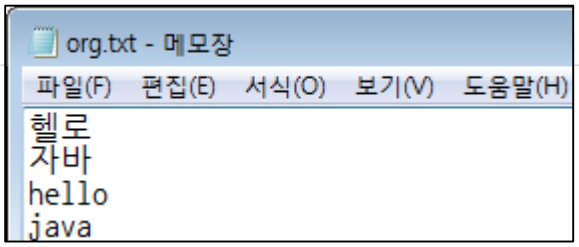
```
BufferedReader(Reader in)
BufferedReader(Reader in, int size)
BufferedWriter(Writer out)
BufferedWriter(Writer out, int size)
```

- BufferedReader 클래스에 추가된 주요 메서드

메서드	설명
Stream<String> lines( )	읽은 행을 스트림으로 반환한다.
String readLine( )	한 행을 읽어 문자열로 반환한다.

- 예제 : [예제 12-6]

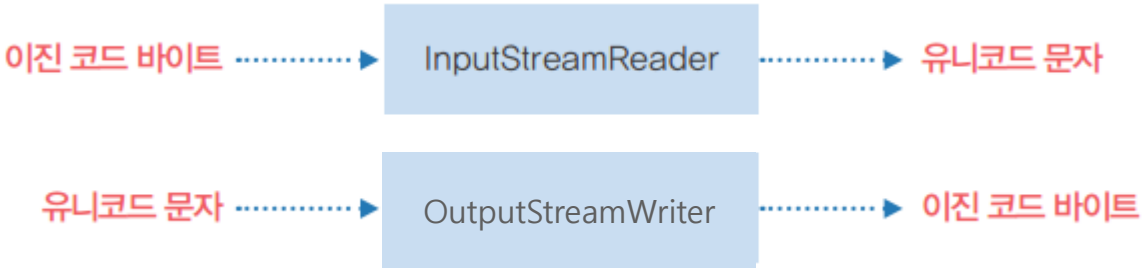
```java
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

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) {
        }
    }
}
```




```
헬로
자바
hello
java
```

InputStreamReader및 OutputStreamWriter

- 바이트 기반의 InputStream과 OutputStream을 포장해 문자 기반의 Reader와 Writer로 변환하는 클래스

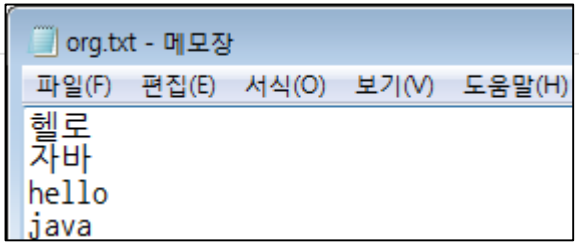


- 생성자

```
InputStreamReader(InputStream in)
InputStreamReader(InputStream in, Charset cs)
OutputStreamWriter(OutputStream out)
OutputStreamWriter(OutputStream out, Charset cs)
```

- 예제 : [예제 12-7]

```java
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStreamReader;

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

        try (FileInputStream fi = new FileInputStream(input);
                InputStreamReader in = new InputStreamReader(fi, "US-ASCII")) {
            int c;

            System.out.println(in.getEncoding());
            while ((c = in.read()) != -1)
                System.out.print((char) c);
        } catch (IOException e) {
        }
    }
}
```




```
ASCII
????
????
hello
java
```

- 예제 : [예제 12-8]

```
3  import java.io.BufferedReader;
4  import java.io.FileReader;
5  import java.io.FileWriter;
6  import java.io.IOException;
7  import java.io.PrintWriter;
8
9  public class PrinterWriterDemo {
10     public static void main(String[] args) {
11         try (BufferedReader br = new BufferedReader(
12                 new FileReader("D:\\Temp\\org.txt"));
13                 PrintWriter pr = new PrintWriter(
14                     new FileWriter("D:\\Temp\\dup.txt"));) {
15
16             br.lines().forEach(x -> pr.println(x));
17         } catch (IOException e) {
18         }
19     }
20 }
```

파일 관리 개념

- 입출력 스트림은 파일이나 장치를 읽거나 쓰기 위해 사용
- 입출력 스트림으로 파일을 생성하거나 삭제하거나 이름을 변경하는 등 관리 기능은 없음
- 자바는 파일을 관리하기 위해 File 클래스, Path 인터페이스, Files 클래스를 제공

File 클래스(1)

- 파일이나 폴더의 경로를 추상화한 클래스로 java.io 패키지에 포함
- 파일 유무, 삭제, 접근 권한 조사 등을 수행
- 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.

File 클래스(2)

- File 클래스의 주요 메서드 ①

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.

File 클래스(2)

- File 클래스의 주요 메서드 ②

메서드	설명
String getPath( )	파일의 경로를 반환한다.
boolean isDirectory( )	폴더 존재 유무를 반환한다.
boolean isFile( )	파일 존재 유무를 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 : [예제 12-9]

```
04  public class FileDemo {
05    public static void main(String[] args) throws IOException {
06       File file = new File("C:\\Windows");
07       File[] fs = file.listFiles();
08
09       for (File f : fs)
10          if (f.isDirectory())
11             System.out.printf("dir : %s\n", f);
12          else
13             System.out.printf("file: %s(%d bytes)\n", f, f.length());
14    }
15  }
```

폴더(C:\Windows)에 포함된 모든 자식 파일과 폴더를 File 배열로 반환한다.

File 객체가 폴더이면 폴더 경로를 출력하고, 파일이면 파일 경로 및 이름과 크기를 출력한다.

```
dir : C:\Windows\addins
file: C:\Windows\AhnInst.log(352824 bytes)
dir : C:\Windows\AppCompat
dir : C:\Windows\Application Data
dir : C:\Windows\AppPatch
dir : C:\Windows\assembly
```

Path 인터페이스

- 운영체제에 따라 일관성 없이 동작하는 File클래스를 대체
- 기존 File 객체도 File 클래스의 toPath( ) 메서드를 이용해 Path 객체로 변환 가능
- Path 인터페이스의 구현 객체는 파일 시스템에서 경로를 의미
- java.nio.file 패키지에 포함
- 주요 메서드

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 메서드로 구성된 클래스
- java.nio.file 패키지에 포함
- 주요 메서드

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 유무를 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.

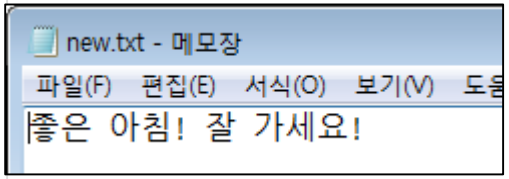
- 예제 : [예제 12-10]

```java
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()));
    }
}
```

```
org.txt는 폴더? false
Temp는 폴더? true
org.txt는 읽을 수 있는 파일? true
org.txt의 크기? 25
```

- 예제 : [예제 12-11]

```java
import java.io.File;
import java.io.IOException;
import java.nio.charset.Charset;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.StandardOpenOption;
import java.util.List;

public class Files2Demo {
    public static void main(String[] args) throws Exception {
        Charset cs = Charset.defaultCharset();
        Path p = new File("D:\\Temp\\new.txt").toPath();

        if (Files.notExists(p))
            Files.createFile(p);

        byte[] data = "좋은 아침!\n잘 가세요!\n".getBytes();
        Files.write(p, data, StandardOpenOption.APPEND);

        try {
            List<String> lines = Files.readAllLines(p, cs);

            for (String line : lines)
                System.out.println(line);
        } catch (IOException e) {
        }
    }
}
```




```
좋은 아침!
잘 가세요!
```

스트림 얻기

- BufferedReader 클래스의 lines( ) 메서드를 이용하면 스트림을 생성
- Files 클래스의 정적 메서드를 사용해 파일이나 폴더의 내용을 행 단위로 읽을 수 있는 스트림 생성
- Files 클래스가 스트림을 반환하는 제공하는 정적 메서드

메서드	설명
Stream<String> lines(Path path)	기본 문자집합을 이용해 파일의 모든 행을 스트림으로 반환한다.
Stream<String> lines(Path path, Charset cs)	주어진 문자집합을 이용해 파일의 모든 행을 스트림으로 반환한다.
Stream<Path> list(Path dir)	서브 폴더를 제외한 폴더에 들어 있는 모든 원소를 스트림으로 반환한다.
Stream<Path> walk(Path start)	서브 폴더를 포함한 폴더에 들어 있는 모든 원소를 스트림으로 반환한다.

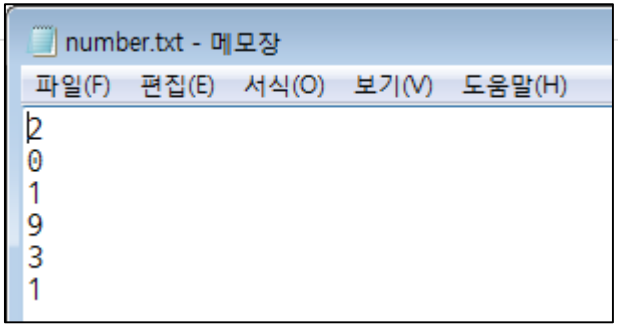
- 예제 : [예제 12-12]

```java
import java.io.File;
import java.io.IOException;
import java.util.Arrays;
import java.util.stream.Stream;

public class Stream1Demo {
    public static void main(String[] args) throws IOException {
        File file = new File("C:\\Windows");
        File[] fs = file.listFiles();

        Stream<File> stream = Arrays.stream(fs);
        long count = stream.filter(x -> x.isDirectory() == false).count();

        System.out.println("C:\\Windows에 있는 파일 개수 : " + count);
    }
}
```

```
C:\Windows에 있는 파일 개수 : 83
```

- 예제 : [예제 12-13]

```java
import java.io.File;
import java.nio.file.Files;
import java.nio.file.Path;
import java.util.stream.Stream;

public class Stream2Demo {
    public static void main(String[] args) throws Exception {
        String[] number = { "zero", "one", "two", "three", "four", "five",
                "six", "seven", "eight", "nine" };
        Path p = new File("C:\\Temp\\number.txt").toPath();

        Stream<String> s = Files.lines(p);

        s.forEach(x -> System.out.println(x));

        s = Files.lines(p);

        s.map(x -> number[Integer.parseInt(x)]).forEach(x -> System.out.print(x + " "));
    }
}
```




```
2
0
1
9
3
1
two zero one nine three one
```