

first Java

1

CHAPTER 12.

JAVA I/O

목차

12.1 JAVA의 입출력

12.2 바이트 기반 스트림

12.3 문자기반 기반스트림

12.4 File 클래스

12.5 인스턴스의 직렬화

12.1 JAVA의 입출력



JAVA의 입출력

- 컴퓨터를 통한 입출력

- 키보드 입력과 모니터 출력
- 하드디스크에 저장되어 있는 파일의 읽기와 저장
- USB와 같은 외부 메모리 장치의 읽기와 저장
- 네트워크로 연결되어 있는 컴퓨터에 있는 자원의 읽기와 저장

12.1 JAVA의 입출력



JAVA의 입출력

- JAVA에서는 위의 예시와 같은 입출력을 다루기 위해 스트림(stream)이라는 흐름의 개념을 통해 데이터의 입출력을 처리.
- 스트림의 특징
 - 스트림은 JAVA 프로그램과 파일 사이에서 데이터의 전송을 도와주는 역할.
 - 데이터의 흐름은 단 방향으로 처리 되기 때문에 JAVA 프로그램에서 파일의 데이터를 읽어올 때와 JAVA 프로그램에서 데이터를 파일로 내보낼 때 각 각의 스트림을 사용해서 처리.



12.1 JAVA의 입출력



JAVA의 입출력

- JAVA에서는 JAVA 프로그램 기준으로 데이터를 읽어오는 스트림을 입력(Input) 스트림, 출력하는(Output) 스트림으로 구분해서 사용.
- 입력과 출력을 줄여서 I/O라고 표현.



12.1 JAVA의 입출력



JAVA Stream의 특징

- 스트림은 FIFO(First In First Out : 먼저 들어간 것이 먼저 나옴) 구조로 되어있어 데이터의 순서가 바뀌지 않고, 순차적인 접근만 허용함.
- 읽기와 쓰기를 동시에 할 수 없어 InputStream과 OutputStream을 필요에 따라 각각 하나씩 생성해서 사용해야 함.
- 사용자의 입력을 기다린다거나 할 때 스레드가 일시 정지 되어 프로그램의 흐름이 지연되는 지연상태가 발생 할 수 있습니다.
- Stream은 파일의 입력과 출력, 네트워크 소켓 통신 (서버와 클라이언트간의 통신), 콘솔의 입출력 등에 활용됨.

12.1 JAVA의 입출력



JAVA Stream의 특징

- JAVA의 입출력 스트림 클래스의 종류

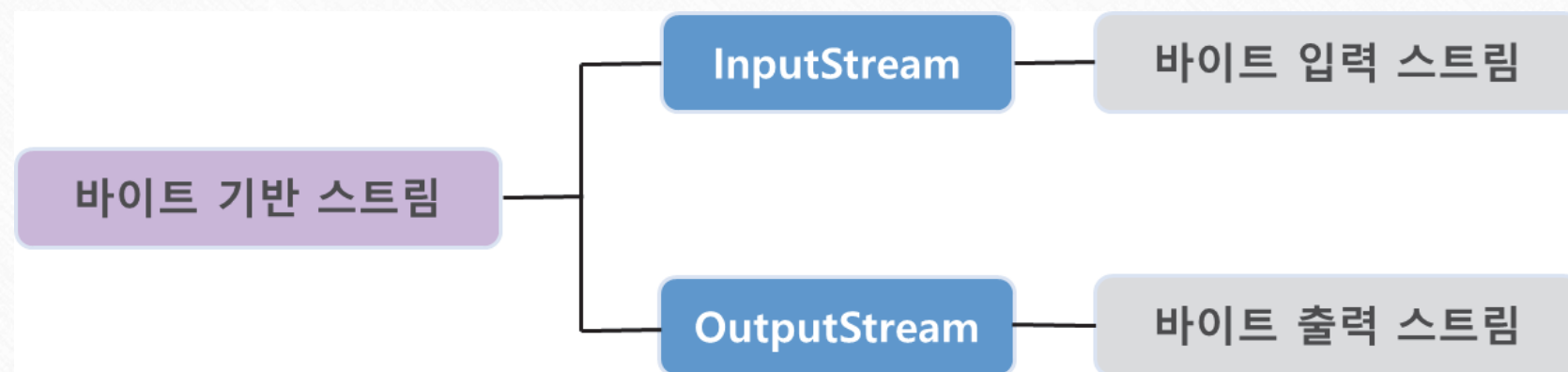
바이트 스트림	바이트 스트림은 1바이트를 입·출력 할 수 있는 스트림으로 일반적으로 바이트로 구성된 이미지 파일이나 동영상 파일의 입출력 처리에 적합합니다.
문자 스트림	유니코드로 된 문자를 입·출력 하는 스트림으로 2바이트를 입·출력 합니다. 유니 코드 단위의 데이터를 사용하기 때문에 세계 모든 언어로 구성된 파일을 입·출력 하기에 적합 합니다.

12.2 바이트 기반 스트림



바이트 기반 스트림

- JAVA에서는 java.io 패키지를 통해 InputStream과 OutputStream 클래스를 제공하고 있고 JAVA에서의 I/O 처리하는 것은 스트림 클래스 타입의 인스턴스를 생성 해서 사용해야 함.



12.2 바이트 기반 스트림



바이트 기반 스트림

- InputStream 클래스와 OutputStream 클래스에 정의된 메소드

클래스	메소드	설명
InputStream	<code>abstract int read()</code>	스트림으로 부터 바이트를 읽어 들입니다.
	<code>int read(byte[] b)</code>	스트림으로 부터 특정 바이트를 읽어 들이고 배열 <code>b</code> 에 저장합니다.
	<code>int read(byte[] b, int off, int len)</code>	스트림으로 부터 바이트 배열 <code>b</code> 를 읽어 들이고 배열 <code>b[off]</code> 부터 <code>len</code> 개의 데이터를 저장합니다.
OutputStream	<code>abstract void write(int b)</code>	스트림에 매개변수의 인자로 받은 바이트 데이터를 저장합니다.
	<code>void write(byte[] b)</code>	스트림에 배열 <code>b</code> 에 저장된 바이트 데이터들을 저장합니다.
	<code>void write(byte[] b, int off, int len)</code>	스트림에 바이트 배열 <code>b</code> 에 저장된 바이트 데이터들 중 <code>b[off]</code> 부터 <code>len</code> 개의 바이트를 저장합니다.

12.2 바이트 기반 스트림

10



바이트 기반 스트림

- 스트림클래스의 메소드들을 사용할 때 주의할 점은 예외처리를 반드시 하도록 되어있기 때문에 API를 잘 확인 한 후 적절한 예외처리가 되어야 함.

read

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

Reads some number of bytes from the input stream and stores them into the buffer array `b`. The number of bytes actually read is returned as an integer. This method blocks until input data is available, end of file is detected, or an exception is thrown.

If the length of `b` is zero, then no bytes are read and 0 is returned; otherwise, there is an attempt to read at least one byte. If no byte is available because the stream is at the end of the file, the value -1 is returned; otherwise, at least one byte is read and stored into `b`.

The first byte read is stored into element `b[0]`, the next one into `b[1]`, and so on. The number of bytes read is, at most, equal to the length of `b`. Let `k` be the number of bytes actually read; these bytes will be stored in elements `b[0]` through `b[k-1]`, leaving elements `b[k]` through `b[b.length-1]` unaffected.

The `read(b)` method for class `InputStream` has the same effect as:

```
read(b, 0, b.length)
```

Parameters:

`b` - the buffer into which the data is read.

Returns:

the total number of bytes read into the buffer, or -1 if there is no more data because the end of the stream has been reached.

Throws:

`IOException` - If the first byte cannot be read for any reason other than the end of the file, if the input stream has been closed, or if some other I/O error occurs.

`NullPointerException` - if `b` is null.

See Also:

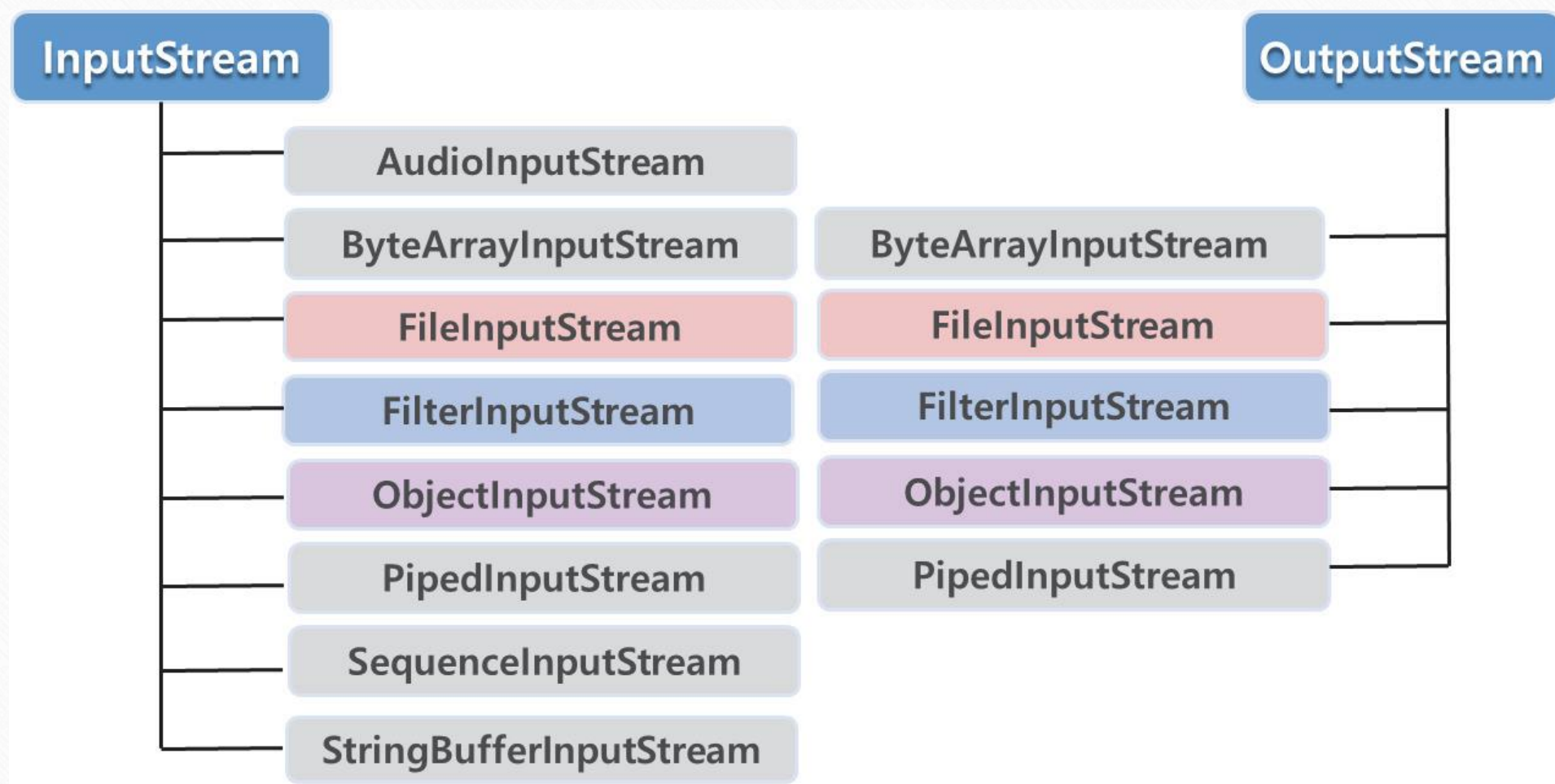
```
read(byte[], int, int)
```


12.2 바이트 기반 스트림



바이트 기반 스트림 클래스

- 스트림은 기본적으로 바이트 단위로 데이터를 전송.
- 바이트 스트림 클래스는 입/출력의 대상에 따라 여러 클래스로 파생되어 다양한 스트림 클래스를 제공하고 있음.



12.2 바이트 기반 스트림



FileInputStream / FileOutputStream

- FileInputStream과 FileOutputStream은 파일을 대상으로 입/출력을 다루는 바이트 타입의 스트림 클래스.
- 기본 스트림 클래스를 상속하고 있고, 상속하는 파일의 바이트 데이터들의 입출력이 가능하도록 스트림 클래스들이 오버라이딩 되어 있음.

```
public class FileOutputStream extends OutputStream
```

```
public class FileInputStream extends InputStream
```


12.2 바이트 기반 스트림



FileInputStream / FileOutputStream

스트림 구분	Method	설명
FileOutputStream	<code>void write(byte[] b)</code>	바이트 배열의 b.length 크기의 바이트 데이터를 해당 파일에 파일 출력 스트림을 통해 씁니다.
	<code>void write(byte[] b, int off, int len)</code>	바이트 배열의 index off 에서 시작하는 바이트 배열의 len 사이즈 크기의 바이트 데이터를 해당 파일에 파일 출력 스트림을 통해 씁니다.
	<code>void write(int b)</code>	매개변수의 인자로 전달된 바이트를 해당 파일에 파일 출력 스트림을 통해 씁니다.
	<code>void close()</code>	파일 출력 스트림을 닫습니다.
FileInputStream	<code>int available()</code>	파일 입력 스트림을 통해서 해당 파일에서 읽을 수 있는 예상 바이트 수를 반환합니다.
	<code>int read()</code>	파일 입력 스트림에서 데이터 바이트를 읽는데 사용됩니다.
	<code>int read(byte[] b)</code>	파일 입력 스트림에서 최대 b.length 바이트의 데이터를 읽는데 사용됩니다.
	<code>int read(byte[] b, int off, int len)</code>	파일 입력 스트림에서 index off 에서 시작해서 len 사이즈의 바이트의 데이터를 읽는데 사용됩니다.

12.2 바이트 기반 스트림

14



FileInputStream / FileOutputStream

```
package chapter12;

import java.io.FileOutputStream;
import java.io.IOException;

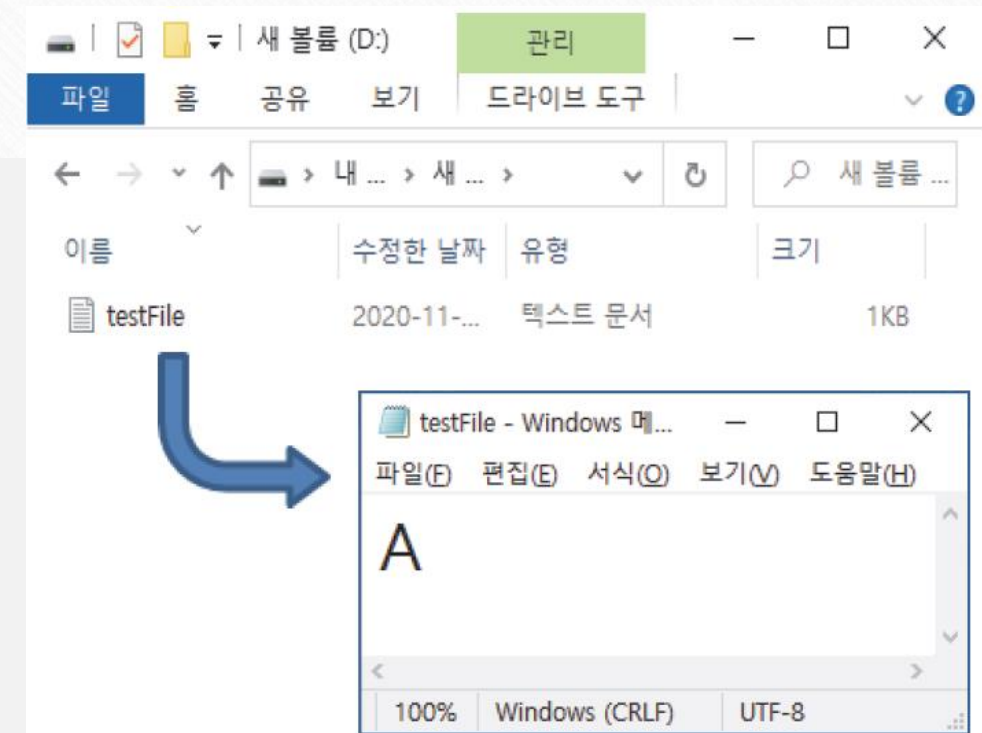
public class FileOutputStreamTest1 {

    public static void main(String[] args) {

        try {
            FileOutputStream out = new FileOutputStream("D:\\testFile.txt"); // 생성자에 파일 경로를
            이용해서 스트림 생성
            out.write(65); // 파일 스트림을 통해 유니코드 65를 쓰기(출력)
            out.close(); // 스트림을 사용후에는 닫아주어야합니다.
            System.out.println("파일에 데이터 쓰기 성공 !");
        } catch (IOException e) {
            System.out.println(e);
        }

    }

}
```



결과(Console)

파일에 데이터 쓰기 성공 !

12.2 바이트 기반 스트림

15



FileInputStream / FileOutputStream

```
package chapter12;

import java.io.FileOutputStream;
import java.io.IOException;

public class FileOutputStreamTest2 {

    public static void main(String[] args) {

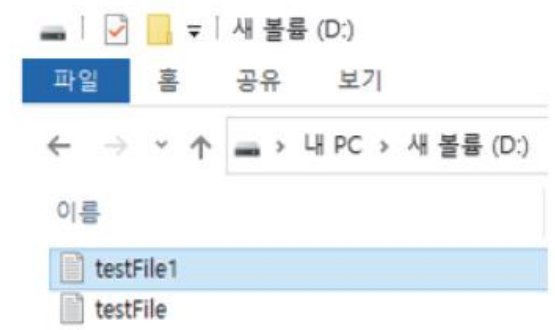
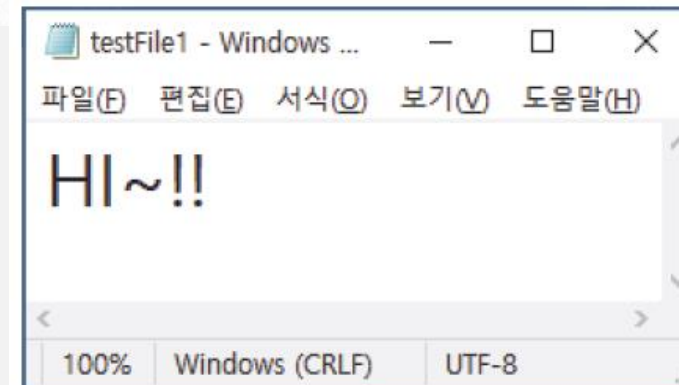
        try {
            FileOutputStream out = new FileOutputStream("D:\\testFile1.txt");

            String s = "HI~!!";
            byte b[] = s.getBytes(); // 문자열을 byte타입의 배열로 변환
            out.write(b);

            System.out.println("파일에 데이터 쓰기 성공 !");

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

    }
}
```



결과(Console)

파일에 데이터 쓰기 성공 !

12.2 바이트 기반 스트림

16



FileInputStream / FileOutputStream

```
package chapter12;

import java.io.FileInputStream;
import java.io.IOException;

public class FileInputStreamTest1 {

    public static void main(String[] args) {

        try {
            FileInputStream in = new FileInputStream("D:\\testFile.txt"); // 생성자에 파일 경로를
            //이동해서 스트림 생성
            int i = in.read(); // read() 메소드를 이용해서 바이트 단위의 데이터를 읽어옵니다.
            System.out.println(i);
            System.out.println((char) i);
            in.close();
            System.out.println("파일 데이터를 모두 읽었습니다.");
        } catch (IOException e) {
            System.out.println(e);
        }
    }
}
```

결과(Console)

65

A

파일 데이터를 모두 읽었습니다.

12.2 바이트 기반 스트림

17



FileInputStream / FileOutputStream

```
package chapter12;

import java.io.FileInputStream;
import java.io.IOException;

public class FileInputStreamTest2 {
    public static void main(String[] args) {
        try {
            FileInputStream in = new FileInputStream("D:\\testFile1.txt");
            int i = 0;
            while (true) {
                i = in.read(); // 읽어올 데이터가 없으면 -1을 반환합니다.
                if(i == -1) { // 읽어올 데이터가 없으면 반복문을 종료 시킵니다.
                    break;
                }
                System.out.print((char) i);
            }
            in.close();
            System.out.println("파일 데이터를 모두 읽었습니다.");
        } catch (IOException e) {
            System.out.println(e);
        }
    }
}
```

결과(Console)

HI~!!
파일 데이터를 모두 읽었습니다.

12.2 바이트 기반 스트림

18



FileInputStream / FileOutputStream

```
package chapter12;

import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;

public class FileTransferTest1 {

    public static void main(String[] args) {
        try {

            InputStream in = new FileInputStream("D:\\testFile1.txt"); // 복사할 원본 데이터
            OutputStream out = new FileOutputStream("D:\\testFile1copy.txt"); // 복사해서 옮길 위치의
스트림 객체 생성

            int byteData = 0; // 읽어올 데이터를 저장할 변수
```


12.2 바이트 기반 스트림

19



FileInputStream / FileOutputStream

```
while (true) {  
    byteData = in.read();  
    if (byteData == -1) {  
        break;  
    }  
    out.write(byteData);  
}  
in.close();  
out.close();  
System.out.println("파일 복사가 완료되었습니다!");  
} catch (IOException e) {  
    System.out.println(e);  
}  
}
```

결과(Console)

파일 복사가 완료되었습니다!

이름	수정한 날짜	유형	크기
testFile1copy	2020-11-...	텍스트 문서	1KB
testFile	2020-11-...	텍스트 문서	1KB
testFile1	2020-11-...	텍스트 문서	1KB

testFile1copy - Windo...

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

HI~!!

100% Windows (CRLF) UTF-8

12.2 바이트 기반 스트림



FileInputStream / FileOutputStream

```
package chapter12;

import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;

public class FileTransferTest2 {

    public static void main(String[] args) {
        try {
            InputStream in = new FileInputStream("D:\\test.zip");
            OutputStream out = new FileOutputStream("D:\\testcopy.zip");

            int copyByte = 0;
            int byteDataSize = 0; // 복사한 파일의 사이즈

            byte[] bufData = new byte[1024*2]; //2kb사이즈의 배열 생성, 배열에 2kb 데이터를 저장해서 복사
```


12.2 바이트 기반 스트림

21



FileInputStream / FileOutputStream

```
while (true) {
    byteDataSize = in.read(bufData);
    if (byteDataSize == -1) {
        break;
    }
    out.write(bufData, 0, byteDataSize);
    copyByte += byteDataSize; // 복사한 바이트 사이즈 증가 연산
}
in.close();
out.close();
System.out.println(copyByte + " byte 파일 복사가 완료되었습니다!");
} catch (IOException e) {
    System.out.println(e);
}
}
```

결과(Console)

1006319469 byte 파일 복사가 완료되었습니다!

이름	수정된 날짜	유형
test	2020-11-...	파일 폴더
test	2020-08-...	압축(ZIP) 파일
testcopy	2020-11-...	압축(ZIP) 파일

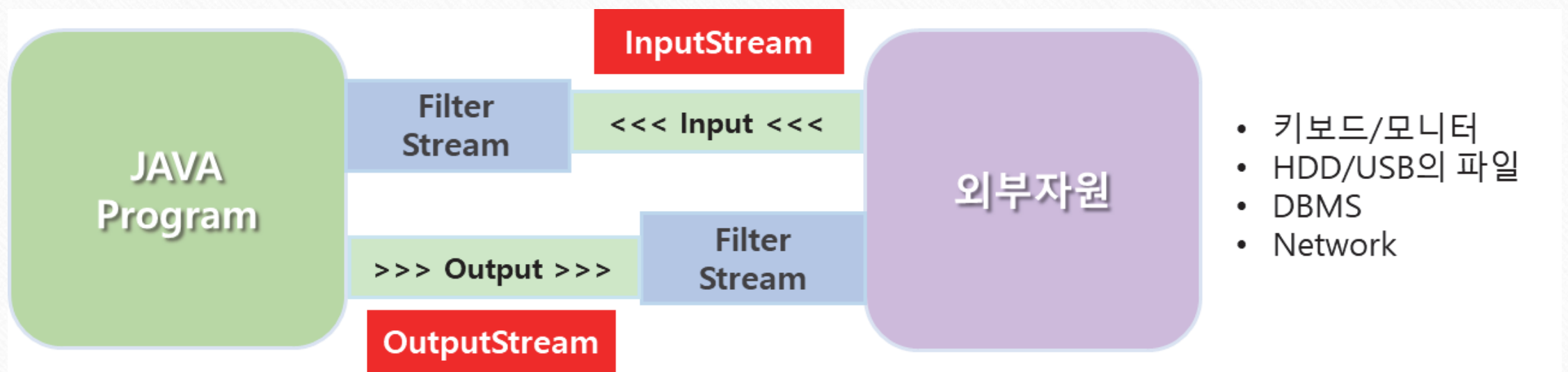
12.2 바이트 기반 스트림

22



BufferedInputStream / BufferedOutputStream 필터 스트림

- JAVA에서 제공하는 필터 스트림은 실제로 데이터를 주고받을 수는 없지만, 다른 스트림의 기능을 향상시키거나 새로운 기능을 추가해 주는 스트림.
- 직접 데이터를 읽거나 출력을 하지 못하지만 기본 스트림에 보조적인 기능을 추가하는 스트림이기 때문에 보조 스트림이라고도 함.
- 필터 스트림은 기본 스트림의 인스턴스를 생성자의 매개변수 인자로 받아 인스턴스를 생성해야 하는 특징을 가지고 있음.



12.2 바이트 기반 스트림

23



BufferedInputStream / BufferedOutputStream 필터 스트림

- 바이트기반 스트림의 필터 스트림

입력 스트림	출력 스트림	설명
FilterInputStream	FilterOutputStream	필터를 이용한 입출력
BufferedInputStream	BufferedOutputStream	버퍼를 이용한 입출력
DataInputStream	DataOutputStream	입출력 스트림으로부터 JAVA의 기본 타입으로 데이터를 읽어올 수 있게 함.
ObjectInputStream	ObjectOutputStream	데이터를 객체 단위로 읽거나, 읽어 들인 객체를 역 직렬화시킴.
SequenceInputStream	X	두 개의 입력 스트림을 논리적으로 연결함.
PushbackInputStream	X	다른 입력 스트림에 버퍼를 이용하여 push back이나 unread와 같은 기능을 추가함.
X	PrintStream	다른 출력 스트림에 버퍼를 이용하여 다양한 데이터를 출력하기 위한 기능을 추가함.

12.2 바이트 기반 스트림



BufferedInputStream / BufferedOutputStream 클래스

- BufferedInputStream / BufferedOutputStream은 파일 입/출력의 성능을 향상시키기 위한 필터 스트림.
- 이 필터 스트림은 FilterInputStream과 FilterOutputStream을 상속해서 파생한 스트림 클래스.

```
public class BufferedOutputStream extends FilterOutputStream
```

```
public class BufferedInputStream extends FilterInputStream
```


12.2 바이트 기반 스트림



BufferedInputStream / BufferedOutputStream 클래스

- 필터 스트림의 생성자

스트림 클래스	생성자
BufferedOutputStream	BufferedOutputStream(OutputStream os)
	BufferedOutputStream(OutputStream os, int size)
BufferedInputStream	BufferedInputStream(InputStream IS)
	BufferedInputStream(InputStream IS, int size)

12.2 바이트 기반 스트림



BufferedInputStream / BufferedOutputStream 클래스

- 필터 스트림의 메소드

스트림 클래스	메소드	설명
BufferedOutputStream	<code>void write(int b)</code>	전달된 바이트 데이터를 지정된 파일에 출력 스트림을 이용해 씁니다.
	<code>void write(byte[] b, int off, int len)</code>	바이트 배열의 <code>index off</code> 에서 시작 하여 <code>len</code> 사이즈의 바이트 데이터를 지정된 파일에 출력 스트림을 이용해 씁니다.
	<code>void flush()</code>	스트림의 버퍼에 버퍼링된 데이터를 비웁니다.
BufferedInputStream	<code>int available()</code>	입력 스트림에서 읽을 수 있는 데이터의 예상 바이트 수를 반환합니다.
	<code>int read()</code>	입력 스트림에서 데이터의 다음 바이트 데이터를 읽습니다.
	<code>int read(byte[] b, int off, int ln)</code>	입력 스트림으로 부터 바이트 배열 <code>b</code> 를 읽어 들이고 배열 <code>b[off]</code> 부터 <code>len</code> 개의 데이터를 저장합니다.
	<code>void close()</code>	입력 스트림을 닫습니다.

12.2 바이트 기반 스트림

27



BufferedInputStream / BufferedOutputStream 클래스

```
package chapter12;

import java.io.BufferedInputStream;
import java.io.BufferedOutputStream;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;

public class FileTransferBufferedFilterTest {

    public static void main(String[] args) {
        try {
            InputStream in = new FileInputStream("D:WWtest.zip");
            BufferedInputStream bin = new BufferedInputStream(in); // 기본 스트림으로 필터스트림을 생성

            OutputStream out = new FileOutputStream("D:WWtestcopy2.zip");
            BufferedOutputStream bout = new BufferedOutputStream(out); // 기본 스트림으로 필터스트림을 생성

            int byteData = 0;
```

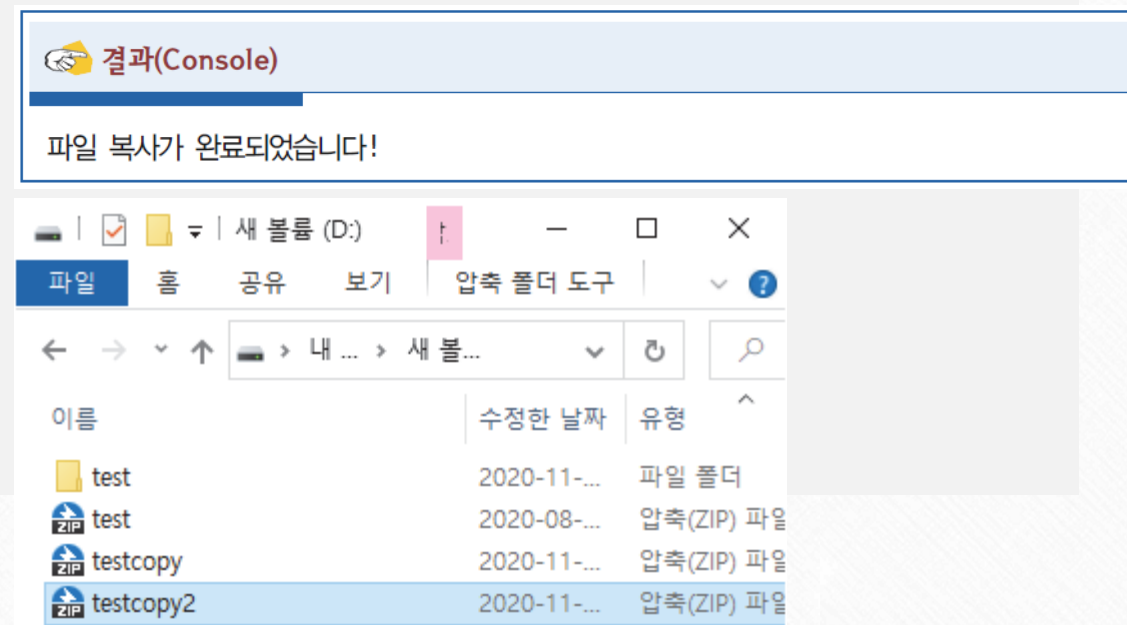
12.2 바이트 기반 스트림

28



BufferedInputStream / BufferedOutputStream 클래스

```
while (true) {  
    byteData = bin.read();  
    if (byteData == -1) {  
        break;  
    }  
    bout.write(byteData);  
}  
  
in.close();  
out.close();  
System.out.println("파일 복사가 완료되었습니다!");  
  
} catch (IOException e) {  
    System.out.println(e);  
}  
  
}
```



12.3 문자기반 기반스트림



문자기반 기반스트림

- 바이트 스트림은 문자를 파일에 쓰고 저장하는 처리도 가능하지만 다른 프로그램을 이용하거나 다른 운영체제에서 처리할 때에는 문제가 발생할 수 있음.
- 운영체제마다 문자를 표현 하는 방식이 다르기 때문에 다른 운영체제에서 만든 파일인 경우 처리가 안 될 수 있음.
- JAVA에서는 이러한 문제를 해결할 수 있도록 유니코드로 된 2바이트 문자 데이터를 입출력 할 수 있는 문자 기반의 스트림을 제공.
- 문자 스트림은 이미지, 동영상과 같은 바이너리 데이터의 입출력은 되지 않고 문자타입만 입출력이 가능.
- 이미지나 동영상 파일의 입출력을 해야 한다면 바이트 타입의 입출력 스트림을 이용.

12.3 문자기반 기반스트림



문자기반 기반스트림

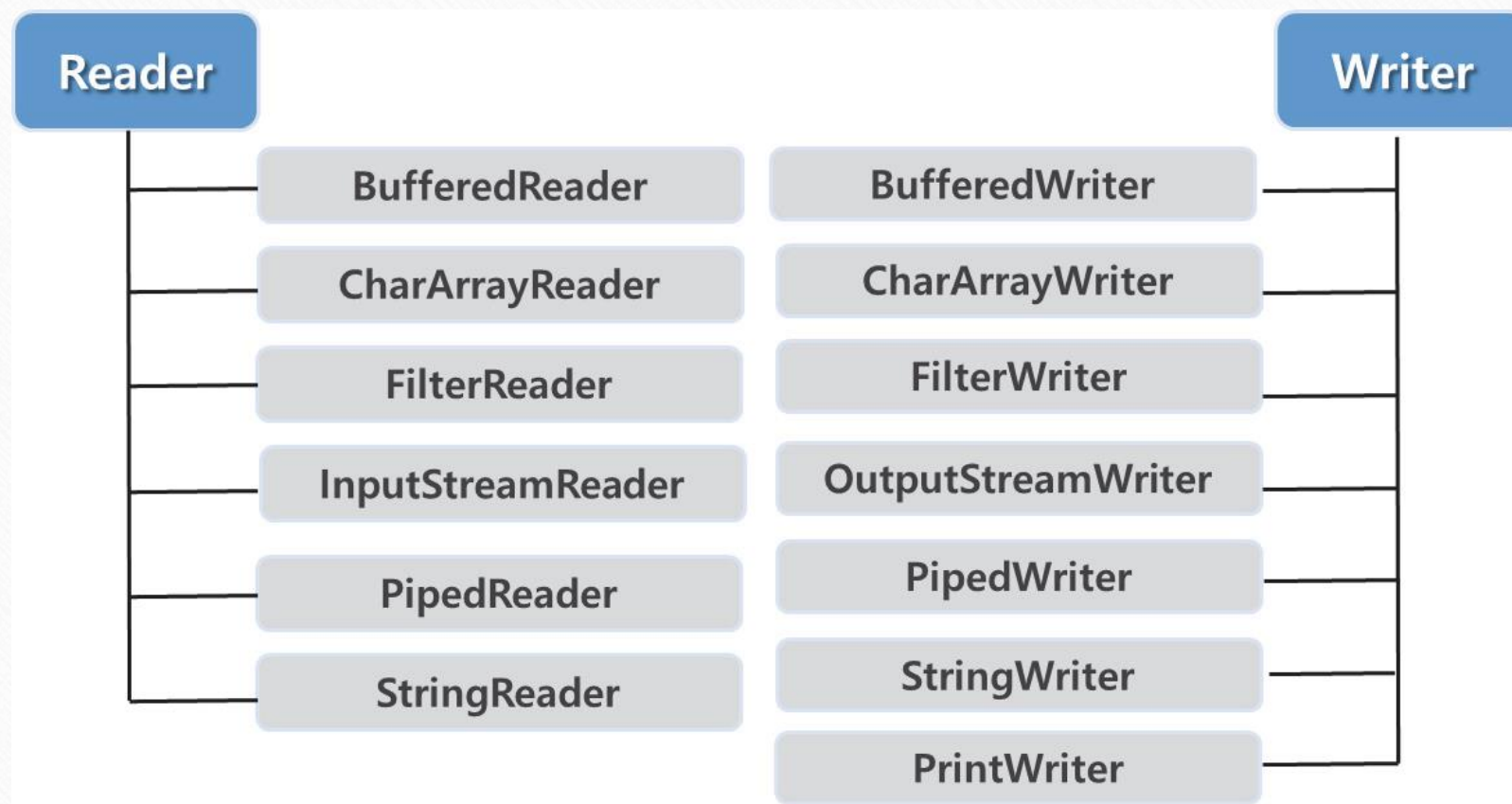
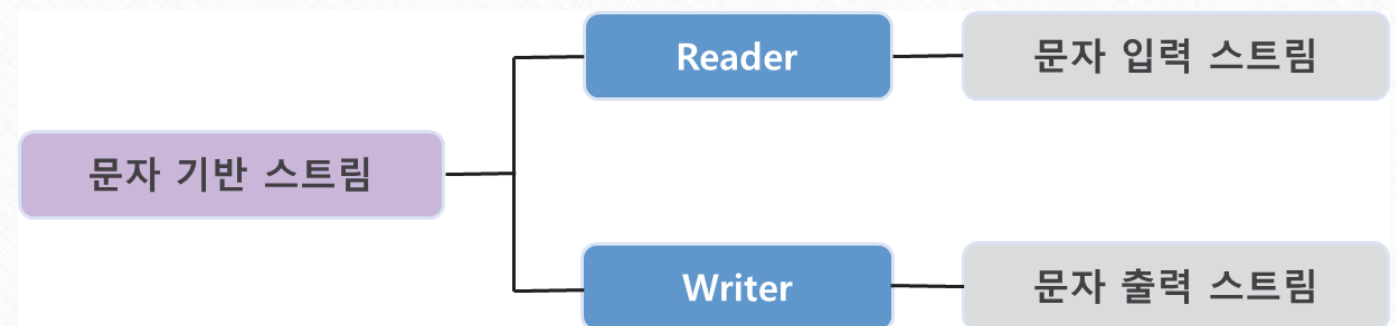
- 바이트 스트림은 다른 프로그램을 이용하거나 다른 운영체제에서 처리할 때에는 문제가 발생할 수 있음.
- 운영체제마다 문자를 표현 하는 방식이 다르기 때문에 다른 운영체제에서 만든 파일인 경우 처리가 안 될 수 있음.
- JAVA에서는 이러한 문제를 해결할 수 있도록 유니코드로 된 2바이트 문자 데이터를 입출력 할 수 있는 문자 기반의 스트림을 제공.
- 문자 스트림은 이미지, 동영상과 같은 바이너리 데이터의 입출력은 되지 않고 문자타입만 입출력이 가능.
- 이미지나 동영상 파일의 입출력을 해야 한다면 바이트 타입의 입출력 스트림을 이용.

12.3 문자기반 기반스트림



문자기반 기반스트림

- 문자를 다루는 스트림



12.3 문자기반 기반스트림



문자기반 기반스트림

- 메소드

클래스	메소드	설명
Reader	<code>abstract int read()</code>	스트림으로 부터 데이터를 읽어 들이도록 메소드를 오버라이딩 할 추상 메소드입니다.
	<code>int read()</code>	스트림으로 부터 2바이트 데이터를 읽어 들입니다.
	<code>int read(char[] cbuf)</code>	스트림으로 부터 <code>char</code> 타입의 배열에 담아 읽어옵니다.
	<code>int read(char[] cbuf, int off, int len)</code>	스트림으로 부터 <code>char</code> 타입의 배열 <code>cbuf</code> 를 읽어 들고 배열 <code>cbuf[off]</code> 부터 <code>len</code> 개의 데이터를 읽어옵니다.
	<code>int read(CharBuffer target)</code>	<code>CharBuffer</code> 타입의 인스턴스에 담아 읽어옵니다.
	<code>abstract void close()</code>	스트림을 종료합니다.

12.3 문자기반 기반스트림



문자기반 기반스트림

Writer	<code>Writer append(char c)</code>	문자 데이터 <code>c</code> 를 추가하여 작성합니다.
	<code>Writer append(CharSequence csq)</code>	문자열 데이터 <code>csq</code> 를 추가하여 작성합니다.
	<code>Writer append(CharSequence csq, int start, int end)</code>	문자열 <code>csq</code> 의 <code>start index</code> 에서 <code>end index</code> 까지 의 문자열을 잘라 추가하여 작성합니다.
	<code>abstract void close()</code>	스트림을 종료합니다.
	<code>abstract void flush()</code>	스트림이 가지고 있는 버퍼들을 모두 플러시 합니다.
	<code>void write(char[] cbuf)</code>	<code>char</code> 타입의 배열에 출력 데이터를 저장하고 저장된 데이터 를 대상 파일에 출력합니다.
	<code>abstract void write(char[] cbuf, int off, int len)</code>	<code>char</code> 타입의 배열에 출력 데이터를 저장하고 저장된 배열 <code>cbuf[off]</code> 부터 <code>len</code> 개의 데이터를 대상 파일에 출력합니다.
	<code>void write(int c)</code>	대상 파일에 2바이트 데이터를 출력합니다.
	<code>void write(String str)</code>	대상 파일에 문자열을 출력합니다.
	<code>void write(String str, int off, int len)</code>	문자열 <code>str</code> 의 <code>off</code> 인덱스부터 <code>len</code> 개의 문자열을 잘라 을 대상 파일에 출력합니다.

12.3 문자기반 기반스트림



FileReader / FileWriter

- FileReader / FileWriter 클래스는 파일에 문자열을 입출력 하는데 사용되는 스트림 클래스.
- FileReader는 InputStreamReader를 상속하고 있고, FileWriter 클래스는 OutputStreamWriter를 상속하고 있음.

```
public class FileReader extends InputStreamReader
```

```
public class FileWriter extends OutputStreamWriter
```


12.3 문자기반 기반스트림



FileReader / FileWriter

- FileReader / FileWriter 클래스의 생성자

스트림 클래스	생성자
FileReader	FileReader(String file)
	FileReader(File file)
FileWriter	FileWriter(String file)
	FileWriter(File file)

12.3 문자기반 기반스트림



FileReader / FileWriter

```
package chapter12;

import java.io.FileWriter;
import java.io.IOException;
import java.io.Writer;

public class FileWriterStreamTest {

    public static void main(String[] args) {

        String str = "StringWn";
        char ch = 'A';
        char[] charArr = { 'B', 'C' };

        Writer out = null;
```


12.3 문자기반 기반스트림

37

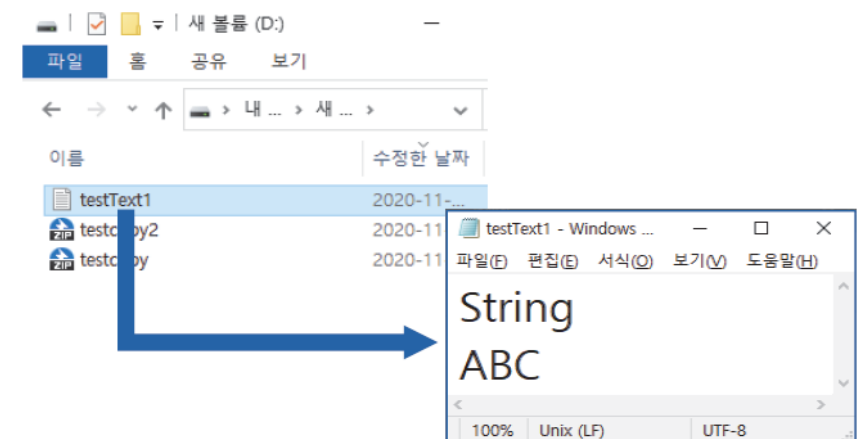


FileReader / FileWriter

```
try {  
    out = new FileWriter("d:\\testText1.txt");  
  
    out.write(str); // 문자 열을 쓸 수 있습니다.  
    out.write(ch); // 문자를 쓸 수 있습니다.  
    out.write(charArr); // char 타입의 배열을 쓸 수 있습니다.  
  
    out.close();  
    System.out.println("파일에 문자들을 출력했습니다.");  
  
} catch (IOException e) {  
    e.printStackTrace();  
}  
  
}
```

결과(Console)

파일에 문자들을 출력했습니다.



12.3 문자기반 기반스트림



FileReader / FileWriter

```
package chapter12;

import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.io.Reader;

public class FileReaderStreamTest {

    public static void main(String[] args) {

        char[] cbuf = new char[10]; // 최대 10개의 문자 읽어 저장
        int readCnt=0;
        Reader reader;
```


12.3 문자기반 기반스트림




FileReader / FileWriter

```
try {
    reader = new FileReader("d:\\testText1.txt");

    readCnt = reader.read(cbuf, 0, cbuf.length);
    for (int i = 0; i < readCnt; i++)
        System.out.print(cbuf[i]);
    reader.close();

} catch (FileNotFoundException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
}

}
```

 결과(Console)

String
ABC

12.3 문자기반 기반스트림



BufferedReader / BufferedWriter

- BufferedReader / BufferedWriter 입출력 효율을 높이기 위해 버퍼(char[])를 사용하는 문자 스트림.
- 라인(line)단위의 입출력이 편리함.
- BufferedReader는 Reader 클래스를 상속하고, BufferedWriter는 Writer 클래스를 상속.

```
public class BufferedReader extends Reader
```

```
public class BufferedWriter extends Writer
```


12.3 문자기반 기반스트림



BufferedReader / BufferedWriter

- BufferedReader / BufferedWriter 클래스의 생성자

스트림 클래스	생성자
BufferedReader	BufferedReader(Reader in)
	BufferedReader(Reader in, int sz)
BufferedWriter	BufferedWriter(Writer out)
	BufferedWriter(Writer out, int sz)

12.3 문자기반 기반스트림



BufferedReader / BufferedWriter

- BufferedReader / BufferedWriter 메소드

스트림 클래스	메소드	설명
BufferedWriter	<code>void newLine()</code>	줄 바꿈, 즉 개행을 합니다.
	<code>void write(int c)</code>	문자를 씁니다.
	<code>void write(char[] cbuf, int off, int len)</code>	문자 타입의 배열에서 off 인덱스 위치에서 len 만큼의 문자를 씁니다.
	<code>void write(String s, int off, int len)</code>	문자열에서 off 인덱스 위치에서 len 만큼의 문자를 씁니다.
	<code>void flush()</code>	문자를 저장하는 버퍼를 비웁니다.
	<code>void close()</code>	스트림을 닫습니다.
BufferedReader	<code>int read()</code>	파일에 있는 문자 하나를 읽어옵니다.
	<code>int read(char[] cbuf, int off, int len)</code>	문자 타입의 배열에서 off 인덱스 위치에서 len 만큼의 문자를 읽어옵니다.
	<code>String readLine()</code>	한 줄의 문자열을 읽어옵니다.
	<code>void close()</code>	스트림을 닫습니다.

12.3 문자기반 기반스트림



BufferedReader / BufferedWriter

```
package chapter12;

import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;

public class BufferedWriterTest {

    public static void main(String[] args) {

        BufferedWriter out = null;

        try {

            out = new BufferedWriter(new FileWriter("d:WWtestText2.txt")); // 기본 스트림을 통해 객체를 생성

            out.write("홍차(紅,茶)는 차잎 내부의 성분이 자체에 들어있는 효소에");
            out.write(" 산화되어 붉은 빛을 띠는 차를 뜻한다.");
            out.newLine();
            out.write("녹차나 보이차와 같이 효소의 작용을 중지시키는 ");
            out.write(" 쇠청(曬靑, 햇볕에 쪼여 말림)");
            out.write(" 과정을 거치지 않기 때문에 잎 자체의 효소로 산화가 된 것이다.");
```

12.3 문자기반 기반스트림

44



BufferedReader / BufferedWriter

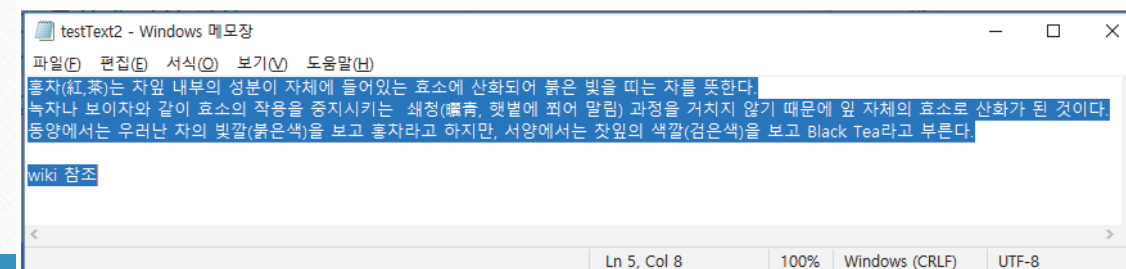
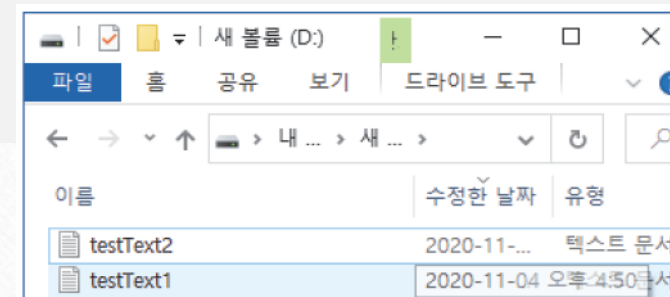
```
out.newLine();
out.write("동양에서는 우려난 차의 빛깔(붉은색)을 보고 홍차라고 하지만, ");
out.write("서양에서는 찻잎의 색깔(검은색)을 보고 Black Tea라고 부른다.");
out.newLine();
out.newLine();
out.write("wiki 참조");
out.close();
System.out.println("문자열 입력 완료.");

} catch (IOException e) {
    e.printStackTrace();
}

}
```

결과(Console)

문자열 입력 완료.



12.3 문자기반 기반스트림



BufferedReader / BufferedWriter

```
package chapter12;

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

public class BufferedReaderTest {

    public static void main(String[] args) {

        BufferedReader in = null;
        String str = null;

        try {
            in = new BufferedReader(new FileReader("d:\\test\\test2.txt")); // 기본 스트림을 통해 객체를
```

생성

12.3 문자기반 기반스트림

46



BufferedReader / BufferedWriter

```
while (true) {
    str = in.readLine();
    if (str == null)
        break;
    System.out.println(str);
}
in.close();

} catch (FileNotFoundException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
}

}
```

결과(Console)

홍차(紅, 茶)는 차잎 내부의 성분이 자체에 들어있는 효소에 산화되어 붉은 빛을 띠는 차를 뜻한다. 녹차나 보이차와 같이 효소의 작용을 중지시키는 쐐청(曬靑, 햇볕에 쬔어 말림) 과정을 거치지 않기 때문에 잎 자체의 효소로 산화가 된 것이다. 동양에서는 우려난 차의 빛깔(붉은색)을 보고 홍차라고 하지만, 서양에서는 찻잎의 색깔(검은색)을 보고 Black Tea라고 부른다.

wiki 참조

12.4 File 클래스



File 클래스

- File 클래스는 파일의 데이터를 다루는 것이 아니라 파일의 경로명을 다루는 클래스.
- File 클래스를 이용하면 파일의 이름 변경, 파일 삭제, 디렉터리 생성과 같은 디렉터리와 파일을 다룰 수 있음.
- 파일이 가지고 있는 데이터가 아닌 파일 자체가 가지는 경로가 처리의 대상.
- File 클래스의 생성자

변수	설명
<code>static String pathSeparator</code>	경로 구분자 문자를 저장하는 변수입니다. 윈도우: "\", 리눅스/유닉스 계열 : "/"
<code>static char pathSeparatorChar</code>	
<code>static String separator</code>	파일간의 구분자 문자를 저장하는 변수입니다. 윈도우: ";", 리눅스 : ":"
<code>static char separatorChar</code>	

12.4 File 클래스

File 클래스

- File 클래스의 클래스 변수

변수	설명
<code>static String pathSeparator</code>	경로 구분자 문자를 저장하는 변수입니다. 윈도우: "\", 리눅스/유닉스 계열 : "/"
<code>static char pathSeparatorChar</code>	
<code>static String separator</code>	파일간의 구분자 문자를 저장하는 변수입니다. 윈도우: ";", 리눅스 : ":"
<code>static char separatorChar</code>	

12.4 File 클래스

49



File 클래스

- File 클래스의 메소드

메소드	설명
static File createTempFile(String prefix, String suffix)	default temporary-file 폴더에 File 인스턴스가 가지는 파일 이름에 prefix와 suffix를 붙여 임시파일을 생성합니다.
boolean createNewFile()	File 인스턴스가 가지는 파일 이름을 가진 파일이 존재하지 않으면 인스턴스가 저장하고 있는 이름으로 새로운 파일을 생성합니다.
boolean isAbsolute()	File 인스턴스가 가지는 파일 경로 이름이 절대 경로인지 테스트합니다.
boolean isDirectory()	File 인스턴스가 가지는 경로가 폴더인지 여부를 테스트합니다.
boolean isFile()	File 인스턴스가 가지는 경로가 일반 파일인지 여부를 테스트합니다.
String getName()	File 인스턴스가 가지는 경로의 파일 또는 폴더의 이름을 반환합니다.
String getParent()	File 인스턴스가 가지는 경로의 부모의 경로명 문자열을 반환합니다.
Path toPath()	File 인스턴스가 가지는 경로의 생성 된 Path 객체를 반환합니다.
URI toURI()	File 인스턴스가 가지는 경로의 이름을 나타내는 URI 파일을 반환합니다.
File[] listFiles()	File 인스턴스가 가지는 경로의 디렉토리의 파일 또는 폴더를 나타내는 이름의 배열을 반환합니다.
boolean mkdir()	File 인스턴스가 가지는 경로의 이름으로 폴더를 생성합니다.

12.4 File 클래스



File 클래스

```
package chapter12;

import java.io.File;

public class FileTest {

    public static void main(String[] args) {

        File originFile = new File("d:\\\\test.zip"); // 경로를 이용해서 file객체 생성

        if (!originFile.exists()) {
            System.out.println("원본파일이 존재하지 않아 프로그램을 종료합니다.");
            return;
        }

        File newDir = new File("d:\\\\backup");
        if (!newDir.exists()) {
            System.out.println("새로운 디렉터리를 생성합니다.");
            newDir.mkdir();
        }
    }
}
```


12.4 File 클래스

51

File 클래스

```
System.out.println("d:// 폴더안의 파일 리스트 =====");
File myDir = new File("d:WW");
File[] list = myDir.listFiles();
for (int i = 0; i < list.length; i++) {

    if (list[i].isDirectory()) {
        System.out.print("[DIR] ");
    } else if (list[i].isFile()) {
        System.out.print("[FILE] ");
    }
    System.out.println(list[i].getName());
}
System.out.println("=====");
System.out.println("파일을 backup 디렉터리로 이동시킵니다.");
for (int i = 0; i < list.length; i++) {
    if (list[i].isFile()) {
        File newFile = new File(newDir, list[i].getName());
        list[i].renameTo(newFile);
        System.out.println("> " + list[i].getName() + "이 동했습니다.");
    }
}
System.out.println(">>>>> 모든파일이 backup 디렉터리로 이동했습니다.");
```

12.4 File 클래스

File 클래스

```
System.out.println("d:\\backup\\폴더안의 파일 리스트 ==");
System.out.println("backup 디렉터를 확인합니다.");
File[] newList = newDir.listFiles();
for (int i = 0; i < newList.length; i++) {

    if (newList[i].isDirectory()) {
        System.out.print("[DIR] ");
    } else if (newList[i].isFile()) {
        System.out.print("[FILE] ");
    }
    System.out.println(newList[i].getName());
}
}
```

결과(Console)

새로운 디렉터를 생성합니다.

d:// 폴더안의 파일 리스트 =====

```
[DIR] $RECYCLE.BIN
[DIR] backup
[DIR] System Volume Information
[DIR] test
[FILE] test.zip
[FILE] testcopy.zip
[FILE] testcopy2.zip
[FILE] testFile.txt
[FILE] testFile1.txt
[FILE] testFile1copy.txt
[FILE] testFile1copy2.txt
[FILE] testText1.txt
[FILE] testText2.txt
=====
```

파일을 backup 디렉터리로 이동시킵니다.

```
> test.zip이 동했습니다.
> testcopy.zip이 동했습니다.
> testcopy2.zip이 동했습니다.
> testFile.txt이 동했습니다.
> testFile1.txt이 동했습니다.
> testFile1copy.txt이 동했습니다.
> testFile1copy2.txt이 동했습니다.
> testText1.txt이 동했습니다.
> testText2.txt이 동했습니다.
>>>>> 모든파일이 backup 디렉터리로 이동했습니다.
```

d:\\backup\\ 폴더안의 파일 리스트 =====

backup 디렉터를 확인합니다.

```
[FILE] test.zip
[FILE] testcopy.zip
[FILE] testcopy2.zip
[FILE] testFile.txt
[FILE] testFile1.txt
[FILE] testFile1copy.txt
[FILE] testFile1copy2.txt
[FILE] testText1.txt
[FILE] testText2.txt
```


12.4 File 클래스



File 클래스

결과(Console)

새로운 디렉터리를 생성합니다.

d:// 폴더안의 파일 리스트 =====

```
[DIR] $RECYCLE.BIN
[DIR] backup
[DIR] System Volume Information
[DIR] test
[FILE] test.zip
[FILE] testcopy.zip
[FILE] testcopy2.zip
[FILE] testFile.txt
[FILE] testFile1.txt
[FILE] testFile1copy.txt
[FILE] testFile1copy2.txt
[FILE] testText1.txt
[FILE] testText2.txt
```

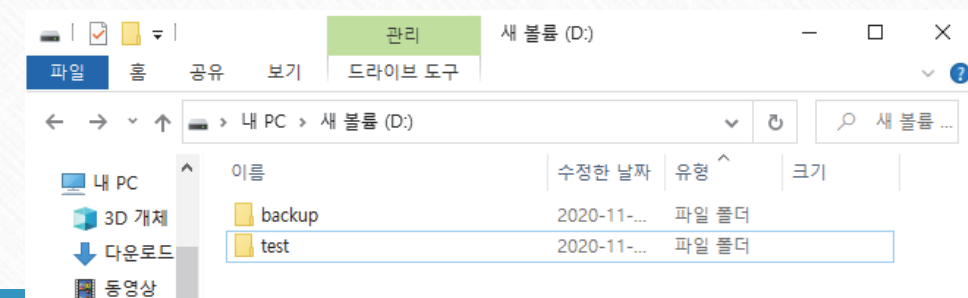
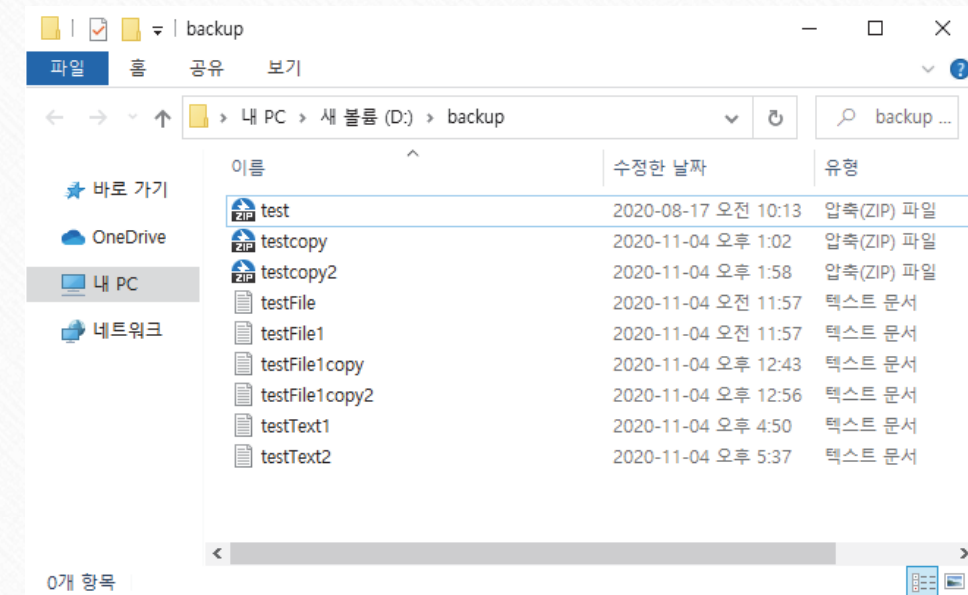
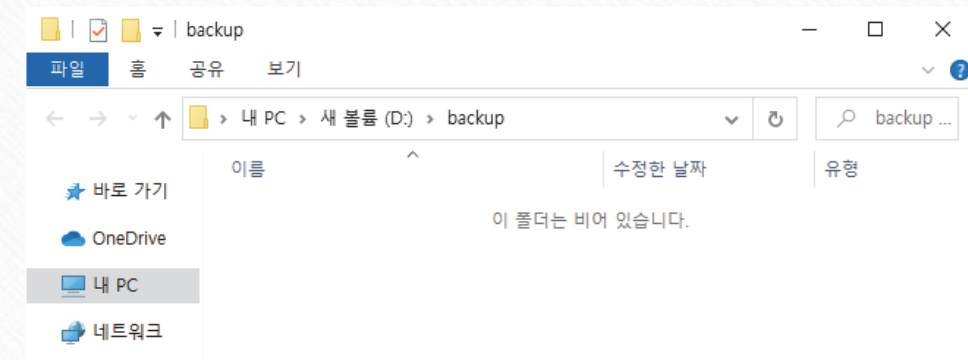
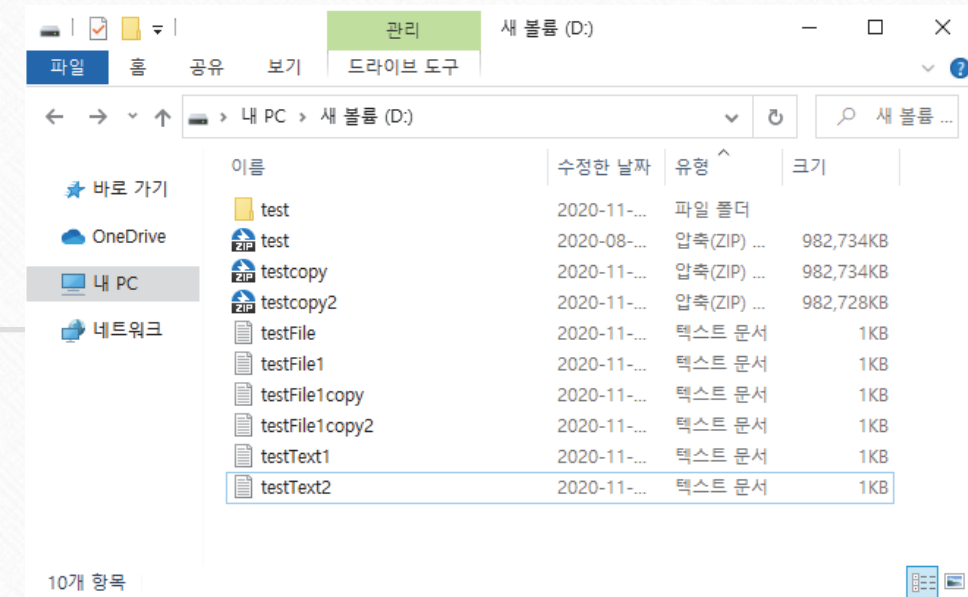
파일을 backup 디렉터리로 이동시킵니다.

```
> test.zip이 동했습니다.
> testcopy.zip이 동했습니다.
> testcopy2.zip이 동했습니다.
> testFile.txt이 동했습니다.
> testFile1.txt이 동했습니다.
> testFile1copy.txt이 동했습니다.
> testFile1copy2.txt이 동했습니다.
> testText1.txt이 동했습니다.
> testText2.txt이 동했습니다.
>>>> 모든파일이 backup 디렉터리로 이동했습니다.
```

d:\backup\ 폴더안의 파일 리스트 =====

backup 디렉터리를 확인합니다.

```
[FILE] test.zip
[FILE] testcopy.zip
[FILE] testcopy2.zip
[FILE] testFile.txt
[FILE] testFile1.txt
[FILE] testFile1copy.txt
[FILE] testFile1copy2.txt
[FILE] testText1.txt
[FILE] testText2.txt
```



12.5 인스턴스의 직렬화

54



인스턴스의 직렬화

- JAVA에서의 직렬화는 바이트기반 스트림으로 인스턴스를 파일로 저장하는 것을 의미하고 파일에서 데이터를 읽어 인스턴스로 만드는 반대 처리를 역 직렬화라고 함.
- 직렬화의 특징은 플랫폼이 독립적이라는 것인데 현재 플랫폼에서 직렬화 하고 다른 플랫폼에서 역 직렬화가 가능해서 네트워크에서 인스턴스 데이터를 전달하는데 사용됨.
- 직렬화를 하기 위해서는 저장하려는 클래스는 Serializable 인터페이스를 구현해야 합니다.
- 직렬화를 위해서는 ObjectOutputStream 클래스를 writeObject() 메소드 이용하여 파일로 저장하고 역 직렬화는 ObjectInputStream 클래스의 readObject() 메소드를 이용해서 인스턴스로 반환 받음.

12.5 인스턴스의 직렬화

55



java.io.Serializable 인터페이스

- Serializable은 구현의 목적이 아닌 마킹의 기능이 있는 인터페이스.
- 클래스가 직렬화가 가능하도록 하는데 사용되므로 인스턴스의 저장이 필요한 클래스에 구현해 주어야 함.
- String 클래스, 래퍼 클래스 등은 Serializable 인터페이스를 구현하고 있어 직렬화가 가능함.
- 배열과 Collection의 경우 저장되어 있는 요소 중 직렬화할 수 없는 요소를 포함하고 있으면 직렬화가 되지 않음.

12.5 인스턴스의 직렬화

56



ObjectOutputStream 클래스

- ObjectOutputStream 클래스는 인스턴스를 OutputStream에 쓰는데 사용.
- Serializable 인터페이스를 구현하는 인스턴스에만 스트림에 쓸 수 있음.

```
public ObjectOutputStream(OutputStream out) throws IOException {}  
  
public final void writeObject(Object obj) throws IOException {}
```


12.5 인스턴스의 직렬화

57



ObjectInputStream 클래스

- ObjectInputStream은 ObjectOutputStream을 사용하여 작성된 객체 및 기본 데이터를 역 직렬화함.

```
public ObjectInputStream(InputStream in) throws IOException {}  
  
public final Object readObject() throws IOException, ClassNotFoundException{}
```

12.5 인스턴스의 직렬화

58



ObjectInputStream 클래스

```
package chapter12;

import java.io.Serializable;

public class Person implements Serializable {

    private String name;
    private int age;

    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public void tell() {
        System.out.println("안녕하세요. " + age + "살 " + name + "입니다.");
    }
}
```


12.5 인스턴스의 직렬화

59



ObjectInputStream 클래스

```
package chapter12;

import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;

public class SerializableTest {

    public static void main(String[] args) {

        Person person = new Person("King", 10);
        String msg = "안녕하세요~!!";

        try {
            ObjectOutputStream outputStream = null;
            FileOutputStream fos = new FileOutputStream("instanceData.ser");
            outputStream = new ObjectOutputStream(fos);
            outputStream.writeObject(person);
            outputStream.writeObject(msg);
        }
    }
}
```

12.5 인스턴스의 직렬화

60



ObjectInputStream 클래스

```
ObjectInputStream inputStream = null;
FileInputStream fis = new FileInputStream("instanceData.ser");
inputStream = new ObjectInputStream(fis);
Person newPerson = (Person) inputStream.readObject();
String newStr = (String) inputStream.readObject();

newPerson.tell();
System.out.println(newStr);

} catch (IOException e) {
    e.printStackTrace();
} catch (ClassNotFoundException e) {
    e.printStackTrace();
}
}
```

결과(Console)

안녕하세요. 10살 King입니다.
안녕하세요~!!

감사합니다
