3. I/O 기본2

#1.인강/0.자바/6.자바-고급2편

문자 다루기1 - 시작

스트림의 모든 데이터는 `byte` 단위를 사용한다. 따라서 `byte`가 아닌 문자를 스트림에 직접 전달할 수 는 없다.
예를 들어서 `String` 문자를 스트림을 통해 파일에 저장하려면 `String`을 `byte`로 변환한 다음에 저장해야 한다.

이번 예제에서 공통으로 다룰 상수를 먼저 만들자.

```
package io.text;

public class TextConst {
    public static final String FILE_NAME = "temp/hello.txt";
}
```

```
package io.text;

import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.Arrays;

import static io.text.TextConst.FILE_NAME;
import static java.nio.charset.StandardCharsets.UTF_8;

public class ReaderWriterMainV1 {

    public static void main(String[] args) throws IOException {
```

```
        String writeString = "ABC";
        // 문자 -> byte UTF-8 인코딩
        byte[] writeBytes = writeString.getBytes(UTF_8);
        System.out.println("write String: " + writeString);
        System.out.println("write bytes: " + Arrays.toString(writeBytes));

        // 파일에 쓰기
        FileOutputStream fos = new FileOutputStream(FILE_NAME);
        fos.write(writeBytes);
        fos.close();

        // 파일에서 읽기
        FileInputStream fis = new FileInputStream(FILE_NAME);
        byte[] readBytes = fis.readAllBytes();
        fis.close();

        // byte -> String UTF-8 디코딩
        String readString = new String(readBytes, UTF_8);

        System.out.println("read bytes: " + Arrays.toString(writeBytes));
        System.out.println("read String: " + readString);
    }
}
```

실행 결과

```
write String: ABC
write bytes: [65, 66, 67]
read bytes: [65, 66, 67]
read String: ABC
```

실행 결과 - hello.txt

```
ABC
```

byte[] writeBytes = writeString.getBytes(UTF_8)

- `String`을 `byte`로 변환할 때는 `String.getBytes(Charset)`을 사용하면 된다.
- 이때 문자를 `byte` 숫자로 변경해야 하기 때문에 반드시 **문자 집합(인코딩 셋)**을 지정해야 한다.
- 여기서는 UTF_8로 인코딩 한다.

- ABC를 인코딩하면 65, 66, 67이 된다.

이렇게 만든 `byte[]`을 `FileOutputStream`에 `write()`로 전달하면 65, 66, 67을 파일에 저장할 수 있다. 결과적으로 우리가 의도한 ABC 문자를 파일에 저장할 수 있다.

String readString = new String(readBytes, UTF_8)

- 반대의 경우도 비슷하다. `String` 객체를 생성할 때, 읽어들인 `byte[]`과 디코딩할 문자 집합을 전달하면 된다.
- 그러면 `byte[]`를 `String` 문자로 다시 복원할 수 있다.

여기서 핵심은 스트림은 `byte`만 사용할 수 있으므로, `String`과 같은 문자는 직접 전달할 수 는 없다는 점이다.
그래서 개발자가 번거롭게 다음과 같은 변환 과정을 직접 호출해주어야 한다.

- `String` + 문자 집합 ➜ `byte[]`
- `byte[]` + 문자 집합 ➜ `String`

이렇게 번거로운 변환 과정을 누군가 대신 처리해주면 더 편리하지 않을까?
우리가 이전에 배운 `BufferedXxx`와 같이 누군가 이러한 변환 기능을 대신 처리해주면 좋지 않을까?

문자 다루기2 - 스트림을 문자로

- **OutputStreamWriter**: 스트림에 byte 대신에 문자를 저장할 수 있게 지원한다.
- **InputStreamReader**: 스트림에 byte 대신에 문자를 읽을 수 있게 지원한다.

```
package io.text;

import java.io.*;

import static io.text.TextConst.FILE_NAME;
import static java.nio.charset.StandardCharsets.UTF_8;

public class ReaderWriterMainV2 {

    public static void main(String[] args) throws IOException {
        String writeString = "ABC";
        System.out.println("write String: " + writeString);
```

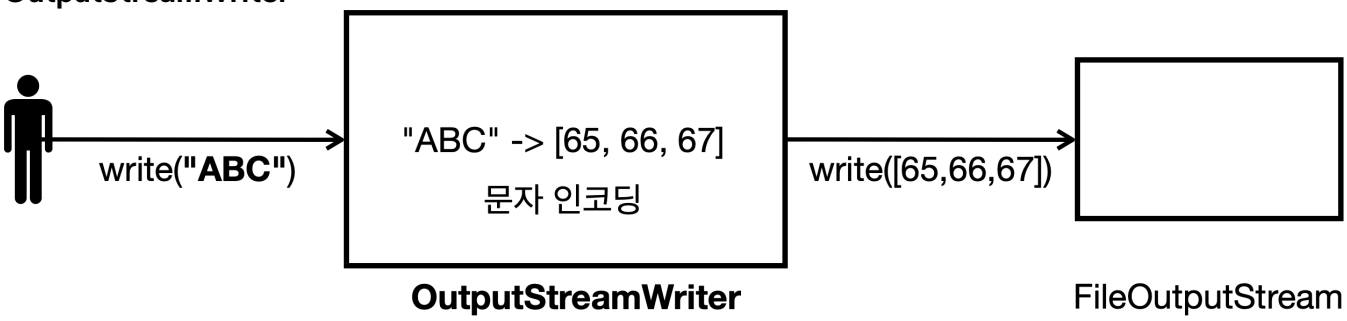
```
        // 파일에 쓰기
        FileOutputStream fos = new FileOutputStream(FILE_NAME);
        OutputStreamWriter osw = new OutputStreamWriter(fos, UTF_8);
        osw.write(writeString);
        osw.close();

        // 파일에서 읽기
        FileInputStream fis = new FileInputStream(FILE_NAME);
        InputStreamReader isr = new InputStreamReader(fis, UTF_8);

        StringBuilder content = new StringBuilder();
        int ch;
        while ((ch = isr.read()) != -1) {
            content.append((char) ch);
        }
        isr.close();

        System.out.println("read String: " + content);
    }
}
```

- 코드를 보면 앞서 작성한 `BufferedXxx`와 비슷한 것을 확인할 수 있다.

실행 결과

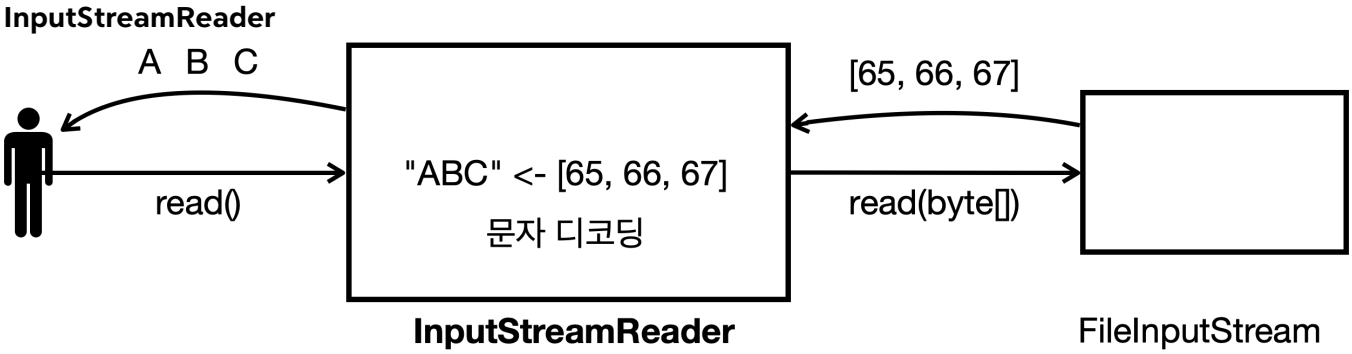
```
write String: ABC
read String: ABC
```

OutputStreamWriter



- `OutputStreamWriter`는 문자를 입력 받고, 받은 문자를 인코딩해서 `byte[]`로 변환한다.
- `OutputStreamWriter`는 변환한 `byte[]`을 전달할 `OutputStream`과 인코딩 문자 집합에 대한 정보가 필요하다. 따라서 두 정보를 생성자를 통해 전달해야 한다.
  - `new OutputStreamWriter(fos, UTF_8)`
- `osw.write(writeString)`를 보면 `String` 문자를 직접 전달하는 것을 확인할 수 있다.

- 그림을 보면 `OutputStreamWriter`가 문자 인코딩을 통해 `byte[]`로 변환하고, 변환 결과를 `FileOutputStream`에 전달하는 것을 확인할 수 있다.



- 데이터를 읽을 때는 `int ch = read()`를 제공하는데, 여기서는 문자 하나인 `char` 형으로 데이터를 받게 된다. 그런데 실제 반환 타입은 `int` 형이므로 `char` 형으로 캐스팅해서 사용하면 된다.
- 자바의 `char` 형은 파일의 끝인 `-1`을 표현할 수 없으므로 대신 `int`를 반환한다.
- 그림을 보면 데이터를 읽을 때 `FileInputStream`에서 `byte[]`을 읽은 것을 확인할 수 있다. `InputStreamReader`는 이렇게 읽은 `byte[]`을 문자인 `char`로 변경해서 반환한다. 물론 `byte`를 문자로 변경할 때도 문자 집합이 필요하다.
  - `new InputStreamReader(fis, UTF_8)`

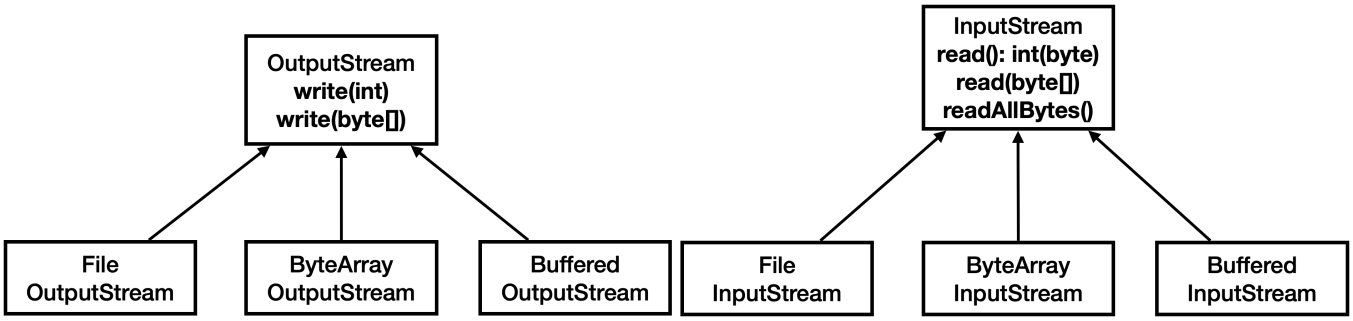
`OutputStreamWriter`, `InputStreamReader` 덕분에 매우 편리하게 문자를 `byte[]`로 변경하고, 그 반대도 가능하다. 덕분에 개발자는 쉽게 `String` 문자를 파일에 저장할 수 있다.

앞서 우리가 스트림을 배울 때 분명 `byte` 단위로 데이터를 읽고 쓰는 것을 확인했다. `write()`의 경우에도 `byte` 단위로 데이터를 읽고 썼다. 최상위 부모인 `OutputStream`의 경우 분명 `write()`가 `byte` 단위로 입력하도록 되어있다.
그런데 `OutputStreamWriter`의 `write()`는 `byte`가 아니라 `String`이나 `char`를 사용한다. 어떻게 된 것일까?

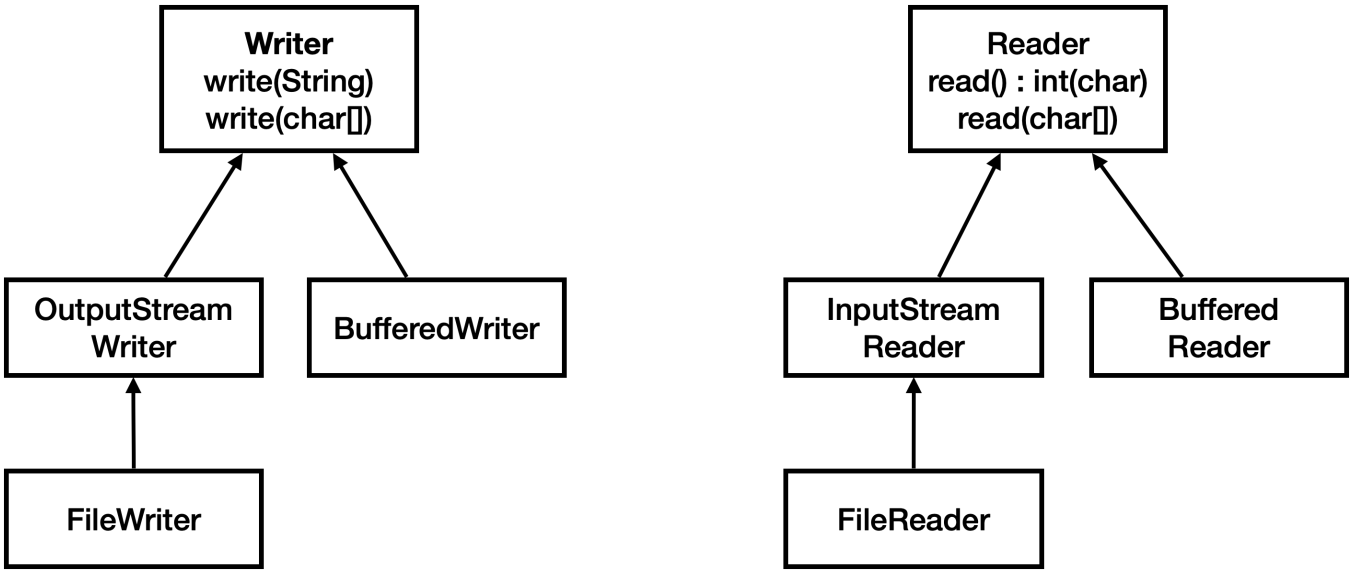
문자 다루기3 - Reader, Writer

자바는 byte를 다루는 I/O 클래스와 문자를 다루는 I/O 클래스를 둘로 나누어두었다.

byte를 다루는 클래스



문자를 다루는 클래스



- byte를 다루는 클래스는 `OutputStream`, `InputStream`의 자식이다.
  - 부모 클래스의 기본 기능도 `byte` 단위를 다룬다.
  - 클래스 이름 마지막에 보통 `OutputStream`, `InputStream`이 붙어있다.
- 문자를 다루는 클래스는 `Writer`, `Reader`의 자식이다.
  - 부모 클래스의 기본 기능은 `String`, `char` 같은 문자를 다룬다.
  - 클래스 이름 마지막에 보통 `Writer`, `Reader`가 붙어있다.

우리가 방금 본 `OutputStreamWriter`는 바로 문자를 다루는 `Writer` 클래스의 자식이다. 그래서 `write(String)`이 가능한 것이다. `OutputStreamWriter`는 문자를 받아서 byte로 변경한 다음에 byte를 다루는 `OutputStream`으로 데이터를 전달했던 것이다.

여기서 **꼭! 기억해야할 중요한 사실이 있다. 처음에 언급했듯이 모든 데이터는 byte 단위(숫자)로 저장된다.** 따라서 `Writer`가 아무리 문자를 다룬다고 해도 문자를 바로 저장할 수는 없다. 이 클래스에 문자를 전달하면 결과적으로 내부에서는 지정된 문자 집합을 사용해서 문자를 byte로 인코딩해서 저장한다.

FileWriter, FileReader

`Writer`, `Reader`를 사용하는 다른 예를 하나 더 보자.

```
package io.text;

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

import static io.text.TextConst.FILE_NAME;
import static java.nio.charset.StandardCharsets.UTF_8;

public class ReaderWriterMainV3 {

    public static void main(String[] args) throws IOException {
        String writeString = "ABC";
        System.out.println("write String: " + writeString);

        // 파일에 쓰기
        FileWriter fw = new FileWriter(FILE_NAME, UTF_8);
        fw.write(writeString);
        fw.close();

        // 파일에서 읽기
        StringBuilder content = new StringBuilder();
        FileReader fr = new FileReader(FILE_NAME, UTF_8);
        int ch;
        while ((ch = fr.read()) != -1) {
            content.append((char) ch);
        }
        fr.close();

        System.out.println("read String: " + content);

    }
}
```

실행 결과

```
write String: ABC
read String: ABC
```

new FileWriter(FILE_NAME, UTF_8)

- `FileWriter`에 파일명과, 문자 집합(인코딩 셋)을 전달한다.
- `FileWriter`는 사실 내부에서 스스로 `FileOutputStream`을 하나 생성해서 사용한다.
- 모든 데이터는 byte 단위로 저장된다는 사실을 다시 떠올려보자.

```
public FileWriter(String fileName, Charset charset) throws IOException {
    super(new FileOutputStream(fileName), charset);
}
```

fw.write(writeString)

- 이 메서드를 사용하면 문자를 파일에 직접 쓸 수 있다. (물론 실제 그런 것은 아니다.)
- 이렇게 문자를 쓰면 `FileWriter` 내부에서는 인코딩 셋을 사용해서 문자를 byte로 변경하고, `FileOutputStream`을 사용해서 파일에 저장한다.
- 개발자가 느끼기에는 문자를 직접 파일에 쓰는 것 처럼 느껴지지만, 실제로는 내부에서 문자를 byte로 변환한다.

new FileReader(FILE_NAME, UTF_8)

- 앞서 설명한 `FileWriter`와 같은 방식으로 작동한다.
- 내부에서 `FileInputStream`를 생성해서 사용한다.

```
public FileReader(String fileName, Charset charset) throws IOException {
    super(new FileInputStream(fileName), charset);
}
```

`ch = fr.read()`

- 데이터를 읽을 때도 내부에서는 `FileInputStream`을 사용해서 데이터를 byte 단위로 읽어들인다. 그리고 문자 집합을 사용해서 `byte[]`을 `char`로 디코딩한다.

FileWriter와 OutputStreamWriter

`FileWriter` 코드와 앞서 작성한 `OutputStreamWriter`를 사용한 코드가 뭔가 비슷하다는 점을 알 수 있다. 딱 하나 차이점이 있다면 이전 코드에서는 `FileOutputStream`을 직접 생성했는데, `FileWriter`는 생성자 내부에서 대신 `FileOutputStream`를 생성해준다. 사실 `FileWriter`는 `OutputStreamWriter`을 상속한다. 그리고 다른 추가 기능도 없다. 딱 하나, 생성자에서 개발자 대신에 `FileOutputStream`을 생성해주는 일만 대신 처리해준다.

따라서 `FileWriter`는 `OutputStreamWriter`를 조금 편리하게 사용하도록 도와줄 뿐이다. 물론 `FileReader`도 마찬가지다.

정리

`Writer`, `Reader` 클래스를 사용하면 바이트 변환 없이 문자를 직접 다룰 수 있어서 편리하다. 하지만 실제로는 내부에서 byte로 변환해서 저장한다는 점을 꼭 기억하자. 모든 데이터는 바이트 단위로 다룬다! 문자를 직접 저장할 수는 없다!
그리고 반드시 기억하자, 문자를 byte로 변경하려면 항상 문자 집합(인코딩 셋)이 필요하다!

참고: 문자 집합을 생략하면 시스템 기본 문자 집합이 사용된다.

문자 다루기4 - BufferedReader

`BufferedOutputStream`, `BufferedInputStream`과 같이 `Reader`, `Writer`에도 버퍼 보조 기능을 제공하는 `BufferedReader`, `BufferedWriter` 클래스가 있다.

추가로 문자를 다룰 때는 한 줄(라인)단위로 다룰 때가 많다.
`BufferedReader`는 한 줄 단위로 문자를 읽는 기능도 추가로 제공한다.

```
package io.text;

import java.io.*;

import static io.text.TextConst.FILE_NAME;
import static java.nio.charset.StandardCharsets.UTF_8;

public class ReaderWriterMainV4 {

    private static final int BUFFER_SIZE = 8192;

    public static void main(String[] args) throws IOException {
        String writeString = "ABC\n가나다";
        System.out.println("== Write String ==");
        System.out.println(writeString);

        // 파일에 쓰기
```

```
        FileWriter fw = new FileWriter(FILE_NAME, UTF_8);
        BufferedWriter bw = new BufferedWriter(fw, BUFFER_SIZE);
        bw.write(writeString);
        bw.close();

        // 파일에서 읽기
        StringBuilder content = new StringBuilder();
        FileReader fr = new FileReader(FILE_NAME, UTF_8);
        BufferedReader br = new BufferedReader(fr, BUFFER_SIZE);

        String line;
        while ((line = br.readLine()) != null) {
            content.append(line).append("\n");
        }
        br.close();

        System.out.println("== Read String ==");
        System.out.println(content);
    }
}
```

실행 결과

```
== Write String ==
ABC
가나다
== Read String ==
ABC
가나다
```

br.readLine()

- 한 줄 단위로 문자를 읽고 `String`을 반환한다.
- 파일의 끝(EOF)에 도달하면 `null`을 반환한다.
    - 반환 타입이 `String`이기 때문에 EOF를 -1로 표현할 수 없다. 대신에 `null`을 반환한다.

기타 스트림

지금까지 설명한 스트림 외에 수 많은 스트림들이 있다.

몇가지 유용한 부가 기능을 제공하는 `PrintStream`, `DataOutputStream` 보조 스트림을 알아보자.

PrintStream

`PrintStream`은 우리가 자주 사용해왔던 바로 `System.out`에서 사용되는 스트림이다.

`PrintStream`과 `FileOutputStream`을 조합하면 마치 콘솔에 출력하듯이 파일에 출력할 수 있다.

```
package io.streams;

import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.PrintStream;

public class PrintStreamEtcMain {

    public static void main(String[] args) throws FileNotFoundException {
        FileOutputStream fos = new FileOutputStream("temp/print.txt");
        PrintStream printStream = new PrintStream(fos);
        printStream.println("hello java!");
        printStream.println(10);
        printStream.println(true);
        printStream.printf("hello %s", "world");
        printStream.close();
    }
}
```

실행 결과 - temp/print.txt

```
hello java!
10
true
hello world
```

`PrintStream`의 생성자에 `FileOutputStream`을 전달했다. 이제 이 스트림을 통해서 나가는 출력은 파일에 저장된다.

이 기능을 사용하면 마치 콘솔에 출력하는 것 처럼 파일이나 다른 스트림에 **문자를 출력**할 수 있다.

DataOutputStream

`DataOutputStream`을 사용하면 자바의 `String`, `int`, `double`, `boolean` 같은 데이터 형을 편리하게 다룰 수 있다.

이 스트림과 `FileOutputStream`을 조합하면 파일에 자바 데이터 형을 편리하게 저장할 수 있다.

```
package io.streams;

import java.io.*;

public class DataStreamEtcMain {

    public static void main(String[] args) throws IOException {
        FileOutputStream fos = new FileOutputStream("temp/data.dat");
        DataOutputStream dos = new DataOutputStream(fos);

        dos.writeUTF("회원A");
        dos.writeInt(20);
        dos.writeDouble(10.5);
        dos.writeBoolean(true);
        dos.close();

        FileInputStream fis = new FileInputStream("temp/data.dat");
        DataInputStream dis = new DataInputStream(fis);
        System.out.println(dis.readUTF());
        System.out.println(dis.readInt());
        System.out.println(dis.readDouble());
        System.out.println(dis.readBoolean());
        dis.close();
    }
}
```

실행 결과

```
회원A
20
10.5
true
```

예제를 보면 자바 데이터 타입을 사용하면서, 회원 데이터를 편리하게 저장하고 불러오는 것을 확인할 수 있다.
이 스트림을 사용할 때 주의할 점으로는 꼭! 저장한 순서대로 읽어야 한다는 것이다. 그렇지 않으면 잘못된 데이터가 조회될 수 있다.

저장한 `data.dat` 파일을 직접 열어보면 제대로 보이지 않는다. 왜냐하면 `writeUTF()`의 경우 UTF-8 형식으로 저장하지만, 나머지의 경우 문자가 아니라 각 타입에 맞는 byte 단위로 저장하기 때문이다.
예를 들어서 자바에서 int는 4byte를 묶어서 사용한다. 해당 byte가 그대로 저장되는 것이다.
텍스트 편집기는 자신의 문자 집합을 사용해서 byte를 문자로 표현하려고 시도하지만 문자 집합에 없는 단어이거나 또는 전혀 예상하지 않은 문자로 디코딩 될 것이다.
참고로 `DataOutputStream`에 대해서는 뒤에 활용에서 더 자세히 다룬다.

정리

정리

- 기본(기반, 메인) 스트림
  - File, 메모리, 콘솔등에 직접 접근하는 스트림
  - 단독으로 사용할 수 있음
  - 예) `FileInputStream`, `FileOutputStream`, `FileReader`, `FileWriter`, `ByteArrayInputStream`, `ByteArrayOutputStream`
- 보조 스트림
  - 기본 스트림을 도와주는 스트림
  - 단독으로 사용할 수 없음, 반드시 대상 스트림이 있어야함
  - 예) `BufferedInputStream`, `BufferedOutputStream`, `InputStreamReader`, `OutputStreamWriter`, `DataOutputStream`, `DataInputStream`, `PrintStream`