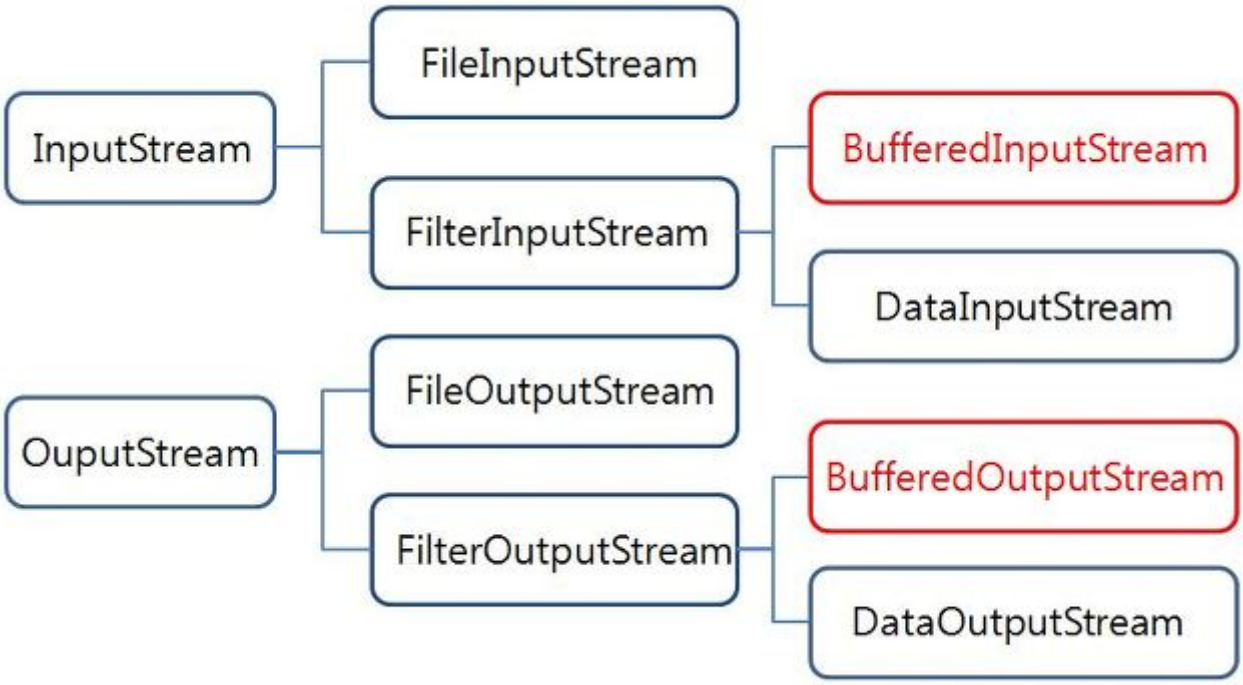

InputStream, OutputStream

- **바이트 단위 IO 클래스는 가장 기본이 되는 IO 클래스이다.**
- 바이트 스트림 클래스들은 모두 추상클래스인 InputStream과 OutputStream의 하위 클래스이며, 입출력과 관련된 모든 바이트 스트림은 InputStream과 OutputStream에 있는 메서드를 포함한다는 의미이다.

1. InputStream의 중요 메서드

```
int available() throws IOException // 현재 읽을 수 있는 바이트 수를 반환한다.
void close() throws IOException // 입력스트림을 닫는다.
int read() throws IOException // 입력스트림에서 한 바이트를 읽어 int 값으로 반환(실제로 읽어들인 값을 반환)하며 더이상 읽을 값이 없을 경우 -1을 반환한다.
int read(byte buf[]) throws Exception // 입력스트림에서 buf[] 크기만큼 읽어 buf에 저장하고, 읽은 바이트 수를 반환한다.
```

2. OutputStream의 중요 메서드

```
void close() throws Exception // 출력스트림을 닫는다.
void flush() throws IOException // 버퍼에 남은 출력스트림을 출력한다.
void write(int i) throws IOException // 정수 i의 하위 8비트를 출력한다.
void write(byte buf[]) throws IOException // buf 배열의 내용을 출력한다.
void write(byte buf[], int index, int size) throws IOException // buf 배열의 index 위치부터 size만큼의 바이트를 출력한다.
```

3. 사용 예제

```
import java.io.IOException;
public class IOTest {
  public static void main(String[] args) {
    int i = 0;
    try {
      while ((i = System.in.read()) != -1) {
        System.out.write(i);
      }
    } catch (IOException e) {
      System.out.println(e);
    }
  }
}
```

System.in 은 InputStream 형식. 이는 InputStream에 있는 모든 메서드를 사용할 수 있다는 의미.
System.out은 PrintStream 형식이며 PrintStream 역시 OutputStream을 상속받고 있기 때문에 write() 메서드를 사용할 수 있다.

위 예제가 실행되는 순서를 살펴보자.
키보드로부터 입력을 받으면 일단 OS가 관리하는 키보드 버퍼에 쌓이고 엔터를 누르면 JVM에게 입력되어진 값(enter 포함) 이 전달된다. 이후 전달된 문자열은 read() 메서드에 의해 차례대로 한 바이트씩 읽어들이게 된다. 그리고 한 바이트씩 읽어들이면서 while문 안에 있는 명령을 실행하게 된다.

마지막으로 파일의 끝을 표시하려면 Ctrl + z 를 누른 후 enter를 치면 된다.

FileInputStream, FileOutputStream

- FileInputStream : InputStream 클래스를 상속받은 자식 클래스. 하드 디스크 상에 있는 파일로부터 바이트 단위의 입력을 받는 클래스이다. 출발 지점과 도착 지점을 연결하는 통로(스트림) 을 생성한다. 생성자의 argument로 File 객체나 파일명을 직접 String으로 줄 수 있다. 일반적으로 파일명을 String 꼴로 주는 경우가 많은데 파일이 존재하지 않을 가능성도 있어 Exception 처리가 필요하다.
- FileOutputStream : OutputStream 클래스를 상속받은 자식 클래스, 파일로 바이트 단위의 출력을 내보내는 클래스이다.

1. FileInputStream 생성자

```
FileInputStream(String filepath) throws FileNotFoundException
// filepath로부터 지정한 파일로부터 바이트 단위로 읽어들이는 스트림 객체 생성
```

2.FileOutputStream 생성자

```
FileOutputStream(String filepath) throws IOException
// filepath로 지정한 파일에 대한 출력 스트림을 생성
FileOutputStream(String filepath, Boolean append) throws IOException
// 지정한 파일로 출력 스트림을 생성, append 변수 값이 true로 설정되면 기존 파일에 이어서 쓰게 된다.
```

DataInputStream, DataOutputStream

- DataInputSteram과 DataOutputStream은 **자바의 기본형 데이터인 int, float, double, boolean, short, byte 등의 정보를 입력하고 출력하는 데 사용하는 클래스.**
- **Data Processing Stream.** 즉 중간에서 데이터를 가공하는 일을 하는 클래스를 말한다. 이러한 클래스를 **"Decorator"** 라고 한다.
- 두 클래스는 각각 **생성자에서 InputStream과 OutputStream을 받아들인다.**
- 즉, InputStream과 OutputStream의 하위 클래스들을 받아들인다는 의미도 포
- 함.

1. DataInputStream의 중요 메서드 (일부)

```
int readInt() throws IOException // 스트림으로부터 읽은 int를 반환한다.
byte readByte() throws IOException // 스트림으로부터 읽은 byte를 반환한다.
```

void readFully(byte buf[]) throws IOException // 스트림으로부터 buf 크기만큼의 바이트를 읽어 buf에 저장한다.
String readUTF() throws IOException // UTF 인코딩 값을 얻어 문자열로 반환하며, 네트워크 프로그래밍을 할 때 네트워크로부터 문자열을 읽어들일 시 사용한다.

2. DataOutputStream의 중요 메서드 (일부)

void write(int i) throws IOException // int형 i값이 갖는 1바이트(하위 8bit)를 출력한다.
void write(byte buf[], int index, int size) throws IOException // 바이트 배열 buf의 주어진 위치 index에서 size 만큼 출력한다.
void writeBoolean(boolean b) throws IOException // boolean을 1바이트 값으로 출력한다.
void writeDouble(double d) throws IOException // Double 클래스의 doubleToBits() 를 사용해서 long으로 변환한 다음 long값을 8바이트 수량으로 상위 바이트부터 출력한다.
void writeFloat(float f) throws IOException // Float를 floatToBits()를 사용해서 변환한 후 int 값을 4바이트 수량으로 상위 바이트부터 출력한다.
void writeInt(int i) throws IOException // int의 상위바이트부터 출력한다.
void writeUTF(String s) throws IOException // UTF-8 인코딩을 사용해서 문자열을 출력하며, 네트워크 프로그래밍을 할 때, 문자열 전송 시 자주 사용된다.

3. 사용 예제

```
import java.io.DataOutputStream;
import java.io.FileOutputStream;

public class DataOutputStreamTest {

  public static void main(String[] args) {

    try (
        DataOutputStream dos = new DataOutputStream(new
FileOutputStream("data.bin"))) {
      dos.writeBoolean(true);
      dos.write(5);
```

```
      dos.writeByte((byte) 5);
      dos.writeInt(100);
      dos.writeDouble(200.5);
      dos.writeUTF("Hi");
      System.out.println("저장했습니다.");
    } catch (Exception e) {
      System.out.println(e.getMessage());
    }
  }
}
```

FileOutputStream이 data.bin이라는 파일에 출력.
그리고 이러한 FileOutputStream의 객체를 DataOutputStream의 생성자에 지정한 후, 해당 메서드를 이용해서 출력했는데 이는 DataOutputStream은 내부적으로 FileOutputStream을 이용한다는 것을 뜻한다.

다음 예제는 생성된 data.bin 파일에서 거꾸로 데이터를 읽어들여 화면에 출력하는 것을 보여준다.

```
import java.io.DataInputStream;
import java.io.FileInputStream;

public class DataInputStreamTest {
  public static void main(String[] args) {
    try (DataInputStream dis = new DataInputStream(new
FileInputStream("data.bin"))) {
      boolean b  = dis.readBoolean();
      int     b2 = dis.read();
      byte    b3 = dis.readByte();
      int     i  = dis.readInt();
      double  d  = dis.readDouble();
      String  s  = dis.readUTF();

      System.out.println("boolean: " + b);
      System.out.println("int    : " + b2);
```

```
      System.out.println("byte   : " + b3);
      System.out.println("int    : " + i);
      System.out.println("double : " + d);
      System.out.println("String : " + s);

    } catch (Exception e) {
      System.out.println(e.getMessage());
    }
  }
}
```

FileInputStream 생성자에 data.bin 파일명을 지정하므로써 FileInputStream은 data.bin 파일로부터 데이터를 읽어들이며, 이러한 FileInputStream의 객체를 DataInputStream 생성자에 인자로 지정함으로써 DataInputStream을 통해서 데이터를 읽어들인다.

BufferedInputStream, BufferedOutputStream

- 버퍼 : 데이터를 한 곳에서 다른 한 곳으로 전송하는 동안 일시적으로 그 데이터를 보관하는 메모리의 영역
- 버퍼링 : 버퍼를 활용하는 방식 또는 버퍼를 채우는 동작을 의미한다.

  FileInputStream /FileOutputStream 에서 이야기한 것처럼, 1 Byte 단위로 입/출력이 이루어지면 기계적인 동작이 많아지므로 효율이 떨어지게 된다. BufferedOutputStream클래스는 flush()메서드가 호출되거나, 버퍼가 꽉 찰 때까지 데이터를 버퍼에 저장했다가 한꺼번에 스트림에 쓰는 방식이다. 예를 들면 100byte에 해당하는 정보를 스트림에 쓰려고 한다면 기존의 방식은 1byte씩 100번의 write()메서드를 호출해야 했지만, 버퍼링을 사용한다면 100byte를 버퍼에 모아서 1번에 write()메서드를 호출하면 되므로 상당히 효율적이라고 할 수 있다.

BufferedInputStream & BufferedOutputStream클래스는 입력과 출력에 대한 버퍼링(buffering)기능을 제공한다. 이것을 다른 말로 해 보면 BufferedInputStream은 InputStream객체와 버퍼를 내부적으로 가지고 있다는 것이고, 마찬가지로 BufferedOutputStream은 OutputStream객체와 버퍼를 가지고 있는 것이다.

BufferedInputStream을 통하여 원하는 자료를 1바이트 단위로 읽는 read() 메소드를 수행하면 시스템 내부적으로 버퍼를 준비하고 이 버퍼를 이용하여 지정된 파일로부터 버퍼의 크기만큼 한꺼번에 많은 데이터를 가져온다.이렇게 채워진 버퍼로부터 1 바이트씩 읽어들여 프로그램으로 전달하게 된다. 이때, 1 바이트씩 읽어들이

는 작업은 파일로부터 읽어오는 것이 아닌 준비된 시스템 버퍼에서 읽어오게 되므로, 파일 입력으로 인한 성능 저하를 최소화 할 수 있다.
(read() - 버퍼준비, 지정파일에서 데이터 추출해서 담기 - 버퍼에서 1바이트씩 프로그램으로 전달)
BufferedOutputStream 역시 마찬가지로 데이터를 1 바이트씩 출력하면 파일이 아닌 준비된 시스템 버퍼에 출력을 쌓는다. 버퍼가 모두 채워지거나 close(), flush() 명령을 만나면 버퍼의 모든 내용을 하드 디스크 파일에 출력한다. 따라서, BufferedOutputStream 역시 하드 디스크 파일에 대한 출력을 최소화 하여 효율을 향상 시킬 수 있다.
((버퍼에 출력데이터 담기 - 버퍼가득/close()/flush() - 버퍼 내용을 하드디스크파일로 출력)

결론적으로 사용자가 BufferedInputStream과 BufferedOutputStream을 이용하여 프로그램을 작성하면 1 바이트씩 읽고 쓰는 모든 작업이 하드 디스크 파일이 아닌 내부적인 버퍼를 대상으로 발생하며, 필요에 따라 버퍼와 하드 디스크 파일간에 입출력이 간헐적으로 발생하므로 전체적인 입출력 성능이 동적으로 향상될 수 있다.

1. BufferedInputStream의 생성자

```
BufferedInputStream(InputStream in) // InputStream에 대한 BufferedInputStream 객체를 생성한다.
BufferedInputStream(InputStream in, int size) // InputStream에 대한
BufferedInputStream 객체를 생성하고 내부 버퍼의 크기를 size 값으로 설정한다.
```

2. BufferedOutputStream의 생성자

```
BufferedOutputStream(OutputStream out) // OutputStream에 대한 BufferedOutputStream 객체를 생성한다.
BufferedOutputStream(OutputSteam out, int size) // OutputStream에 대한
BufferedOutputStream 객체를 생성하고 내부 버퍼의 크기를 Size 값으로 설정한다.
```