Java I/O Stream I - ByteStream

I/O

- Data의 입력(Input), 출력(Output)
- Java Program과 입출력 대상 간의 Data 입출력은 입출력 대상(File, Server, Device…..)에 따라 입출력 방법이 완전히 달라진다.

	Data	
	↔	File
Java Program	↔	Server
	↔	Device

만약에 Java Program이 3개의 입출력 대상과 입출력을 진행하는데, 3개 모두 각기 다른 종류이면 어떻게 되겠는가?
3개의 입출력 방법을 써야한다.

Java I/O

- **Java Program과 입출력 대상 사이에 인터페이스 역할을 하는 I/O Stream Class를 정의하고 표준화시켜 입출력 대상에 상관없이 입출력 진행방식이 동일하도록 설계된 모델.**

- 입출력 대상 또는 제조사 측에서 Java의 I/O Stream Class를 **기반으로** 상속한 서브 클래스들을 설계하기 때문에, 개발자는 I/O Stream Class에서 정의된 메소드를 가지고 I/O를 진행할 수 있다.

- I/O의 인터페이스화의 이점
  i. 입출력 대상이 각기 달라도 오버라이딩에 의해 메소드 이름이 통일되도록 제한하고 최상위인 I/O Stream Class를 기반으로 제작하도록 표준화

ii. **입출력 대상 측이 오버라이딩 한 메소드들을 설계해 주면,**
Java Program 개발자는 입출력 대상 측의 데이터 처리까지 신경 쓸 필요가 없다.

	Data		Data	
			↔	**Target A**
Java Program	↔	**Input Stream Class** **Output Stream Class**	↔	**Target B**
			↔	**Target C**

- 쉽게 이해하기 위해서 간단하게 정의해보자

```
public abstract class IO {
     public abstract void input();
     public abstract void output();
}

public class DeviceIO extend IO {
     @Override
     public void input() { System.out.println("디바이스 방식으로 데이터 입력..."); }
     @Override
     public void output() { System.out.println("디바이스 방식으로 데이터 출력..."); }
}

public class ConsoleIO extend IO {
     @Override
     public void input() { System.out.println("콘솔 방식으로 데이터 입력..."); }
     @Override
     public void output() { System.out.println("콘솔 방식으로 데이터 출력..."); }
}
```

```
public class IOMain {
     public static void main(String[] args) {

          (입출력 대상과 상관없이 동일한 메소드로 입출력이 진행된다!)
          IO deviceIO = new DeviceIO();
          deviceIO.input();
          deviceIO.output();

          IO consoleIO = new ConsoleIO();
          consoleIO.input();
          consoleIO.output();
     }
}
```

ByteStream

입력 스트림 (InputStream)	프로그램으로 데이터를 읽어 들이는 스트림
출력 스트림 (OutputStream)	프로그램으로 부터 데이터를 내보내는 스트림

- 스트림 (Stream)?
  스트림의 본래 뜻은 강의 흐름이라는 뜻인데, Byte단위 데이터열의 단방향 흐름을 의미한다.
  (흐름이 단방향 이므로 입력과 출력의 스트림이 따로 존재해야 한다.)

- 스트림의 생성은 스트림 인스턴스의 생성이다!
- 입출력 스트림의 입출력 데이터 단위는 Byte이고, 데이터의 아무런 변환작업을 하지 않는다.
- 입출력 스트림은 대부분 쌍을 이룬다.

1Byte	1Byte	1Byte	1Byte	1Byte	1Byte	1Byte	1Byte	1Byte	1Byte	.....

InputStream

- 프로그램으로 데이터를 읽어 들이는 스트림
- 생성된 시점부터 데이터를 순차적으로 차례대로 읽는다.

- 대표적인 메소드

`public abstract int read() throws IOExeption`	데이터 읽기 (1byte)
`public void close() throws IOExeption`	데이터 읽기 종료

- read() 메소드의 반환형이 int인 이유는 **실제 데이터 크기는 1byte(0~255)**까지 인데, 읽기오류시 -1 반환
  -1을 반환하려면 표현 최대개수가 257개가 필요하므로 int형으로 반환함.
  (반환하는 int형 4Byte 데이터 중 4번째 1Byte가 실제 데이터이고 이 부분만 읽는다.)

compact1, compact2, compact3
java.io

Class InputStream

java.lang.Object
    java.io.InputStream

All Implemented Interfaces:
Closeable, AutoCloseable

Direct Known Subclasses:
AudioInputStream, ByteArrayInputStream, FileInputStream, FilterInputStream, InputStream, ObjectInputStream, PipedInputStream, SequenceInputStream, StringBufferInputStream

---

파일 기반의 InputStream

```
InputStream in = new FileInputStream("run.exe");
byte byteData =  in.read();
```

- 스트림의 생성은 결국 스트림 인스턴스의 생성을 의미
- FileInputStream 클래스는 InputStream 클래스를 상속한다.
- InputStream의 read() 호출은 오버라이딩에 의해 결국 FileInputStream의 read() 메소드를 호출하게 된다.

OutputStream

- 프로그램으로 부터 데이터를 내보내는 스트림
- 생성된 시점부터 데이터를 순차적으로 차려대로 쓴다.

- 대표적인 메소드

public abstract void write(int b) throws IOExeption	데이터 쓰기 (1byte)
public void close() throws IOExeption	데이터 쓰기 종료

- write() 메소드는 인자인 4Byte 정수 데이터 중 하위 1Byte만 전달

compact1, compact2, compact3
java.io

Class OutputStream

java.lang.Object
    java.io.OutputStream

All Implemented Interfaces:

Closeable, Flushable, AutoCloseable

Direct Known Subclasses:

ByteArrayOutputStream, FileOutputStream, FilterOutputStream, ObjectOutputStream, OutputStream, PipedOutputStream

---

파일 기반의 OutputStream

```
OutputStream out = new FileOutputStream("home.bin");
out.write(1);
out.write(2);
out.close();
```

- **FileOutputStream 클래스는 OutputStream 클래스를 상속한다.**

스트림 기반의 파일 입출력 예제

```
public class IOStreamDemo {
     public static void main(String[] args) throws IOException {
          InputStream in = new FileInputStream("D:/Documents/org.txt"); // 해당 위치의 파일읽기
          OutputStream out = new FileOutputStream("D:/Documents/copy.txt"); // 해당 위치에 파일을 생성

          int bValue = 0;
          int bData;

          while(true){
               // 반복문을 돌며 처음부터 1btye 씩 꺼내서 읽는다.
```

```
            bData = in.read();

            // 반복문을 돌며 1byte 씩 순서대로 읽다가 데이터를 다 읽었으면 탈출
            if(bData == -1){
                break;
            }

            // 꺼낸 데이터를 생성 할 파일에 처음부터 기록한다.
            out.write(bData);

            bValue++;
        }

        // 작업이 끝나면 반드시 IOStream 종료 시킬 것
        out.close();
        in.close();

        System.out.println("복사된 데이터 크기 : "+bValue+"Byte");
        System.out.println("프로그램 종료");
    }
}
```

바이트 배열로 파일 입출력 진행하기

`public abstract void read(byte[] b) throws IOExeption`	Byte 배열 단위로 읽기
`public abstract void write(byte[] b, int offset, int len) throws IOExeption`	Byte 배열 단위로 쓰기

```
public class IOStreamDemo {

	private static final String ORG_FILE_PATH = "D:/Documents/org.txt";
	private static final String COPY_FILE_PATH = "D:/Documents/copy.txt";

	public static void main(String[] args) throws FileNotFoundException{
		int bData;
		int byteCount = 0;

		// 1024 byte 만큼 읽을 임시 버퍼
		byte[] buf = new byte[1024];

		InputStream in = new FileInputStream(ORG_FILE_PATH);
		OutputStream out = new FileOutputStream(COPY_FILE_PATH);

		try {
			while(true){
				bData = in.read(buf); // buf 만큼 읽는다.(1024byte)
				if(bData == -1){
					break;
				}

				out.write(buf, 0, buf.length); // buf 배열의 인덱스 0 부터, buf의 크기만큼 보낸다.
				byteCount++;
			}

			out.close();
			in.close();
		} catch(IOException e) {
			e.printStackTrace();
		}
	}
}
```
