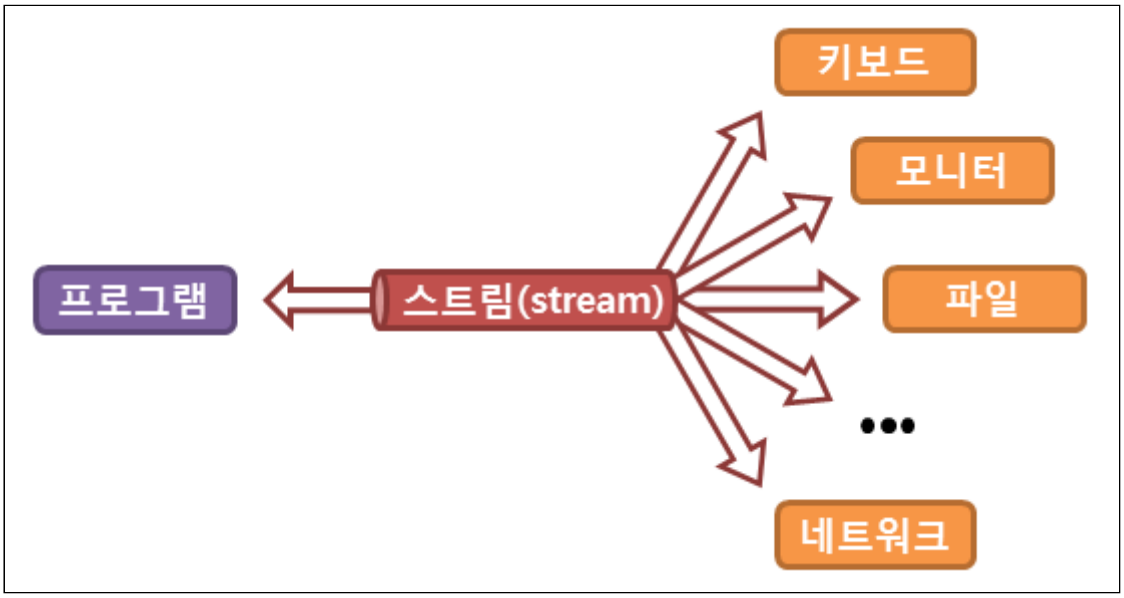
[TODO] 입출력

스트림의 개념

- 자바에서는 파일, 콘솔 혹은 네트워크 입출력을 직접 다루지 않고 **스트림(stream)이라는 추상화된 개념**을 통해서 다룸
    - 스트림 => 데이터의 원천(소스)에서 목적지로 흐르는 데이터의 흐름(flow)을 의미함



- 입출력 스트림
    - 스트림은 **한 방향으로만 통신**할 수 있고 **입력과 출력을 동시에 처리할 수 없음**
    - 따라서 스트림은 사용 목적에 따라 **입력 스트림과 출력 스트림으로 구분**됨
- 자바에서는 java.io 패키지를 통해 InputStream과 OutputStream 클래스를 별도로 제공

클래스	메소드	설명
InputStream	abstract int read()	해당 입력 스트림으로부터 다음 바이트를 읽어들임.
	int read(byte[] b)	해당 입력 스트림으로부터 특정 바이트를 읽어들인 후, 배열 b에 저장함.
	int read(byte[] b, int off, int len)	해당 입력 스트림으로부터 len 바이트를 읽어들인 후, 배열 b[off]부터 저장함.
OutputStream	abstract void write(int b)	해당 출력 스트림에 특정 바이트를 저장함.
	void write(byte[] b)	배열 b의 특정 바이트를 배열 b의 길이만큼 해당 출력 스트림에 저장함.
	void write(byte[] b, int off, int len)	배열 b[off]부터 len 바이트를 해당 출력 스트림에 저장함.

스트림의 종류

바이트 기반 스트림

- 바이트 스트림을 기본적으로 **하나의 바이트(1 byte => 8 bits)를 최소 단위로 하는 입출력 작업**을 지원하는 스트림
- 바이트 기반 스트림 클래스들의 최고 조상 클래스는 **InputStream과 OutputStream 클래스**
    - 추상 클래스이므로 추상 메서드만 존재하며 구체적인 쓰기, 읽기 동작 방식은 하위 클래스에서 구현

문자 기반 스트림

- 문자열 스트림은 하나의 글자를 최소 단위로 하는 입출력 작업을 지원하는 스트림
    - 자바에서 기본적으로 글자의 크기는 2바이트(=char의 크기)
- 문자 기반 스트림 클래스들의 최고 조상 클래스는 **Reader와 Writer 클래스**

보조 스트림

- 보조 스트림은 실제로 데이터를 주고받을 수는 없지만, 다른 **스트림의 기능을 향상시키거나 새로운 기능을 추가**해 주는 스트림
    - 가령 버퍼 입출력을 지원하는 BufferedInputStream, BufferedOutputStream은 버퍼 저장소를 제공하여 더 효율적인 입출력이 진행될 수 있도록 함

대표적인 보조 스트림 클래스

입력 스트림	출력 스트림	설명
BufferedInputStream	BufferedOutputStream	버퍼를 이용한 입출력 지원
DataInputStream	DataOutputStream	자바의 기본 타입 데이터 입출력 지원
ObjectInputStream	ObjectOutputStream	객체 타입의 입출력 지원
FilterInputStream	FilterOutputStream	입출력 과정에서 필터링 작업 지원

바이트 기반 스트림 클래스 활용

FileInputStream, FileOuputStream

- 입출력의 대상이 파일인 입출력 스트림

```
public class FileInputOutputStreamDemo {
    public static void main(String[] args) throws IOException {
        // 생성할 파일 이름 전달
```

```
        FileOutputStream fos = new FileOutputStream("temp" + File.separator +
"test.bin");
        // 한 바이트 쓰기 (최소 단위는 바이트)
        fos.write(0x00);
        // 바이트 배열 쓰기
        fos.write(new byte[]{ 0x01, 0x02, 0x03 });
        // 바이트 배열을 쓰되, 쓰기 시작할 배열의 시작 위치 및 크기를 정해주기
        fos.write(new byte[]{ 0x0a, 0x0b, 0x0c, 0x0d, 0x0e }, 1, 3);
        // 입출력 리소스는 반드시 close 메서드로 닫아줄 것
        fos.close();

        FileInputStream fis = new FileInputStream("temp" + File.separator +
"test.bin");
        byte first = (byte) fis.read();
        System.out.println(first);
        byte[] buffer1 = new byte[3];
        fis.read(buffer1);
        System.out.println(Arrays.toString(buffer1));
        byte[] buffer2 = new byte[]{ -1, -1, -1, -1, -1 };
        fis.read(buffer2, 1, 3);
        System.out.println(Arrays.toString(buffer2));
        fis.close();

        // 바이트 기반 스트림을 통한 문자열 저장 및 읽어오기도 가능
        fos = new FileOutputStream("temp" + File.separator + "test.txt");
        byte[] stringBytes = "Hello\nWorld".getBytes("UTF-8");
        fos.write(stringBytes);
        fos.close();

        fis = new FileInputStream("temp" + File.separator + "test.txt");
        byte[] buffer = new byte[255];
        // read의 반환값은 읽어온 바이트 총 개수
        int readBytes = fis.read(buffer);
        // 총 11바이트를 읽을 수 있으므로 11이 반환
        System.out.println(readBytes);
        byte[] bytesForString = Arrays.copyOf(buffer, 11);
        String s = new String(bytesForString, "UTF-8");
        System.out.println(s);
        fis.close();
    }
}
```

ByteArrayInputStream, ByteArrayOutputStream

- 내부 메모리에 저장된 바이트 배열이 입출력 대상인 입출력 스트림

```
public class ByteArrayInputOutputStreamDemo {
    public static void main(String[] args) throws IOException {
        // 내부 메모리에 저장된 바이트 배열에 쓰기 위해서 ByteArrayOutputStream 클래스
사용
        ByteArrayOutputStream baos = new ByteArrayOutputStream();
        // 메서드 사용법은 다르지 않음
        baos.write(new byte[]{ 0x0a, 0x0b, 0x0c });
        // 입력이 끝난 후 스트림에 저장된 내용을 바이트 배열로 변환 가능
        byte[] result = baos.toByteArray();
```

```
        System.out.println(Arrays.toString(result));
        baos.close();

        byte[] inputBytes = new byte[100];
        for(int i=0;i<100;i++) inputBytes[i] = (byte) i;
        // 특정 바이트 배열의 내용을 읽어올 수 있도록 ByteArrayInputStream 클래스 사용
        ByteArrayInputStream bais = new ByteArrayInputStream(inputBytes);
        byte[] buffer = new byte[10];
        bais.read(buffer);
        System.out.println(Arrays.toString(buffer));
    }
}
```

문자 기반 스트림 클래스 활용

FileReader, FileWriter

-

```
public class FileReaderWriterDemo {
    public static void main(String[] args) throws IOException {
        // 파일에 쓰되, 문자열(char 배열 혹은 String)을 이용하여 파일 내용 작성 가능
        FileWriter fw = new FileWriter("temp" + File.separator +
"writer_test.txt");
        // OutputStream과 비슷한 방식으로 동작
        fw.write("Hello");
        fw.write(new char[] { 'W', 'o', 'r', 'l', 'd' });
        fw.write("\n");
        fw.write("Hello", 1, 4);
        fw.write(new char[] { 'W', 'o', 'r', 'l', 'd' }, 1, 4);
        /*
        파일 입력 내용
        HelloWorld
        elloorld
        */
        fw.close();

        // 파일을 읽되 문자열을 중심으로 내용을 읽어오게 됨
        FileReader fr = new FileReader("temp" + File.separator +
"writer_test.txt");
        // 첫 글자 읽어오기
        char first = (char) fr.read();
        System.out.println(first);
        char[] buffer = new char[255];
        int readChars = fr.read(buffer);
        System.out.println("read chars : " + readChars);
        for(int i=0;i<readChars;i++) System.out.print(buffer[i]);
        System.out.println("\n");
        fr.close();
    }
}
```

InputStreamReader, OutputStreamWriter

-

```
public class InputStreamReaderWriterDemo {
    public static void main(String[] args) throws IOException {
        FileOutputStream fos = new FileOutputStream("temp" + File.separator +
"osw_test.txt");
        // OutputStreamWriter의 경우 직접 인코딩을 지정 가능
        OutputStreamWriter osw = new OutputStreamWriter(fos, "UTF-8");
        osw.write("abc안녕");
        osw.close();

        FileInputStream fis = new FileInputStream("temp" + File.separator +
"osw_test.txt");
        // 반드시 생성한 시점의 인코딩과 같은 인코딩으로 지정해야 함
        InputStreamReader isr = new InputStreamReader(fis, "UTF-8");
        char[] buffer = new char[512];
        // 읽어낸 문자의 개수를 반환
        int readChars = isr.read(buffer);
        System.out.println("read bytes : " + readChars);
        String s = new String(buffer, 0, readChars);
        System.out.println(s);
        isr.close();
    }
}
```

CharArrayReader, CharArrayWriter

- 바이트 기반 스트림인 ByteArrayInputStream, ByteArrayOutputStream의 문자 버전이라고 봐도 무방함

보조 스트림 활용

BufferedInputStream, BufferedOutputStream, BufferedReader, BufferedWriter

- 버퍼를 이용하여 더 효율적인 입출력을 지원하도록 설계된 보조 클래스

```
public class BufferedInputOutputStreamDemo {
    public static void main(String[] args) throws IOException {
        // 버퍼 스트림은 모두 보조 스트림 (따라서 다른 소스가 될 수 있는 스트림이 반드시
필요함)
        // 버퍼 사이즈는 4바이트로 지정 (이렇게 작게 지정하면 안되며, 일반적으로 버퍼 사이
즈는 4k, 8k 정도로 지정 (4096, 8192), 참고로 기본 버퍼사이즈는 8192 bytes)
        // http://stackoverflow.com/questions/236861/how-do-you-determine-the-
ideal-buffer-size-when-using-fileinputstream
```

```
        BufferedOutputStream bos = new BufferedOutputStream(new
FileOutputStream("temp" + File.separator + "out4.txt"), 4);

        /*
        // 버퍼가 채워져야 쓰기때문에 4번의 반복이 있을때마다 output 파일에 결과 확인 가능
        // 6번을 'A'를 썼으나 실제로 확인해보면 4개의 A만 있음!
        for(int i=0;i<6;i++) bos.write(65);
        */

        /*
        // 남은 버퍼에 있는 것들을 flush 메소드 호출하여 쓸 수 있음
        for(int i=0;i<6;i++) bos.write(65);
        bos.flush();
        */

        /*
        // BufferedOutputStream 클래스에서 close 메소드 호출 시 flush 메소드도 자동 호
출하므로 사실은 close만 호출해도 무관하지만 close 메소드 호출 전 반드시 flush를 호출할 수
있도록 하는 것을 권장
        // http://stackoverflow.com/questions/2732260/in-java-when-i-call-
outputstream-close-do-i-always-need-to-call-outputstream
        for(int i=0;i<6;i++) bos.write(65);
        bos.close();
        */

        // 가장 일반적인 용례
        // 1. 내용을 쓰고
        for(int i=0;i<6;i++) bos.write(65 + i);
        // 2. 모두 다 썼으면 flush 호출하여 남은 버퍼에 있는 내용을 써주고
        bos.flush();
        // 3. 자원 반환
        bos.close();

        // BufferedInputStream 사용시 버퍼를 경유하여 바이트를 읽어들임
        BufferedInputStream bis = new BufferedInputStream(new
FileInputStream("temp" + File.separator + "out4.txt"), 4);

        int readByte;
        while((readByte = bis.read()) != -1) {
            System.out.println(String.format("%X : %d", readByte,
bis.available()));
        }
        bis.close();
    }
}
```

DataInputStream, DataOutputStream

- 자바에서 제공하는 원시형 값들과 문자열을 저장할 수 있도록 설계된 보조 클래스

```
public class DataInputOutputStreamDemo {
    public static void main(String args[]) throws IOException {
        FileOutputStream fos = new FileOutputStream("temp" + File.separator +
"data_output_stream_test.bin");
        // DOS 단독으로는 사용되지 않으며 다른 Stream을 보조하는 역할 수행
```

```
        DataOutputStream dos = new DataOutputStream(fos);
        // 여러 primitive 값과 String 값을 쓸 수 있는 "write타입이름" 메소드 제공
        // 단, 객체는 쓸 수 없음! (객체는 직렬화를 통하여 저장 가능)
        dos.writeBoolean(true); // false(00), true(01)
        dos.writeByte(127);
        // UTF-16 인코딩으로 문자열 저장
        dos.writeChar('c');
        dos.writeInt(100);
        // 처음 2 바이트는 총 바이트 개수, 그 이후 UTF-8 인코딩 형식으로 바이트 저장
        dos.writeUTF("가나다");
        dos.close();

        DataInputStream dis = new DataInputStream(new FileInputStream("temp" +
File.separator + "data_output_stream_test.bin"));
        // 저장된 순서와 같은 순서로 읽어야 함
        Boolean b = dis.readBoolean();
        System.out.println(b);
        System.out.println(dis.readByte());
        System.out.println(dis.readChar());
        System.out.println(dis.readInt());
        System.out.println(dis.readUTF());
        dis.close();
    }
}
```

FilterInputStream, FilterOutputStream

-

RandomAccessFile

-

```
public class RandomAccessFileDemo {
    public static void main(String args[]) throws IOException {
        System.out.println("RandomAccessFile Example");

        FileOutputStream fos = new FileOutputStream("temp" + File.separator +
"rout.bin");

        for(int i=0;i<256;i++) {
            // 0 (00) ~ 255 (FF)까지 각 바이트값을 기록
            fos.write(i);
        }

        fos.close();

        // RandomAccessFile 인스턴스 생성
        /*
        RandomAccessFile의 특징

        일반적으로 InputStream, OutputStream이 각각 입출력을 담당하며 "순차적"으로 바이
트를 읽거나 내보내는 것과 다르게
```

```
        RandomAccessFile 클래스를 이용하여 특정 offset의 바이트를 "임의 접근"(random
access)할 수 있으며 또한 입, 출력을
        동시에 사용 가능함!
        */
        RandomAccessFile raf = new RandomAccessFile("temp" + File.separator +
"rout.bin", "rw");

        for(int i=0;i<raf.length();i++) {
            // seek 메소드 사용하여 바이트 offset을 지정 가능
            raf.seek(i); // 내부적으로 참조 위치를 변경함 (따로 반환형은 없음)
            int data = raf.read(); // 현재 지정된 offset에서 1바이트 read
            System.out.printf("offset " + i + " : %x\n", data);
        }

        // 완전히 random하게 바이트 읽어오기 가능
        raf.seek(0);
        System.out.printf("%x\n", raf.read());
        raf.seek(255);
        System.out.printf("%x\n", raf.read());
        raf.seek(128);
        System.out.printf("%x\n", raf.read());

        // 처음 16바이트를 00으로 채우기
        // 다시 처음 byte를 지정하도록 offset 변경
        raf.seek(0);
        System.out.println("current offset : " + raf.getFilePointer()); // 현재
offset은 0
        for(int i=0;i<16;i++) {
            // write 메소드 호출하면 해당 offset에 byte를 써주고 다음 offset으로 이동함
            raf.write(0);
        }

        // 현재 offset에서 16바이트를 이동하고 00으로 채움
        raf.skipBytes(16);
        System.out.println("current offset : " + raf.getFilePointer()); // 현재
offset은 32 (hex값으로 20)
        raf.write(0);

        // 파일 크기 지정 가능 (늘어난 크기 만큼 바이트를 00으로 채움)
        raf.setLength(512);

        raf.close();
    }
}
```

File 클래스

-

```
public class FileClassDemo {
    public static void main(String args[]) throws IOException {
        System.out.println("File Class Example");
        String userName = "Mirim";

        // username은 자기 컴퓨터 이름으로 적절히 수정
```

```
        // Mac을 사용하는 경우 아예 path를 새로 써야 함!
        File f = new File("C:" + File.separator + "Users" + File.separator +
userName + File.separator + "Downloads");

        // 해당 폴더 혹은 파일이 존재하는지 여부를 돌려줌
        System.out.println("f.exists() : " + f.exists());

        // 해당 폴더 혹은 파일의 절대 경로를 얻어냄 (가령 윈도우의 경우 C: 혹은 D: 시작하
는 경로를 얻어냄)
        System.out.println("f.getAbsolutePath() : " + f.getAbsolutePath());

        // 파일인지 여부를 돌려줌
        System.out.println("f.isFile() : " + f.isFile());

        // 디렉토리인지 여부를 돌려줌
        System.out.println("f.isDirectory() : " + f.isDirectory());

        File newDir = new File("C:\\Users\\" + userName + "\\Downloads\\Hello");

        // 디렉토리를 생성 (하지만 해당 디렉토리를 생성할 부모 디렉토리가 존재하지 않으면
실패함!)
        if(newDir.mkdir()) System.out.println("디렉토리 생성 완료");

        // 해당 파일, 디렉토리를 제거
        // 디렉토리의 경우 디렉토리 내부에 파일이나 다른 디렉토리 존재하면 지울 수 없음
        // (따라서 재귀적으로 내부의 모든 파일과 디렉토리를 제거한 후 제거하는 로직이 필요
함)
        // 혹은 라이브러리 사용 가능
        // http://commons.apache.org/proper/commons-io/javadocs/api-
release/org/apache/commons/io/FileUtils.html#deleteDirectory(java.io.File)
        if(newDir.delete()) System.out.println("파일, 디렉토리 제거 완료");

        // 디렉토리 검사
        if(f.isDirectory()) {
            /*
            // 단순히 문자열만 필요하다면
            String[] dirList = f.list();
            for(String item : dirList) {
                System.out.println(item);
            }
            */

            /*
            // 파일 정보가 필요하다면
            File[] files = f.listFiles();
            for(File itemFile : files) {
                // 폴더 사이즈는 0으로 뜸
                // http://stackoverflow.com/questions/2149785/get-size-of-
folder-or-file
                System.out.println((itemFile.isDirectory()
                        ? "Directory : " : ("File (" + itemFile.length() + "
bytes) : "))
                        + itemFile.getName());
            }
            */
        }
    }
}
```

- https://crystalcube.co.kr/123