스트림과 IO

입출력(I/O) 이란?

- 입력과 출력을 줄여 부르는 말
- 두 대상 간의 데이터를 주고 받는 것

스트림(Stream) 이란?

- 데이터를 운반(입출력)하는데 사용되는 연결 통로
- 연속적인 데이터의 흐름을 물에 비유해서 붙여진 이름
- 하나의 스트림으로 입출력을 동시에 수행할 수 없다 (단방향 통신)
- 입출력을 동시에 수행하려면, 2개의 스트림이 필요하다.

바이트기반 스트림 – InputStream, OutputStream

- 데이터를 바이트(byte) 단위로 주고 받는다.

InputStream	OutputStream
abstract **int** read()	abstract void write(**int** b)
int read(**byte[]** b)	void write(**byte[]** b)
int read(**byte[]** b, int off, int len)	void write(**byte[]** b, int off, int len)

입력스트림	출력스트림	대상
FileInputStream	**File**OutputStream	파일
ByteArrayInputStream	**ByteArray**OutputStream	메모리
PipedInputStream	**Piped**OutputStream	프로세스
AudioInputStream	**Audio**OutputStream	오디오장치

```
import java.io.*;
import java.util.Arrays;

class IOEx1 {
    public static void main(String[] args) {
          byte[] inSrc = {0,1,2,3,4,5,6,7,8,9};
          byte[] outSrc = null;

          ByteArrayInputStream input = null;
          ByteArrayOutputStream output = null;

          input = new ByteArrayInputStream(inSrc);
          output = new ByteArrayOutputStream();

          int data = 0;

          while((data = input.read())!=-1) {
               output.write(data);  // void write(int b)
          }

          outSrc = output.toByteArray(); // 스트림의 내용을 byte배열로 반환한다.

          System.out.println("Input Source  :" + Arrays.toString(inSrc));
          System.out.println("Output Source :" + Arrays.toString(outSrc));
    }
}
```

[실행결과]

```
Input Source  :[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
Output Source :[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

보조 스트림

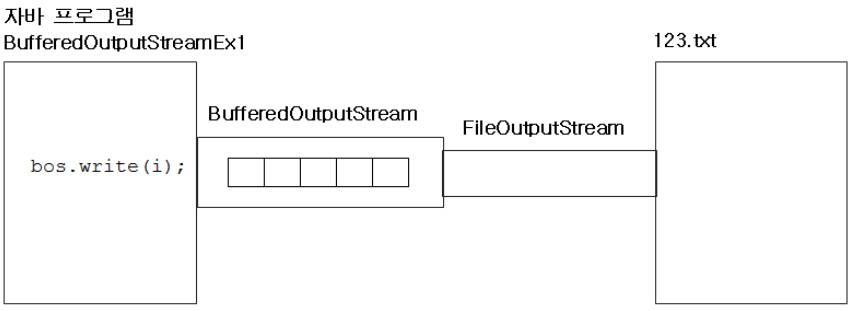
- 스트림의 기능을 향상시키거나 새로운 기능을 추가하기 위해 사용
- 독립적으로 입출력을 수행할 수 없다.

```
// 먼저 기반스트림을 생성한다.
FileInputStream fis = new FileInputStream("test.txt");
// 기반스트림을 이용해서 보조스트림을 생성한다.
BufferedInputStream bis = new BufferedInputStream(fis);

bis.read();  // 보조스트림인 BufferedInputStream으로부터 데이터를 읽는다.
```

```
import java.io.*;

class BufferedOutputStreamEx1 {
    public static void main(String args[]) {
        try {
            FileOutputStream fos = new FileOutputStream("123.txt");
            // BufferedOutputStream의 버퍼 크기를 5로 한다.
             BufferedOutputStream bos = new BufferedOutputStream(fos, 5);
            // 파일 123.txt에  1 부터 9까지 출력한다.
            for(int i='1'; i <= '9'; i++) {
                bos.write(i);
            }

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

[실행결과]

```
C:\jdk1.5\work\ch14>java BufferedOutputStreamEx1

C:\jdk1.5\work\ch14>type 123.txt
12345
```



문자 기반 스트림 – Reader, Writer

문자기반의 파일 입출력, 텍스트 파일의 입출력에 사용한다.

```
import java.io.*;

class FileReaderEx1 {
    public static void main(String args[]) {
        try {
            String fileName = "test.txt";
            FileInputStream fis = new FileInputStream(fileName);
            FileReader fr = new FileReader(fileName);

            int data =0;
            // FileInputStream을 이용해서 파일내용을 읽어 화면에 출력한다.
            while((data=fis.read())!=-1) {
                System.out.print((char)data);
            }
            System.out.println();
            fis.close();

            // FileReader를 이용해서 파일내용을 읽어 화면에 출력한다.
            while((data=fr.read())!=-1) {
                System.out.print((char)data);
            }
            System.out.println();
            fr.close();

        } catch (IOException e) {
                e.printStackTrace();
        }
    } // main
}
```

[실행결과]

```
C:\jdk1.5\work\ch14>type test.txt
Hello, 안녕하세요?

C:\jdk1.5\work\ch14>java FileReaderEx1
Hello, ¾?³???¼¼¿??
Hello, 안녕하세요?
```

File

생성자 / 메서드	설 명
File(String fileName)	주어진 문자열(fileName)을 이름으로 갖는 파일을 위한 File인스턴스를 생성한다. 파일 뿐만 아니라 디렉토리도 같은 방법으로 다룬다. 여기서 fileName은 주로 경로(path)를 포함해서 지정해주지만, 파일 이름만 사용해도 되는 데 이 경우 프로그램이 실행되는 위치가 경로(path)로 간주된다.
File(String pathName, String fileName) File(File pathName, String fileName)	파일의 경로와 이름을 따로 분리해서 지정할 수 있도록 한 생성자. 이 중 두 번째 것은 경로를 문자열이 아닌 File인스턴스인 경우를 위해서 제공된 것이다.
String getName()	파일이름을 String으로 반환한다.
String getPath()	파일의 경로(path)를 String으로 반환한다.
String getAbsolutePath() File getAbsoluteFile()	파일의 절대경로를 String으로 반환한다. 파일의 절대경로를 File로 반환한다.
String getParent() File getParentFile()	파일의 조상 디렉토리를 String으로 반환한다. 파일의 조상 디렉토리를 File로 반환한다.
String getCanonicalPath() File getCanonicalFile()	파일의 정규경로를 String으로 반환한다. 파일의 정규경로를 File로 반환한다.

멤버변수	설 명
static String pathSeparator	OS에서 사용하는 경로(path) 구분자. 윈도우 ";", 유닉스 ":"
static char pathSeparatorChar	OS에서 사용하는 경로(path) 구분자. 윈도우에서는 ';', 유닉스 ':'
static String separator	OS에서 사용하는 이름 구분자. 윈도우 "₩", 유닉스 "/"
static char separatorChar	OS에서 사용하는 이름 구분자. 윈도우 '₩', 유닉스 '/'

```
File f = new File("c:\\jdk1.5\\work\\ch14\\FileEx1.java");
String fileName = f.getName();
int pos = fileName.lastIndexOf(".");
```

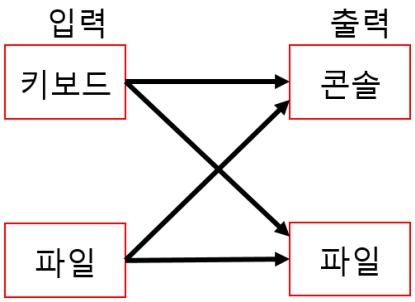
```
System.out.println("경로를 제외한 파일이름 - " + f.getName());
System.out.println("확장자를 제외한 파일이름 - " + fileName.substring(0,pos));
System.out.println("확장자 - " + fileName.substring(pos+1));
```

```
System.out.println("경로를 포함한 파일이름 - " + f.getPath());
System.out.println("파일의 절대경로       - " + f.getAbsolutePath());
System.out.println("파일이 속해 있는 디렉토리 - " + f.getParent());
```

```
경로를 제외한 파일이름 - FileEx1.java
확장자를 제외한 파일이름 - FileEx1
확장자 - java
```

```
경로를 포함한 파일이름 - c:\jdk1.5\work\ch14\FileEx1.java
파일의 절대경로       - c:\jdk1.5\work\ch14\FileEx1.java
파일의 정규경로       - C:\jdk1.5\work\ch14\FileEx1.java
파일이 속해 있는 디렉토리 - c:\jdk1.5\work\ch14
```

이제 아래 4가지의 예를 들어 실습해 보도록 합시다.



1. KeyboardToConsole.java

```
public class KeyboardToConsole {
  public static void main(String[] args) throws IOException {
    int readData = 0;
    while((readData = System.in.read()) != -1) {
       System.out.write(readData); // 버퍼에 모아 놓았다가 enter를 치면 출력됨.
    }
  }
}
```

2. KeyboardToFile.java

```
public class KeyboardToFile {
  public static void main(String[] args) {
    int readData = 0;
    FileOutputStream fos = null;

    try {
      fos = new FileOutputStream("c:\\Temp\\output.txt");
      while((readData = System.in.read()) != -1) {
        fos.write(readData);
      }

    } catch(Exception e) {
      e.printStackTrace();
    } finally {
      if(fos != null) try{fos.close(); } catch(Exception e){}
    }
  }
}
```

3. FileToConsole.java

```
public class FileToConsole {
  public static void main(String[] args) {
        int readData = 0;

    try(FileInputStream fis = new FileInputStream("c:\\Temp\\output.txt");) {
      while((readData = fis.read()) != -1) {
        System.out.write(readData);
      }
    } catch(Exception e) {
      e.printStackTrace();
    }
  }
}
```

4. FileToFile

```
public class FileToFile {
  public static void main(String[] args) {
    int readData = 0;
```

```
   try(FileInputStream fis = new FileInputStream("c:\\Temp\\output.txt");
     FileOutputStream fos = new FileOutputStream("c:\\Temp\\output2.txt");) {

     while((readData = fis.read()) != -1) {
       fos.write(readData);
     }
   } catch(Exception e) {
     e.printStackTrace();
   }
  }
}
```

보조 스트림인 Buffered 를 사용하여 성능 향상 예제 (copy 시간 확인해 보세요)

1. 하드디스크에 저장된 20M 크기의 파일을 copy 할 경우 소요되는 시간 측정

```
public class FileCopyPerformance {
  public static void main(String[] args) {
     long start = System.currentTimeMillis();
     copyFile("C:\\Program Files\\Java\\jdk1.8.0_45\\src.zip", "c:\\Temp\\javasrc.zip");
     long finish = System.currentTimeMillis();
     System.out.println("소요시간: " + (finish-start) + "ms");
   }

   private static void copyFile(String org, String dest) {
      int readData = 0;

      try(FileInputStream fis = new FileInputStream(org);
        FileOutputStream fos = new FileOutputStream(dest);) {

        while((readData = fis.read()) != -1) {
          fos.write(readData);
        }
      } catch(Exception e) {
        e.printStackTrace();
      }
     }
}
```

2. 하드디스크에 저장된 20M 크기의 파일을 보조스트림을 사용하여 copy 할 경우 소요되는 시간 측정

```
public class FileCopyPerformance {

  public static void main(String[] args) {
     long start = System.currentTimeMillis();
     copyFileUseBuffer("C:\\Program Files\\Java\\jdk1.8.0_45\\src.zip", "c:\\Temp\\javasrc.zip");
     long finish = System.currentTimeMillis();
     System.out.println("소요시간: " + (finish-start) + "ms");
  }

   private static void copyFileUseBuffer(String org, String dest) {
```

```
        int readData = 0;

        try(BufferedInputStream fis = new BufferedInputStream(new FileInputStream(org));
        BufferedOutputStream fos = new BufferedOutputStream(new FileOutputStream(dest));) {

        while((readData = fis.read()) != -1) {
           fos.write(readData);
        }
        } catch(Exception e) {
            e.printStackTrace();
        }
    }
}
```

3. 하드디스크에 저장된 20M 크기의 파일을 직접 보조스트림을 만들어서 copy 할 경우 소요되는 시간 측정

```
public class FileCopyPerformance {

  public static void main(String[] args) {
     long start = System.currentTimeMillis();
     copyFileCustomBuffer("C:\\Program Files\\Java\\jdk1.8.0_45\\src.zip", "c:\\Temp\\javasrc.zip");
     long finish = System.currentTimeMillis();
     System.out.println("소요시간: " + (finish-start) + "ms");
  }

private static void copyFileCustomBuffer(String org, String dest) {
    int readSize = 0;

    try(FileInputStream fis = new FileInputStream(org);
      FileOutputStream fos = new FileOutputStream(dest);) {

      byte[] buff = new byte[1024*8];

      while((readSize = fis.read(buff)) != -1) {
        fos.write(buff, 0, readSize);   // 마지막 읽은 사이즈 만큼
      }
    } catch(Exception e) {
      e.printStackTrace();
    }
  }
}
```

수고했습니다. 스트림은 특히, 파일입출력, 네트워크 프로그래밍에 많이 사용되고 있음으로 특징 위주로 이해하고 있으면 될 것 같습니다.