■ 입출력 범위, 입출력 모델

일반적인 입출력의 대상

- 키보드와 모니터
- 하드디스크에 저장되어 있는 파일
- USB와 같인 외부 메모리 장치
- 네트워크로 연결되어 있는 컴퓨터
- 사운드카드, 오디오카드와 같은 멀티미디어 장치
- 프린터, 팩시밀리와 같은 출력장치

입출력 대상이 달라지면 프로그램상에서의 입출력 방식도 달라지는 것이 보통이다. 그런데 자바에서는 입출력 대상에 상관없이 입출력의 진행 방식이 동일하도록 별도의 'I/O 모델'을 정의하고 있다.

I/O 모델의 정의로 인해서 입출력 대상의 차이에 따른 입출력 방식의 차이는 크지 않다. 기본적인 입출력의 형태는 동일하다. 그리고 이것이 JAVA의 I/O 스트림이 갖는 장점이다.

■ Byte Stream

- 바이트 스트림은 1byte 단위로 입출력
- 일반적으로 동영상이나 이미지 파일과 같은 바이너리 파일에 대한 처리

■ Character Stream

- 문자 스트림은 2byte 단위로 입출력
- 일반적으로 문자나 문자열 파일과 같은 텍스트 파일에 대한 처리

■ 파일 대상 바이트 입력 스트림

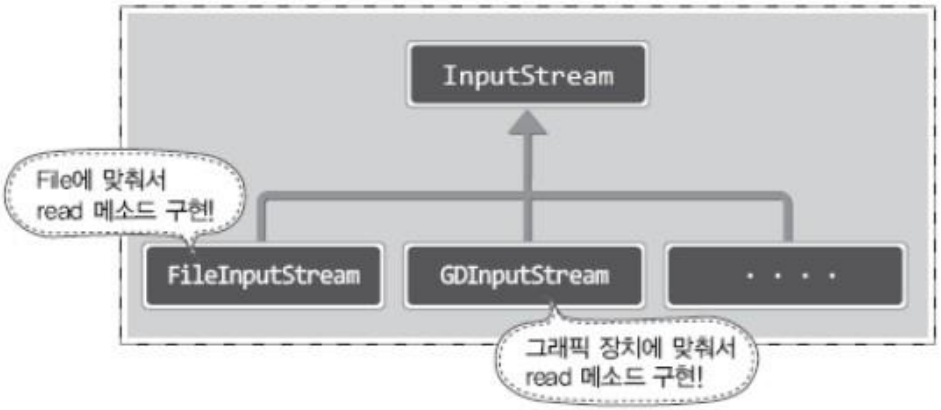
파일 run.exe 대상의 입력 스트림 생성

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

✈ 스트림의 생성은 결국 인스턴스의 생성.

✈ FileInputStream 클래스는 InputStream 클래스를 상속한다.

InputStream 클래스는 모든 입력 스트림 클래스의 최상위 클래스



InputStream 클래스의 대표적인 두 메소드

- `public abstract int read() throws IOException`
- `public void close() throws IOException`

이렇듯 입력의 대상에 적절하게 read 메소드가 정의되어 있다. 그리고 입력의 대상에 따라서 입력 스트림을 의미하는 별도의 클래스가 정의되어 있다.

파일 대상의 입력 스트림 생성

```
InputStream in=new FileInputStream("run.exe");
int bData=in.read();    // 오버라이딩에 의해 FileInputStream의 read 메소드 호출!
```

■ 파일 대상 바이트 출력 스트림

InputStream	↔	OutputStream
FileInputStream	↔	FileOutputStream

입출력 스트림은 대부분 쌍(Pair)을 이룬다.

OutputStream 클래스의 대표적인 메소드

- public abstract void write(int b) throws IOException
- public void close() throws IOException

```
OutputStream out=new FileOutputStream("home.bin");
out.write(1);    // 4바이트 int형 정수 1의 하위 1바이트만 전달된다.
out.write(2);    // 4바이트 int형 정수 2의 하위 1바이트만 전달된다.
out.close;       // 입력 스트림 소멸
```

파일 대상의 출력 스트림 생성 및 데이터 전송

■ Byte Stream 사용 – 1

```
public class FileIOStreamExam1 {
    public static void main(String[] args) {
        InputStream in = null;
        OutputStream out = null;
        try {
            in = new FileInputStream("c:/test.bin");
            out = new FileOutputStream(
                        "c:/copy_test.bin");
            int bData = 0;
            while (true) {
                bData = in.read();
                if (bData == -1) {
                    break;
                }
                out.write(bData);
            }
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            if(in != null) {
                try {
                    in.close();
                } catch (IOException e) {}
            }
            if(out != null) {
                try {
                    out.close();
                } catch (IOException e) {}
            }
        }
    }
}
```

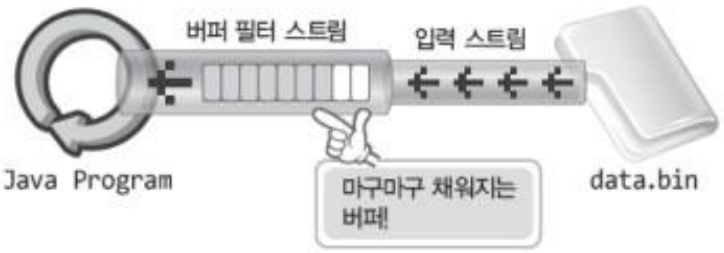
■ Byte Stream 사용 – 2 (Buffer 사용)

```
public class FileIOStreamExam2 {
    public static void main(String[] args) {
        InputStream in = null;
        OutputStream out = null;
        try {
            in = new FileInputStream("c:/test.bin");
            out = new FileOutputStream(
                        "c:/copy_test.bin");
            int bData = 0;
            byte buf[] = new byte[1024];
            while (true) {
                bData = in.read(buf);
                if (bData == -1) {
                    break;
                }
                out.write(buf);
            }
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            if(in != null) {
                try {
                    in.close();
                } catch (IOException e) {}
            }
            if(out != null) {
                try {
                    out.close();
                } catch (IOException e) {}
            }
        }
    }
}
```

■ 버퍼링 기능을 제공하는 보조 스트림

```
public static void main(String[] args) throws IOException
{
    InputStream in=new FileInputStream("org.bin");
    OutputStream out=new FileOutputStream("cpy.bin");
    BufferedInputStream bin=new BufferedInputStream(in);
    BufferedOutputStream bout=new BufferedOutputStream(out);
    int copyByte=0;
    int bData;
    while(true)
    {
        bData=bin.read();
        if(bData==-1)
            break;
        bout.write(bData);
        copyByte++;
    }
    bin.close();
    bout.close();
    System.out.println("복사된 바이트 크기 "+ copyByte);
}
```

버퍼링 되므로,
read, write 함수의 호출이 빠르게 진행된다.



- BufferedInputStream 버퍼 필터 입력 스트림
- BufferedOutputStream 버퍼 필터 출력 스트림

BufferedInputStream은 입력버퍼,
BufferedOutputStream은 출력버퍼 제공!

BufferedOutputStream 클래스의 flush 메소드 호출을 통해서 버퍼링 된 데이터의 목적지 전송이 가능하다! 또한 close 메소드를 통해서 스트림을 종료하면, 스트림의 버퍼는 flush! 된다.

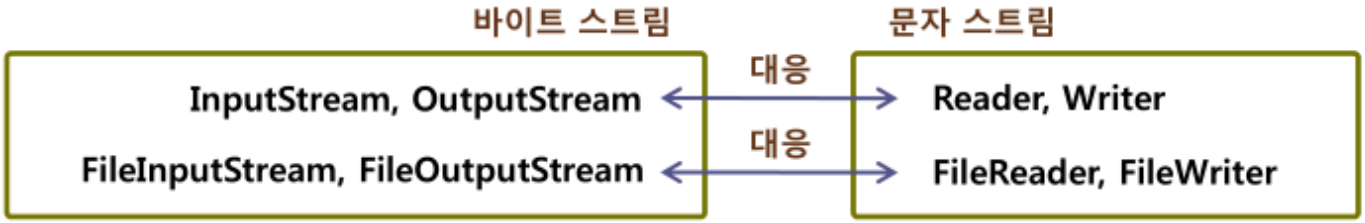
■ 보조 스트림 사용 – 1

```
public class FileIOStreamExam3 {
    public static void main(String[] args) {
        InputStream in = null;
        OutputStream out = null;
        BufferedInputStream bis = null;
        BufferedOutputStream bos = null;
        try {
            in = new FileInputStream("c:/test.bin");
            out = new FileOutputStream(
                        "c:/copy_test.bin");
            bis = new BufferedInputStream(in);
            bos = new BufferedOutputStream(out);
            int bData = 0;
            while((bData = bis.read()) > -1) {
                bos.write(bData);
            }
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            if(bis != null) {
                try {
                    bis.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
            /* bos close */
            /* in close */
            /* out close */
        }
    }
}
```

■ 보조 스트림 사용 – 2

```
public class FileIOStreamExam4 {
    public static void main(String[] args) {
        InputStream in = null;
        OutputStream out = null;
        BufferedInputStream bis = null;
        BufferedOutputStream bos = null;
        try {
            in = new FileInputStream("c:/test.bin");
            out = new FileOutputStream(
                        "c:/copy_test.bin");
            bis = new BufferedInputStream(in);
            bos = new BufferedOutputStream(out);
            int bData = 0;
            byte buf[] = new byte[1024];
            while((bData = bis.read(buf)) > -1) {
                bos.write(buf);
            }
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            if(bis != null) {
                try {
                    bis.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
            /* bos close */
            /* in close */
            /* out close */
        }
    }
}
```

■ 파일 대상 문자 입/출력 스트림



Reader의 대표적인 메소드

```
public int read() throws IOException
public abstract int read(char[] cbuf, int off, int len) throws IOException
```

Writer의 대표적인 메소드

```
public void write(int c) throws IOException
public abstract void write(char[] cbuf, int off, int len) throws IOException
```

```
public static void main(String[] args) throws IOException
{
    char ch1='A';
    char ch2='B';

    Writer out=new FileWriter("hyper.txt");
    out.write(ch1);
    out.write(ch2);
    out.close();
}
```

자바에서는 각각 2바이트로 표현됨

실행 운영체제에 따라서 1바이트씩 인코딩 되어서 저장!

```
public static void main(String[] args) throws IOException
{
    char[] cbuf=new char[10];
    int readCnt;

    Reader in=new FileReader("hyper.txt");
    readCnt=in.read(cbuf, 0, cbuf.length);
    for(int i=0; i<readCnt; i++)
        System.out.println(cbuf[i]);

    in.close();
}
```

읽어 들이는 과정에서 1바이트 데이터가 2바이트로 표현될 수 있다.

■ Character Stream 사용 – 1 (Reader / Writer)

```
public class FileRWExam1 {
    public static void main(String[] args) {
        Reader reader = null;
        Writer writer = null;
        try {
            reader = new FileReader("c:/test.bin");
            writer = new FileWriter(
                                "c:/copy_test.bin");
            int bData = 0;
            while ((bData = reader.read()) > -1) {
                writer.write(bData);
            }
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            if(reader != null) {
                try {
                    reader.close();
                } catch (IOException e) {}
            }
            if(writer != null) {
                try {
                    writer.close();
                } catch (IOException e) {}
            }
        }
    }
}
```

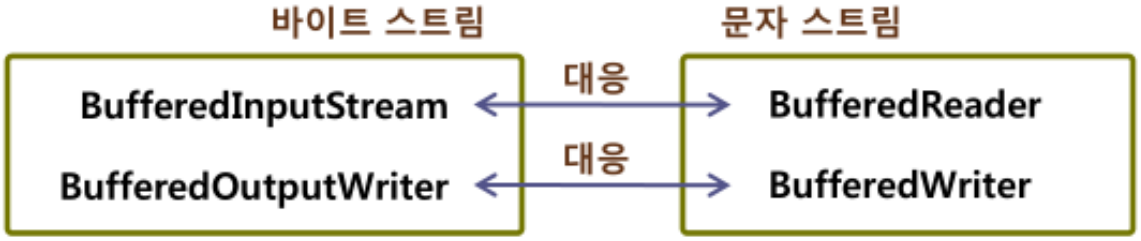
■ Character Stream 사용 – 2 (Writer)

```
public class FileRWExam2 {
    public static void main(String[] args) {
        FileWriter fw = null;
        try {
            fw = new FileWriter("c:/text.txt");
            String text = "나는 눈이 좋아서 꿈에 눈이 오나봐₩n" +
                          "온세상이 모두 하얀 나라였지 어젯밤 꿈속에~";
            fw.write(text);
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            if(fw != null) {
                try {
                    fw.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }
    }
}
```

■ Byte Stream 사용 – 3

```
public class FileIOStreamExam5 {
    public static void main(String[] args) {
        FileOutputStream fos = null;
        try {
            fos = new FileOutputStream("c:/text2.txt");
            String text = "동해물과 백두산이 마르고 닳도록₩n" +
                          "하느님이 보우하사 우리나라 만세";
            fos.write(text.getBytes());
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            /* fos close */
        }
    }
}
```

■ BufferedReader, BufferedWriter



- 문자열의 입력

  BufferedReader 클래스의 다음 메소드

  ```
  public String readLine() throws IOException
  ```

- 문자열의 출력

  Writer 클래스의 다음 메소드

  ```
  public void write(String str) throws IOException
  ```

일관성 없는 문자열의 입력방식과 출력방식!
그러나 문자열의 입력뿐만 아니라 출력도 버퍼링의 존재는 도움이 되므로 입력과 출력 모두에 버퍼링 필터를 적용하자!

문자열 출력을 위한 스트림의 구성

```
BufferedWriter out= new BufferedWriter(new FileWriter("Strint.txt"));
```

문자열 입력을 위한 스트림의 구성

```
BufferedReader in= new BufferedReader(new FileReader("Strint.txt"));
```

■ 일반적으로 많이 사용되는 파일 읽기 코드

```
public class FileIOStreamExam6 {
    public static void main(String[] args) {
        InputStream in = null;
        InputStreamReader isr = null;
        BufferedReader br = null;
        try {
            in = new FileInputStream("c:/text.txt");
            isr = new InputStreamReader(in);
            br = new BufferedReader(isr);
            String data = "";
            while ((data = br.readLine()) != null) {
                System.out.println(data);
            }
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            if(br != null) {
                try {
                    br.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
            if(isr != null) {
                try {
                    isr.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
            if(in != null) {
                try {
                    in.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }
    }
}
```

■ File 클래스

- 디렉터리의 생성, 소멸
- 파일의 소멸
- 디렉터리 내에 존재하는 파일이름 출력

File 클래스가 지원하는 기능

```
public static void main(String[] args) throws IOException
{
    File myDir=new File("C:\\YourJava\\JavaDir");    // 디렉터리 위치 정보
    myDir.mkdir();   // 디렉터리 생성
    . . . .
}
```

디렉터리 생성의 예

```
public static void main(String[] args) throws IOException
{
    File myFile=new File("C:\\MyJava\\my.bin");
    File reFile=new File("C:\\YourJava\\my.bin");
    myFile.renameTo(reFile);      // 파일의 이동
    . . . .
}
```

파일 이동의 예

renameTo 는 파일의 이름을 변경하는 메소드인데, 경로의 변경에 사용이 가능하다.

```
File myFile=
    new File("C:"+File.separator+"MyJava"+File.separator+"my.bin");
if(myFile.exists()==false)
{
    System.out.println("원본 파일이 준비되어 있지 않습니다.");
    return;
}
```

File.separator는 운영체제에 따른 구분자로 각각 치환된다.

■ File 클래스 기반 I/O 스트림 생성

- `public FileInputStream(File file)` // FileInputStream의 생성자
  `throws FileNotFoundException`
- `public FileOutputStream(File file)` // FileOutputStream의 생성자
  `throws FileNotFoundException`
- `public FileReader(File file)` // FileReader의 생성자
  `throws FileNotFoundException`
- `public FileWriter(File file)` // FileWriter의 생성자
  `throws IOException`

```
File inFile=new File("data.bin");
if(inFile.exists()==false)
{
    // 데이터를 읽어 들일 대상 파일이 존재하지 않음에 대한 적절한 처리
}
InputStream in=new FileInputStream(inFile);
```

■ File 클래스 사용

```
public class FileExam {
    public static void main(String[] args) {
        // 파일 확인
        File file = new File("c:/test.bin");
        System.out.println(file.isFile());
        // 파일 생성
        File file2 = new File("c:/test2.bin");
        try {
            boolean isSuccess =
                    file2.createNewFile();
            System.out.println("결과 : " + isSuccess);
        } catch (IOException e) {
            e.printStackTrace();
        }
        // 디렉토리 생성
        File dir1 = new File("c:/temp");
        if(!dir1.isDirectory()) {
            boolean isSuccess = dir1.mkdir();
            System.out.println("결과 : " + isSuccess);
        }
        // 하위 디렉토리 생성
        File dir2 = new File("c:/test/test");
        if(!dir2.isDirectory()) {
            boolean isSuccess = dir2.mkdirs();
            System.out.println("결과 : " + isSuccess);
        }
        // 파일 이동
        file.renameTo(new File("c:/temp/test.bin"));
    }
}
```

■ 상대경로 기반

실제 프로그램 개발에서는 절대경로가 아닌 상대경로를 이용하는 것이 일반적이다. 그래야 실행환경 및 실행위치의 변경에 따른 문제점을 최소화할 수 있기 때문이다.

```
File subDir1=new File("AAA");  현재 디렉터리 기준으로...
File subDir2=new File("AAA\\BBB"); 현대 디렉터리에 존재하는 AAA의 하위 디렉터리인 BBB...
File subDir3=new File("AAA"+File.separator+"BBB"); AAA\\BBB의 운영체제 독립버전...
```

```
class RelativePath
{
    public static void main(String[] args)
    {
        File curDir=new File("AAA");
        System.out.println(curDir.getAbsolutePath());

        File upperDir=new File("AAA"+File.separator+"BBB");
        System.out.println(upperDir.getAbsolutePath());
    }
}
```

위의 예제는 운영체제에 상관없이 실행이 가능하다!

실행결과

```
C:\MyJava\YourJava>java RelativePath
C:\MyJava\YourJava\AAA
C:\MyJava\YourJava\AAA\BBB
```

■ 상대경로 사용

```
public class RelativePath {
    public static void main(String[] args) {
        File curDir = new File("AAA");
        System.out.println(curDir.getAbsolutePath());

        File upperDir = new File("AAA" + File.separator + "BBB");
        System.out.println(upperDir.getAbsolutePath());
    }
}
```

■ 하위 경로 파일 탐색 – 1 (재귀호출)

```
public class RecursiveCall {
    public static void main(String[] args) {
        System.out.println("재귀호출 시작");
        int number = 5;
        long result = factorial(number);
        System.out.println("재귀호출 결과 : " + result);
        System.out.println("재귀호출 끝");
    }

    public static long factorial(int n) {
        long result = 0;
        if(n == 1) {
            result = 1;
        } else {
            result = n * factorial(n - 1);
        }
        return result;
    }
}
```

■ 하위 경로 파일 탐색 – 2 (재귀호출)

```
public class FileSearch {
    public static void main(String[] args) {
        FileSearch fs = new FileSearch();
        fs.subDirList("c:/Windows");
    }

    public void subDirList(String source) {
        File dir = new File(source);
        File[] fileList = dir.listFiles();
        for (int i = 0; i < fileList.length; i++) {
            File file = fileList[i];
            if (file.isFile()) {
                System.out.println("₩t 파일 이름 = " + file.getName());
            } else if (file.isDirectory()) {
                System.out.println("디렉토리 이름 = " + file.getName());
                subDirList(file.getAbsolutePath());
            }
        }
    }
}
```

■ 하위 경로 및 모든 파일 삭제

```
public class FileDelete {
    public static void main(String[] args) {
        FileDelete dd = new FileDelete();
        dd.deleteDirectory("c:/test");
    }

    public boolean deleteDirectory(String path) {
        File file = new File(path);
        if(!file.exists()) {
            return false;
        }

        File[] files = file.listFiles();
        for(File f : files) {
            if(f.isDirectory()) {
                deleteDirectory(f.getAbsolutePath());
            } else {
                f.delete();
            }
        }
        return file.delete();
    }
}
```