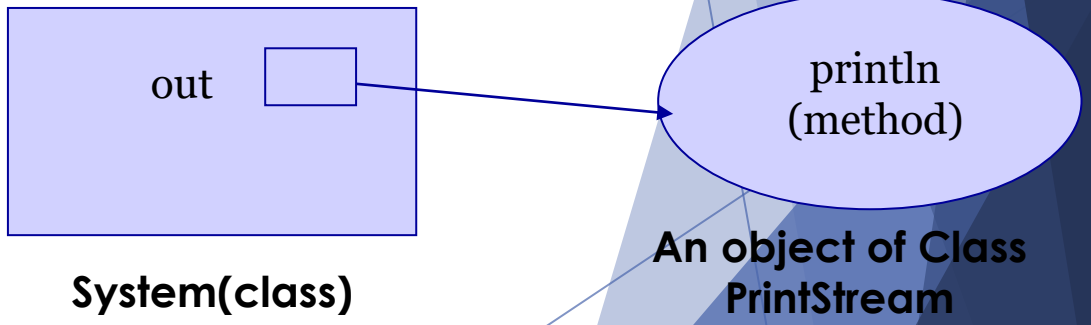
Java I/O Programming

Stream and I/O

Stream and I/O
: I/O

- I/O (Input and Output: 입출력)
  - 프로그램과 외부 소스 혹은 목적지와의 데이터 및 정보의 교환
  - 입력 : 키보드, 파일 등으로부터 프로그램으로 들어오는 데이터
  - 출력 : 화면, 파일 등으로 프로그램으로부터 나가는 데이터

```
class HelloJAVA {
    public static void main (String[] args) {
            System.out.println("Hello, JAVA");
    }
}
```

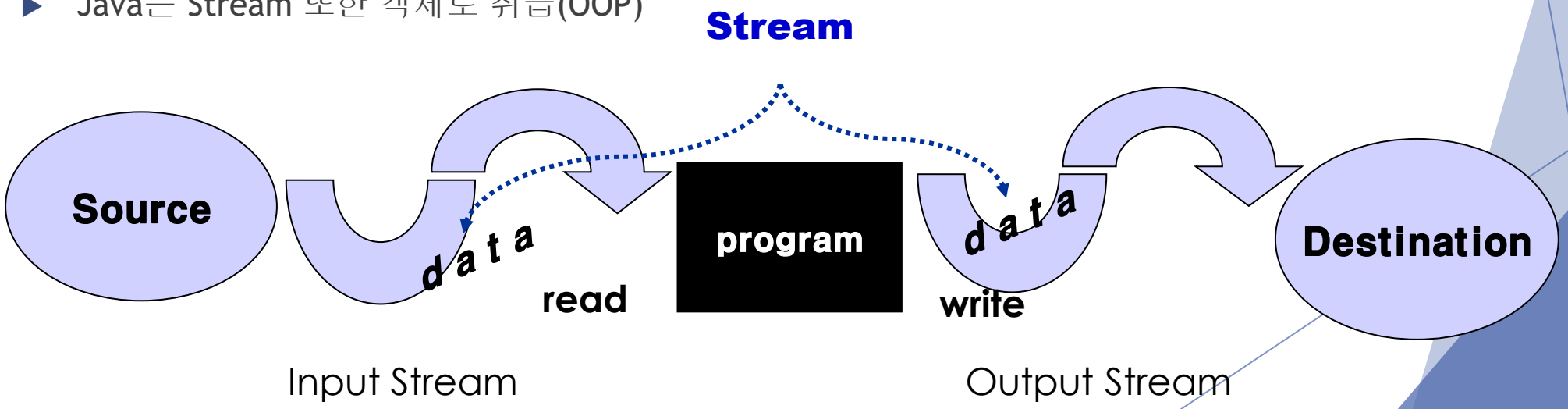


- System.out – PrintStream 객체를 참조하는 System Class의 static 멤버
- System.in – InputStream 객체를 참조하는 System Class의 static 멤버

Stream and I/O

: Stream

- 프로그램과 I/O 객체를 연결, 데이터를 소스에서 전달하거나 목적지로 수신하는 길 (Path)
  - 목적지/소스: 키보드 및 스크린을 포함한 파일, Network Connections, 다른 프로그램 등
- 일방향 (Unidirectional)으로 프로그램과 I/O 객체를 연결
  - InputStream - 데이터를 읽어들이는 객체 ex) System.in
  - OutputStream - 데이터를 써서 내보내는 객체 ex) System.out
- Java는 Stream 또한 객체로 취급(OOP)



Stream and I/O

: Byte Stream vs Character Stream

- 바이트 스트림 (Byte Stream)
  - 1과 0으로 구성된 바이너리 데이터의 입출력 처리 ex) 이미지, 사운드 등
- 문자 스트림 (Character Stream)
  - 문자, 텍스트 형태의 데이터 입출력 처리를 위한 스트림 ex) 텍스트, 웹페이지, 키보드 입력 등
- Java I/O 클래스
  - Java.io 패키지에 I/O를 위한 4개의 추상 클래스 정의
  - 해당 클래스에서 상속받아 파일에 데이터를 읽고 씀

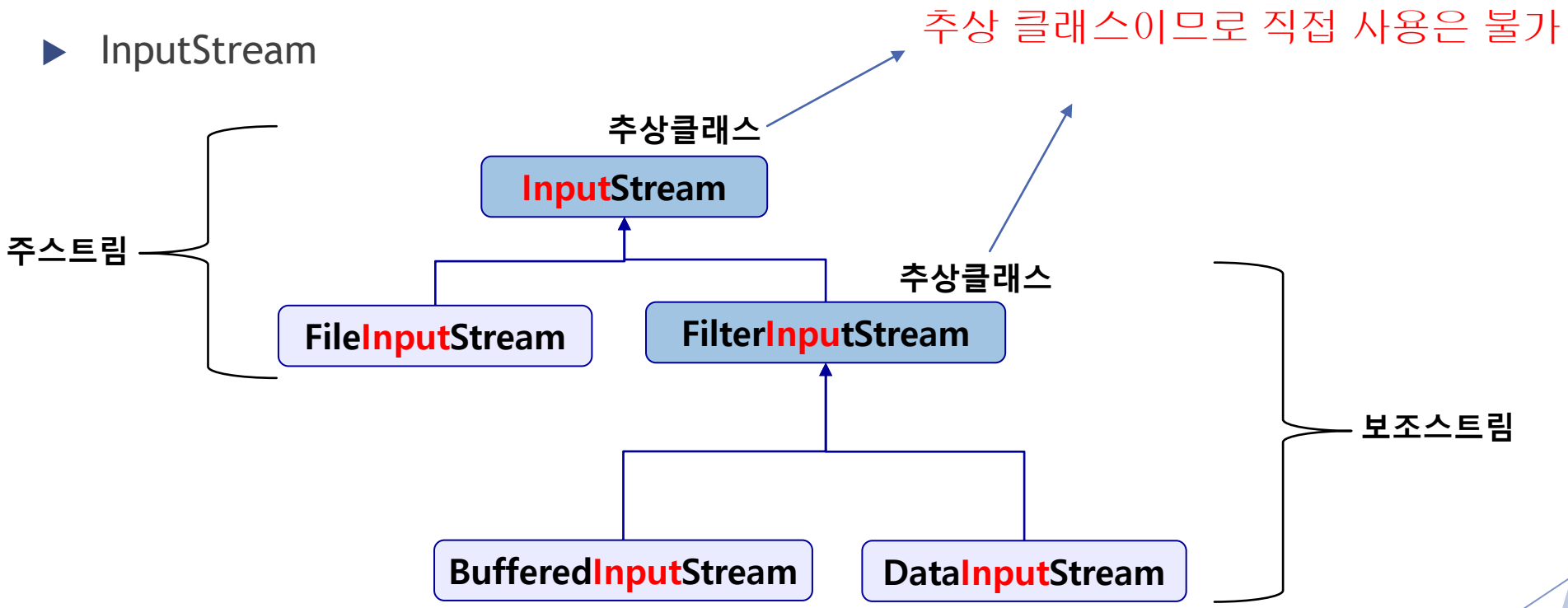
InputStream	Reader
abstract int read() int read(**byte[] b**) int read(**byte[] b**, int off, int len)	int read() int read(**char[]** cbuf) **abstract** int read(**char[]** cbuf, int off, int len)
OutputStream	**Writer**
abstract void write(int b) void write(**byte[]** b) void write(**byte[]** b, int off, int len)	void write(int c) void write(**char[]** cbuf) **abstract** void write(**char[]** cbuf, int off, int len) void write(String str) void write(String str, int off, int len)

바이트 스트림용 문자 스트림용

Stream and I/O

: Byte Stream

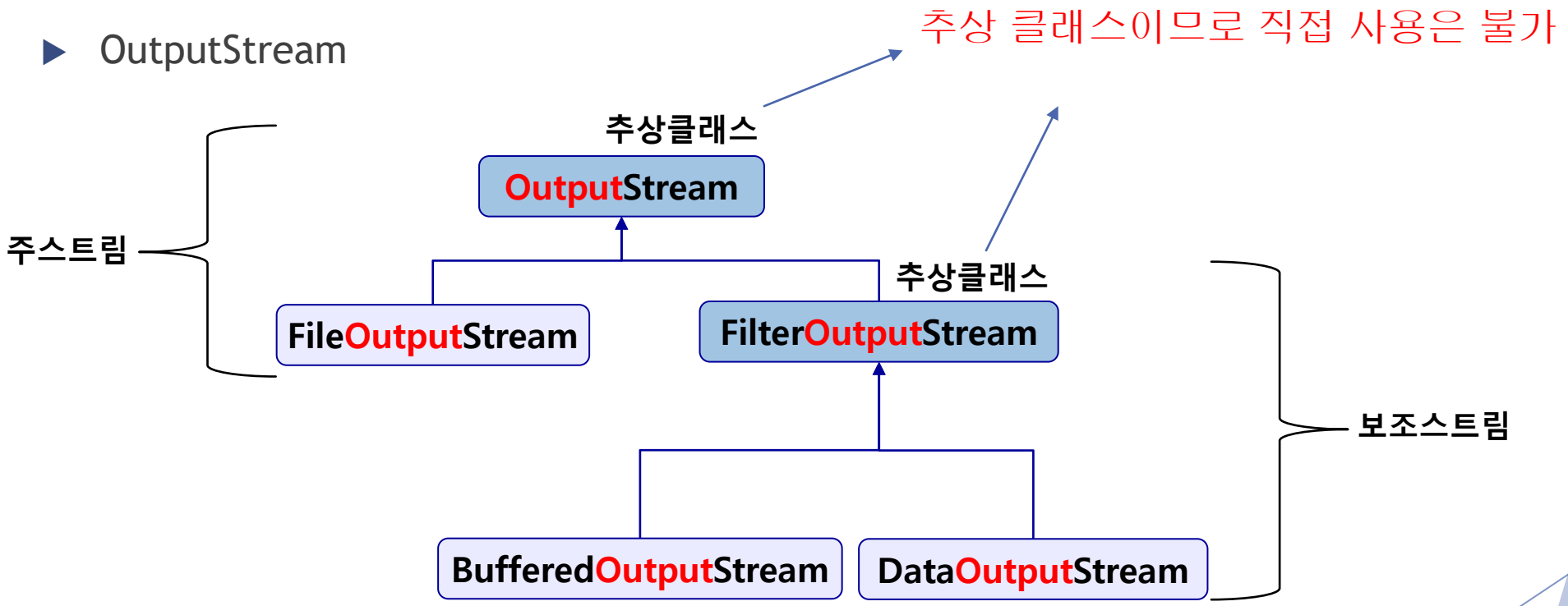
- InputStream



Stream and I/O

: Byte Stream

- OutputStream



Stream and I/O
: Byte Stream

- InputStream의 메서드

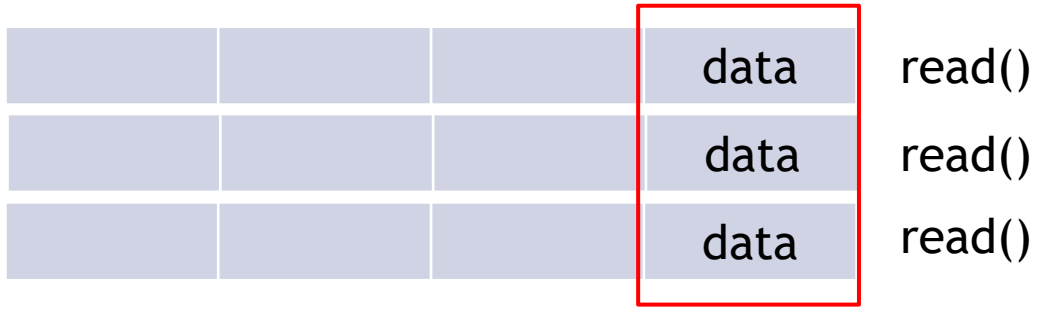
메서드명	설 명
int available()	스트림으로부터 읽어 올 수 있는 데이터의 크기를 반환한다.
void close()	스트림을 닫음으로써 사용하고 있던 자원을 반환한다.
void mark(int readlimit)	현재위치를 표시해 놓는다. 후에 reset()에 의해서 표시해 놓은 위치로 다시 돌아갈 수 있다. readlimit은 되돌아갈 수 있는 byte의 수이다.
boolean markSupported()	mark()와 reset()을 지원하는지를 알려 준다. mark()와 reset()기능을 지원하는 것은 선택적이므로, mark()와 reset()을 사용하기 전에 markSupported()를 호출해서 지원여부를 확인해야한다.
abstract int read()	1 byte를 읽어 온다(0~255사이의 값). 더 이상 읽어 올 데이터가 없으면 -1을 반환한다. abstract메서드라서 InputStream의 자손들은 자신의 상황에 알맞게 구현해야한다.
int read(byte[] b)	배열 b의 크기만큼 읽어서 배열을 채우고 읽어 온 데이터의 수를 반환한다. 반환하는 값은 항상 배열의 크기보다 작거나 같다.
int read(byte[] b, int off, int len)	최대 len개의 byte를 읽어서, 배열 b의 지정된 위치(off)부터 저장한다. 실제로 읽어 올 수 있는 데이터가 len개보다 적을 수 있다.
void reset()	스트림에서의 위치를 마지막으로 mark()이 호출되었던 위치로 되돌린다.
long skip(long n)	스트림에서 주어진 길이(n)만큼을 건너뛴다.

- OutputStream의 메서드

메서드명	설 명
void close()	입력소스를 닫음으로써 사용하고 있던 자원을 반환한다.
void flush()	스트림의 버퍼에 있는 모든 내용을 출력소스에 쓴다.
abstract void write(int b)	주어진 값을 출력소스에 쓴다.
void write(byte[] b)	주어진 배열 b에 저장된 모든 내용을 출력소스에 쓴다.
void write(byte[] b, int off, int len)	주어진 배열 b에 저장된 내용 중에서 off번째부터 len개 만큼만 읽어서 출력소스에 쓴다.

Stream and I/O
: InputStream 주요 메서드

- read()
  - 입력 스트림으로부터 **1바이트를 읽고 4바이트 int 타입으로 리턴**
  - 리턴된 4 바이트 중 끝의 1바이트에만 데이터가 들어가 있다
  - 더 이상 입력스트림으로부터 바이트를 읽을 수 없다면 -1을 반환



이곳에만 데이터가 들어 있음

Stream and I/O
: InputStream 주요 메서드

- read(byte[] b) 메서드
  - 매개값으로 주어진 바이트 배열의 길이만큼 바이트를 읽고 배열에 저장
  - 실제 읽은 바이트 수가 배열의 길이보다 짧을 경우, 읽은 수만큼만 리턴
  - 더이상 읽을 바이트가 없다면 -1을 리턴

Stream and I/O
: InputStream 주요 메서드

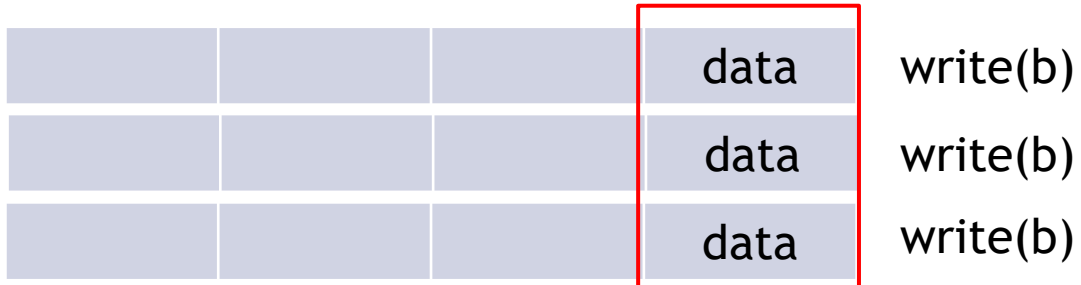
- read(byte[] b, int offset, int len) 메서드
  - 매개값으로 주어진 offset만큼을 건너뛴 후, len 길이만큼 바이트를 읽고 배열에 저장
  - 실제 읽은 바이트 수가 배열의 길이보다 짧을 경우, 읽은 수만큼만 리턴
  - 더이상 읽을 바이트가 없다면 -1을 리턴

Stream and I/O
: InputStream 주요 메서드

- close() 메서드
  - InputStream을 더이상 사용하지 않을 경우, close() 메서드를 호출하여 시스템 자원을 풀어준다
  - 시스템 자원을 풀어주는 것은 매우 중요하다

Stream and I/O
: OutputStream 주요 메서드

- write(int b) 메서드
  - 매개변수로 주어진 int 값에서 끝에 있는 1바이트만 출력 스트림으로 전송



이곳 데이터만 전송함

Stream and I/O
: OutputStream 주요 메서드

- write(byte[] b) 메서드
  - 매개변수로 바이트 배열의 모든 바아트를 출력 스트림으로 전송

- write(byte[] b, int offset, int len) 메서드
  - 매개변수로 주어진 바이트 배열의 b[offset] 부터 len개의 바이트를 출력 스트림으로 전송

Stream and I/O
: OutputStream 주요 메서드

- flush() 메서드
  - OutputStream은 바로 출력되는 것이 아니라 내부에 있는 작은 버퍼에 쌓여 있다가 순서대로 출력됨
  - flush() 메서드는 버퍼에 쌓여 있는 내용을 모두 출력시키고 버퍼를 비우는 메서드

- close() 메서드
  - OutputStream을 더이상 사용하지 않을 때 시스템 자원을 풀어준다

Stream and I/O

: ByteArrayInputStream and ByteArrayOutputStream

```
import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.util.Arrays;

public class IOEx1 {
    public static void main(String[] args) throws IOException {
        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);
        }

        outSrc = output.toByteArray();          //Stream의 내용을 byte 배열로 반환

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

        output.close();
        input.close();
    }
}
```

Stream and I/O

: FileInputStream and FileOutputStream

- 파일에 데이터를 입출력하는 바이트 기반 스트림

생성자	설 명
FileInputStream(String name)	지정된 파일이름(name)을 가진 실제 파일과 연결된 FileInput Stream을 생성한다.
FileInputStream(File file)	파일의 이름이 String이 아닌 File인스턴스로 지정해주어야 하는 점을 제외하고 FileInputStream(String name)와 같다.
FileOutputStream(String name)	지정된 파일이름(name)을 가진 실제 파일과의 연결된 File OutputStream을 생성한다.
FileOutputStream(String name, boolean append)	지정된 파일이름(name)을 가진 실제 파일과 연결된 File OutputStream을 생성한다. 두번째 인자인 append를 true로 하면, 출력 시 기존의 파일내용의 마지막에 덧붙인다. false면, 기존의 파일내용을 덮어쓰게 된다.
FileOutputStream(File file)	파일의 이름을 String이 아닌 File인스턴스로 지정해주어야 하는 점을 제외하고 FileOutputStream(String name)과 같다.

Java File Basic
: File 클래스

- java.io 패키지에서 제공
- 주요 기능
  - 파일 정보: 파일 크기, 속성, 이름 등의 정보를 알아내는 기능
  - 파일 조작: 파일 생성 및 삭제
  - 디렉토리 관련: 디렉토리 생성, 파일 리스트 얻기 등
- 파일에 데이터를 읽고 쓰는 기능은 지원하지 않음
  - 파일의 입출력은 Stream을 사용해야 한다

```
File file = new File("D:\\javastudy\\file\\phonedb.txt")
```

파일 객체를 생성한 것이지, 파일을 생성한 것은 아니다

Java File Basic
: File 클래스

- 파일/디렉터리 생성 및 삭제 메소드
  - 먼저, exists() 메소드를 호출하여 이미 존재하는 파일인지 체크하자

리턴타입	메소드	설명
boolean	createNewFile()	새로운 파일을 생성
booelan	mkdir()	새로운 디렉토리를 생성
boolean	mkdirs()	경로상에 없는 모든 디렉토리를 생성
boolean	delete()	파일 또는 디렉토리 삭제

```
boolean isExist = file.exists(); //   파일이 이미 있는지 확인
```

.

Java File Basic
: File Class

- 파일 및 디렉터리 정보를 확인하는 메소드 목록

리턴타입	메소드	설명
boolean	canExecute()	실행할 수 있는 파일인지 여부
boolean	canRead()	읽을 수 있는 파일인지 여부
boolean	canWrite()	수정 및 저장할 수 있는 파일인지 여부
String	getName()	파일의 이름을 리턴
String	getParent()	부모 디렉토리를 리턴
File	getParentFile()	부모 디렉토리를 File 객체로 생성후 리턴
String	getPath()	전체 경로를 리턴
boolean	isDirectory()	디렉토리인지 여부
boolean	isFile()	파일인지 여부
boolean	isHidden()	숨김 파일인지 여부
long	lastModified()	마지막 수정 날짜 및 시간을 리턴
long	length()	파일의 크기 리턴
String[]	list()	디렉토리에 포함된 파일 및 서브디렉토리 목록 전부를 String 배열로 리턴
String[]	list(FilenameFilter filter)	디렉토리에 포함된 파일 및 서브디렉토리 목록 중에 FilenameFilter 에 맞는 것만 String 배열로 리턴
File[]	listFiles()	디렉토리에 포함된 파일 및 서브 디렉토리 목록 전부를 File 배열로 리턴
File[]	listFiles(FilenameFilter filter)	디렉토리에 포함된 파일 및 서브디렉토리 목록 중에 FilenameFilter 에 맞는 것만 File 배열로 리턴

L

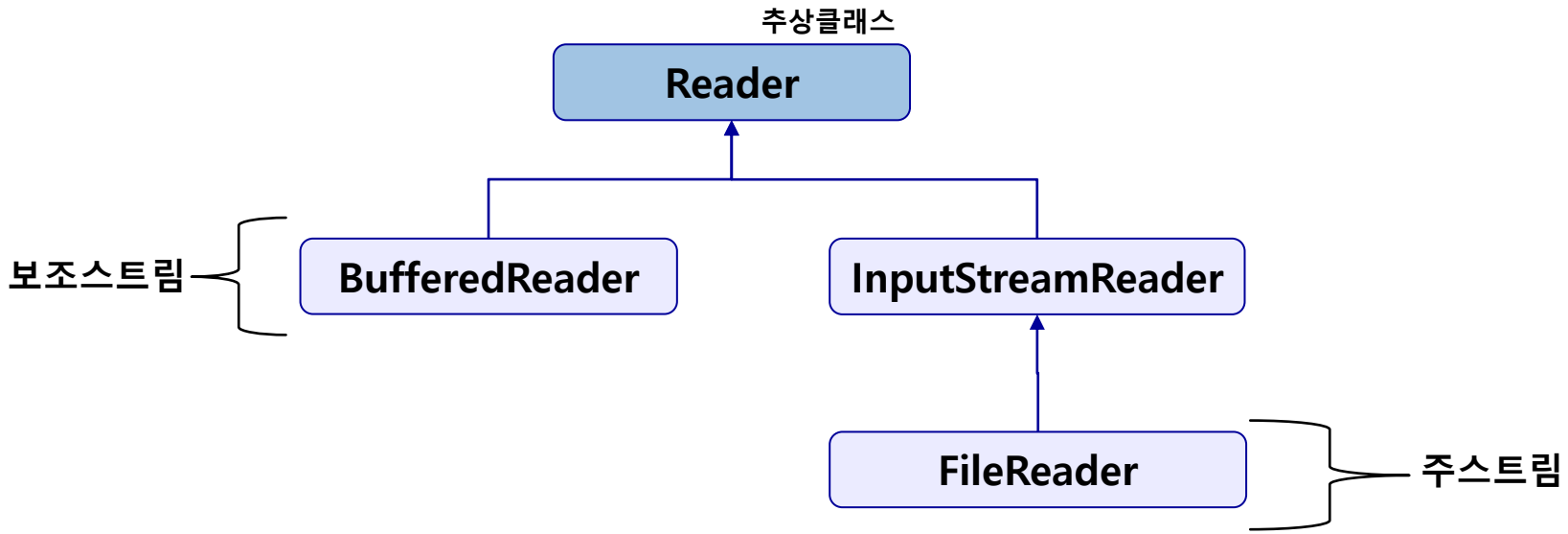
Stream and I/O

: FileInputStream and FileOutputStream – Example

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;

public class FileCopy {
    public static void main(String[] args) {
        try {
            FileInputStream fis = new FileInputStream(args[0]);
            FileOutputStream fos = new FileOutputStream(args[1]);

            int data = 0;
            while((data = fis.read()) != -1) {
                fos.write(data);//void write(int b)
            }

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

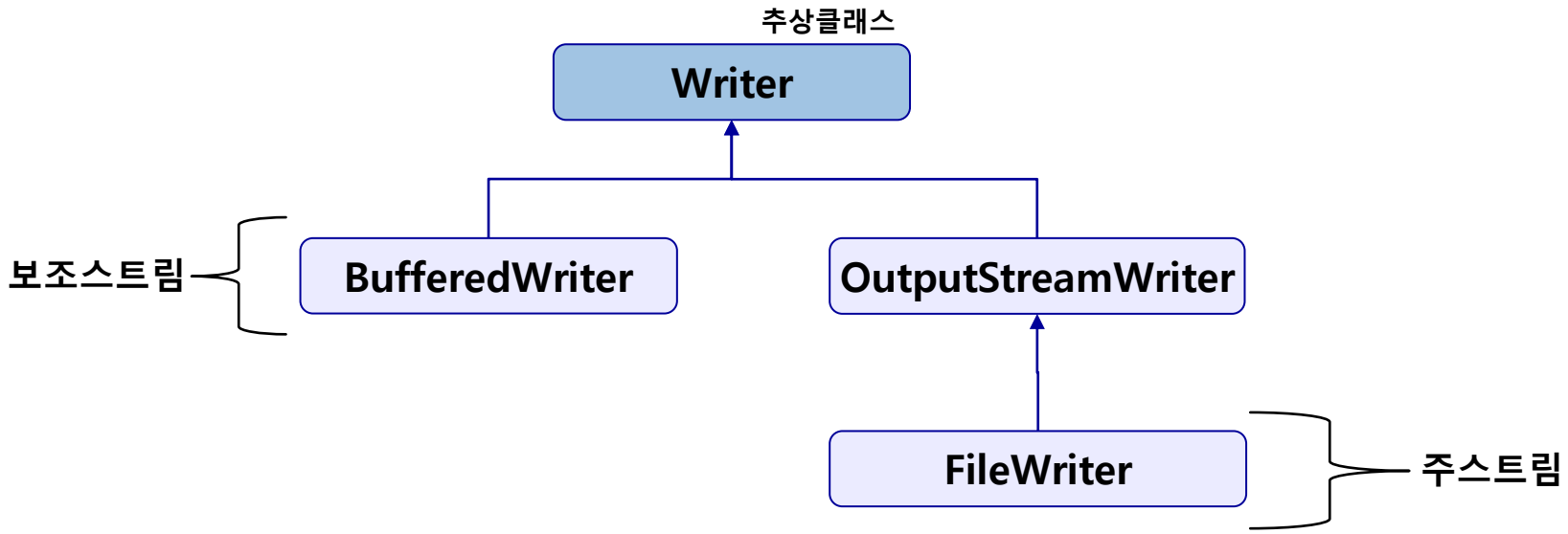
Stream and I/O
: Character Stream

- Reader (문자기반 입력스트림 최고 조상)



Stream and I/O
: Character Stream

- Writer (문자기반 출력스트림 최고 조상)



Stream and I/O
: Character Stream

- Reader 메서드

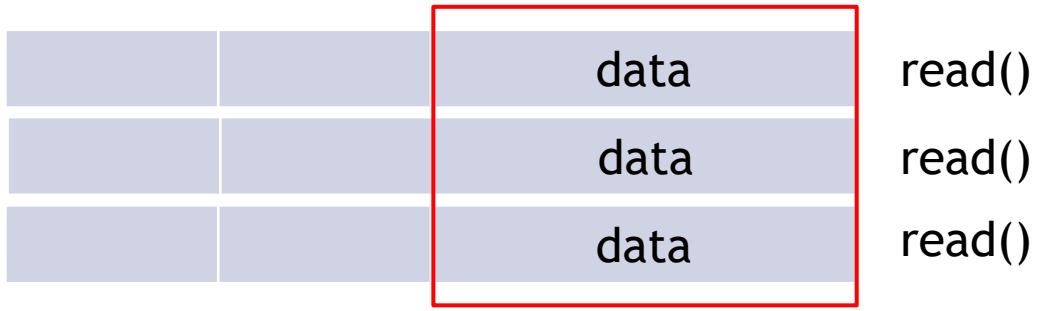
메서드	설 명
abstract void close()	입력스트림을 닫음으로써 사용하고 있던 자원을 반환한다.
void mark(int readlimit)	현재위치를 표시해놓는다. 후에 reset()에 의해서 표시해 놓은 위치로 다시 돌아갈 수 있다.
boolean markSupported()	mark()와 reset()을 지원하는지를 알려 준다.
int read()	입력소스로부터 하나의 문자를 읽어 온다. char의 범위인 0~65535범위의 정수를 반환하며, 입력스트림의 마지막 데이터에 도달하면, -1을 반환한다.
int read(char[] c);	입력소스로부터 매개변수로 주어진 배열 c의 크기만큼 읽어서 배열 c에 저장한다. 읽어 온 데이터의 개수 또는 -1을 반환한다.
abstract int read(char[] c, int off, int len)	입력소스로부터 최대 len개의 문자를 읽어서 , 배열 c의 지정된 위치(off)부터 읽은 만큼 저장한다. 읽어 온 데이터의 개수 또는 -1을 반환한다.
boolean ready()	입력소스로부터 데이터를 읽을 준비가 되어있는지 알려 준다.
void reset()	입력소스에서의 위치를 마지막으로 mark()가 호출되었던 위치로 되돌린다.
long skip(long n)	현재 위치에서 주어진 문자 수(n)만큼을 건너뛴다.

- Writer 메서드

메서드	설 명
abstract void close()	출력스트림를 닫음으로써 사용하고 있던 자원을 반환한다.
abstract void flush()	스트림의 버퍼에 있는 모든 내용을 출력소스에 쓴다.(버퍼가 있는 스트림에만 해당됨)
void write(int b)	주어진 값을 출력소스에 쓴다.
void write(char[] c)	주어진 배열 c에 저장된 모든 내용을 출력소스에 쓴다.
abstract void write(char[] c, int off, int len)	주어진 배열 c에 저장된 내용 중에서 off번째부터 len길이만큼만 출력소스에 쓴다.
void write(String str)	주어진 문자열(str)을 출력소스에 쓴다.
void write(String str, int off, int len)	주어진 문자열(str)의 일부를 출력소스에 쓴다.(off번째 문자부터 len개 만큼의 문자열)

Stream and I/O
: Reader 주요 메서드

- read() 메서드
  - 입력 스트림으로부터 한 개의 문자(2byte)를 읽고 4바이트 int 타입으로 리턴
  - 메서드가 리턴한 int 값을 byte로 변환하면 읽은 문자를 읽을 수 있음
  - 더 이상 읽어들일 문자가 없다면 -1을 반환



Stream and I/O
: Reader 주요 메서드

- read(char[] buffer) 메서드
  - 입력 스트림으로부터 매개값으로 주어진 문자 배열만큼의 문자를 읽고 저장
  - 리턴값은 읽은 문자 수
  - 실제 읽은 문자 수가 배열의 길이보다 작을 경우, 읽은 수만큼을 리턴
  - 더 이상 읽어들일 문자가 없다면 -1을 반환

Stream and I/O
: Reader 주요 메서드

- read(char[] buffer, int offset, int len) 메서드
  - 입력 스트림으로부터 매개값으로 주어진 offset 만큼 건너뛰고 len 만큼의 문자를 읽고 저장
  - 리턴값은 읽은 문자 수
  - 실제 읽은 문자 수가 len 보다 작을 경우, 읽은 수만큼을 리턴
  - 더 이상 읽어들일 문자가 없다면 -1을 반환
- close() 메서드
  - Reader를 더이상 사용하지 않을 경우, 시스템 자원을 풀어준다

Stream and I/O
: Writer 주요 메서드

- write(int c) 메서드
  - int 값에 들어있는 뒤의 2바이트(한 개의 문자)만 출력 스트림으로 전송
- write(char[] cbuffer) 메서드
  - 매개값으로 주어진 char[] 배열의 모든 문자를 출력 스트림으로 전송
- write(char[] cbuffer, int offset, int len) 메서드
  - cbuffer[offset] 부터 len개의 문자를 출력 스트림으로 전송

Stream and I/O
: Writer 주요 메서드

- write(String str) 메서드 : 편의 기능
  - 매개변수로 주어진 str 문자열을 출력 스트림으로 전송
- write(String str, int offset, int len) : 편의 기능
  - 매개값으로 주어진 str에서 offset 만큼 건너뛰고 len 만큼의 문자열을 출력 버퍼로 전송
- flush()
  - 버퍼에 쌓여있는 내용을 모두 출력하고 버퍼를 비움
- close()
  - 시스템 자원을 해제

Stream and I/O

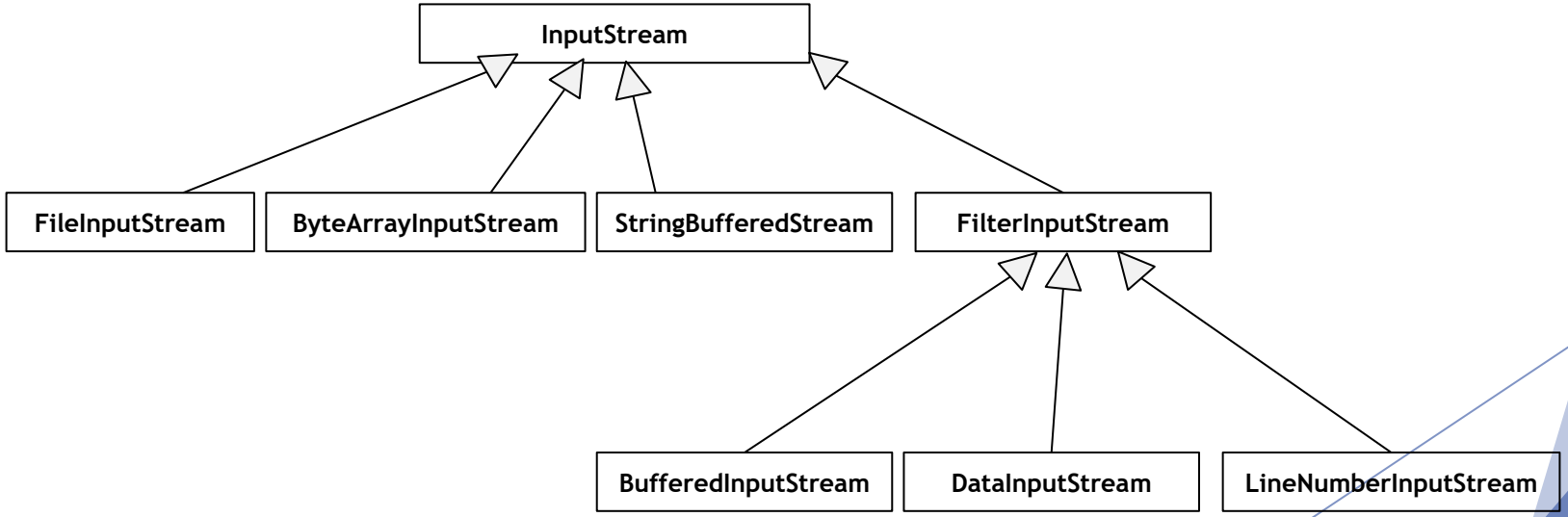
: Character Stream : FileReader and FileWriter

- 문자 기반 파일 입출력,
  텍스트 파일의 입출력에 사용

```
import java.io.FileInputStream;
import java.io.FileReader;
import java.io.IOException;

public class FireReaderEx1 {
    public static void main(String args[]) {
        try {
            String fileName = "test.txt";
            FileInputStream fis = new FileInputStream(fileName);
            FileReader fr = new FileReader(fileName);

            int data = 0;
            //  FileInputStream으로 파일 내용을 읽어 화면에 출력
            while((data = fis.read()) != -1) {
                System.out.println((char)data);
            }
            System.out.println();
            fis.close();

            //  FileReader로 파일 내용을 읽어 화면에 출력
            while((data = fr.read()) != -1) {
                System.out.println((char)data);
            }
            System.out.println();
            fr.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Stream and I/O
: 보조 스트림

- 데코레이터 패턴
  - 객체의 추가적인 요건을 동적으로 첨가
  - 서브클래스를 만드는 방식으로 기능을 유연하게 확장할 수 있는 방법을 제공
  - 장식할 대상과 장식을 동일시하는 패턴



Stream and I/O
: 바이트 기반 보조 스트림

- FilterInputStream과 FilterOutputStream
  - 모든 바이트 기반 보조스트림의 최고조상
  - 보조스트림은 자체적으로 입출력을 수행할 수 없다
  - 상속을 통해 FilterInputStream/FilterOutputStream의 read()와 write()를 오버라이딩 해야 한다

```
public class FilterInputStream extends InputStream {
    protected volatile InputStream in;
    protected FilterInputStream(InputStream in) {
        this.in = in;
    }

    public int read() throws IOException {
        return in.read();
    }
    ...
}
```

FilterInputStream의 자손 - BufferedInputStream, DataInputStream, PushbackInputStream 등
FilterOutputStream의 자손 - BufferedOutputStream, DataOutputStream, PrintStream 등

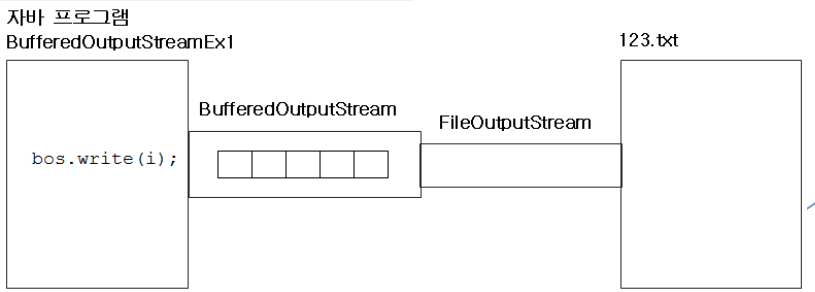
Stream and I/O
: 바이트 기반 보조 스트림

- BufferedInputStream과 BufferedOutputStream
  - 입출력 효율을 높이기 위해 버퍼(byte[])를 사용하는 보조 스트림
  - 보조스트림을 닫으면 기반스트림도 닫힌다

```
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
```



Stream and I/O
: 문자 기반 보조 스트림

- BufferedReader와 BufferedWriter
  - 입출력 효율을 높이기 위해 버퍼(char[])를 사용
  - 라인(line) 단위의 입출력에 편리

String readLine() - 한 라인을 읽어온다. (BufferedReader의 메서드)
void newLine() - '라인 구분자(개행문자)'를 출력한다. (BufferedWriter의 메서드)

```
import java.io.*;

class BufferedReaderEx1 {
    public static void main(String[] args) {
        try {
            FileReader fr = new FileReader("BufferedReaderEx1.java");
            BufferedReader br = new BufferedReader(fr);

            String line = "";
            for(int i=1;(line = br.readLine())!=null;i++) {
                //  ";"를 포함한 라인을 출력한다.
                if(line.indexOf(";")!=-1)
                    System.out.println(i+":"+line);
            }

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

[실행결과]

```
1:import java.io.*;
6:              FileReader fr = new FileReader("BufferedReaderEx1.java");
7:              BufferedReader br = new BufferedReader(fr);
9:              String line = "";
10:             for(int i=1;(line = br.readLine())!=null;i++) {
11:                 //  ";"를 포함한 라인을 출력한다.
12:                 if(line.indexOf(";")!=-1)
13:                     System.out.println(i+":"+line);
```

Stream and I/O
: 문자 기반 보조 스트림

- InputStreamReader와 OutputStreamWriter
  - 바이트 기반 스트림을 문자 기반 스트림처럼 사용할 수 있게 해준다
  - 인코딩(encoding)을 변환하여 입출력할 수 있게 해준다
  - 콘솔(console)로부터 라인 단위로 입력받기

```
InputStreamReader isr = new InputStreamReader(System.in);
BufferedReader br = new BufferedReader(isr);
String line = br.readLine();
```

  - 인코딩 변환하기

```
FileInputStream fis = new FileInputStream("korean.txt");
InputStreamReader isr = new InputStreamReader(fis, "KSC5601");
```

Stream and I/O
: 문자 기반 보조 스트림

- InputStreamReader와 OutputStreamWriter 예제

```
import java.io.*;

class ReadKBLine() throws IO Exception {
     public static void main(String[] arg) throws Exception {
               try {
                 BufferedReader keybd = new BufferedReader(new InputStreamReader(System.in));
                 System.out.println(keybd.readLine());
               } catch (IOException e) {}
     }
}
```

- BufferedReader 클래스의 Constructor : BufferedReader(Reader in)
- InputStreamReader 클래스의 Constructor : InputStreamReader(InputStream in)
- System.in 자체가 InputStream 객체

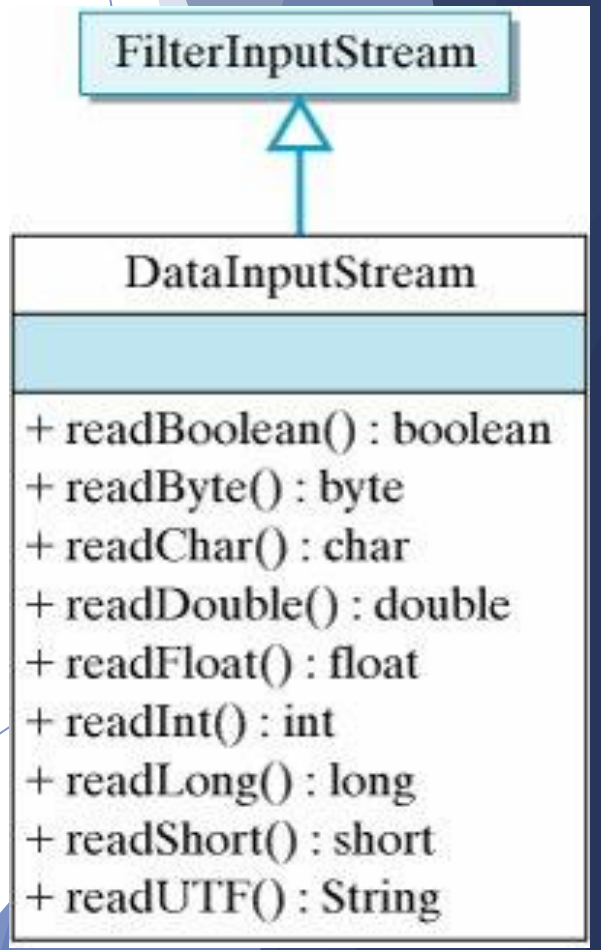
Stream and I/O

: 문자 기반 보조 스트림 - InputStreamReader와 OutputStreamWriter 예제 2

```
public class eggMonster {
    public static void main(String[] args) {
            String morningEgg;
            String lunchEgg;
            String dinnerEgg;

            try {
                BufferedReader keybd = new BufferedReader(new InputStreamReader(System.in));
                System.out.print("아침에 먹은 계란 갯수: ");
                morningEgg = keybd.readLine();
                System.out.println("아침에 계란" + morningEgg + "개");
                System.out.println();
                System.out.print("점심에 먹은 계란 갯수: ");
                lunchEgg = keybd.readLine();
                System.out.println("점심에 계란" + lunchEgg + "개");
                System.out.println();
                System.out.print("저녁에 먹은 계란 갯수: ");
                dinnerEgg = keybd.readLine();
                System.out.println("저녁에 계란" + dinnerEgg + "개");
            } catch (IOException e) {
                e.printStackTrace();
                System.exit(1);
            }
    }
}
```

Stream and I/O
: 기본 타입 입출력 보조스트림

- DataInputStream과 DataOutputStream
  - 바이트 기반 스트림은 기본 데이터 타입(Primitive Types)을 입출력할 수 없다
  - DataInputStream과 DataOutputStream을 이용하면 기본 타입 입출력이 가능하다
  - 주의: DataInputStream으로 읽을 때는 DataOutputStream에서 출력한 순서대로 읽어와야 한다
    - 각 데이터 타입의 크기가 모두 다르기 때문
    - 예) 출력 순서 : String -> int -> double
      입력 순서 : String -> int -> double

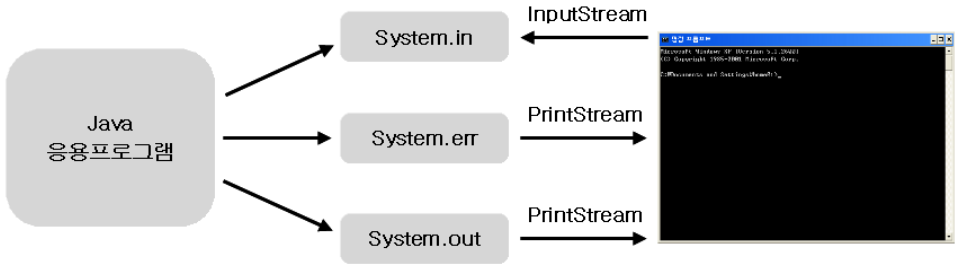


L

Stream and I/O
: 표준입출력

- 콘솔을 통한 데이터의 입출력을 '표준입출력 ' 이라 한다
- JVM이 시작되면서 자동적으로 생성되는 스트림

```
System.in  - 콘솔로부터 데이터를 입력받는데 사용
System.out - 콘솔로 데이터를 출력하는데 사용
System.err - 콘솔로 데이터를 출력하는데 사용
```




```
public final class System {
   public final static InputStream in = nullInputStream();
   public final static PrintStream out = nullPrintStream();
   public final static PrintStream err = nullPrintStream();
   ...
}
```

```
static void setOut(PrintStream out)
static void setErr(PrintStream err)
static void setIn(InputStream in)
```

Stream and I/O
: 파일 닫기

- 정상적으로 수행되었을 경우, Java가 알아서 파일을 close 해줌
- close() 메서드를 호출하는 이유
  - 비정상적으로 프로그램이 끝났을 경우 파일이 손상되는 것을 막아줌
  - 파일에 출력한 이후 close 해주어야 읽을 수 있음
    - RandomAccessFile 클래스 객체를 이용, 파일에서 출력과 읽어들이는 것을 동시에 할 수 있으나 일반적으로 Transaction 관점에서 바람직하지 않음

Stream and I/O
: StringTokenizer

- java.util.StringTokenizer
  - 문자열을 특정 구분자(Delimeter)로 분리해내는 기능 수행
- StringTokenizer의 Constructor
  - StringTokenizer(String str)
    - Default 생성자는 구분자로 “\t\n\r”을 가정
  - StringTokenizer(String str, String delim)

```
String tel = “010-9000-7777”;
StringTokenizer st = new StringTokenizer(tel, “#-”); // #과 -을 구분자로 사용
while(st.hasMoreTokens()) {
    System.out.println(st.nextToken());
}
```

Stream and I/O

: StringTokenizer - Example

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

public class StringTokenizerTest {

    public static void main(String[] args) {

            try {
                BufferedReader br = new BufferedReader(new FileReader("c:\\temp\\phone.txt"));
                String phoneList;

                while ((phoneList = br.readLine()) != null) {
                    StringTokenizer st = new StringTokenizer(phoneList, " -.\t");
                    while(st.hasMoreTokens()) {
                        System.out.println(st.nextToken());}                    }
            } catch (IOException e) {
                e.printStackTrace();
                System.exit(1);
            }
    }
}
```

phone.txt

```
둘리 010-2222-2222
고길동      011-0000-2345
도우너      010-5555-5555
마이콜      011-9999-7777
```

Stream and I/O
: Scanner

- Scanner
  - Java 5부터 java.util.Scanner 제공
  - BufferedReader와 FileReader, StringTokenizer의 조합 대신 사용
  - 키보드 입력 사용의 예

```
Scanner skb = new Scanner(System.in);
System.out.print("Enter name: ");
String name = stdin.nextLine();
System.out.print("Enter age:  ");
int age = stdin.nextInt();
```

  - File에서 사용 예

```
File file = new File(  “inputFile.txt” );
Scanner sf = new Scanner( file );
```

Stream and I/O

: Scanner – Example

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

public class ScannerTest {

    public static void main(String[] args) {
        try {
            File file = new File( "c:\\temp\\phone2.txt" );
            Scanner sf = new Scanner( file );
            String name;
            int number1;    // 전화번호 첫번째 자리
            int number2;    // 전화번호 두번째 자리
            int number3;    // 전화번호 세번째 자리
            while ( sf.hasNextLine() ) {
                name = sf.next();
                // 첫번째 Delimeter(이 경우는 스페이스)까지 String으로 읽음
                number1 = sf.nextInt();  // Integer값으로 Parsing
                number2 = sf.nextInt();
                number3 = sf.nextInt();
                System.out.println( name + number1 + number2 + number3 );
            }
        } catch(FileNotFoundException ex ) {

        }
    }
}
```

phone2.txt

```
둘리 010 2222 2222
고길동 011 0000 2345
```

L

Stream and I/O

: Scanner vs BufferedReader

- Scanner 클래스가 있는데도, BufferedReader 클래스를 InputStream, InputStreamReader, FileReader, StringTokenizer 등과 조합해서 사용하는 이유
  - 하나의 업무를 하나의 클래스에서 수행하여 클래스의 변경 없이 다양한 경우에 조합해서 사용 가능
  - Scanner 클래스로 처리 가능한 경우는 Scanner 클래스 사용을 추천

	Scanner	BufferReader
Parsing	▪ Parsing 자료형에 따라 nextInt(), nextFloat(), next() 등 사용	▪ Read(), readLine() 등의 메쏘드로는 Parsing 지원하지 않음 ▪ StringTokenizer와 Wrapper 클래스 메쏘드 사용 Ex) Integer.parseInt(Delimeteredtoken)
EOF/EOS*	▪nextLine(), nextInt()는 exception을 throw함 ▪hasNextLine(), hasNextInt() 등을 사용해서 미리 확인할 필요	▪readLine() – null 값은 return ▪read() – "-1"값을 return

* EOF/EOS – End of File / End of Stream