



## 14-2. 보조 스트림

혼자 공부하는 자바(개정판) (신용권 저)

- 시작하기 전에
- 보조 스트림 연결하기
- 문자 변환 보조 스트림
- 성능 향상 보조 스트림
- 기본 타입 입출력 보조 스트림
- 프린터 보조 스트림
- 객체 입출력 보조 스트림
- 키워드로 끝내는 핵심 포인트
- 확인문제



## 시작하기 전에

[핵심 키워드] : 보조 스트림, 문자 변환, 성능 향상, 기본 타입 입출력, 개행 출력

### [핵심 포인트]

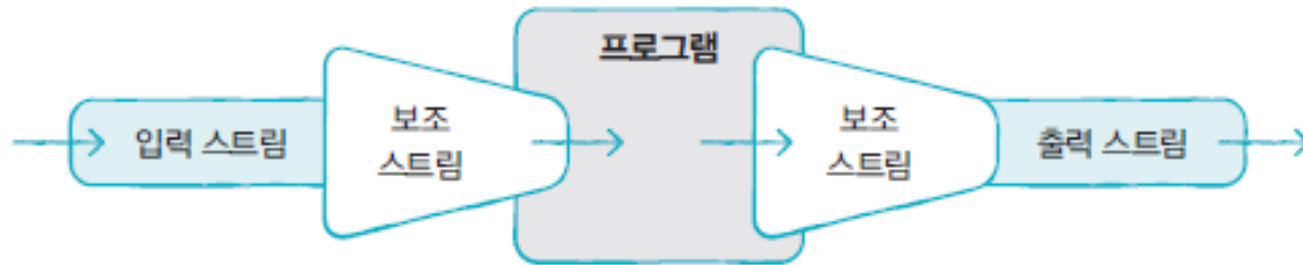
기본 스트림을 직접 사용해서 데이터를 입출력할 수도 있지만, 데이터를 변환해서 입출력하거나 데이터의 출력 형식을 지정하려는 경우 등에는 보조 스트림을 연결하여 사용하면 편리하다.



# 시작하기 전에

## ❖ 보조 스트림

- 다른 스트림과 연결되어 여러가지 편리한 기능을 제공하는 스트림
- 자체적으로 입출력 수행할 수 없기 때문에 입출력 소스와 바로 연결되는 InputStream, OutputStream, Reader, Writer 등에 연결하여 입출력 수행



- 프로그램은 입력 스트림으로부터 직접 데이터 읽지 않고, 보조 스트림에서 제공하는 기능 이용하여 데이터 읽음. 또한 출력 스트림으로 직접 데이터 보내지 않고 보조 스트림에서 제공하는 기능 이용하여 데이터 보냄



# 보조 스트림 연결하기

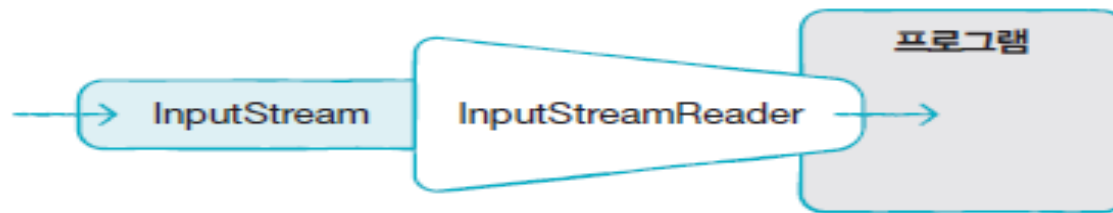
## ❖ 보조 스트림 연결하기

- 보조 스트림 생성 시 자신이 연결될 스트림을 생성자의 매개값으로 제공

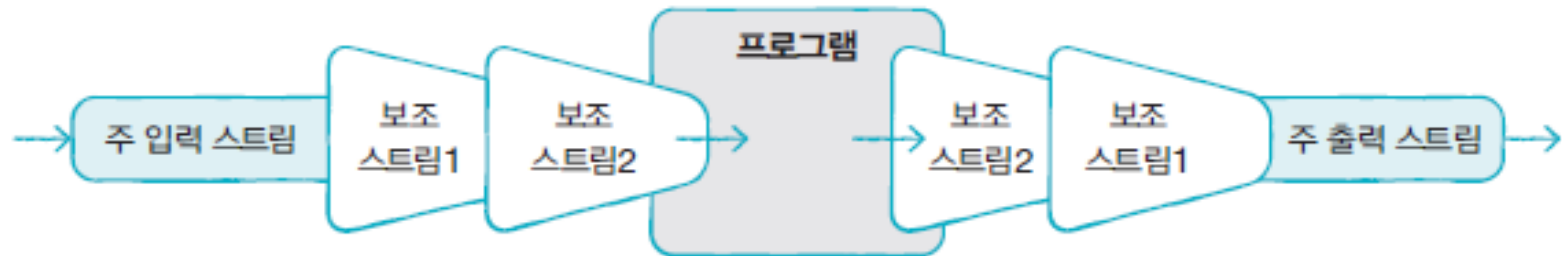
```
보조스트림 변수 = new 보조스트림(연결스트림)
```

```
InputStream is = ...;
```

```
InputStreamReader reader = new InputStreamReader(is);
```



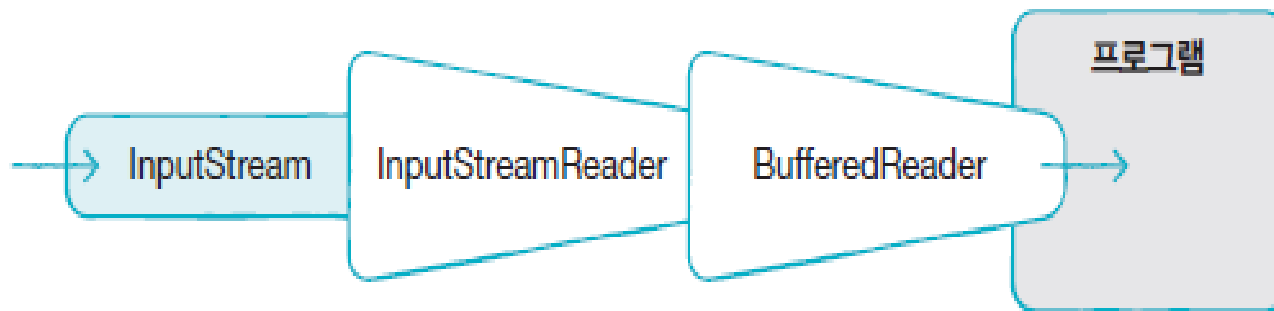
- 보조 스트림을 연속적으로 연결할 수도 있음



# 보조 스트림 연결하기

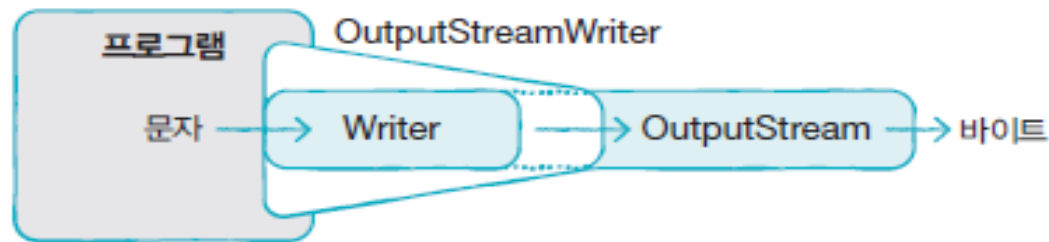
## ■ 예시

```
InputStream is = System.in;  
InputStreamReader reader = new InputStreamReader(is);  
BufferedReader br = new BufferedReader(reader);
```



## 문자 변환 보조 스트림

- ❖ 소스 스트림이 바이트 기반 스트림이면서 입출력 데이터가 문자일 경우 Reader와 Writer로 변환해서 사용하는 것이 편리함
- ❖ **OutputStreamWriter**
  - 바이트 기반 출력 스트림에 연결되어 문자 출력 스트림인 Writer로 변환하는 보조 스트림



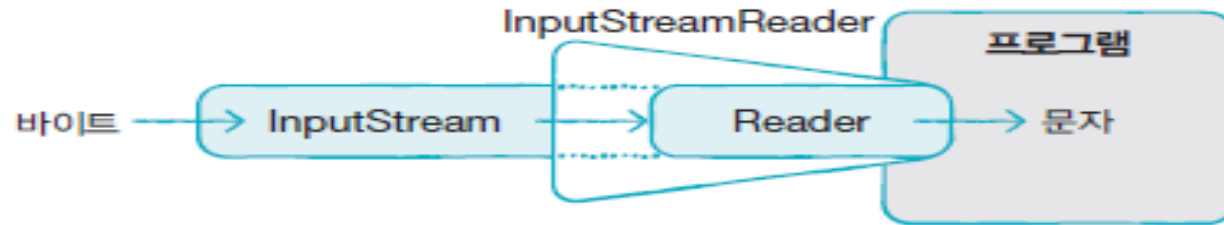
```
Writer writer = new OutputStreamWriter(바이트 기반 출력 스트림);
```

```
FileOutputStream fos = new FileOutputStream("C:/Temp/test1.txt");  
Writer writer = new OutputStreamWriter(fos);
```

# 문자 변환 보조 스트림

## ❖ InputStreamReader

- 바이트 기반 입력 스트림에 연결되어 문자 입력 스트림인 Reader로 변환하는 보조 스트림



```
Reader reader = new InputStreamReader(바이트 기반 입력 스트림);
```

```
FileInputStream fis = new FileInputStream("C:/Temp/test1.txt");  
Reader reader = new InputStreamReader(fis);
```





# 문자 변환 보조 스트림

## ■ 예시 - 문자 변환 보조 스트림

```
01 package sec02.exam01;
02
03 import java.io.FileInputStream;
04 import java.io.FileOutputStream;
05 import java.io.InputStreamReader;
06 import java.io.OutputStreamWriter;
07 import java.io.Reader;
08 import java.io.Writer;
09
10 public class CharacterConvertStreamExample {
11     public static void main(String[] args) throws Exception {
12         write("문자 변환 스트림을 사용합니다.");
13         String data = read();
14         System.out.println(data);
15     }
16
17     public static void write(String str) throws Exception {
18         FileOutputStream fos = new FileOutputStream("C:/Temp/test1.txt");
19         Writer writer = new OutputStreamWriter(fos);
```

FileOutputStream에  
OutputStreamWriter  
보조 스트림을 연결



# 문자 변환 보조 스트림

```
20 writer.write(str);
21 writer.flush();
22 writer.close();
23 }
24
25 public static String read() throws Exception {
26     FileInputStream fis = new FileInputStream("C:/Temp/test1.txt");
27     Reader reader = new InputStreamReader(fis);
28     char[] buffer = new char[100];
29     int readCharNum = reader.read(buffer);
30     reader.close();
31     String data = new String(buffer, 0, readCharNum);
32     return data;
33 }
34 }
```

OutputStreamWriter 보조 스트림을  
이용해서 문자 출력

FileInputStream에  
InputStreamReader  
보조 스트림을 연결

InputStreamReader 보조 스트림을  
이용해서 문자 입력

char 배열에서 읽은  
수만큼 문자열로 변환

**실행결과**  
문자 변환 스트림을 사용합니다.



# 성능 향상 보조 스트림

## ❖ 성능 향상 보조 스트림

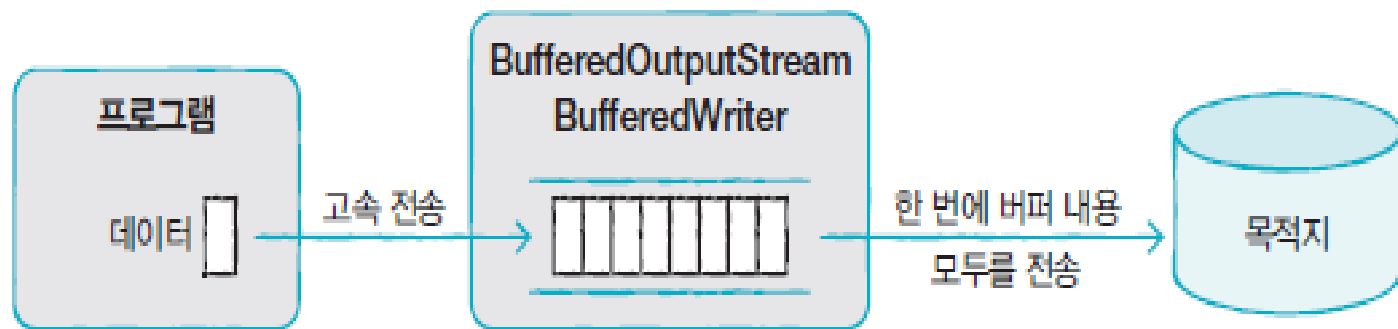
- 메모리 버퍼를 추가로 제공하여 프로그램의 실행 성능을 향상
- 바이트 기반 스트림에서는  
    `BufferInputStream`, `BufferedOutputStream`
- 문자 기반 스트림에서는  
    `BufferedReader`, `BufferWriter`

## ❖ `BufferedOutputStream`과 `BufferedWriter`

- 바이트 기반 출력 스트림에 연결되어 버퍼 제공하는 보조 스트림
- 문자 기반 출력 스트림에 연결되어 버퍼 제공하는 보조 스트림
- 프로그램에서 전송한 데이터를 내부 버퍼에 쌓아두었다가 버퍼가 꽉 차면 한꺼번에 보냄
- 생성자의 매개값으로 준 출력 스트림과 연결되어 추가 내부 버퍼 제공



## 성능 향상 보조 스트림



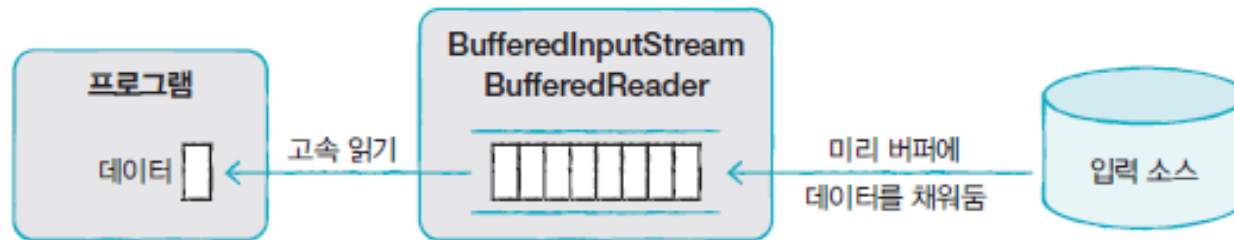
```
BufferedOutputStream bos = new BufferedOutputStream(바이트 기반 출력 스트림);  
BufferedWriter bw = new BufferedWriter(문자 기반 출력 스트림);
```



# 성능 향상 보조 스트림

## ❖ BufferedInputStream과 BufferedReader

- 바이트 기반 입력 스트림에 연결되어 버퍼 제공하는 보조 스트림
- 문자 기반 입력 스트림에 연결되어 버퍼를 제공하는 보조 스트림
- 입력 소스로부터 내부 버퍼 크기만큼 데이터를 미리 읽고 버퍼에 저장해둠
- 프로그램이 외부 소스로부터 직접 읽는 것이 아닌 버퍼로부터 읽음으로써 읽기 성능 향상



- 생성자의 매개값으로 준입력 스트림과 연결되어 추가 내부 버퍼 제공

```
BufferedInputStream bis = new BufferedInputStream(바이트 기반 입력 스트림);  
BufferedReader br = new BufferedReader(문자 기반 입력 스트림);
```

# 성능 향상 보조 스트림

## ■ 예시 - 파일 복사 성능 테스트

```
01 package sec02.exam02;
02
03 import java.io.*;
04
05 public class NonBufferVsBufferExample {
06     public static void main(String[] args) throws Exception {
07         String originalFilePath1 = 기본 스트림 생성
08             NonBufferVsBufferExample.class.getResource("originalFile1.jpg").getPath();
09         String targetFilePath1 = "C:/Temp/targetFile1.jpg";
10         FileInputStream fis = new FileInputStream(originalFilePath1);
11         FileOutputStream fos = new FileOutputStream(targetFilePath1);
12
13         String originalFilePath2 = 버퍼 보조 스트림 연결
14             NonBufferVsBufferExample.class.getResource("originalFile2.jpg").getPath();
15         String targetFilePath2 = "C:/Temp/targetFile2.jpg";
16         FileInputStream fis2 = new FileInputStream(originalFilePath2);
17         FileOutputStream fos2 = new FileOutputStream(targetFilePath2);
18         BufferedInputStream bis = new BufferedInputStream(fis2);
19         BufferedOutputStream bos = new BufferedOutputStream(fos2);
20
21         long nonBufferTime = copy(fis, fos); FileInputStream, FileOutputStream을  
이용한 복사 시간 측정
22         System.out.println("버퍼를 사용하지 않았을 때:\t" + nonBufferTime + "ns");
23
24         long bufferTime = copy(bis, bos); BufferedInputStream, BufferedOutputStream을  
이용한 복사 시간 측정
25         System.out.println("버퍼를 사용했을 때:\t\t" + bufferTime + "ns");
```

# 성능 향상 보조 스트림

```
26
27     fis.close();
28     fos.close();
29     bis.close();
30     bos.close();
31 }
32
33 static int data = -1;
34 public static long copy(InputStream is, OutputStream os) throws Exception {
35     long start = System.nanoTime(); ← 시작 시간 저장
36     while(true) {
37         data = is.read();
38         if(data == -1) break; ← [파일 복사] 원본 파일에서 읽은 1byte를
39         os.write(data);       ← 타겟 파일로 바로 출력
40     }
41     os.flush();
42     long end = System.nanoTime(); ← 끝 시간 저장
43     return (end-start); ← 복사에 걸린 시간 리턴
44 }
45 }
```

실행결과	
버퍼를 사용하지 않았을 때:	5330067700ns
버퍼를 사용했을 때:	42501200ns



# 성능 향상 보조 스트림

- BufferedReader는 라인 단위로 문자열 읽는 readLine() 메소드 제공  
[Enter]키 이전의 모든 문자열 읽고 리턴  
키보드에서 입력한 내용 및 파일 내용을 라인 단위로 읽을 수 있음

예시 - 라인 단위로 구분된 문자열

---

```
01    C 언어
02    Java 언어
03    Python 언어
```

---

라인 단위로 문자열 얻으려면 BufferedReader 보조 스트림 생성하고 readLine()으로 반복해서 읽음





# 성능 향상 보조 스트림

```
01 package sec02.exam03;
02
03 import java.io.*;
04
05 public class ReadLineExample {
06     public static void main(String[] args) throws Exception {
07         Reader reader = new FileReader(
08             ReadLineExample.class.getResource("language.txt").getPath()
09         );
10         BufferedReader br = new BufferedReader(reader);
11
12         while(true) {
13             String data = br.readLine();
14             if(data == null) break;
15             System.out.println(data);
16         }
17
18         br.close();
19     }
20 }
```

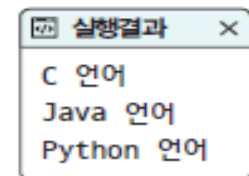
문자 기반  
입력 스트림 연기

BufferedReader 보조 스트림 연결

라인 단위 문자열을 읽고 리턴

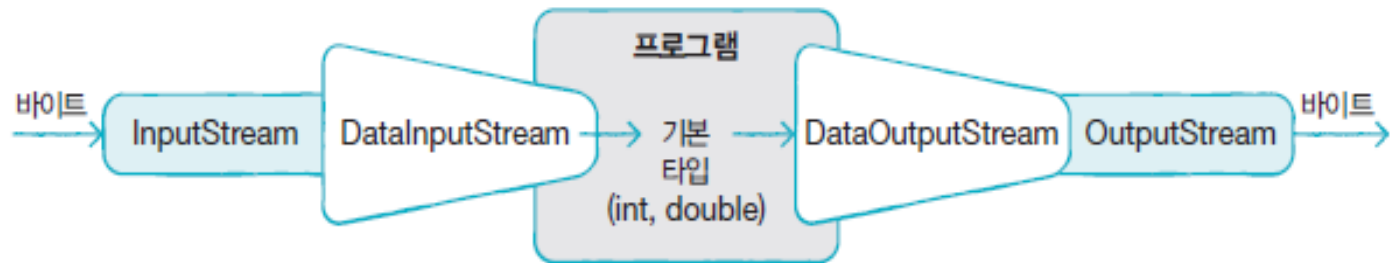
파일 끝에 도달했을 경우

입력 스트림 닫기



# 기본 타입 입출력 보조 스트림

- ❖ **DataInputStream**과 **DataOutputStream** 보조 스트림 연결하면 기본 타입인 boolean, char, short, int, long, float, double 입출력할 수 있음



## ■ 객체 생성 코드

```
DataInputStream dis = new DataInputStream(바이트 기반 입력 스트림);  
DataOutputStream dos = new DataOutputStream(바이트 기반 출력 스트림);
```



# 기본 타입 입출력 보조 스트림

DataInputStream		DataOutputStream	
boolean	readBoolean()	void	writeBoolean(boolean v)
byte	readByte()	void	writeByte(int v)
char	readChar()	void	writeChar(int v)
double	readDouble()	void	writeDouble(double v)
float	readFloat()	void	writeFloat(float v)
int	readInt()	void	writeInt(int v)
long	readLong()	void	writeLong(long v)
short	readShort()	void	writeShort(int v)
String	readUTF()	void	writeUTF(String str)

- 데이터 타입의 크기가 모두 다르므로 DataOutputStream으로 출력한 데이터를 다시 DataInputStream으로 읽어올 때는 출력한 순서와 동일한 순서로 읽어야 함



# 기본 타입 입출력 보조 스트림

## ■ 예시 - 기본 타입 입출력

```
01 package sec02.exam04;
02
03 import java.io.*;
04
05 public class DataInputStreamExample {
06     public static void main(String[] args) throws Exception {
07         FileOutputStream fos = new FileOutputStream("C:/Temp/primitive.db");
08         DataOutputStream dos = new DataOutputStream(fos);
09
10         dos.writeUTF("홍길동");
11         dos.writeDouble(95.5);
12         dos.writeInt(1);
13
14         dos.writeUTF("감자바");
15         dos.writeDouble(90.3);
16         dos.writeInt(2);
```

← 기본 타입 값 출력

← 바이트 기반 출력 스트림을 생성하고  
DataOutputStream 보조 스트림 연결

# 기본 타입 입출력 보조 스트림

```
17
18     dos.flush(); dos.close(); ← 출력 스트림 닫기
19
20     FileInputStream fis = new FileInputStream("C:/Temp/primitive.db");
21     DataInputStream dis = new DataInputStream(fis);
22
23     for(int i=0; i<2; i++) {
24         String name = dis.readUTF();
25         double score = dis.readDouble(); ← 기본 타입 값 읽기
26         int order = dis.readInt();
27         System.out.println(name + " : " + score + " : " + order);
28     }
29
30     dis.close(); ← 입력 스트림 닫기
31 }
32 }
```

바이트 기반 입력 스트림을 생성하고  
DataInputStream 보조 스트림 연결

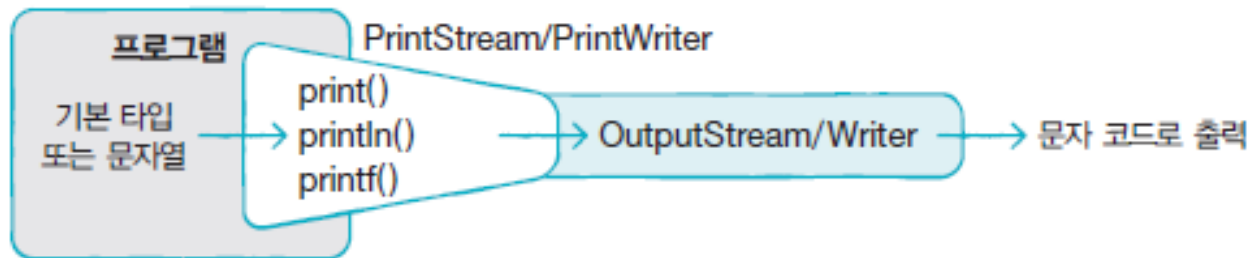
실행결과	
홍길동	: 95.5 : 1
감자바	: 90.3 : 2



# 프린터 보조 스트림

❖ **PrintStream**과 **PrintWriter**는 프린터와 유사하게 출력하는 `print()`, `println()` 메소드 가진 보조 스트림

- 각기 바이트 기반 출력 스트림 및 문자 기반 출력 스트림과 연결



```
PrintStream ps = new PrintStream(바이트 기반 출력 스트림);  
PrintWriter pw = new PrintWriter(문자 기반 출력 스트림);
```

- `println()` 메소드는 `print()`와 달리 출력할 데이터 끝에 개행 문자 `\n` 추가하여 줄바꿈 일어남



# 프린터 보조 스트림

- 출력할 데이터 타입에 따른 오버로딩

PrintStream / PrintWriter			
void	print(boolean b)	void	println(boolean b)
void	print(char c)	void	println(char c)
void	print(double d)	void	println(double d)
void	print(float f)	void	println(float f)
void	print(int i)	void	println(int i)
void	print(long l)	void	println(long l)
void	print(Object obj)	void	println(Object obj)
void	print(String s)	void	println(String s)
		void	println()





# 프린터 보조 스트림

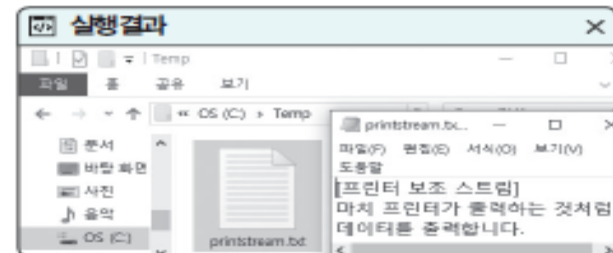
## ■ 예시 - 라인 단위로 출력하기

```
01 package sec02.exam05;
02
03 import java.io.FileOutputStream;
04 import java.io.PrintStream;
05
06 public class PrintStreamExample {
07     public static void main(String[] args) throws Exception {
08         FileOutputStream fos = new FileOutputStream("C:/Temp/printstream.txt");
09         PrintStream ps = new PrintStream(fos);
10
11         ps.println("[프린터 보조 스트림]");
12         ps.print("마치 ");
13         ps.println("프린터가 출력하는 것처럼 ");
14         ps.println("데이터를 출력합니다.");
15
16         ps.flush();
17         ps.close();
18     }
19 }
```

바이트 기반 출력 스트림을 생성하고  
PrintStream 보조 스트림 연결

라인 단위로 문자열 출력

버퍼에 잔류하는  
문자열을 모두 보내고,  
출력 스트림 닫음

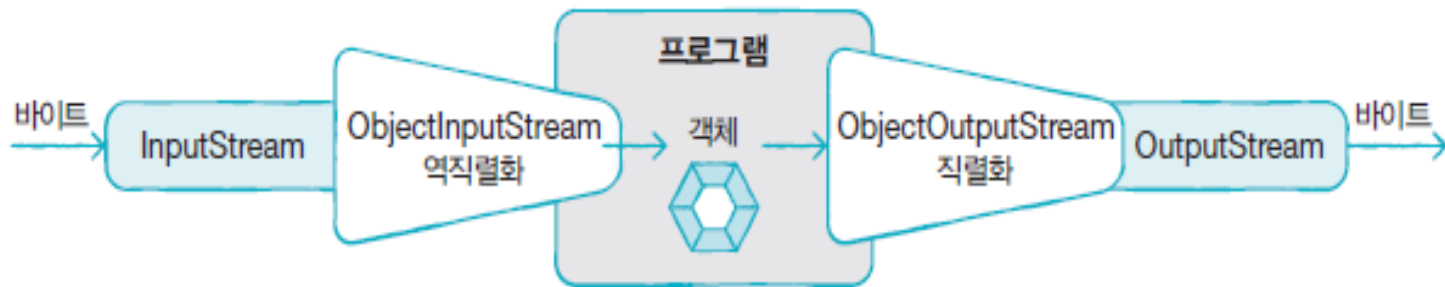




# 객체 입출력 보조 스트림

❖ **ObjectOutputStream**과 **ObjectInputStream** 보조 스트림 연결하면 메모리에 생성된 객체를 파일 또는 네트워크로 출력

- 각기 객체 직렬화 및 역직렬화 역할



- 연결할 바이트 기반 입출력 스트림을 생성자 매개값으로 받음

```
ObjectInputStream ois = new ObjectInputStream(바이트 기반 입력 스트림);  
ObjectOutputStream oos = new ObjectOutputStream(바이트 기반 출력 스트림);
```



# 객체 입출력 보조 스트림

- ObjectOutputStream의 writeObject() 메소드는 객체 직렬화하여 출력 스트림으로 보냄

```
oos.writeObject(객체);
```

- ObjectInputStream의 readObject() 메소드는 입력 스트림에서 읽은 바이트를 역직렬화하여 객체로 다시 복원해 리턴함

원래 타입으로 강제변환 필요

```
객체타입 변수 = (객체타입) ois.readObject();
```

- 자바는 java.io.Serializable 인터페이스 구현한 객체만 직렬화함

Serializable 인터페이스는 메소드 선언 없으므로, 네트워크로 전송하려는 경우 클래스 선언 시 implements Serializable 추가해야 함

```
public class XXX implements Serializable { ... }
```



# 객체 입출력 보조 스트림

## ■ 예시 - 객체 입출력 보조 스트림

```
01 package sec02.exam06;
02
03 import java.io.*;
04 import java.text.SimpleDateFormat;
05 import java.util.*;
06
07 public class ObjectOutputStreamExample {
08     public static void main(String[] args) throws Exception {
09         writeList(); ← List를 파일에 저장
10         List<Board> list = readList(); ← 파일에 저장된 List 읽기
11
12         SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
13         for(Board board : list) {
14             System.out.println(
15                 board.getBno() + "\t" + board.getTitle() + "\t" +
16                 board.getContent() + "\t" + board.getWriter() + "\t" +
17                 sdf.format(board.getDate())
18             );
19         }
20     }
21
22     public static void writeList() throws Exception {
23         List<Board> list = new ArrayList<>(); ← List 생성
```

← List 내용을 모니터에 출력

## 객체 입출력 보조 스트림

```
24
25 list.add(new Board(1, "제목1", "내용1", "글쓴이1", new Date()));
26 list.add(new Board(2, "제목2", "내용2", "글쓴이2", new Date()));
27 list.add(new Board(3, "제목3", "내용3", "글쓴이3", new Date()));
28
29 FileOutputStream fos = new FileOutputStream("C:/Temp/board.db");
30 ObjectOutputStream oos = new ObjectOutputStream(fos);
31 oos.writeObject(list);
32 oos.flush();
33 oos.close();
34 }
35
36 public static List<Board> readList() throws Exception {
37     FileInputStream fis = new FileInputStream("C:/Temp/board.db");
38     ObjectInputStream ois = new ObjectInputStream(fis);
39     List<Board> list = (List<Board>) ois.readObject();
40     return list;
41 }
42 }
```

list에 Board 객체 저장

객체 출력 스트림을 이용해서 list 출력


객체 입력 스트림을 이용해서 list 읽기

실행결과				
1	제목1	내용1	글쓴이1	2019-05-07
2	제목2	내용2	글쓴이2	2019-05-07
3	제목3	내용3	글쓴이3	2019-05-07

# 객체 입출력 보조 스트림

```
01 package sec02.exam06;
02
03 import java.io.Serializable;
04 import java.util.Date;
05
06 public class Board implements Serializable {
07     private int bno;
08     private String title;
09     private String content;
10     private String writer;
11     private Date date;
12
13     public Board(int bno, String title, String content, String writer, Date date) {
14         this.bno = bno;
15         this.title = title;
16         this.content = content;
17         this.writer = writer;
18         this.date = date;
19     }
```

Serializable 인터페이스 구현



## 객체 입출력 보조 스트림

```
20
21     public int getBno() { return bno; }
22     public void setBno(int bno) { this.bno = bno; }
23     public String getTitle() { return title; }
24     public void setTitle(String title) { this.title = title; }
25     public String getContent() { return content; }
26     public void setContent(String content) { this.content = content; }
27     public String getWriter() { return writer; }
28     public void setWriter(String writer) { this.writer = writer; }
29     public Date getDate() { return date; }
30     public void setDate(Date date) { this.date = date; }
31 }
```

---



# 키워드로 끝내는 핵심 포인트

- **보조 스트림** : 다른 스트림과 연결되어 여러 편리한 기능 제공해주는 스트림. 자체적으로 입출력 수행할 수 없기 때문에 입출력 소스와 바로 연결되는 InputStream, OutputStream, Reader, Writer 등에 연결해서 입출력 수행. 문자 변환, 입출력 성능 향상, 기본 타입 입출력 등 기능 제공
- **문자 변환** : 소스 스트림이 바이트 기반 스트림이면서 입출력 데이터가 문자라면 Reader와 Writer로 변환해서 사용하는 것이 편리함. OutputStreamWriter는 Writer로 변환하는 보조 스트림이며, InputStreamReader는 Reader로 변환하는 보조 스트림
- **성능 향상** : 메모리 버퍼 추가로 제공하여 프로그램의 실행 성능을 향상시키는 것들이 있음. 바이트 기반 스트림에서는 BufferedInputStream, BufferedOutputStream, 문자 기반 스트림에서는 BufferedReader, BufferedWriter가 있음
- **기본 타입 입출력** : DataInputStream과 DataOutputStream 보조 스트림 연결하면 기본 타입인 boolean, char, short, int, long, float, double 입출력 가능
- **개행 출력** : PrintStream/PrintWriter의 println() 메소드는 출력할 데이터 끝에 개행 문자인 \n 추가하여 출력 시 콘솔이나 파일에서 줄 바꿈 일어남





## 확인문제

❖ 보조 스트림에 대한 설명 중 맞는 것에 O, 틀린 것에 X표하세요

- InputStreamReader는 InputStream을 Reader로 변환하는 보조 스트림이다 ( )
- BufferedInputStream은 데이터 읽기 성능을 향상시키는 보조 스트림이다 ( )
- DataInputStream은 객체를 입출력하는 보조 스트림이다 ( )
- PrintStream은 라인 단위로 출력할 수 있는 보조 스트림이다 ( )





# 확인문제

- ❖ FileReader와 BufferedReader를 이용해서 source.txt 내용을 읽고, 각 라인 번호를 추가해 모니터로 출력하는 프로그램을 작성해보세요

라인 번호를 출력

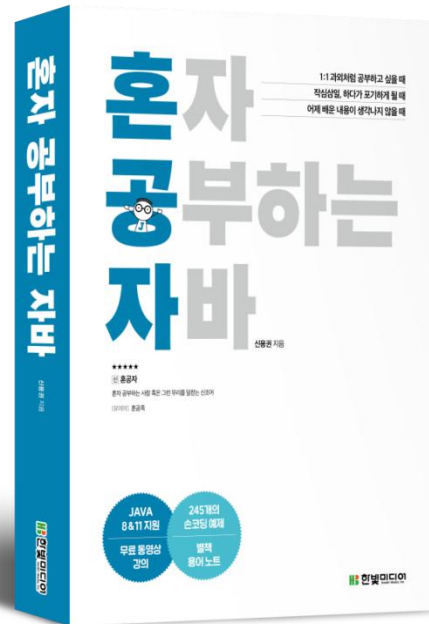
소스 코드 AddLineNumberExample.java

```
01 package sec02.verify.exam02;
02
03 import java.io.BufferedReader;
04 import java.io.FileReader;
05
06 public class AddLineNumberExample {
07     public static void main(String[] args) throws Exception {
08         String filePath = "src/sec02/verify/exam02/AddLineNumberExample.java";
09
10
11
12
13
14
15
16
17
18
19
20     }
21 }
```

# 확인문제

```
실행결과
1: package sec02.verify.exam02;
2:
3: import java.io.BufferedReader;
4: import java.io.FileReader;
5:
6: public class AddLineNumberExample {
7:     public static void main(String[] args) throws Exception {
8:         String filePath = "src/sec02/verify/exam02/AddLineNumberExample.java";
9:         ...
10:     }
11: }
```





Thank You!