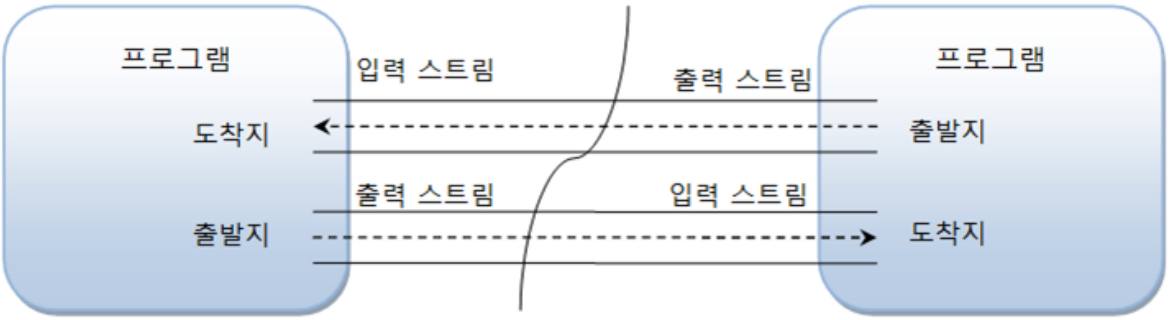
IO 패키지

입출력은 컴퓨터 내부 또는 외부의 장치와 프로그램 간 데이터를 주고받는 것을 말한다. 어느 한쪽에서 다른 쪽으로 데이터를 전달하려면, 두 대상을 연결하고 데이터를 전송할 수 있는 선이 필요하다. 이것이 스느림(stream)이라고 한다. 즉, 스트림이란 데이터를 운반하는데 사용되는 연결통로이다.

1. 입력 스트림과 출력 스트림



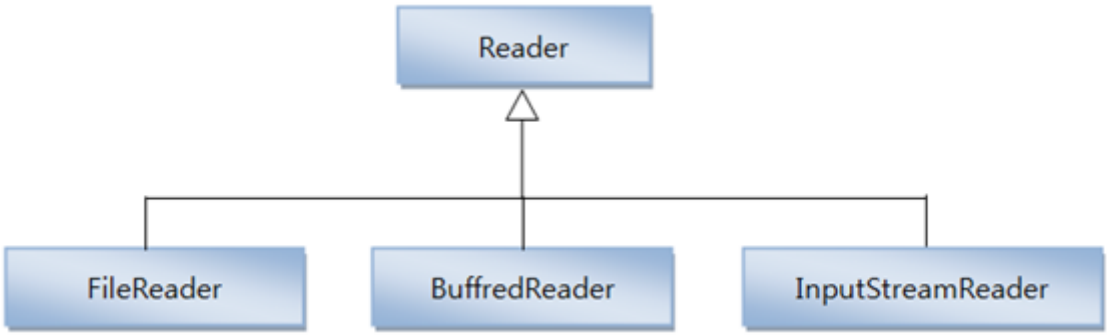
프로그램이 출발지냐 도착지냐에 따라서 스트림의 종류가 결정된다. 데이터를 입력받을 때에는 입력 스트림(InputStream) 라고 부르고, 프로그램이 데이터를 보낼 때에는 출력스트림(OutputStream)이라고 부른다.

자바의 기본적인 데이터 입출력(IO:Input/Output) API는 java.io패키지에서 제공한다.

java.io 패키지의 주요 클래스	설명
File	파일 시스템의 파일의 정보를 얻기위한 클래스
Console	콘솔로부터 문자를 입출력하기 위한 클래스
InputStream / OutputStream	바이트 단위 입출력을 위한 최상위 입출력 스트림 클래스
FileInputStream / FileOutputStream DataInputStream / DataOutputStream ObjectInputStream / ObjectOutputStream PrintStream BufferedInputStream / BufferedOutputStream	바이트 단위 입출력을 위한 하위 스트림 클래스
Reader / Writer	문자 단위 입출력을 위한 최상위 입출력 스트림 클래스
FileReader / FileWriter InutStreamReader / OutputStreamWriter PrintWriter BufferedReader / BufferedWriter	문자 단위 입출력을 위한 하위 스트림 클래스

1.1 Reader

Reader는 문자 기반 입력 스트림의 최상위 클래스로 추상 클래스이다. 모든 문자 기반 입력 스트림은 이 클래스를 상속받아서 만들어진다.



<Reader클래스의 주요 메소드>

메소드		설명
int	read()	입력 스트림으로부터 한개의 문자를 읽고 리턴한다.
int	read(char[] cbuf)	입력 스트림으로부터 읽은 문자들을 매개값으로 주어진 문자 배열 cbuf 에 저장하고 실제로 읽은 문자 수를 리턴한다.
int	read(char[] cbuf, int off, int len)	입력 스트림으로부터 len 개의 문자를 읽고 매개값으로 주어진 문자 배열 cbuf[off] 부터 len 개까지 저장한다. 그리고 실제로 읽은 문자 수인 len 개를 리턴한다.
void	close()	사용한 시스템 자원을 반납하고 입력 스트림을 닫는다.

read() 메소드

입력 스트림으로부터 한 개의 문자(2byte)를 읽고 4btye **int타입으로 반환**한다. 읽은 문자를 얻기 위해서는 (char)타입으로 변환이 필요하다.

```
public class ReadExample1{
    public static void main(String[] args) throws Exception{
        Reader reader = new FileReader(" /*파일의 경로+파일명*/ ");
```

```
        int readData;
        while(true){
            readData = reader.read();
            if(readData == -1){
                break;
            }
            System.out.print((char)readData);
        }
        reader.close();
    }
}
```

read(char[] cbuf) 메소드

입력 스트림으로부터 매개값으로 주어진 문자 배열의 길이만큼 문자를 읽고 배열에 저장한다. 그리고 읽은 문자 수를 반환한다.

read(char[] cbuf, int off, int len) 메소드

입력 스트림으로부터 len개의 문자만큼 읽고 매개값으로 주어진 문자 배열 cbuf(off)부터 len개까지 저장한다. 그리고 읽은 문자 수인 len개를 반환한다.

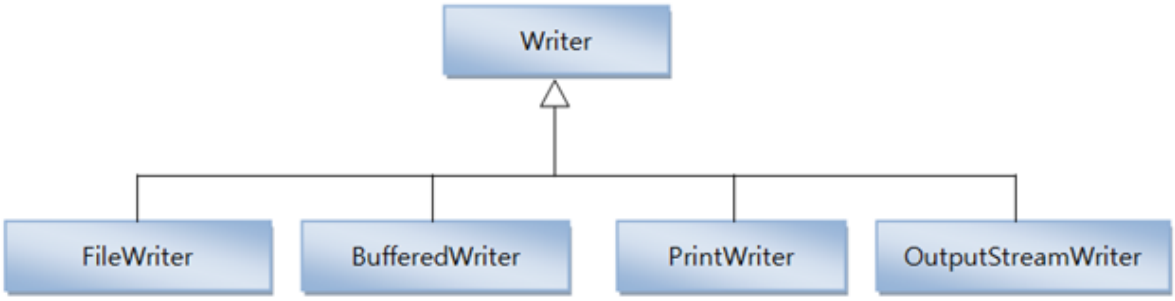
```
public class ReadExample2{
    public static void main(String[] args) throws Exception{
        Reader reader = new FileReader(" /*파일의 경로+파일명*/ ");
        int readCharNo;
        char[] cbuf = new char[2];
        readCharNo = reader.read()
        String data = "";
        while(readCharNo = reader.read(cbuf) != -1){
            data += new String(cbuf, 0, readCharNo);
        }
        System.out.print(daya);
        reader.close();
    }
}
```

close() 메소드

마지막으로 Reader를 더 이상 사용하지 않는 경우에는 close() 메소드를 호출해서 Reader에서 사용했던 시스템 자원을 풀어준다.

1.2 Writer 클래스

문자 기반 입력 스트림의 최상위 클래스로 추상 클래스이다. 모든 문자 기반 출력스트림은 이 클래스를 상속받아서 만들어진다.



<Writer의 주요 메소드>

리턴타입	메소드	설명
void	wrtie(int c)	출력 스트림으로 매개값으로 주어진 한 문자를 보낸다.
void	write(char[] cbuf)	출력 스트림에 매개값으로 주어진 문자 배열 cbuf 의 모든 문자를 보낸다.
void	write(char[] cbuf, int off, int len)	출력 스트림에 매개값으로 주어진 문자 배열 cbuf[off] 부터 len 개까지의 문자를 보낸다.
void	write(String str)	출력 스트림에 매개값으로 주어진 문자열을 전부 보낸다.
void	write(String str, int off, int len)	출력 스트림에 매개값으로 주어진 문자열 off 순번부터 len 개까지의 문자를 보낸다.
void	flush()	버퍼에 잔류하는 모든 문자열을 출력한다.
void	close()	사용한 시스템 자원을 반납하고 출력 스트림을 닫는다.

write(int c) 메소드

매개 변수로 주어진 int 값에서 끝에 있는 **2바이트만** 출력 스트림으로 보낸다.

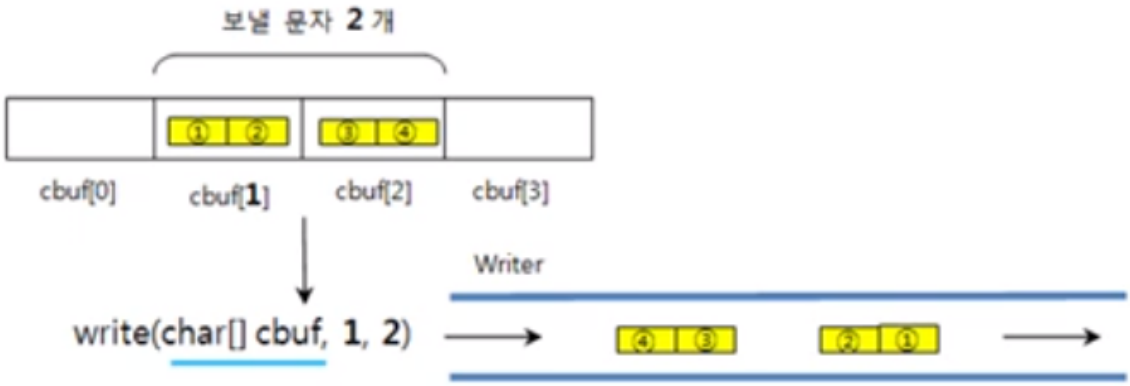

```
public class WriteExample1{
    public static void main(String[] args) throws Exception{
        Writer writer = new FileWriter(" /*파일의 경로+파일명*/ ");
        char[] data = "홍길동".toCharArray();
        for(int i=0; i<data.length; i++){
            writer.write(data[i])
        }
       os.flush()
       os.close();
    }
}
```

write(char[] cbuf) 메소드

매개값으로 주어진 char[] 배열의 모든 문자를 출력 스트림으로 보낸다.

write(char[] cbuf, int off, int len) 메소드

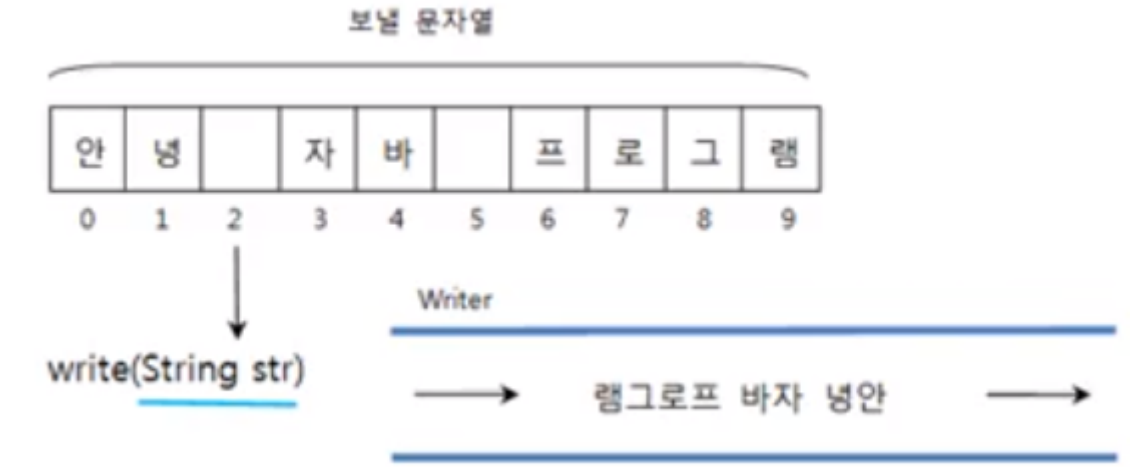
cbuf[off]부터 len개의 문자를 출력 스트림으로 보낸다.

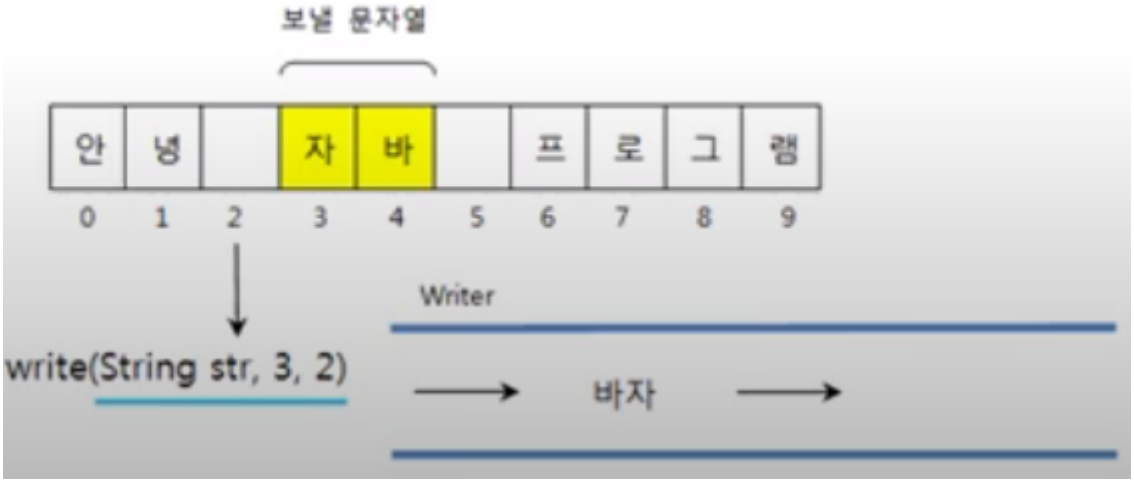



```
public class WriteExample1{
    public static void main(String[] args) throws Exception{
        Writer writer = new FileWriter(" /*파일의 경로+파일명*/ ");
        char[] data = "홍길동".toCharArray();
        writer.write(data,1,2);
        os.flush()
        os.close();
    }
}
```

write(String str)과 write(String str, int off, int len)메소드

문자열을 조금 더 쉽게 보내기 위해서 고안





2. 콘솔 입출력

콘솔(Console)은 시스템을 사용하기 위해 키보드로 입력을 받고 화면에 출력하는 소프트웨어이다. 콘솔로부터 데이터를 입력받을 때 System.in을 사용하고, 데이터를 출력할 때는 System.out, 에러를 출력할 때에는 System.err를 사용한다.

2.1 System.* 필드

System.in필드

```
InputStream is = System.in;
int ascilCode = is.read();
char inputChar = (char)is.read();
```

InputStream의 read() 메소드는 1바이트만 읽기 때문에 1바이트의 아스키 코드로 표현되는 숫자, 영어, 특수문자는 프로그램에서 잘 읽을 수 있지만, 한글과 같이 2바이트를 필요로 하는 유니코드는 read() 메소드로 읽을 수 없다. 키보드로 한글을 얻기 위해서는 전체 입력된 내용을 바이트배열(read(byte b))로 받고, 이 배열을 이용해서 String객체를 생성해야 한다.

```
byte [] byteData = new byte[15];
int readByteNo = System.in.read(byteData);
```

```
public class SystemOutExample {
    public static void main(String[] args) throws IOException {

        InputStream is = System.in;
        byte[] datas = new byte[100];

        System.out.print("이름 : ");
        int nameBytes = is.read(datas);
        String name = new String(datas, 0, nameBytes-2);

        System.out.print("코멘트 : ");
        int commentBytes = is.read(datas);
        String comment = new String(datas, 0, nameBytes-2);

        System.out.println("입력한 이름은 : "+name);
        System.out.println("입력한 코멘트는 : "+comment);

    }
}
```

System.out 필드

in과 반대로 콘솔로 데이터를 출력하기 위해서는 out필드를 사용한다.

```
OutputStream os = System.out;
```

OutputStream의 write(int b) 메소드는 1바이트만 보낼 수 있기 때문에 1바이트로 표현 가능한 숫자, 영어, 특수문자는 출력이 가능하지만, 2바이트로 표현되는 한글을 출력할 수 없다. 한글을 출력하기 위해서는 한글을 바이트 배열로 얻은 다음, write(byte[] b) 메소드로 콘솔에 출력하면 된다.

```
String name = "홍길동";
byte[] nameBytes = name.getBytes();
os.write(nameBytes);
os.flush();
```

2.2 Scanner 클래스

```
Scanner scanner = new Scanner(System.in);
```

리턴타입	메소드	설명
boolean	nextBoolean()	boolean(true/false) 값을 읽는다.
byte	nextByte()	byte 값을 읽는다.
short	nextShort()	short 값을 읽는다.
int	nextInt()	int 값을 읽는다.
long	nextLong()	long 값을 읽는다.
float	nextFloat()	float 값을 읽는다.
double	nextDouble()	double 값을 읽는다.
String	nextLine()	String 값을 읽는다.

3. 파일 입출력

3.1 File클래스

java.io에서 제공하는 File 클래스는 파일 크기, 파일 속성, 파일 이름 등의 정보를 얻어내는 기능과 파일 생성 및 삭제 기능을 제공한다.

```
File file = new File( /* 파일위치+파일명 */ );
```

File 객체를 생성했다고 해서 파일이나 디렉토리가 생성되는 것이 아니다. 생성자 매개값으로 주어진 경로가 유효하지 않더라도 컴파일 에러나 예외가 발생하지 않는다. File객체를 통해 해당 경로에 실제로 파일이나 폴더가 있는지 exists( ) 메소드를 통하여 확인할 수 있다.

```
boolean isExist = file.exists();
```

만약 exists() 메소드의 리턴값이 false라면 아래 메소드를 통하여 파일 또는 폴더를 생성할 수 있다.

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

파일이나 폴더가 이미 존재할 경우에는 아래 메소도를 통하여 정보를 얻을 수 있다.

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

3.2 FileInputStream 클래스

파일로부터 바이트 단위로 읽어들일 때 사용하는 바이트 기반 입력 스트림이다. 바이트 단위로 읽기 때문에 그림, 오디오, 비디오, 텍스트 파일 등 모든 종류의 파일을 읽을 수 있다.

```
File file = new FileInputStream("/*파일의 경로*/")
```

```
public class FileInputStreamExample {
    public static void main(String[] args) throws Exception  {

        FileInputStream fis = new FileInputStream("null"/*텍스트파일*/);

        int data;
        while((data = fis.read())!=-1) {
            System.out.println(data);
        }
        fis.close();
    }
}
```

3.3 FileOutputStream 클래스

바이트 단위로 데이터를 저장할 때 사용하는 바이트 기반 출력 스트림이다. 모든 종류의 데이터를 파일로 저장할 수 있다.

```
FileOutputStream fos = new FileOutputStream("/*경로*/", true)
```

```
public class File_IO_stream_example {

    public static void main(String[] args) throws Exception {

        String originalName = "/*경로명 + 파일 이름*/";
        String targetName = "/*경로명 + 파일 이름*/";

        FileInputStream fis = new FileInputStream(originalName);
        FileOutputStream fos = new FileOutputStream(targetName);

        int readByteNo;
        byte[] readBytes = new byte[10];

        while((readByteNo = fis.read(readBytes))!=-1) {
            fos.write(readBytes, 0, readByteNo);
        }

        fos.flush();
        fos.close();
        fis.close();
    }
}
```

3.4 FileReader

텍스트 파일을 프로그램으로 읽어들일 때 사용하는 문자 기반 스트림이다. 문자 단위로 읽기 때문에 그림, 오디오, 비디오 등의 파일은 읽을 수 없다.

```
FileReader fr = new FileReader(/*.txt*/)
```

```
public class FileReaderExample {

    public static void main(String[] args) throws Exception {
        FileReader fr = new FileReader("/*파일.txt*/");


        char[] cbuf = new char[100];
        int i = fr.read(cbuf);

        while(i != -1) {
            String data = new String(cbuf, 0, i);
            System.out.println(data);
        }

        fr.close();
    }
}
```

3.5 FileWriter

텍스트 데이터를 저장할 때 사용하는 문자 기반 스트림이다.

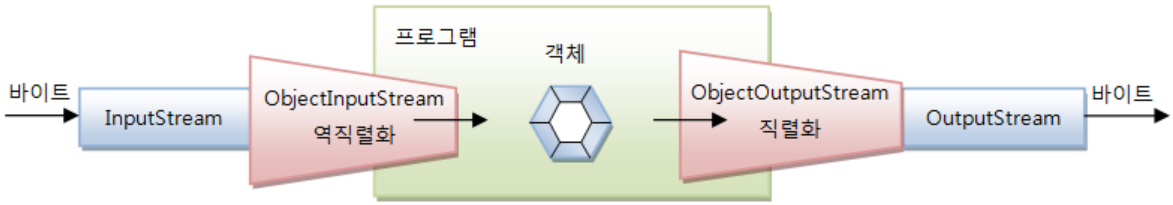
```
FileWriter fw = new FileWriter(""/*.txt*/"",true);
```

```
public class FileWriterExample {
    public static void main(String[] args) throws Exception {

        File file = new File("/*.txt*/");
        FileWriter fw = new FileWriter("/*.txt*/", true);
        fw.write("FileWriter는 한글로된");
        fw.write("문자를 바로 출력할 수 있습니다.");
        fw.flush();
        fw.close();
    }
}
```

4. 직렬화(serialization)

메모리에 생성된 객체를 파일 또는 네트워크로 출력할 수 있다. 객체는 문자가 아니기 때문에 바이트 기반 스트림으로 출력해야 하는데 객체를 출력하기 위해서는 객체의 데이터(필드값)를 일렬로 늘어선 연속적인 바이트로 변경해야한다. 이것을 객체 직렬화(serialization)이라고 한다.




```
//다른 보조 스트림과 마찬가지로 연결할 ㅂ바이트 입출력 스트림을 생성자의 매개값으로 받는다.
ObjectInputStream ois = new ObjectInputStream(바이트입력스트림);
ObjectOutputStream oos = new ObjectOutputStream(바이트출력스트림);

//객체를 적렬화하기 위해 writeObject() 메소드 사용
oos.writeObject(객체);

//반대로 ObjectInputStream의 readObject() 메소드는 입력 스트림에서 읽은 바이트를 역직렬화해서 객체로 생성한다.
객체타입 변수 = (객체타입)ois.readObject();
```

- 직렬화 가능한 클래스(Serializable)
  - Serializable 인터페이스를 구현한 클래스만 직렬화 가능(**transient 필드 제외**)
- writeObject() 메소드는 직렬화 직전 자동으로 호출되며 추가로 직렬화 내용을 작성할 수 있다.
- readObject()메소드는 역직렬화 직전 자동으로 호출되며 추가로 역직화화 내용을 작성할 수 있다.

```
public class ClassA implements Serializable {
    int field1;
    ClassB field2 = new ClassB();
    static int field3;
    transient int field4;
}
```

```
public class ClassB implements Serializable {
    int field1;
}
```

```
public class SerializableWriter {
    public static void main(String[] args) throws Exception {
        FileOutputStream fos = new FileOutputStream("/*파일 경로*/");
        ObjectOutputStream oos = new ObjectOutputStream(fos);

        ClassA c = new ClassA();
        c.field1 = 1;
        c.field2.field1 = 2;
        c.field3 = 3;
        c.field4 = 4;

        oos.writeObject(c);
        oos.flush();
        oos.close();
        fos.close();

        FileInputStream fis = new FileInputStream("/*파일 경로*/");
        ObjectInputStream ois = new ObjectInputStream(fis);
```

```
        ClassA v = (ClassA)ois.readObject();
        System.out.println(v.field1);
        System.out.println(v.field2.field1);
        System.out.println(v.field3);
        System.out.println(v.field4);
    }
}
```