# 파일 입출력

강사 : 강병준

# 입출력(I/O)과 스트림(stream)

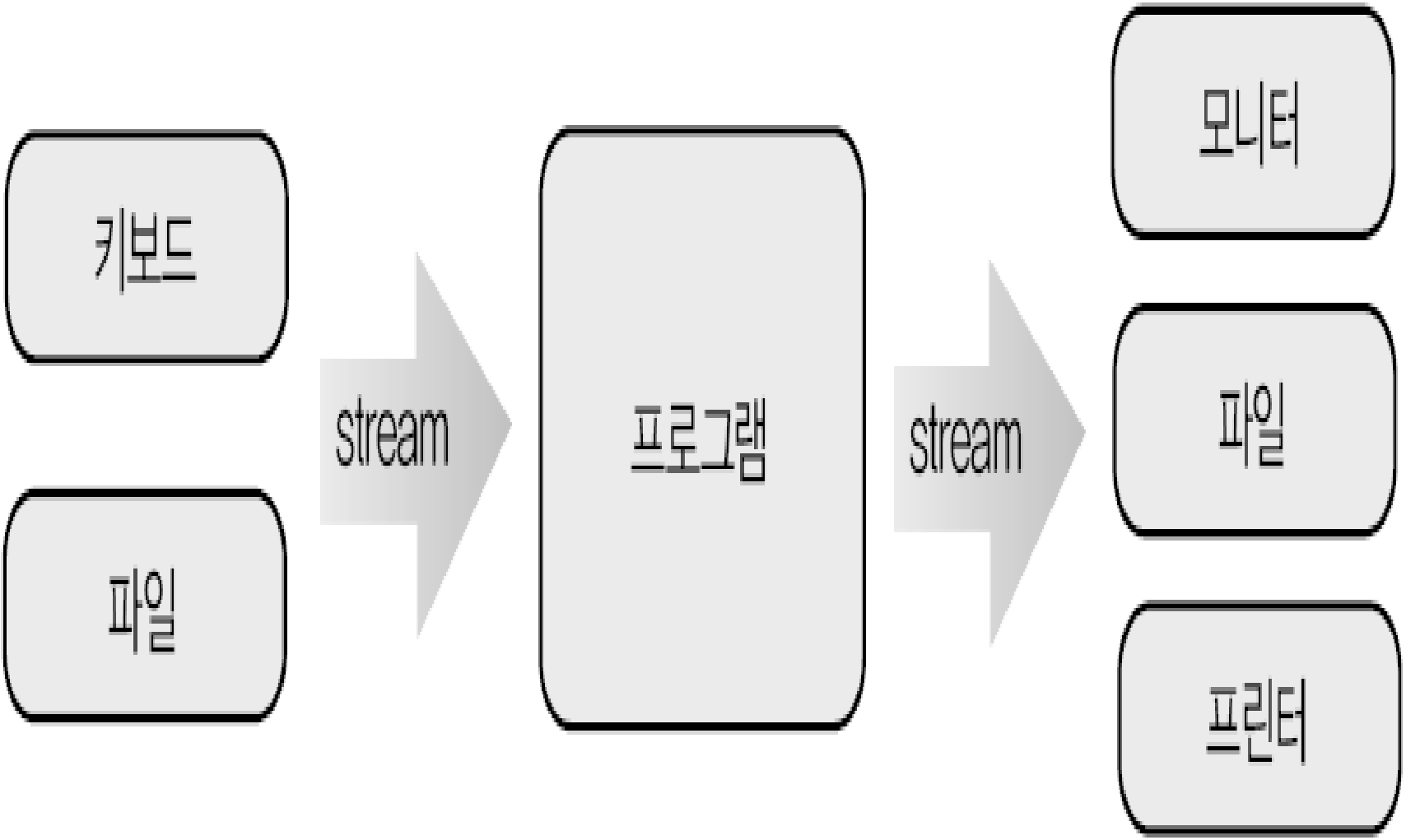
▶ 입출력(I/O)이란?

- 입력(Input)과 출력(Output)을 줄여 부르는 말
- 두 대상 간의 데이터를 주고 받는 것

▶ 스트림(stream)이란?

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

# 스트림의 이해과 File 객체



# 바이트기반 스트림

## 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 |
|---|---|---|
| **File**InputStream | **File**OutputStream | 파일 |
| **ByteArray**InputStream | **ByteArray**OutputStream | 메모리 |
| **Piped**InputStream | **Piped**OutputStream | 프로세스 |
| **Audio**InputStream | **Audio**OutputStream | 오디오장치 |

# 보조스트림

- 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.
```

| 입력 | 출력 | 설명 |
|---|---|---|
| FilterInputStream | FilterOutputStream | 필터를 이용한 입출력 처리 |
| BufferedInputStream | BufferedOutputStream | 버퍼를 이용한 입출력 성능향상 |
| DataInputStream | DataOutputStream | int, float와 같은 기본형 단위(primitive type)로 데이터를 처리하는 기능 |
| SequenceInputStream | SequenceOutputStream | 두 개의 스트림을 하나로 연결 |
| LineNumberInputStream | 없음 | 읽어 온 데이터의 라인 번호를 카운트<br>(JDK1.1부터 LineNumberReader로 대체) |
| ObjectInputStream | ObjectOutputStream | 데이터를 객체단위로 읽고 쓰는데 사용.<br>주로 파일을 이용하며 객체 직렬화와 관련있음 |
| 없음 | PrintStream | 버퍼를 이용하며, 추가적인 print관련 기능(print, printf, println메서드) |
| PushbackInputStream | 없음 | 버퍼를 이용해서 읽어 온 데이터를 다시 되돌리는 기능<br>(unread, push back to buffer) |

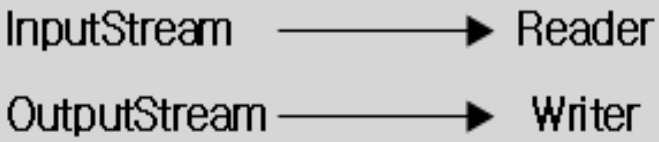
# 문자기반 스트림 – Reader, Writer

- 입출력 단위가 문자(char, 2 byte)인 스트림. 문자기반 스트림의 최고조상

| 바이트기반 스트림 | 문자기반 스트림 | 대상 |
|---|---|---|
| **File**InputStream<br>**File**OutputStream | **File**Reader<br>**File**Writer | 파일 |
| **ByteArray**InputStream<br>**ByteArray**OutputStream | **CharArray**Reader<br>**CharArray**Writer | 메모리 |
| **Piped**InputStream<br>**Piped**OutputStream | **Piped**Reader<br>**Piped**Writer | 프로세스 |
| ***StringBuffer****InputStream*<br>***StringBuffer****OutputStream* | **String**Reader<br>**String**Writer | 메모리 |

| 바이트기반 보조스트림 | 문자기반 보조스트림 |
|---|---|
| **Buffered**InputStream<br>**Buffered**OutputStream | **Buffered**Reader<br>**Buffered**Writer |
| **Filter**InputStream<br>**Filter**OutputStream | **Filter**Reader<br>**Filter**Writer |
| ***LineNumber****InputStream* | **LineNumber**Reader |
| **Print**Stream | **Print**Writer |
| **Pushback**InputStream | **Pushback**Reader |

# InputStream과 OutputStream




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

# InputStream과 OutputStream

▶ InputStream(바이트기반 입력스트림의 최고 조상)의 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. |

# InputStream과 OutputStream

▶ OutputStream(바이트기반 출력스트림의 최고 조상)의 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. |

## ByteArrayInputStream과 ByteArrayOutputStream

- 바이트배열(byte[])에 데이터를 입출력하는 바이트기반 스트림

| | |
|---|---|
| abstract int read() | 1 byte를 읽어 온다(0~255사이의 값). 더 이상 읽어 올 데이터가 없으면 -1을 반환한다. |

# InputStream과 OutputStream

```
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();

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

# File 클래스

| 표 11-5 | File 클래스의 유용한 생성자

| public File(String pathname) |
| --- |
| pathname을 경로로 하는 파일 객체를 만든다. |
| **public File(String parent, String child)** |
| 'parent + child'를 경로로 하는 파일 객체를 만든다. |
| **public File(File parent, String child)** |
| 'parent(부모 파일) + child'를 경로로 하는 파일 객체를 만든다. |

```
File f=new File("sample.txt");
File f=new File("c:\\java_work\\chapter11\\sample.txt");
```

# File 클래스

| 표 11-6 | File 클래스의 유용한 메소드

| |
|---|
| public boolean canRead( ), public boolean canWrite( ) |
| 읽을 수 있거나(can Read) 쓸 수 있는(can Write) 파일이면 true를, 아니면 false를 반환한다. |
| public boolean createNewFile( ) throws IOException |
| this가 존재하지 않는 파일이면 새 파일을 만들고 true를 반환하다. this가 존재하는 파일이면 false를 반환한다. |
| public static File createTempFile(String prefix, String suffix) throws IOException |
| 파일 이름이 prefix이고 확장자는 suffix인 임시 파일을 임시(Temp) 디렉터리에 만든다. |
| public boolean isDirectory( ) |
| 디렉터리이면 true를, 아니면 false를 반환한다. |

# File 클래스

| |
|---|
| public boolean isFile( ) |
| 파일이면 true를, 아니면 false를 반환한다. |
| public boolean isHidden( ) |
| 숨겨진 파일이면 true를, 아니면 false를 반환한다. |
| public long lastModified( ) |
| 마지막으로 수정된 날짜를 long으로 반환한다. |
| public long length( ) |
| 파일의 크기(bytes)를 반환한다. |
| public String getName( ), public String getParent( ), public String getPath( ) |
| 각각 파일의 이름, 파일이 있는 디렉터리의 경로, 파일의 경로를 반환한다. |

# File 클래스

파일 : 관련된 데이터 묶음
디렉토리 : 파일의 묶음
**Java.io의 File**

```
import java.io.File;

public class Dir {
        public static void main (String args[]) {
                // 현재 디렉토리 출력
                File dir = new File (".");
                String[] strs = dir.list();
                for(int i = 0; i < strs.length; i++) {
                        System.out.println (strs[i]);
                }
    }
}
```

# File 클래스

```
import java.io.*;
import java.util.*;
class FileInfo {
public static void main(String [] args) throws Exception {
    File file = new File( “.” );
    System.out.println(  " 파일 상세 정보 *****" );
    System.out.println("절대 경로 :" + file.getAbsolutePath());
    System.out.println("표준 경로 :" + file.getCanonicalPath());
    System.out.println("생성일    :" +
                              new Date(file.lastModified()));
    System.out.println( "파일 크기 : " + file.length() );
    System.out.println( "읽기 속성 : " + file.canRead() );
    System.out.println( "쓰기 속성 : " + file.canWrite() );
    System.out.println( "파일 경로 : " + file.getParent() );
    System.out.println( "숨김 속성 : " + file.isHidden() );
    }
}
```

# File 클래스

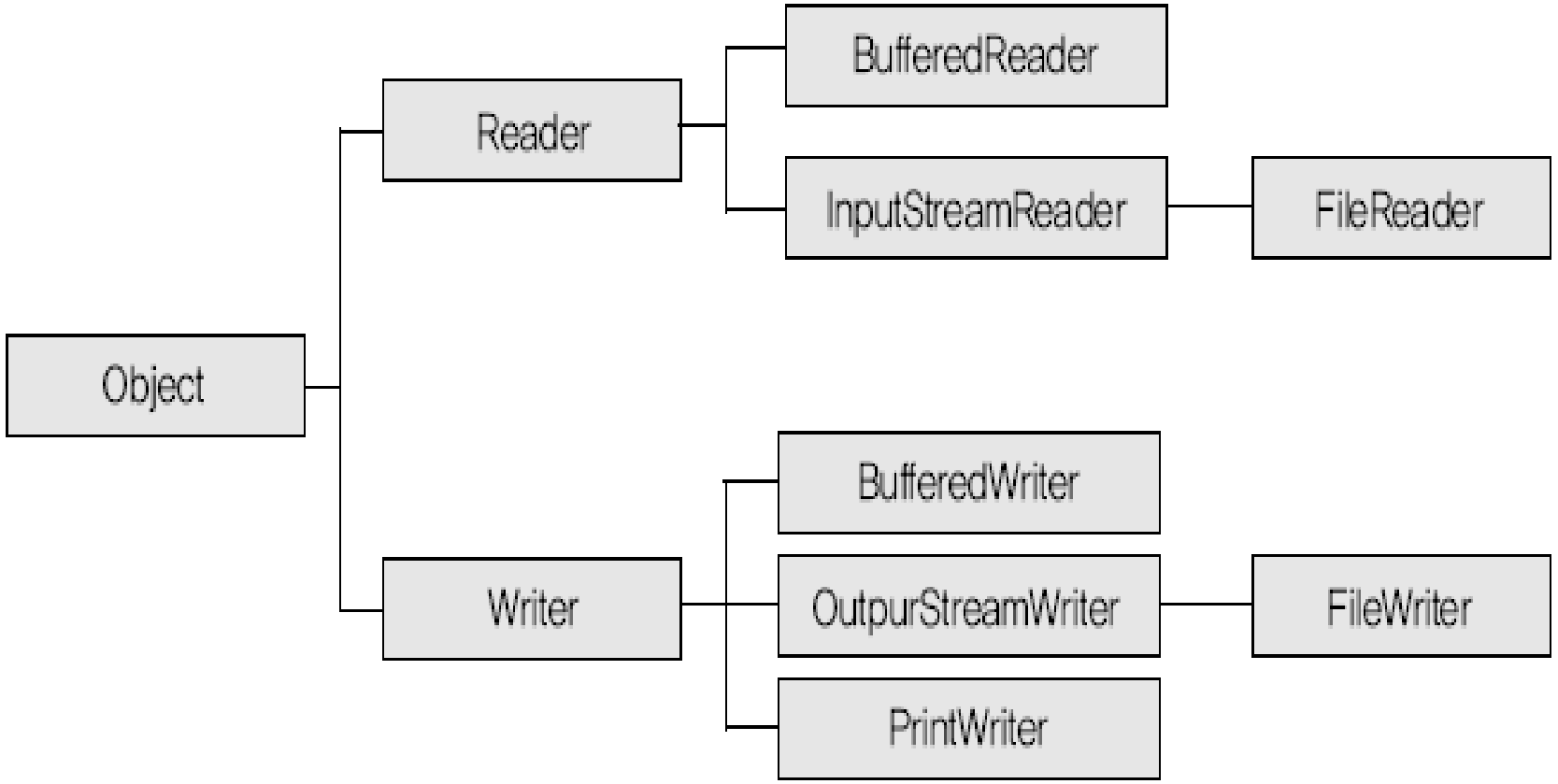
```
import java.io.*;      import java.util.*;
class JDir {
   public static void main(String[] args) throws Exception {
      File file = new File(".classpath");
      if(! file.exists())   {
          System.out.println( " 파일이 존재하지 않습니다.");
         System.exit(1);
      } // if
     System.out.println("절대 경로 :" + file.getAbsolutePath());
     System.out.println("표준 경로 :" + file.getCanonicalPath());
     System.out.println("생성일    :" + new Date(file.lastModified()));
     if(file.isDirectory()) {   // outer if
         File [] list = file.listFiles() ;
         System.out.println(" 디렉토리 파일 목록 출력 **");
          for( int i = 0 ; i < list.length ; i++){
              if(list[i].isFile()) {// inner if
                System.out.print( list[i].getName() +"\t" + list[i].length());
                 System.out.println();
              }  else { // inner if
                System.out.print( list[i].getName() +"\t [디렉토리]");
                System.out.println();
```

# File 클래스

```
} // else
        } // for
    } // outer if

else { System.out.println( " 파일 상세 정보 *****" );
            System.out.println( "파일 크기 : " + file.length() );
            System.out.println( "읽기 속성 : " + file.canRead() );
            System.out.println( "쓰기 속성 : " + file.canWrite() );
            System.out.println( "파일 경로 : " + file.getParent() );
            System.out.println( "숨김 속성 : " + file.isHidden() );
        } // else
    } // main
} //class
```

# 문자 기반 스트림

**자바의 Writer 클래스와 Reader 클래스는 2byte 유니코드로 입출력할 수 있는 문자 기반 스트림을 제공한다.**



# InputStreamReader, OutputStreamWriter


InputStreamReader 클래스의 유용한 생성자

| 생성자 | 설 명 |
| --- | --- |
| InputStreamReader(InputStream instreams) | instreams : 입력될 바이트 타입의 스트림(InputStream 클래스) |
| InputStreamReader(InputStream instreams, String encoding) | encoding : 변환 방법 |

```
InputStreamReader isr=new InputStreamReader(System.in);
```

# BufferedReader, BufferedWriter


## BufferedReader, BufferedWriter

버퍼를 사용하면 보다 효율적으로 데이터를 처리할 수 있는데, 이를 위해 BufferedReader, BufferedWriter 클래스가 제공된다.

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

# BufferedReader, BufferedWriter

```
import java.io.*;
class ReadLine {
   public static void main(String [] args)throws IOException{
         String name=null, addr=null ;
         BufferedReader reader = new BufferedReader(new
              InputStreamReader(System.in));
         System.out.print("이름을 입력하세요 => " );
         name = reader.readLine();
         System.out.print("주소를 입력하세요 => ");
         addr = reader.readLine();
         System.out.println(name+"님 반가워요! 당신은 "+ addr +
            "에 살고 계시군요..^^");
   }
}
```

# FileReader, FileWriter

## FileReader, FileWriter

FileReader 클래스와 FileWriter 클래스는 파일에서 데이터를 읽거나 저장하는 각종 메소드를 제공한다.

FileReader 클래스는 파일 정보를 쉽게 읽을 수 있도록 InputStreamReader 클래스를 상속받아 만든 클래스

FileWriter 클래스는 출력할 유니코드 문자를 바이트로 변환하여 파일에 저장

# FileReader, FileWriter

```
import java.util.*;
class FileWrite{
    public static void main(String[] args) throws Exception {
        String file, str;
        Date date = new Date();
        str = "파일 생성시간\n" + date +"\n" ;
        BufferedReader read = new BufferedReader( new
                InputStreamReader(System.in));
        System.out.print("파일 이름을 입력하세요 >> ");
        file = read.readLine();
        System.out.println("저장할 문자열을 입력하세요 ==> ");
        str += read.readLine();
        char ch_str[] = new char[str.length()];
        str.getChars(0, str.length(), ch_str, 0);
        FileWriter fw = new FileWriter(file);
        fw.write(ch_str);
        fw.close();
        System.out.println( file + " 파일을 성공적으로 저장했습니다.");
    }
}
```

# FileReader, FileWriter

```
import java.io.*;
class FileRead{
   public static void main(String[] args) throws Exception {
        int i = 0;
        String file;

        BufferedReader read = new BufferedReader( new
            InputStreamReader(System.in));

        System.out.print("읽어올 파일 이름을 입력하세요 >> ");
        file = read.readLine();
        System.out.println(file +": 문서 내용 ********** ");
        FileReader fr = new FileReader(file);

        while(( i = fr.read() ) != -1){
             System.out.print((char)i);
        }
        fr.close();
    }
}
```

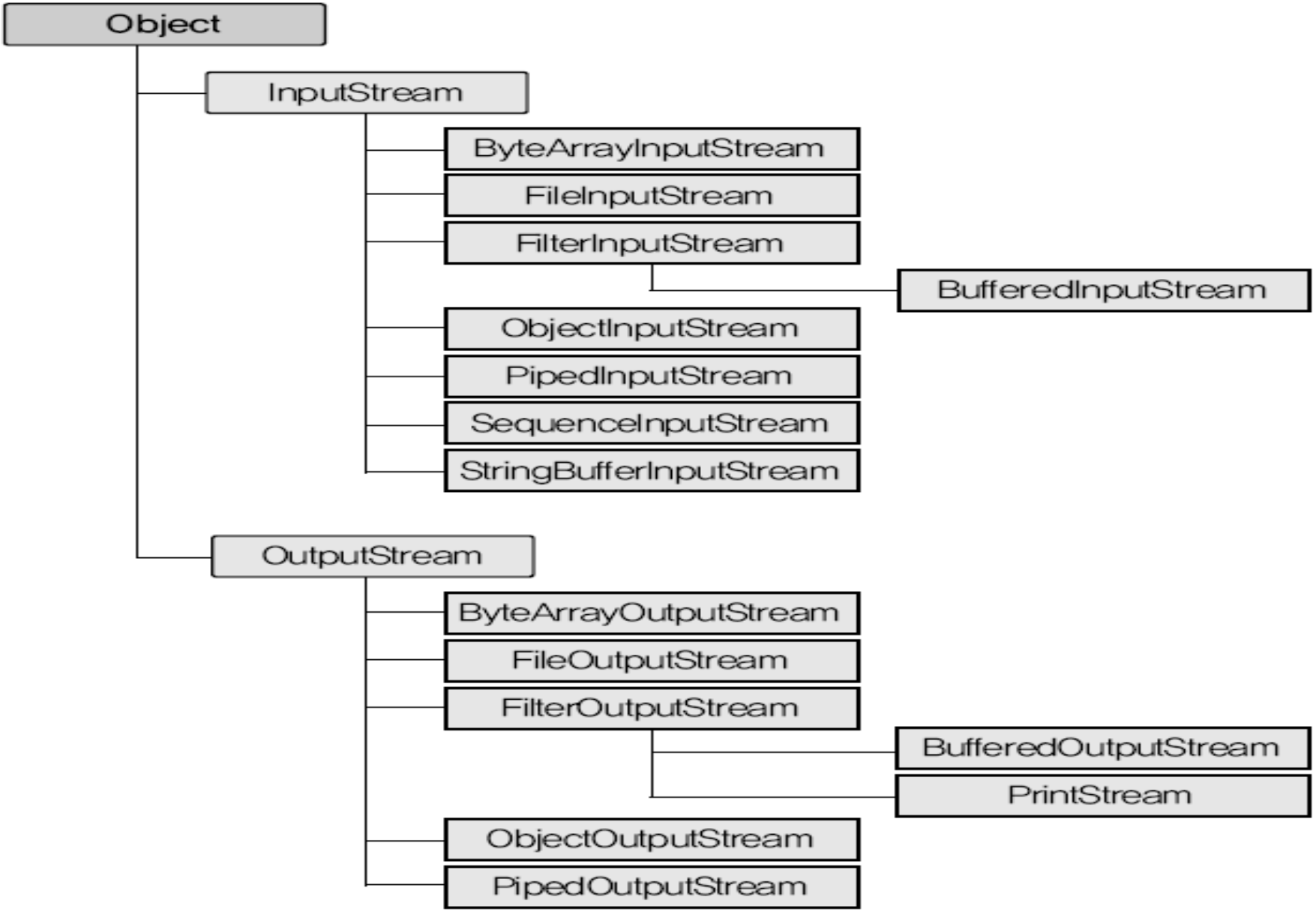
# FileReader, FileWriter

```
import java.io.*;     import java.util.*;
class FileBufferWrite{
   public static void main(String [] args) throws Exception {
         String file , str ;
         Date date = new Date();
         str = "파일 생성시간\n" + date +"\n" ;
         BufferedReader read = new BufferedReader( new
              InputStreamReader(System.in));
         System.out.print("파일 이름을 입력하세요 >> ");

         file = read.readLine();
         System.out.println("저장할 문자열을 입력하세요 ==> ");
         str += read.readLine();
         FileWriter fw = new FileWriter(file);
         BufferedWriter bw = new BufferedWriter(fw);
         bw.write(str);
         bw.close();
         System.out.println( file + " 파일을 성공적으로 저장했습니다.");
   }
}
```

# FileReader, FileWriter

```
import java.io.*;
class FileBufferRead{
    public static void main(String [] args) throws Exception {
        String str, file ;
        BufferedReader read = new BufferedReader( new
            InputStreamReader(System.in));
        System.out.print("읽어올 파일 이름을 입력하세요 >> ");
        file = read.readLine();
        System.out.println(file +": 문서 내용 ********** ");
        FileReader fr = new FileReader(file);
        BufferedReader br = new BufferedReader(fr);

        while( (str = br.readLine())!= null){
            System.out.println(str);
        }
        br.close();
    }
}
```

# InputStream과 OutputStream



# InputStream과 OutputStream

InputStream 클래스의 주요 메소드

| public abstract int read( ) throws IOException; |
| --- |
| 스트림으로부터 다음의1 byte를 읽어와서 int로 반환한다. 파일의 모든 데이터를 읽어왔거나 더 이상 읽어 올 데이터가 없으면 –1을 반환한다. |
| public int read(byte b[ ]) throws IOException |
| 스트림으로부터 다수의 byte를 읽어와서 b에 대입한다. 실제로 읽어온 byte 수를 반환하거나 읽어 올 것이 없으면 –1을 반환한다. |
| public int read(byte b[ ], int off, int len) throws IOException |
| 스트림으로부터 len 만큼의 byte를 읽어와서 b[off]부터 차례대로 대입한다. 실제로 읽어온 byte 수를 반환하거나 읽어 올 것이 없으면 –1을 반환한다. |
| public void close( ) throws IOException |
| 스트림을 닫는다. 스트림과 관련된 시스템 자원(메모리)을 반납한다. |

# InputStream과 OutputStream

OutputStream 클래스의 주요 메소드

| |
|---|
| public void write(byte b[ ], int off, int len) throws IOException |
| b[off]부터 b[len+off-1]까지의 데이터를 스트림에 출력한다. |
| public void write(byte b[ ]) throws IOException |
| 바이트 배열 b를 스트림에 출력한다. |
| public abstract void write(int b) throws IOException; |
| b를 byte로 형변환하여 스트림에 출력한다. 1byte를 기록한다. |
| public void close( ) throws IOException |
| 스트림을 닫는다. 스트림과 관련된 시스템 자원(메모리)을 반납한다. |

# FileInputStream과 FileOutputStream

- 파일(file)에 데이터를 입출력하는 바이트기반 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. |

# FileInputStream과 FileOutputStream

```
import java.io.*;
import java.util.*;
class FileOutputStreamExample{
    public static void main(String[] args) throws Exception {
        String file, str;
        Date date = new Date();
        str = "파일 생성시간\n" + date +"\n" ;
        BufferedReader read = new BufferedReader( new
            InputStreamReader(System.in));
        System.out.print("파일 이름을 입력하세요 >> ");
        file = read.readLine();
        System.out.println("저장할 문자열을 입력하세요 ==> ");
        str += read.readLine();
        byte byte_str[] = str.getBytes();
        FileOutputStream fos = new FileOutputStream(file);
        fos.write(byte_str);
        fos.close();
        System.out.println( file + " 파일을 성공적으로 저장했습니다.");
    }
}
```

# FileInputStream과 FileOutputStream

```
import java.io.*;
class FileInputStreamExample{
    public static void main(String[] args) throws Exception {
        int i = 0;
        String file;
        BufferedReader read = new BufferedReader( new
            InputStreamReader(System.in));
        System.out.print("읽어올 파일 이름을 입력하세요 >> ");
        file = read.readLine();
        System.out.println(file +": 문서 내용 ********** ");
        FileInputStream fis = new FileInputStream(file);

        while((i = fis.read()) != -1){
                System.out.print((char)i);
        }
        fis.close();
    }
}
```

# FileInputStream과 FileOutputStream

```
import java.io.*;     import java.util.*;
class BufferedOutputStreamExample{
   public static void main(String[] args) throws Exception {
         String file, str;
         Date date = new Date();
         str = "파일 생성시간\n" + date +"\n" ;
         BufferedReader read = new BufferedReader( new
             InputStreamReader(System.in));
         System.out.print("파일 이름을 입력하세요 >> ");
         file = read.readLine();
         System.out.println("저장할 문자열을 입력하세요 ==> ");
         str += read.readLine();

         byte byte_str[] = str.getBytes();
         FileOutputStream fos = new FileOutputStream(file);
         BufferedOutputStream bos = new BufferedOutputStream(fos);
         bos.write(byte_str);
         bos.close();
         System.out.println( file + " 파일을 성공적으로 저장했습니다.");
    }
}
```

# FileInputStream과 FileOutputStream

```
import java.io.*;
class BufferedInputStreamExample{
    public static void main(String[] args) throws Exception {
        int i = 0;
        String file;
        BufferedReader read = new BufferedReader( new
            InputStreamReader(System.in));
        System.out.print("읽어올 파일 이름을 입력하세요 >> ");
        file = read.readLine();
        System.out.println(file +": 문서 내용 ********** ");

        FileInputStream fis = new FileInputStream(file);
        BufferedInputStream bis = new BufferedInputStream(fis);

        while((i = bis.read()) != -1){
                System.out.print((char)i);
        }
        bis.close();
    }
}
```

# DataInputStream과 DataOutputStream

- 기본형 단위로 읽고 쓰는 보조스트림
- 각 자료형의 크기가 다르므로 출력할 때와 입력할 때 순서에 주의

| 메서드 / 생성자 | 설 명 |
|---|---|
| DataInputStream(InputStream in) | 주어진 InputStream인스턴스를 기반스트림으로 하는 DataInputStream인스턴스를 생성한다. |
| boolean readBoolean()<br>byte readByte()<br>char readChar()<br>short readShort()<br>int readInt()<br>long readLong()<br>float readFloat()<br>double readDouble() | 각 자료형에 알맞은 값들을 읽어 온다.<br>더 이상 읽을 값이 없으면 EOFException을 발생시킨다. |
| String readUTF() | UTF형식으로 쓰여진 문자를 읽는다. 더 이상 읽을 값이 없으면 EOFException을 발생시킨다. |
| int skipBytes(int n) | 현재 읽고 있는 위치에서 지정된 숫자(n) 만큼을 건너뛴다. |

# DataInputStream과 DataOutputStream

| 메서드 / 생성자 | 설 명 |
| --- | --- |
| DataOutputStream(OutputStream out) | 주어진 OutputStream인스턴스를 기반스트림으로 하는 DataOutputStream인스턴스를 생성한다. |
| void writeBoolean(boolean b)<br>void writeByte(int b)<br>void writeChar(int c)<br>void writeShort(int s)<br>void writeInt(int l)<br>void writeLong(long l)<br>void writeFloat(float f)<br>void writeDouble(double d) | 각 자료형에 알맞은 값들을 출력한다. |
| void writeUTF(String s) | UTF형식으로 문자를 출력한다. |
| void writeChars(String s) | 주어진 문자열을 출력한다. writeChar(char c)메서드를 여러 번 호출한 결과와 같다. |
| int size() | 지금까지 DataOutputStream에 쓰여진 byte의 수를 알려준다. |

# DataInputStream과 DataOutputStream

```
import java.io.*;    import java.util.Arrays;
class DataOutputStreamEx2 {
    public static void main(String args[]) {
        ByteArrayOutputStream bos = null; DataOutputStream dos = null;
        byte[] result = null;
        try {     bos = new ByteArrayOutputStream();
                  dos = new DataOutputStream(bos);
                  dos.writeInt(10);      dos.writeFloat(20.0f);
                  dos.writeBoolean(true);
                  result = bos.toByteArray();
                  String[] hex = new String[result.length];
                  for(int i=0;i<result.length; i++) {
                      if(result[i] < 0) {
                            hex[i] = Integer.toHexString(result[i]+256);
                      } else {   hex[i] = Integer.toHexString(result[i]);  }
                      // hex[i] = "0"+hex[i];
                      // hex[i] = hex[i].substring(hex[i].length()-2);
                  }
                  System.out.println("10진수  :" + Arrays.toString(result));
                  System.out.println("16진수  :" + Arrays.toString(hex));
                  dos.close();
        } catch (IOException e) {  e.printStackTrace();     }
    } // main
}
```

# PrintStream

- 데이터를 다양한 형식의 문자로 출력하는 기능을 제공하는 보조스트림
- System.out과 System.err이 PrintStream이다.
- PrintStream보다 PrintWriter를 사용할 것을 권장한다.

| 생성자 / 메서드 | | 설 명 |
|---|---|---|
| PrintStream(File file)<br>PrintStream(File file, String csn)<br>PrintStream(OutputStream out)<br>PrintStream(OutputStream out,boolean autoFlush)<br>PrintStream(OutputStream out,boolean autoFlush, String encoding)<br>PrintStream(String fileName)<br>PrintStream(String fileName, String csn) | | 지정된 출력스트림을 기반으로 하는 PrintStream인스턴스를 생성한다. autoFlush의 값을 true로 하면 println메서드가 호출되거나 개행문자가 출력될 때 자동으로 flush된다. 기본값은 false이다. |
| boolean checkError() | | 스트림을 flush하고 에러가 발생했는지를 알려 준다. |
| void print(boolean b)<br>void print(char c)<br>void print(char[] c)<br>void print(double d)<br>void print(float f)<br>void print(int i)<br>void print(long l)<br>void print(Object o)<br>void print(String s) | void println(boolean b)<br>void println(char c)<br>void println(char[] c)<br>void println(double d)<br>void println(float f)<br>void println(int l)<br>void println(long l)<br>void println(Object o)<br>void println(String s) | 인자로 주어진 값을 출력소스에 문자로 출력한다.<br>println메서드는 출력 후 줄바꿈을 하고, print메서드는 줄을 바꾸지 않는다. |
| void println() | | 줄바꿈 문자(line separator)를 출력함으로써 줄을 바꾼다. |
| PrintStream printf(String format, Object... args) | | 정형화된(formatted) 출력을 가능하게 한다. |
| protected void setError() | | 작업 중에 오류가 발생했음을 알린다.(setError()를 호출한 후에, checkError()를 호출하면 true를 반환한다.) |

# PrintStream

| format | 설 명 | 결 과(int i=65) |
|---|---|---|
| %d | 10진수(decimal integer) | 65 |
| %o | 8진수(octal integer) | 101 |
| %x | 16진수(hexadecimal integer) | 41 |
| %c | 문자 | A |
| %s | 문자열 | 65 |
| %5d | 5자리 숫자. 빈자리는 공백으로 채운다. | 65 |
| %-5d | 5자리 숫자. 빈자리는 공백으로 채운다.(왼쪽 정렬) | 65 |
| %05d | 5자리 숫자. 빈자리는 0으로 채운다. | 00065 |

| format | 설 명 | 결 과 |
|---|---|---|
| %s | 문자열(string) | ABC |
| %5s | 5자리 문자열. 빈자리는 공백으로 채운다. | ABC |
| %-5s | 5자리 문자열. 빈자리는 공백으로 채운다.(왼쪽 정렬) | ABC |

| format | 설 명 | 결과 |
|---|---|---|
| %e | 지수형태표현(exponent) | 1.234568e+03 |
| %f | 10진수(decimal float) | 1234.56789 |
| %3.1f | 출력될 자리수를 최소 3자리(소수점포함), 소수점 이하 1자리(2번째 자리에서 반올림) | 1234.6 |
| %8.1f | 소수점이상 최소 6자리, 소수점 이하 1자리.<br>출력될 자리수를 최소 8자리(소수점포함)를 확보한다.<br>빈자리는 공백으로 채워진다.(오른쪽 정렬) | 1234.6 |
| %08.1f | 소수점이상 최소 6자리, 소수점 이하 1자리.<br>출력될 자리수를 최소 8자리(소수점포함)를 확보한다.<br>빈자리는 0으로 채워진다. | 001234.6 |
| %-8.1f | 소수점이상 최소 6자리, 소수점 이하 1자리.<br>출력될 자리수를 최소 8자리(소수점포함)를 확보한다.<br>빈자리는 공백으로 채워진다.(왼쪽 정렬)<br>1234.6 | 1234.6 |

| format | 설 명 |
|---|---|
| \t | 탭(tab) |
| \n | 줄바꿈 문자(new line) |
| %% | % |

| format | 설 명 | 결 과 |
|---|---|---|
| %tR<br>%tH:%tM | 시분(24시간) | 21:05<br>21:05 |
| %tT<br>%tH:%tM:%tS | 시분초(24시간) | 21:05:33<br>21:05:33 |
| %tD<br>%tm/%td/%ty | 연월일 | 02/16/07<br>02/16/07 |
| %tF<br>%tY-%tm-%td | 연월일 | 2007-02-16<br>2007-02-16 |

# PrintStream

```
import java.util.Date;
class PrintStreamEx1 {
    public static void main(String[] args) {
        int i = 65;
        float f = 1234.56789f;
        Date d = new Date();

        System.out.printf("문자 %c의 코드는 %d\n", i, i);
        System.out.printf("%d는 8진수로 %o, 16진수로 %x\n", i ,i, i);
        System.out.printf("%3d%3d%3d\n", 100, 90, 80);
        System.out.println();
        System.out.printf("12345678901234567890123456789O\n");
        System.out.printf("%s%-5s%5s\n", "123", "123", "123");
        System.out.println();
        System.out.printf("%-8.1f%8.1f %e\n",f,f,f);
        System.out.println();
        System.out.printf("오늘은 %tY년 %tm월 %td일 입니다.\n", d,d,d,d );
        System.out.printf("지금은 %tH시 %tM분 %tS초 입니다.\n", d,d,d,d );
        System.out.printf("지금은 %1$tH시 %1$tM분 %1$tS초 입니다.\n", d );
    }
}
```

# StringReader와 StringWriter

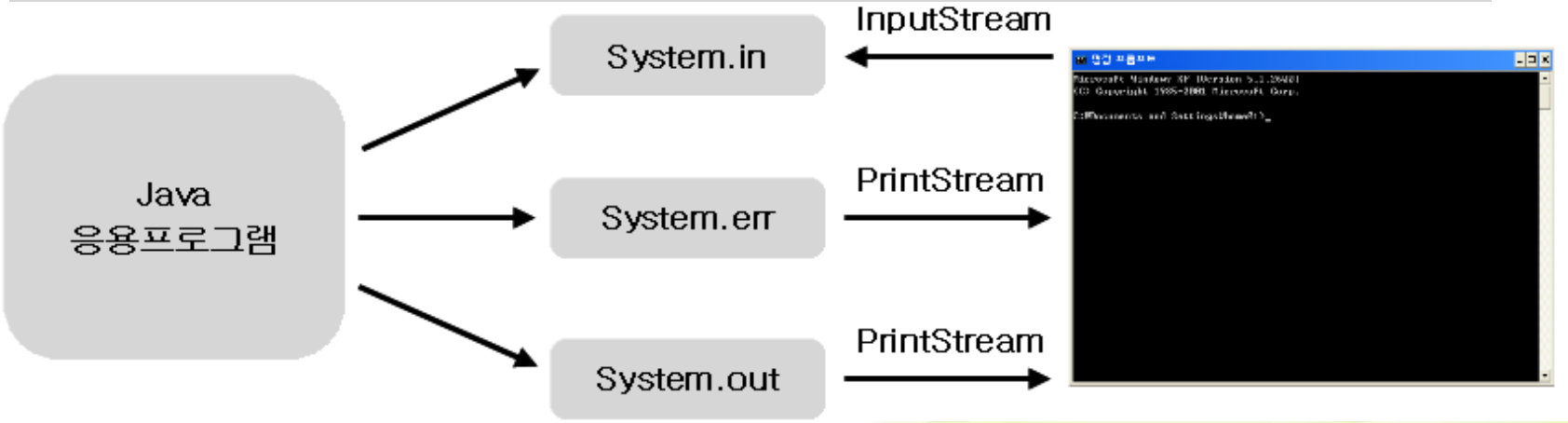
- CharArrayReader, CharArrayWriter처럼 메모리의 입출력에 사용한다.
- StringWriter에 출력되는 데이터는 내부의 StringBuffer에 저장된다.

```
import java.io.*;
class StringReaderWriterEx {
    public static void main(String[] args)          {
        String inputData = "ABCD";
        StringReader input = new StringReader(inputData);
        StringWriter output = new StringWriter();

        int data = 0;

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

        System.out.println("Input Data  :" + inputData);
        System.out.println("Output Data :" + output.toString());
        // System.out.println("Output Data :" +output.getBuffer().toString());
    }
}
```

# 표준입출력 – System.in, System.out, System.err

- 콘솔(console, 화면)을 통한 데이터의 입출력을 '표준 입출력'이라 한다.
- JVM이 시작되면서 자동적으로 생성되는 스트림이다.

**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();
   ...
}
```

# 콘솔 입출력

## ❖ Scanner 클래스

- Console 클래스의 단점
  - 문자열은 읽을 수 있지만 기본 타입(정수, 실수) 값을 바로 읽을 수 없음

- java.util.Scanner
  - 콘솔로부터 기본 타입의 값을 바로 읽을 수 있음

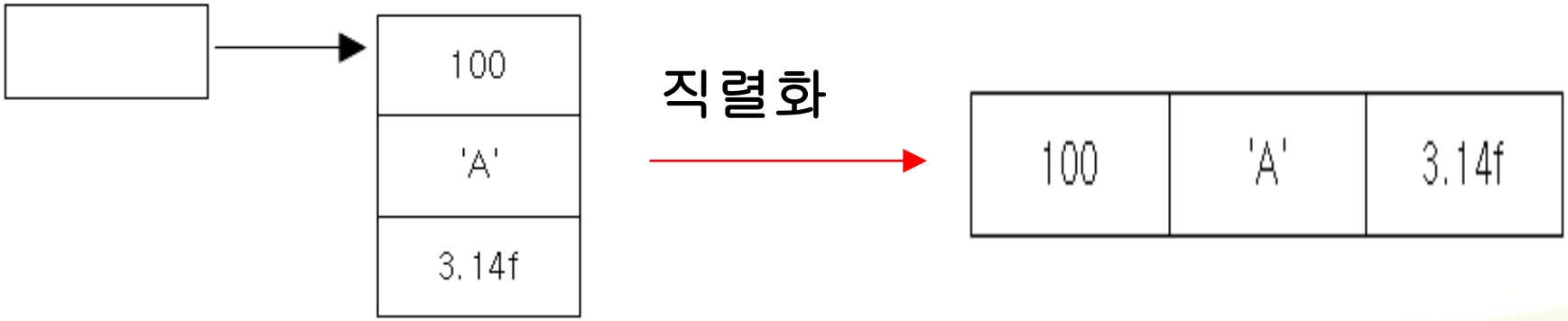
```
Scanner scanner = new Scanner(System.in)
```

  - 제공하는 메소드

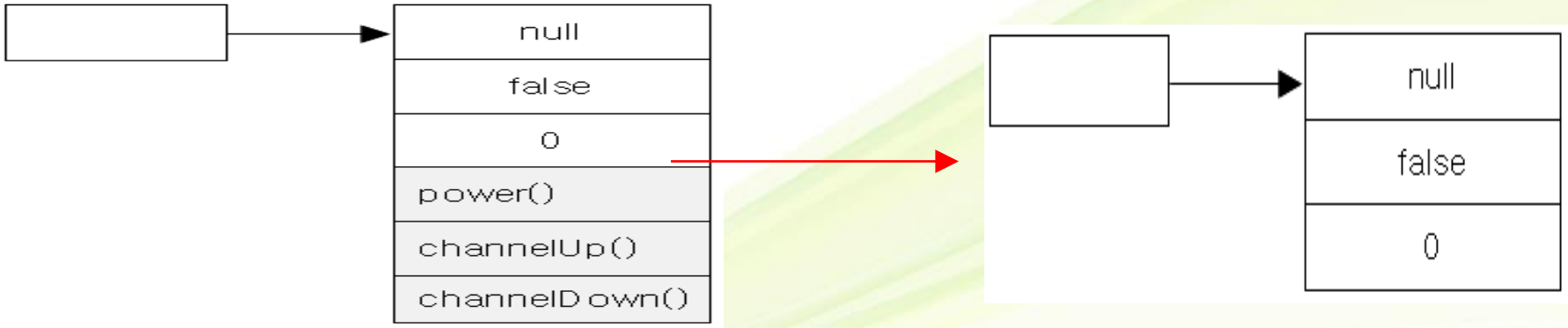
| 리턴타입 | 메소드 | 설명 |
|---|---|---|
| boolean | nextBoolean() | boolean(true/false) 값을 읽는다. |
| byte | nextByte() | byte 값을 읽는다. |
| short | nextShort() | short 값을 읽는다. |
| int | nextInt() | int 값을 읽는다. |
| long | nextLong() | long 값을 읽는다. |
| float | nextFloat() | float 값을 읽는다. |
| double | nextDouble()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

# 직렬화(serialization)란?

- 객체를 '연속적인 데이터'로 변환하는 것. 반대과정은 '역직렬화'라고 한다.
- 객체의 인스턴스변수들의 값을 일렬로 나열하는 것



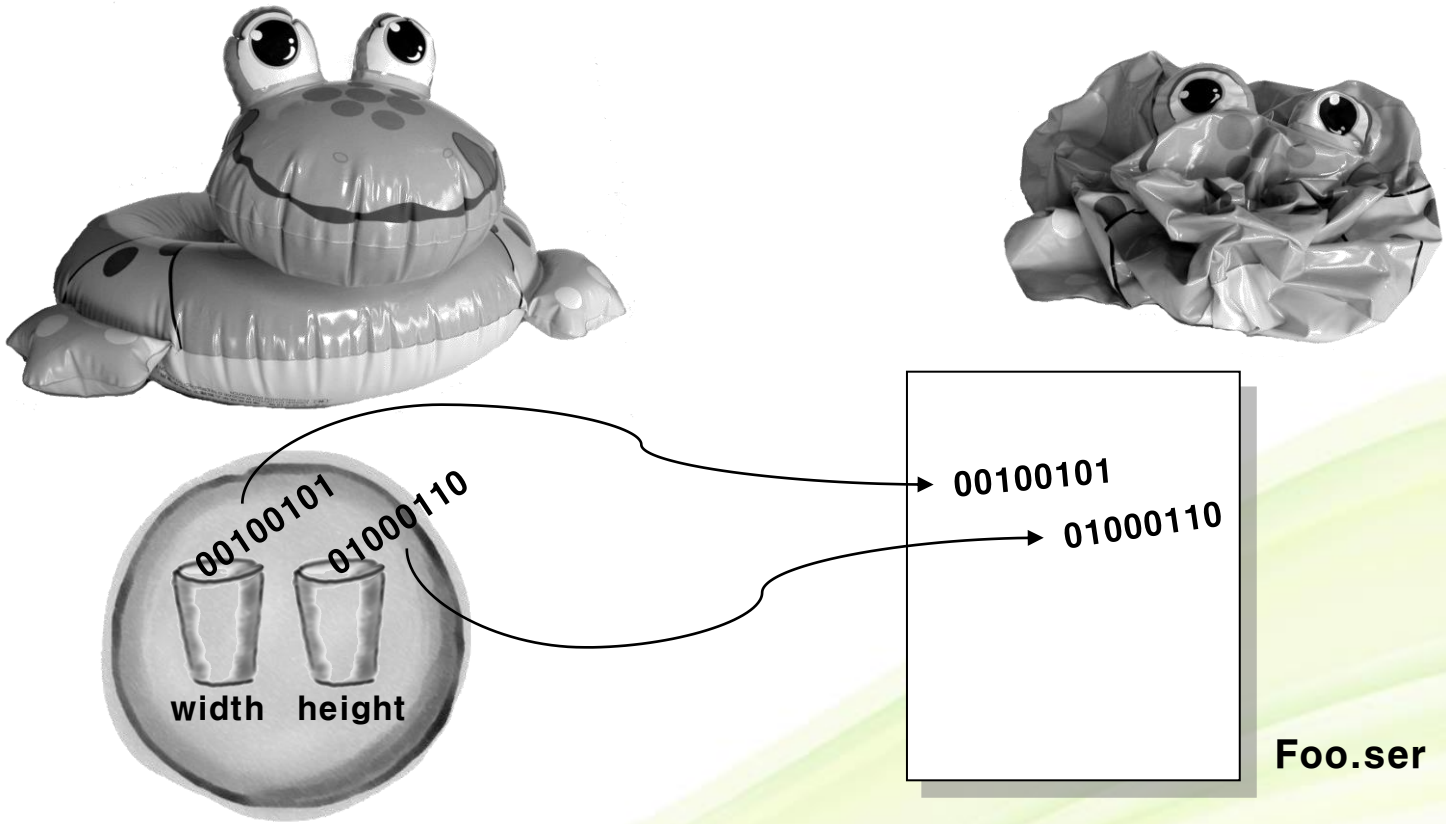

- 객체를 저장하기 위해서는 객체를 직렬화해야 한다.
- 객체를 저장한다는 것은 객체의 모든 인스턴스변수의 값을 저장하는 것



# 직렬화된 객체

힙 안에 들어있는 객체

직렬화된 객체




```
Foo myFoo = new Foo();
myFoo.setWidth(37);
myFoo.setHeight(70);
```

```
FileOutputStream fs = new FileOutputStream("Foo.ser");
ObjectOutputStream os = new ObjectOutputStream(fs);
os.writeObject(myFoo);
```

# 역직렬화(deserialization)

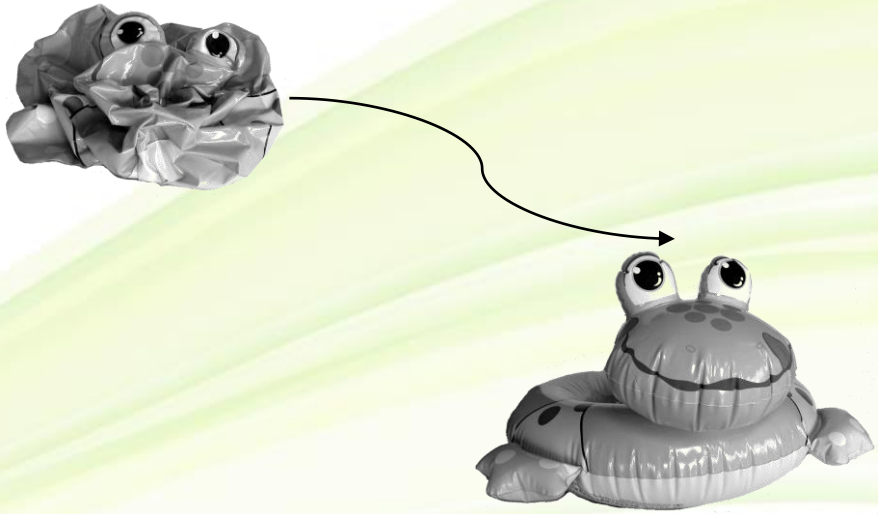
**직렬화된 객체를 원래 상태로 돌려놓는 것**

1. **FileInputStream 만들기**

   FileInputStream fileStream = new FileInputStream(“MyGame.ser”);

2. **ObjectInputStream 만들기**

   ObjectInputStream os = new ObjectInputStream(fileStream);



# 역직렬화(deserialization)

## 3. 객체 읽기

```
Object one = os.readObject();
Object two = os.readObject();
Object three = os.readObject();
```

## 4. 객체 캐스팅

```
GameCharacter elf = (GameCharacter) one;
GameCharacter troll = (GameCharacter) two;
GameCharacter magician = (GameCharacter) three;
```

## 5. ObjectInputStream 닫기

```
os.close();
```

# ObjectInputStream, ObjectOutputStream

- 객체를 직렬화하여 입출력할 수 있게 해주는 보조스트림

```
ObjectInputStream(InputStream in)
ObjectOutputStream(OutputStream out)
```

- 객체를 파일에 저장하는 방법

```
FileOutputStream fos = new FileOutputStream("objectfile.ser");
ObjectOutputStream out = new ObjectOutputStream(fos);

out.writeObject(new UserInfo());
```

- 파일에 저장된 객체를 다시 읽어오는 방법

```
FileInputStream fis = new FileInputStream("objectfile.ser");
ObjectInputStream in= new ObjectInputStream(fis);

UserInfo info = (UserInfo)in.readObject();
```

# ObjectInputStream, ObjectOutputStream

| ObjectOutputStream |
| --- |
| void defaultWriteObject() |
| void write(byte[] buf) |
| void write(byte[] buf, int off, int len) |
| void write(int val) |
| void writeBoolean(boolean val) |
| void writeByte(int val) |
| void writeBytes(String str) |
| void writeChar(int val) |
| void writeChars(String str) |
| void writeDouble(double val) |
| void writeFloat(float val) |
| void writeInt(int val) |
| void writeLong(long val) |
| void writeObject(Object obj) |
| void writeShort(int val) |
| void writeUTF(String str) |

| ObjectInputStream |
| --- |
| void defaultReadObject() |
| int read() |
| int read(byte[] buf, int off, int len) |
| boolean readBoolean() |
| byte readByte() |
| char readChar() |
| double readDouble() |
| float readFloat() |
| int readInt() |
| long readLong() |
| short readShort() |
| Object readObject() |
| String readUTF |

# 객체를 직열화하는 프로그램

```
import java.io.*;
import java.util.GregorianCalendar;
class ObjectOutputExample1 {
    public static void main(String args[]) {
        ObjectOutputStream out = null;
        try {
            out = new ObjectOutputStream(
                    new FileOutputStream("output.dat"));
            out.writeObject(new GregorianCalendar(2006, 0, 14));
            out.writeObject(new GregorianCalendar(2006, 0, 15));
            out.writeObject(new GregorianCalendar(2006, 0, 16));
        }
        catch (IOException ioe) {
            System.out.println("파일로 출력할 수 없습니다.");
        }
        finally {
            try {   out.close();
            }       catch (Exception e) {          }
        }
    }
}
```

# 객체를 역직렬화하는 프로그램

```
import java.io.*;   import java.util.GregorianCalendar;
import java.util.Calendar;
class ObjectInputExample1 {
    public static void main(String args[]) {
        ObjectInputStream in = null;
        try { in = new ObjectInputStream(new FileInputStream("output.dat"));
            while (true) {
                GregorianCalendar calendar = (GregorianCalendar) in.readObject();
                int year = calendar.get(Calendar.YEAR);
                int month = calendar.get(Calendar.MONTH) + 1;
                int date = calendar.get(Calendar.DATE);
                System.out.println(year + "/" + month + "/" + date);
            }
        } catch (FileNotFoundException fnfe) {
              System.out.println("파일이 존재하지 않습니다.");
        } catch (EOFException eofe) { System.out.println("끝");
        } catch (IOException ioe) { System.out.println("파일을 읽을 수 없습니다.");
        } catch (ClassNotFoundException cnfe) {
            System.out.println("해당 클래스가 존재하지 않습니다.");
        } finally {  try { in.close();  } catch (Exception e) { }
        }
    }
}
```

# 직렬화 가능한 클래스 만들기

- java.io.Serializable을 구현해야만 직렬화가 가능하다.

```
public class UserInfo {
    String name;
    String password;
    int age;
}
```

```
public class UserInfo
  implements java.io.Serializable {
    String name;
    String password;
    int age;
}
```

```
public interface Serializable { }
```

- 제어자 transient가 붙은 인스턴스변수는 직렬화 대상에서 제외된다.

```
public class UserInfo implements Serializable {
    String name;
    transient String password;  // 직렬화 대상에서 제외된다.
    int age;
}
```

- Serializable을 구현하지 않은 클래스의 인스턴스도 직렬화 대상에서 제외

```
public class UserInfo implements Serializable {
    String name;
    transient String password;
    int age;

    Object obj = new Object();  // Object객체는 직렬화할 수 없다.
}
```

# 직렬화 가능한 클래스 만들기

- Serializable을 구현하지 않은 조상의 멤버들은 직렬화 대상에서 제외된다.

```
public class SuperUserInfo {
    String name;     // 직렬화되지 않는다.
    String password; // 직렬화되지 않는다.
}

public class UserInfo extends SuperUserInfo implements Serializable {
    int age;
}
```

- readObject()와 writeObject()를 오버라이딩하면 직렬화를 마음대로...

```
private void writeObject(ObjectOutputStream out)
        throws IOException {
        out.writeUTF(name);
        out.writeUTF(password);
        out.defaultWriteObj
}
```

```
private void readObject(ObjectInputStream in)
        throws IOException, ClassNotFoundException {
        name = in.readUTF();
        password = in.readUTF();
        in.defaultReadObject();
}
```

# 직렬화 가능한 클래스 만들기

```
import java.io.Serializable;
public class GoodStock implements Serializable {
        // transient 숫자는 0, 문자는 null
        String code;
        int num;
        public GoodStock(String code, int num) {
                this.code = code;
                this.num  = num;
        }
        public String toString() {
                return "주식[코드:"+code+", 수량:"+num+"]";
        }
}
```

# 직렬화 가능한 클래스 만들기

```
import java.io.*;
public class ObjectWrite2 {
        public static void main(String[] args) throws
FileNotFoundException, IOException {
                ObjectOutputStream oos = new
ObjectOutputStream(
                        new FileOutputStream("stock.txt"));
                oos.writeObject(new GoodStock("111",100));
                oos.writeObject(new GoodStock("222",150));
                oos.writeObject(new GoodStock("333",230));
                oos.close();
                System.out.println("저장완료");
        }
}
```

# 직렬화 가능한 클래스 만들기

```
import java.io.*;
public class ObjectRead2 {
	public static void main(String[] args) throws IOException {
		ObjectInputStream ois = null;
		try {
			ois = new ObjectInputStream(
				new FileInputStream("stock.txt"));
			while(true) {
				GoodStock gs = (GoodStock)ois.readObject();
				System.out.println(gs);
			}
		} catch (Exception e) {
			System.out.println("다 처리했다");
		}
		ois.close();
	}
}
```

# 직렬화 가능한 클래스 만들기

```
import java.io.Serializable;
public class Product {
        String name;      int price;
        Product() {}
        public Product(String name, int price) {
                this.name = name; this.price = price;
        }
        public String toString() {
                return "제품[이름:"+name+", 가격:"+price+"]";
        }
}
class Book extends Product implements Serializable {
        String writer;      int page;
        public Book(String name, int price, String writer, int page) {
                super(name, price);
                this.writer = writer; this.page = page;
        }
        public String toString() {
                return "책[이름:"+name+",가격:"+price+",작가:"+writer+
                        ",쪽수:"+page+"]";
        }
}
```

# 직렬화 가능한 클래스 만들기

```
import java.io.*;
public class BookWrite3 {
    public static void main(String[] args) throws FileNotFoundException,
                    IOException {
        ObjectOutputStream oos = new ObjectOutputStream(
                    new FileOutputStream("book"));
        oos.writeObject(new Book("대지",2000,"펄벅", 500));
        oos.writeObject(new Book("하수와고수",15000,"강병호",200));
        oos.writeObject(new Book("선생님이  때렸어요",20000,"강병철",300));
        oos.close();
        System.out.println("저장완료 ㅋㅋ");
    }
}
```

# 직렬화 가능한 클래스 만들기

```
import java.io.*;
public class BookRead3 {
   public static void main(String[] args) throws IOException {
          ObjectInputStream ois = null;
          try {
                    ois=new ObjectInputStream(new FileInputStream("book"));
                    while(true) {
                              Book b = (Book)ois.readObject();
                              System.out.println(b);
                    }
          } catch (Exception e) {
                    if (e != null)      System.out.println(e.getMessage());
                    System.out.println("처리 완료");
          }
          ois.close();
    }
}
```

# 직렬화 가능한 클래스 만들기

```
import java.io.Serializable;
public class Book2 extends Product implements Serializable {
        String writer;     int page;
        public Book2(String name,int price,String writer,int page) {
                super(name, price);
                this.writer = writer; this.page = page;
        }
        private void writeObject(ObjectOutputStream oos) throws
IOException {
                oos.writeUTF(name);   oos.writeInt(price);
                // oos.writeUTF(writer); oos.writeInt(page);
                oos.defaultWriteObject();
        }
        private void readObject(ObjectInputStream ois) throws IOException,
ClassNotFoundException {
                name =ois.readUTF();  price=ois.readInt();
                // writer=ois.readUTF(); page=ois.readInt();
                ois.defaultReadObject();
        }
        public String toString() {
                return "책[이름:"+name+",가격:"+price+",작가:"+writer+
                        ",쪽수:"+page+"]";
        }
}
```

# 직렬화 가능한 클래스 만들기

```
package ch16;
import java.io.*;
public class BookWrite3 {
   public static void main(String[] args) throws
FileNotFoundException, IOException {
        ObjectOutputStream oos = new ObjectOutputStream(
                new FileOutputStream("book"));
        oos.writeObject(new Book2("대지",2000,"펄벅", 500));
        oos.writeObject(new Book2("하수와고수",15000,"강병호",200));
        oos.writeObject(new Book2("선생님이 ",20000,"강병철",300));
        oos.close();
        System.out.println("저장완료 ㅋㅋ");
  }
}
```

# 직렬화 가능한 클래스 만들기

```
import java.io.*;
public class BookRead3 {
    public static void main(String[] args) throws IOException {
        ObjectInputStream ois = null;
        try {
                ois=new ObjectInputStream(
                        new FileInputStream("book"));
                while(true) {
                        Book2 b = (Book2)ois.readObject();
                        System.out.println(b);
                }
        } catch (Exception e) {
                if (e != null)    System.out.println(e.getMessage());
                System.out.println("처리 완료");
        }
        ois.close();
    }
}
```

# NIO 기반 입출력

# NIO 소개

## ❖ NIO(New Input/Output)

- 기존 java.io API와 다른 새로운 입출력 API
- 자바4부터 추가 → 자바7부터 네트워크 지원 강화된 NIO.2 API 추가
- 관련 패키지

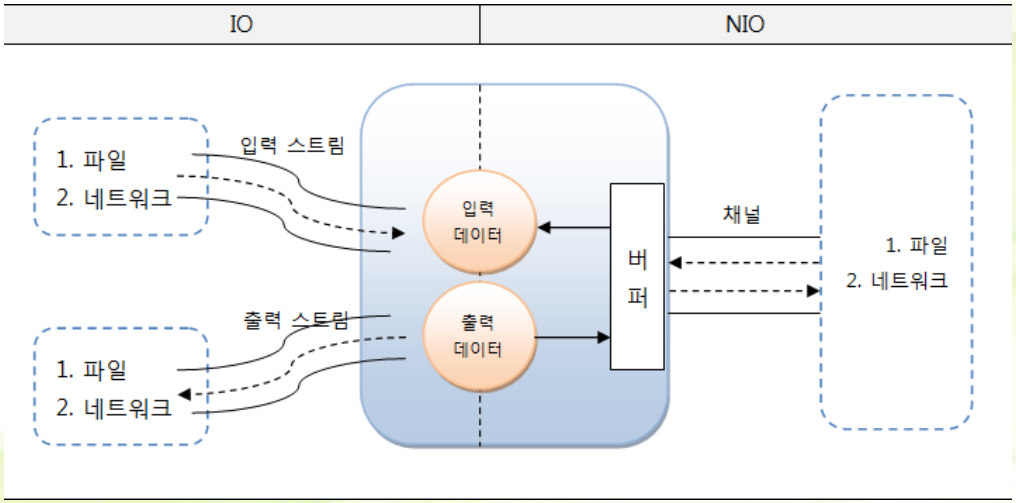
| NIO 패키지 | 포함되어 있는 내용 |
|---|---|
| java.nio | 다양한 버퍼 클래스 |
| java.nio.channels | 파일 채널, TCP 채널, UDP 채널등의 클래스 |
| java.nio.channels.spi | java.nio.channels 패키지를 위한 서비스 제공자 클래스 |
| java.nio.charset | 문자셋, 인코더, 디코더 API |
| java.nio.charset.spi | java.nio.charset 패키지를 위한 서비스 제공자 클래스 |
| java.nio.file | 파일 및 파일 시스템에 접근하기 위한 클래스 |
| java.nio.file.attribute | 파일 및 파일 시스템의 속성에 접근하기 위한 클래스 |
| java.nio.file.spi | java.nio.file 패키지를 위한 서비스 제공자 클래스 |

# NIO 소개

## ❖ IO와 NIO의 차이점

| 구분 | IO | NIO |
|---|---|---|
| 입출력 방식 | 스트림 방식 | 채널 방식 |
| 버퍼 방식 | 넌버퍼(non-buffer) | 버퍼(buffer) |
| 동기 / 비동기 방식 | 동기 방식 | 동기 / 비동기 방식 모두 지원 |
| 블로킹 / 넌블로킹 방식 | 블로킹 방식 | 블로킹 / 넌블로킹 방식 모두 지원 |

- **스트림 vs. 채널**
  - **IO 스트림: 입력 스트림과 출력 스트림으로 구분되어 별도 생성**
  - **NIO 채널: 양방향으로 입출력이 가능하므로 하나만 생성**



# NIO 소개

## ❖ IO와 NIO의 차이점

### ▪ 넌버퍼 vs. 버퍼

- IO 스트림 – 넌버퍼(non-buffer)
  - IO에서는 1바이트씩 읽고 출력 – 느림
  - 보조 스트림인 BufferedInputStream, BufferedOutputStream를 사용해 버퍼 제공 가능
  - 스트림으로부터 입력된 전체 데이터를 별도로 저장해야
    - 저장 후 입력 데이터의 위치 이동해가면서 자유롭게 이용 가능

- NIO 채널 – 버퍼(buffer)
  - 기본적으로 버퍼 사용해 입출력 – 성능 좋음
  - 읽은 데이터를 무조건 버퍼(Butter: 메모리 저장소)에 저장
    - 버퍼 내에서 데이터 위치 이동해 가며 필요한 부분만 읽고 쓸 수 있음

# NIO 소개

## ❖ IO와 NIO의 차이점

### ▪ 블로킹 vs 넌블로킹

#### • IO 스트림 – 블로킹

- 입력 스트림의 read() 메소드 호출
  - 데이터 입력 전까지 스레드는 블로킹(대기상태)
- 출력 스트림의 write() 메소드 호출
  - 데이터 출력 전까지 스레드는 블로킹
- 스레드 블로킹 –다른 일을 할 수가 없고 interrupt 해 블로킹 빠져나올 수도 없음
- 블로킹을 빠져 나오는 유일한 방법 – 스트림을 닫는 것

#### • NIO 채널 – 블로킹, 넌블로킹

- NIO 블로킹은 스레드를 interrupt 함으로써 빠져나올 수 있음
- NIO는 넌블로킹 지원
  - 입출력 작업 시 스레드가 블로킹되지 않음
- 준비 완료된 채널을 선택하는 기능!!!

# NIO 소개

- 네트워크 프로그램 개발 시 IO와 NIO의 선택
  - IO 방식 선택하는 경우
    - 연결 클라이언트의 수가 적고,
    - 전송되는 데이터가 대용량이면서
    - 순차적으로 처리될 필요성 있을 경우
  - NIO 방식 선택하는 경우
    - 연결 클라이언트의 수가 많고
    - 전송되는 데이터 용량이 적으면서,
    - 입출력 작업 처리가 빨리 끝나는 경우

# 파일과 디렉토리

## ❖ 파일 관련 패키지

- IO는 파일의 속성 정보 읽기 위해 File 클래스만 제공
- NIO는 좀 더 다양한 파일의 속성 정보 제공
  - 클래스와 인터페이스를 java.nio.file, java.nio.file.attribute 패키지에서 제공

## ❖ 경로 정의(Path)

- java.nio.file.Path 인터페이스
  - IO의 java.io.File 클래스에 대응
  - NIO의 여러 곳에서 파일 경로 지정 위해 Path 사용
- Path 구현 객체
  - java.nio.file.Paths 클래스의 정적 메소드인 get() 메소드로 얻음

# PathExample.java

```
package ch21;
import java.nio.file.*;
import java.util.Iterator;
public class PathExample {
   public static void main(String[] args) throws Exception {
        Path path = Paths.get("src/ch21/PathExample.java");
        System.out.println("[파일명] " + path.getFileName());
        System.out.println("[부모 디렉토리명]: " +
path.getParent().getFileName());
        System.out.println("중첩 경로수: " + path.getNameCount());
        System.out.println();
        for(int i=0; i<path.getNameCount(); i++) {
              System.out.println(path.getName(i));
         }
         System.out.println();
         Iterator<Path> iterator = path.iterator();
         while(iterator.hasNext()) {
               Path temp = iterator.next();
               System.out.println(temp.getFileName());
          }
    }
}
```

# 파일과 디렉토리

## ❖ 파일 시스템 정보(FileSystem)

- 운영체제의 파일 시스템은 FileSystem인터페이스 통해 접근
- FileSystem 구현 객체는 FileSystems의 정적 메소드인 getDefault()로 얻을 수 있음

```
FileSystem fileSystem = FileSystems.getDefault();
```

- FileSystem에서 제공하는 메소드

| 리턴타입 | 메소드(매개변수) | 설명 |
|---|---|---|
| Iterable<FileStore> | getFileStores() | 드라이버 정보를 가진 FileStore 객체들을 리턴 |
| Iterable<Path> | getRootDirectories() | 루트 디렉토리 정보를 가진 Path 객체들을 리턴 |
| String | getSeparator() | 디렉토리 구분자 리턴 |

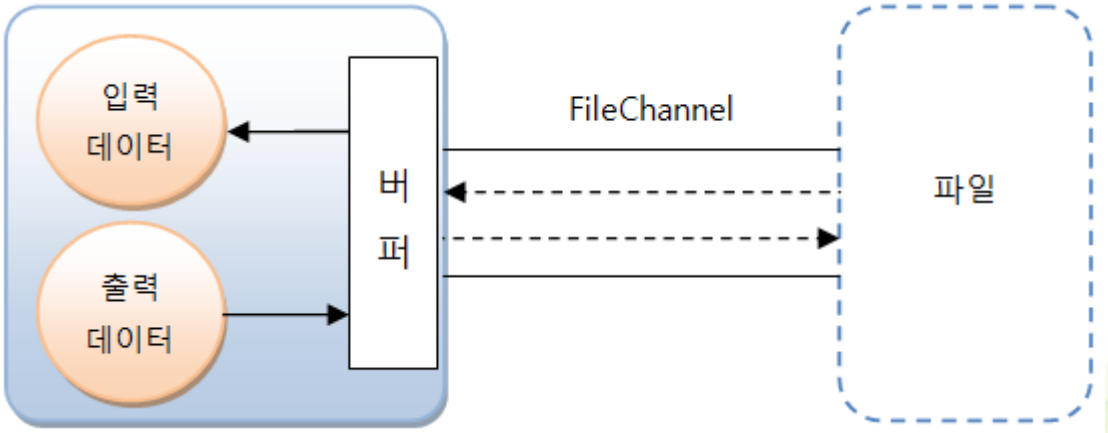
# 파일 시스템 정보 출력

```
import java.nio.file.*;
public class FileSystemExample {
    public static void main(String[] args) throws Exception {
        FileSystem fileSystem = FileSystems.getDefault();
        for(FileStore store : fileSystem.getFileStores()) {
            System.out.println("드라이버명: " + store.name());
            System.out.println("파일시스템: " + store.type());
            System.out.println("전체 공간: ₩t₩t" +
                    store.getTotalSpace() +" 바이트");
            System.out.println("사용 중인 공간: ₩t" + (store.getTotalSpace()
                    - store.getUnallocatedSpace()) + " 바이트");
            System.out.println("사용 가능한 공간: ₩t" +
                    store.getUsableSpace() + " 바이트");
            System.out.println();
        }
        System.out.println("파일 구분자: " + fileSystem.getSeparator());
        System.out.println();
        for(Path path : fileSystem.getRootDirectories()) {
            System.out.println(path.toString());
        }
    }
}
```

# 파일 채널

## ❖ 파일 채널(FileChannel)

- 파일 읽기와 쓰기 가능하게 해주는 역할
- 동기화 처리가 되어 있기 때문에 멀티 스레드 환경에서 사용해도 안전



# 파일 채널

## ❖ FileChannel 생성과 닫기

- FileInputStream, FileOutputStream의 getChannel() 메소드 호출
- FileChannel.open()

```
FileChannel fileChannel = FileChannel.open(Path path, OpenOption... options);
```

  - 첫 번째 path 매개값 – 열거나, 생성하고자 하는 파일 경로
  - 두 번째 options 매개값 – 열기 옵션 값
    - StandardOpenOption 의 다음 열거 상수

| 열거 상수 | 설명 |
|---|---|
| READ | 읽기용으로 파일을 연다. |
| WRITE | 쓰기용으로 파일을 연다. |
| CREATE | 파일이 없다면 새 파일을 생성한다. |
| CREATE_NEW | 새 파일을 만든다. 파일이 이미 있으면 예외와 함께 실패한다. |
| APPEND | 파일 끝에 데이터를 추가한다(WRITE 나 CREATE 와 함께 사용됨) |
| DELETE_ON_CLOSE | 스트림을 닫을 때 파일을 삭제한다(임시파일을 삭제할 때 사용) |
| TRUNCATE_EXISTING | 파일을 0 바이트로 잘라낸다(WRITE 옵션과 함께 사용됨) |

- 채널 닫기: FileChannel을 더 이상 이용하지 않을 경우

```
fileChannel.close();
```

# 파일 채널

## ❖ 파일 쓰기와 읽기

### ▪ 파일 쓰기

```
int bytesCount = fileChannel.write(ByteBuffer src);
```

- 파일에 쓰여지는 바이트는 ByteBuffer의 position 부터 limit 까지

### ▪ 파일 읽기

```
int bytesCount = fileChannel.read(ByteBuffer dst);
```

- 파일에서 읽혀지는 바이트는 ByteBuffer의 position부터 저장
  - 버퍼에 한 바이트를 저장할 때마다 position이 1씩 증가
  - 버퍼에 저장한 마지막 바이트의 위치는 position-1 인덱스까지
  - 한 번 읽을 수 있는 최대 바이트 수는 position부터 ByteBuffer의 capacity까지
- 리턴값은 파일에서 ByteBuffer로 읽혀진 바이트 수
  - 0~(position-1)까지의 바이트 수
  - 더 이상 읽을 바이트가 없다면 read() 메소드는 -1 리턴

# 파일 쓰기와 읽기

```
public class FileChannelWriteExample {
        public static void main(String[] args) throws IOException {
                Path path = Paths.get("C:/Temp/file.txt");
                FileChannel fileChannel = FileChannel.open(
                        path, StandardOpenOption.CREATE,
                         StandardOpenOption.WRITE);
                String data = "안녕하세요";
                Charset charset = Charset.defaultCharset();
                ByteBuffer byteBuffer = charset.encode(data);
                int byteCount = fileChannel.write(byteBuffer);
                System.out.println("file.txt : " + byteCount
                  + " bytes written");

                fileChannel.close();
        }
}
```

# 파일 쓰기와 읽기

```
public class FileChannelReadExample {
        public static void main(String[] args) throws IOException {
                Path path = Paths.get("file.txt");
                FileChannel fileChannel = FileChannel.open(
                        path, StandardOpenOption.READ);
                ByteBuffer byteBuffer = ByteBuffer.allocate(100);
                Charset charset = Charset.defaultCharset();
                String data = "";
                int byteCount;
                while(true) {
                        byteCount = fileChannel.read(byteBuffer);
                        if(byteCount == -1) break;
                        byteBuffer.flip();
                        data += charset.decode(byteBuffer).toString();
                        byteBuffer.clear();
                }
                fileChannel.close();
                System.out.println("file.txt : " + data);
        }
}
```