13. 입/출력 프로그래밍

일반적으로 입출력은 파일로부터 데이터를 읽어오거나 파일에 데이터를 쓰는 작업을 말합니다. 자바에서 입출력은 파일 입출력은 물론이고 인터넷에 있는 URL도 파일처럼 입출력할 수 있는 강력한 API를 제공합니다. 그러나 기능이 뛰어난 만큼 다루기 어려운 점도 있습니다. 따라서 많은 사람들이 입출력 관련 API를 사용할 때 어려움을 겪는 것이 사실입니다. 입출력을 이용한 프로그램을 개발할 때, 무엇보다도 API문서를 잘 살펴보는 것이 매우 중요합니다.

13장의 주요 내용입니다.

- 노드스트림 클래스와 필터스트림 클래스
- 바이트스트림 클래스와 문자스트림 클래스
- 파일(File)과 URL
- RandomAccessFile
- Serialization

13.1. 스트림(Streams)

입출력을 이해하기 전에 스트림이라는 용어부터 알아야 합니다. 스트림은 source에서, sink로의 데이터 흐름을 말합니다. source는 데이터 흐름의 출발점이며, sink는 데이터 흐름이 끝나는 지점을 의미합니다. source로부터 나오는 데이터 흐름 입력 스트림이라고 하며, sink로의 데이터 흐름을 출력 스트림이라고 합니다. 예를 들어 c:₩temp₩in.txt 파일을 읽는 프로그램을 개발하려면 in.txt 파일이 입력 스트림이 되고, 반대로 out.txt 파일에 무언가를 쓰려고 하면 그 파일이 출력 스트림이 되는 것 입니다. source와 sink를 합해서 노드라고 하는데, 노드의 종류로는 디스크상의 파일, 메모리, 또는 스레드나 프로세스 사이에서 채널역할을 하는 파이프 등이 있습니다. 디스크 파일이나 메모리 영역과 같은 노드에서 읽거나 쓰는 스트림을 노드 스트림(Node Stream)이라 합니다. 일단 스트림에서 데이터를 읽으려면 노드 스트림이 필요한데, 노드 스트림을 통해서만 노드로부터 데이터를 읽거나 쓸 수 있기 때문입니다.

자바는 입출력을 수행할 때 문자(char) 단위 스트림과 바이트(byte) 단위 스트림 형식을 지원합니다.

다음 표와 같이 문자자료의 입출력은 Reader와 Writer에 의해 이루어지고, 바이트자료의 입출력은 InputStream과 OutputStream에 의해 이루어집니다.

	Byte Streams	Character Streams
Source Streams	InputStream	Reader
Sink Streams	OutputStream	Writer

13.1.1. 노드스트림

앞에서도 잠깐 언급한바와 같이 자바에서는 3가지 종류의 노드(파일, 메모리, 파이프)를 지원하는데 각 노드와 연결될 수 있는 클래스는 다음 표와 같습니다.

	Byte Streams	Character Streams
File	FileInputStream FileOutputStream	FileReader FileWriter
Memory : Array	ByteArrayInputStream ByteArrayOutputStream	CharArrayReader CharArrayWriter
Memory : String	-	StringReader StringWriter
Pipe	PipedInputStream PipedOutputStream	PipedReader PipedWriter

입출력 수행을 위해 노드와 연결하려면 앞의 표에 있는 클래스의 객체를 생성하면 됩니다. 그 다음 각 클래스에 정의되어 있는 메서드를 이용하여 원하는 작업을 수행합니다.

다음은 노드스트림의 사용 예를 보인 것입니다.

```
FileReader input = new FileReader("c:\in.txt");
input.read();
```

앞의 예를 보면 "c:₩in.txt" 파일을 읽기 위해 노드스트림인 FileReader객체를 생성했습니다. FileReader 클래스의 메서드들을 이용하면 파일에서 데이터를 읽을 수 있습니다. 예를 들어 FileReader 클래스에는 read() 메서드가 있는데, 이를 호출하면 파일로부터 데이터를 읽어올 수 있습니다. 노드스트림 클래스들을 이용하면 데이터를 읽거나 쓸 수 있습니다.

FileReader 클래스의 read() 메서드는 다음과 같습니다.

- public int read() throws IOException : 파일로부터 한 문자를 읽어 리턴합니다.
- public int read(char[] cbuf) throws IOException : 파일로부터 최대 cbuf 크기만큼 읽어 cbuf에 저장하고, 읽은 문자의 수를 리턴합니다.
- public int read(char[] cbuf, int offset, int length) throws IOException : 파일로부터 데이터를 읽어 cbuf의 offset 인덱스 위치부터 length 개수만큼 cbuf에 저장하고 읽은 문자의 수를 리턴합니다.

먼저 텍스트 파일의 내용을 읽어 화면에 출력하는 예를 보겠습니다.

ReadFileExample.java

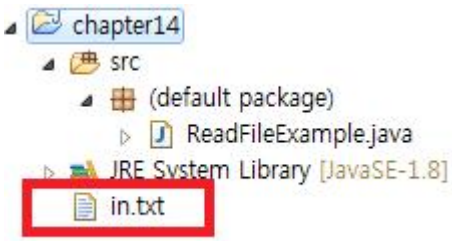
```
1:  import java.io.FileInputStream;
2:  import java.io.FileNotFoundException;
3:  import java.io.IOException;
4:
5:  public class ReadFileExample {
6:
7:     public static void main(String[] args) {
8:        FileInputStream fis = null;
9:
10:       try {
11:          //입력 스트림 객체 생성
12:          fis = new FileInputStream("in.txt");
13:
14:          //입력한 데이터를 저장할 변수 선언
15:          byte[] buffer = new byte[256];
16:
17:          int readCount = fis.read(buffer); //처음 블록을 읽는다.
18:
19:          while(readCount != -1) {
20:             String data = new String(buffer, 0, readCount);
21:             System.out.println(data);
22:
23:             readCount = fis.read(buffer);  //다음 블록을 읽는다.
```

```
24:            }
25:
26:        } catch (FileNotFoundException e) {
27:            System.out.println("파일이 존재하지 않습니다.");
28:        } catch (IOException e) {
29:            System.out.println(e.getMessage());
30:        } finally {
31:            if(fis!=null)
32:                try {fis.close();} catch(Exception e){}
33:        }
34:    }
35: }
```

```
            String data = new String(buffer, 0, readCount);
```

위의 코드에서 buffer 의 내용을 출력하기 위해 String 객체로 생성합니다. String 객체 생성시 buffer의 내용에서 0번 인덱스부터 readCount 개수만큼만 문자열로 생성해야 마지막 직전에 읽었던 내용이 다시 출력되는 현상을 방지할 수 있습니다.

예제를 이클립스에서 실행시키려면 in.txt파일을 **프로젝트 폴더에 작성**하세요. in.txt 파일은 src 폴더 또는 패키지 폴더 안에 넣으면 안 됩니다. in.txt 파일이 없으면 실행이 안 됩니다.



위의 예제는 파일로부터 데이터를 읽을 때 byte 단위로 읽어 문자열 객체로 생성합니다. in.txt 파일의 내용이 한글을 포함할 경우 출력되는 데이터의 일부가 깨질 수 있습니다. 파일이 한글을 포함한다면 FileReader를 이용하면 됩니다.

다음 코드는 FileReader 클래스를 이용하여 데이터를 읽고 지정한 파일에 내용을 출력하는 프로그램입니다. in.txt 파일을 읽어 out.txt 파일에 내용을 복사합니다.

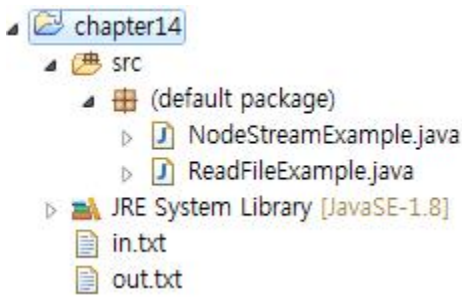
NodeStreamExample.java

```
1:  import java.io.FileReader;
2:  import java.io.FileWriter;
3:  import java.io.IOException;
4:
5:  public class NodeStreamExample {
6:      public static void main(String[] args) {
```

```
7:          FileReader input = null;
8:          FileWriter output = null;
9:          try {
10:            String inFile = "in.txt";
11:            String outFile = "out.txt";
12:            input = new FileReader(inFile);
13:            output = new FileWriter(outFile);
14:
15:            char[] buffer = new char[128];
16:
17:            int readCount = input.read(buffer);
18:            while ( readCount != -1 ) {
19:               output.write(buffer, 0, readCount);
20:               readCount = input.read(buffer);
21:            }
22:            System.out.println("파일이 복사되었습니다.");
23:         } catch (IOException e) {
24:            e.printStackTrace();
25:         } finally {
26:            if(input != null)
27:               try {input.close();} catch (IOException e) {}
28:            if(output != null)
29:               try {output.close();} catch (IOException e) {}
30:         }
31:      }
32: }
```

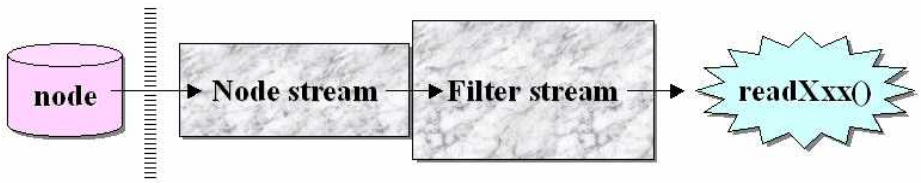
```
Console
<terminated> NodeStreamExamp
파일이 복사되었습니다.
```

위의 예제코드를 실행 하면 out.txt파일이 생성됩니다. 내용은 in.txt 파일에 작성한 내용과 동일합니다. 이클립스에서 예제코드를 실행시켰다면, 실행 후 생성되는 out.txt 파일이 이클립스의 패키지 익스플로러에 나타나게 하려면 프로젝트를 Refresh(F5) 해야 합니다.



13.1.2 필터스트림

필터스트림(Filter Stream)은 처리스트림(Processing Stream)이라고도 하며 다른 객체를 둘러싸는 역할을 합니다.



필터스트림들을 사용하는 이유는 노드스트림의 부족한 기능을 보완하여 좀 더 정밀한 입출력을 하기 위해서입니다. 예들 들어 FileReader클래스의 노드스트림은 파일에서 텍스트를 읽어올 때 한 문자씩 읽는 낮은 수준의 메서드(read() 메서드)만 가지고 있는 반면, BufferedReader클래스 등의 필터스트림은 줄 단위로 읽어 String으로 반환하는 고급 메서드(readLine() 메서드)를 포함하고 있기 때문에 좀 더 편리하게 입출력 작업을 할 수 있는 것입니다.

다음 표는 필터스트림 클래스를 나타낸 것입니다.

Type	Byte Streams	Character Streams
Buffering	BufferedInputStream BufferedOutputStream	BufferedReader BufferedWriter
Data conversion	DataInputStream DataOutputStream	-
Filtering	*FilterInputStream* *FilterOutputStream*	*FilterReader* *FilterWriter*
Converting bytes↔character	-	InputStreamReader OutputStreamWriter
Object Serialization	ObjectInputStream ObjectOutputStream	-
Counting	LineNumberInputStream	LineNumberReader
Printing	PrintStream	PrintWriter

FilterXxx 클래스는 추상(abstract) 클래스이며 다른 필터 클래스들의 상위 클래스입니다. 사용자 정의 필터스트림 클래스를 만들 때 사용할 수 있습니다.

다음은 필터 스트림의 사용 예를 보인 것입니다.

```
FileReader input = new FileReader("c:\in.txt");
BufferedReader bufInput = new BufferedReader(input);
bufInput.readLine();
```

앞의 예에서 BufferedReader클래스의 인자로 FileReader클래스의 객체가 사용되었습니다. 이렇게 BufferedReader클래스를 사용하면 readLine() 메서드를 이용하여 데이터를 라인 단위로 읽을 수 있습니다.

다음 프로그램은 앞의 예제와 같이 파일을 복사하는 프로그램입니다. 앞 프로그램과 다른 점은 문자단위로 읽지 않고, BufferedReader와 BufferedWriter를 이용하여 라인 단위로 입/출력 처리를 합니다.

FilterStreamExample.java

```
1:  import java.io.BufferedReader;
2:  import java.io.BufferedWriter;
3:  import java.io.FileReader;
4:  import java.io.FileWriter;
5:  import java.io.IOException;
6:
7:  public class FilterStreamExample {
8:     public static void main(String[] args) {
9:        FileReader input = null;
10:       FileWriter output = null;
11:       BufferedReader bufInput = null;
12:       BufferedWriter bufOutput = null;
13:       try {
14:          String inFile = "in.txt";
15:          String outFile = "out2.txt";
16:
17:          input = new FileReader(inFile);
18:          output = new FileWriter(outFile);
19:
20:          bufInput = new BufferedReader(input);
21:          bufOutput = new BufferedWriter(output);
22:
23:          String line;
24:          line = bufInput.readLine();
25:          while ( line != null ) {
26:             bufOutput.write(line, 0, line.length());
27:             bufOutput.newLine();
28:             line = bufInput.readLine();
29:          }
30:          System.out.println(inFile + " >> " + outFile);
31:       } catch (IOException e) {
32:          e.printStackTrace();
33:       } finally {
```

```
34:           if(bufInput != null)
35:              try {bufInput .close();} catch (IOException e) {}
36:           if(bufOutput != null)
37:              try {bufOutput.close();} catch (IOException e) {}
38:        }
39:     }
40: }
```

이 예제에서는 FileReader와 FileWriter클래스 외에 BufferedReader와 BufferedWriter 클래스가 사용되었는데 BufferedReader와 BufferedWriter클래스가 필터스트림 클래스에 해당됩니다.

사용자가 원하는 필터스트림 클래스를 만들려면 FilterXxx(예 : FilterReader)형식의 abstract 클래스를 상속받아 구현할 수 있습니다. 자바에서의 입출력은 노드스트림이나 필터스트림 중에서 상황에 맞는 스트림을 선택하여 사용합니다. 특히 자바에서는 다양한 입출력 API를 제공하기 때문에 자주 API 문서를 참고해야 합니다. 또, 어떤 클래스가 노드스트림이고 어떤 클래스가 필터스트림인지 암기할 필요는 없고 API문서에서 사용할 클래스 생성자가 노드스트림 역할을 하는지 필터스트림 역할을 하는지를 구별하면 됩니다. 필터스트림 클래스와 노드스트림 클래스가 같이 사용됐을 경우 finally 블록에서는 어플리케이션과 가장 가까이 있는 스트림, 즉 필터스트림만 닫아줘도 됩니다.

다음 표는 노드스트림 클래스에 해당하는 FileInputStream 클래스의 생성자입니다.

생성자와 설명
FileInputStream(File file) 파일 시스템에서 주어진 파일 객체에 의해 지정된 파일의 연결을 열어 FileInputStream 객체를 생성합니다.
FileInputStream(FileDescriptor fdObj) 파일 시스템에서 주어진 파일 기술자 객체에 의해 지정된 파일의 연결을 열어 FileInputStream 객체를 생성합니다.
FileInputStream(String name) 파일 시스템에서 주어진 파일명으로 파일의 연결을 열어 FileInputStream 객체를 생성합니다.

다음 표는 필터스트림 클래스에 해당하는 DataInputStream 클래스의 생성자입니다.

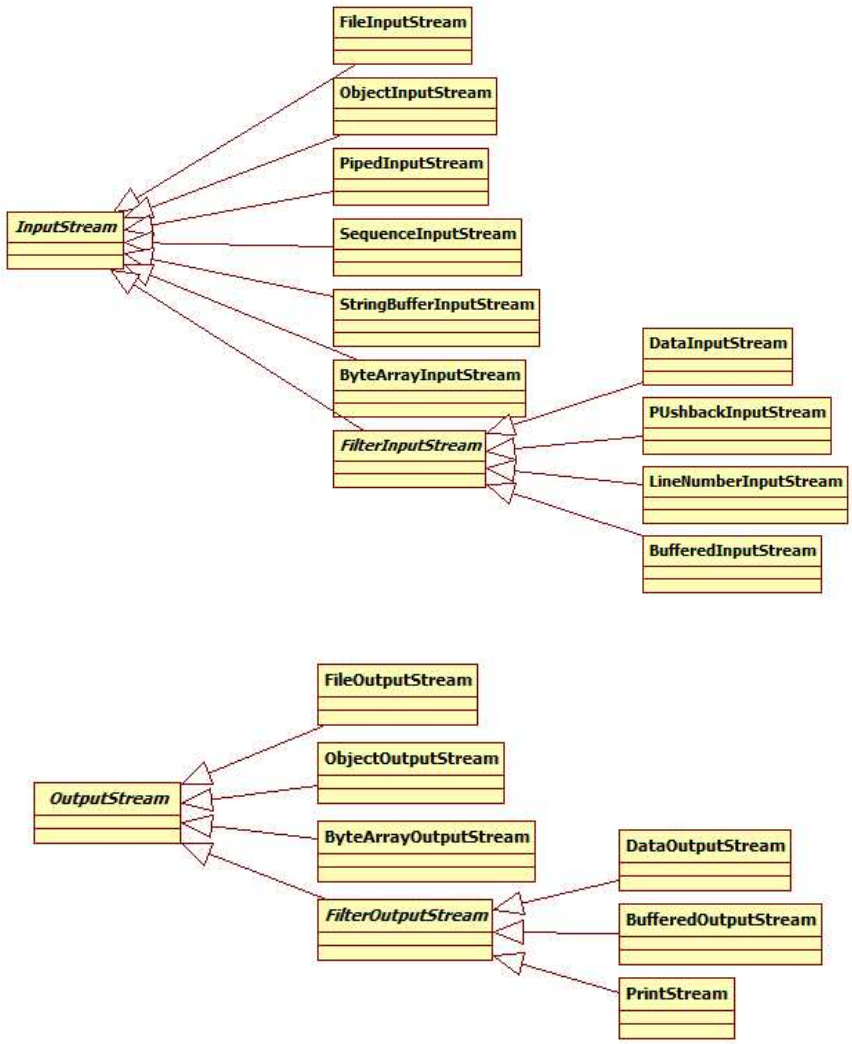
생성자와 설명
DataInputStream(InputStream in) 지정된 기본 InputStream을 사용하는 DataInputStream을 생성합니다.

위의 두 스트림 클래스 생성자의 인자를 주의해서 보세요. FileInputStrem클래스의 경우 3개의 생성자가 있습니다. FileInputStrem클래스의 생성자 인자는 모두 파일과 관련된 것들입니다. 그리고 이들 생성자는 파일에 직접 연결을 열기 때문에 노드스트림임을 알 수 있습니다. 그러나 DataInputStream클래스의 생성자 인자로 노드스트림인 InputStream 클래스의 객체가 사용되었습니다. 이렇게 생성자의 인수로 다른 스트림이 오는 클래스들을 필터 스트림 클래스라고 부릅니다.

13.2. 입/출력 클래스

13.2.1. 바이트스트림 클래스

다음 그림은 자주 사용되는 바이트스트림들의 상속 관계를 나타낸 것입니다. 모든 클래스들을 다 나타내진 않았습니다. 더 자세한 내용을 API문서를 참고하세요.



FileInputStream과 FileOutputStream

이 클래스들은 파일 입력 및 출력에 사용되는 노드스트림 클래스입니다. 객체를 생성할 때 FileInputStream클래스는 파일이 읽기 가능한 상태여야 하며, FileOutputStream클래스는 파일이 없으면 새로운 파일을 생성하고, 있으면 기존 파일을 덮어쓰게 됩니다.

```
FileInputStream infile = new FileInputStream("in.txt");
FileOutputStream outfile = new FileOutputStream("out.txt");
```

다음 프로그램은 앞의 파일복사 예제를 FileInpuStream과 FileOutputStream 클래스를 사용하여 작성한 것입니다.

FileIOExample.java

```
1:  import java.io.FileInputStream;
2:  import java.io.FileNotFoundException;
3:  import java.io.FileOutputStream;
4:  import java.io.IOException;
5:
6:  public class FileIOExample {
7:     public static void main(String [] args) {
8:        String inFile = "in.txt";
9:        String outFile = "out3.txt";
10:
11:       FileInputStream fis = null;
12:       FileOutputStream fos = null;
13:
14:       try {
15:          fis = new FileInputStream(inFile);
16:          fos = new FileOutputStream(outFile);
17:          int readByte = 0;
18:          while ((readByte = fis.read()) != -1) {
19:             fos.write(readByte);
20:          }
21:          System.out.println(inFile + " >> " + outFile);
22:       } catch (FileNotFoundException e) {
23:          System.out.println("파일이 존재하지 않습니다.");
24:       } catch (IOException e) {
25:          System.out.println(e.getMessage());
26:       } finally {
27:          if(fis!=null)
28:             try { fis.close(); } catch (IOException e) { }
29:          if(fos!=null)
30:             try { fos.close(); } catch (IOException e) { }
31:       }
32:    }
33: }
```

FileIOExample 예제 프로그램은 노드스트림인 FileInputStream과 FileOuputStream만을 사용하고 있습니다. read()와 write() 메서드는 1 바이트를 읽고 쓰는 메서드입니다.

DataInputStream과 DataOutputStream

이 클래스는 필터스트림으로 Primitive 형 데이터를 읽거나 쓸 때 사용합니다. 자바에서 사용하는 여러 가지 데이터 유형을 쉽게 입출력 처리하기 위한 많은 메서드를 가지고 있습니다.

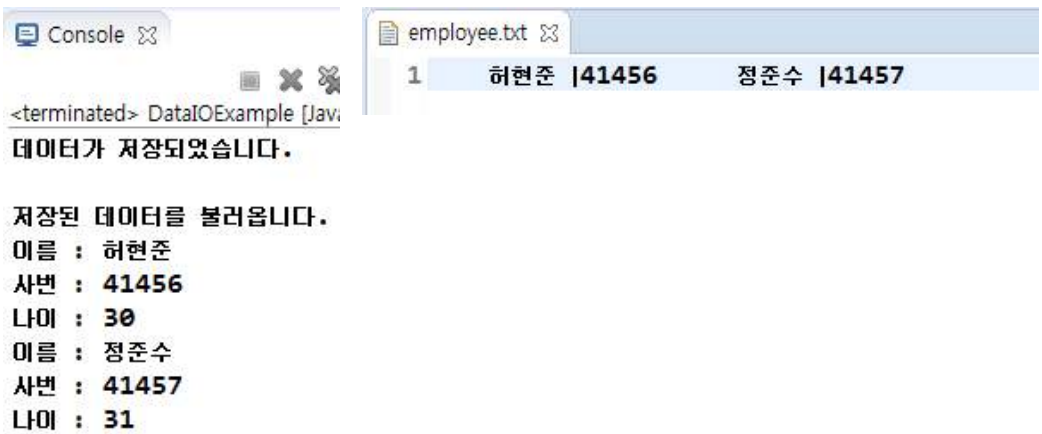
다음 프로그램은 DataInputStream클래스를 이용하여 파일에 "이름", "사번", "나이"를 읽고 쓰는 예입니다.

DataIOExample.java

```
1:  import java.io.DataInputStream;
2:  import java.io.DataOutputStream;
3:  import java.io.FileInputStream;
4:  import java.io.FileNotFoundException;
5:  import java.io.FileOutputStream;
6:  import java.io.IOException;
7:
8:  public class DataIOExample {
9:     public static void main(String [] args) {
10:       String outFile = "employee.txt";
11:
12:       FileOutputStream fos = null;
13:       DataOutputStream dos = null;
14:       try {
15:          fos = new FileOutputStream(outFile);
16:          dos = new DataOutputStream(fos);
17:          dos.writeUTF("허현준");
18:          dos.writeUTF("41456");
19:          dos.writeInt(30);
20:          dos.writeUTF("정준수");
21:          dos.writeUTF("41457");
22:          dos.writeInt(31);
23:          System.out.println("데이터가 저장되었습니다.");
24:       } catch (FileNotFoundException e) {
25:          System.out.println("파일이 존재하지 않습니다.");
26:       } catch (IOException e) {
27:          System.out.println(e.getMessage());
28:       } finally {
29:          if(dos!=null)
30:             try { dos.close(); } catch (IOException e) { }
31:       }
32:
```

```
33:         System.out.println("\n저장된 데이터를 불러옵니다.");
34:         FileInputStream fis = null;
35:         DataInputStream dis = null;
36:         try {
37:            fis = new FileInputStream(outFile);
38:            dis = new DataInputStream(fis);
39:            System.out.println("이름 : " + dis.readUTF());
40:            System.out.println("사번 : " + dis.readUTF());
41:            System.out.println("나이 : " + dis.readInt());
42:            System.out.println("이름 : " + dis.readUTF());
43:            System.out.println("사번 : " + dis.readUTF());
44:            System.out.println("나이 : " + dis.readInt());
45:         } catch (FileNotFoundException e) {
46:            System.out.println("파일이 존재하지 않습니다.");
47:         } catch (IOException e) {
48:            System.out.println(e.getMessage());
49:         } finally {
50:            if(dis!=null)
51:               try { dis.close(); } catch (IOException e) { }
52:         }
53:      }
54: }
```

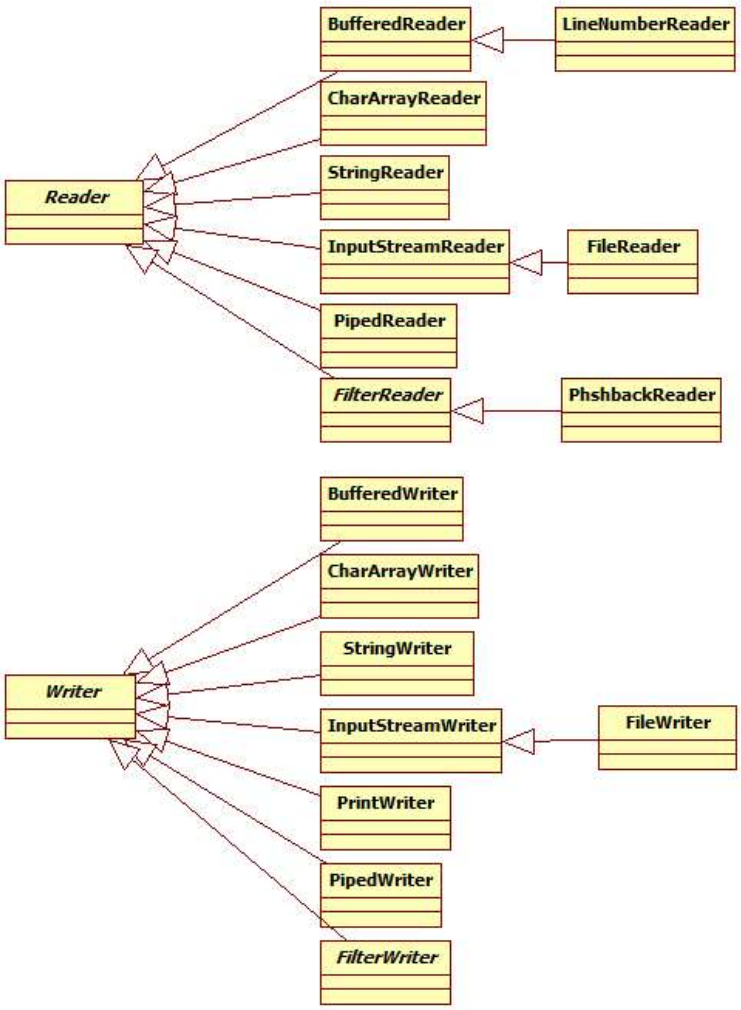
이 예제 프로그램은 DataInputStream클래스와 DataOutputStream클래스의 사용형태를 잘 보여주고 있습니다. 이처럼 기본데이터 타입의 입출력에는 DataInputStream클래스와 DataOutputStream클래스를 사용합니다.



13.2.2. 문자 스트림 클래스

앞에서 설명한 InputStream, OutputStream은 바이트 단위로 입출력 동작을 수행하지만 한글의 경우 2바이트이기 때문에 바이트 단위 처리 시 글자가 깨질 수 있습니다. 따라서 이러한 문제점을 없애기 위해 자바에서는 문자나 문자열을 다룰 때, 유니코드(16bit-Unicode)방식을 제공하고 있습니다. 유니코드를 사용하는 입출력 클래스를 Reader와 Writer라고 부르며, 이들 Reader와 Writer 클래스들을 사용하면 자동으로 바이트가 유니코드화 되므로 한글 같은 2 바이트 문자도 처리할 수 있습니다.

다음 그림은 자주 사용되는 문자 스트림 클래스들의 상속도를 나타낸 것입니다.



InputStreamReader와 OutputStreamWriter

- 이 클래스는 byte 스트림(InputStream, OutputStream)과 char 스트림(Character Reader, Writer 사이의 인터페이스 역할을 합니다. InputStreamReader는 byte 단위 입력 스트림을 char 단위 입력 스트림으로 변환하며 OutputStreamWriter 는 char 단위 출력 스트림을 byte 단위 출력스트림으로 변환합니다.

FileReader와 FileWriter

- 이 클래스는 노드스트림 클래스이며 FileInputStream이나 FileOutputStream과 같은 역할을 하는데, 바이트단위가 아닌 유니코드 문자 단위로 파일 입/출력 처리를 합니다.

BufferedReader와 BufferedWriter

- 이 클래스는 필터스트림 클래스이며 입출력 효율을 증가시켜줍니다.

StringReader와 StringWriter

- 이 클래스는 노드스트림 클래스이며 문자열객체를 읽고 쓰는데 사용합니다.

PipedReader와 PipedWriter

- 이 클래스는 스레드 사이의 통신을 위한 연결 통로로 사용됩니다.

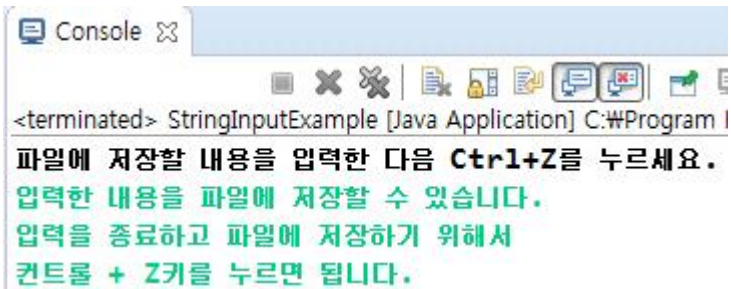
다음 프로그램은 키보드에서 문자열을 입력받아 파일에 저장하는 예입니다.

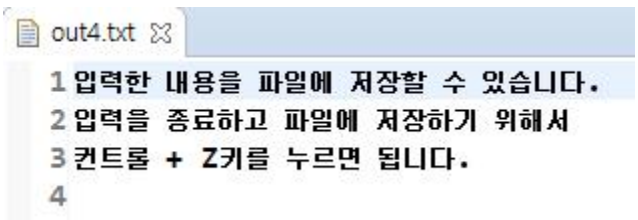
StringInputExample.java

```
1:  import java.io.BufferedReader;
2:  import java.io.BufferedWriter;
3:  import java.io.FileNotFoundException;
4:  import java.io.FileOutputStream;
5:  import java.io.IOException;
6:  import java.io.InputStreamReader;
7:  import java.io.OutputStreamWriter;
8:
9:  public class StringInputExample {
10:    public static void main(String [] args) {
11:       String outFile = "out4.txt";
12:       InputStreamReader isr = null;
13:       BufferedReader br = null;
14:
15:       FileOutputStream fos = null;
16:       OutputStreamWriter osw = null;
17:       BufferedWriter bw = null;
18:
```

```
19:         String inputString;
20:         System.out.println("파일에 저장할 내용을 입력한 다음 Ctrl+Z를 누르세요.");
21:
22:         try {
23:            isr = new InputStreamReader(System.in);
24:            br = new BufferedReader(isr);
25:
26:            fos = new FileOutputStream(outFile);
27:            osw = new OutputStreamWriter(fos);
28:            bw = new BufferedWriter(osw);
29:
30:            while((inputString = br.readLine()) != null) {
31:               bw.write(inputString + "\n");
32:            }
33:         } catch (FileNotFoundException e) {
34:            System.out.println("파일이 존재하지 않습니다.");
35:         } catch (IOException e) {
36:            System.out.println(e.getMessage());
37:         } finally {
38:            if(br!=null)
39:               try { br.close(); } catch (IOException e) { }
40:            if(bw!=null)
41:               try { bw.close(); } catch (IOException e) { }
42:         }
43:      }
44: }
```

이 프로그램이 실행될 때 저장할 문장을 입력한 다음 Ctrl+Z(유닉스 시스템에서는 Ctrl+D)를 눌러 스트림의 끝을 알려줍니다.





예제 코드에서 InputStreamReader가 사용되었습니다. InputStreamReader 클래스는 필터스트림 클래스입니다. 따라서 InputStreamReader의 인자로 System.in을 주었습니다. API문서를 살펴보면 System.in이 InputStream임을 알 수 있습니다.(앞에서 설명했지만, InputStream은 노드스트림입니다.)

BufferedReader클래스는 내부적으로 버퍼를 이용해서 입출력의 성능을 극대화시키기 위해서 사용합니다. readLine() 메서드는 줄 단위로 읽는 메서드입니다. 이 예에서는 엔터키를 누르기 전까지 사용자가 입력한 내용을 한꺼번에 읽습니다. 사용자의 입력을 한 번에 읽어오고 입출력 성능을 높이기 위해 BufferedReader클래스를 사용하였습니다.

finally 블록의 close() 메서드는 open된 파일을 닫아줍니다. 스트림을 닫을 때에는 어플리케이션과 가장 가까운 스트림을 닫아줘야 합니다. 위 예제에서 출력 스트림 객체 fos, osw, bw 중에서 bw 객체는 반드시 닫아야 입력한 내용이 파일에 저장됩니다. fos와 osw 객체를 close() 하더라도 bw 객체를 close() 하지 않으면 파일에 입력한 내용이 저장되지 않습니다.

이 예제 코드에서 파일에 저장되는 데이터의 입출력 스트림 단위를 char 단위로 하려면 아래의 코드를

```
bw = new BufferedWriter(osw);
```

다음과 같이 변경하면 됩니다.

```
bw = new BufferedWriter(new FileWriter(outFile));
```

다음 그림은 InputStreamReader의 생성자를 나타낸 것입니다.

생성자와 설명
InputStreamReader(InputStream in) 디폴트 문자셋을 사용하여 InputStreamReader 객체를 생성합니다.
InputStreamReader(InputStream in, Charset cs) 주어진 문자셋을 사용하여 InputStreamReader 객체를 생성합니다.
InputStreamReader(InputStream in, CharsetDecoder dec) 주어진 문자셋 디코더를 사용하여 InputStreamReader 객체를 생성합니다.
InputStreamReader(InputStream in, String charsetName) 주어진 문자셋 이름을 사용하여 InputStreamReader 객체를 생성합니다.

이 외에도 많은 입/출력 API들이 있으나, 사용방법은 위에서 설명한 내용과 크게 다르지 않습니다.

13.3. 입출력과 관련된 클래스들

13.3.1. 파일(File) 객체

이제 구체적으로 파일을 다루는 법에 대해서 알아보겠습니다. 앞의 예에서는 파일을 지정할 때, 파일이름을 문자열 형태로 주었습니다. 물론 그런 방법으로 파일을 지정할 수도 있지만, 자바에서 제공하는 File클래스를 이용할 수도 있습니다. 파일클래스는 단순하게 파일을 지정하는 목적으로만 사용되는 것은 아니고 매우 다양한 API가 있습니다. 이를 이용하면 파일과 관련된 대부분 작업을 수행할 수 있습니다.

다음 코드는 File객체를 생성하는 방법을 기술한 것입니다. 파일객체는 파일만 가리키는 것은 아니고 디렉터리도 지칭할 수 있음을 기억하기 바랍니다.

```
File someFile, someDir, otherFile;
someFile = new File("c:\autoexec.bat");
someDir = new File("c:\");
otherFile = new File(someDir, "config.sys");
```

다음은 파일클래스의 자주 사용되는 메서드입니다.

파일이름과 관련된 메서드

- String getName() : 파일이름을 반환합니다.
- String getPath() : 파일경로를 반환합니다.
- String getAbsolutePath() : 파일의 절대경로를 반환합니다.
- String getParent() : 파일이 속한 경로를 반환합니다.
- boolean renameTo(File newName) : 파일명을 변경합니다. 변경하고자 하는 파일이 이미 존재할 경우나, 접근권한 등의 이유로 파일명이 변경되지 않으면 false를 반환합니다.
- boolean delete() : 파일을 삭제합니다. 삭제되지 않으면 false를 반환합니다.

파일정보를 알아내는 메서드

- long lastModified() : 마지막으로 수정된 날짜를 long형으로 반환합니다.
- long length() : 파일길이를 반환합니다.

파일을 테스트하는 메서드

- boolean exist() : 파일의 존재여부를 알아냅니다. 파일이 있으면 true, 없으면 false를 반환합니다.
- boolean canWrite() : 쓰기 권한을 가졌는지 알아봅니다. 쓰기 가능하면 true를 반환합니다.
- boolean canRead() : 읽을 권한이 있는지를 알아봅니다. 읽기 가능하면 true를 반환합니다.
- boolean isFile() : 파일인지 알아봅니다. 파일이면 true를 반환하고, 디렉터리 폴더이면 false를 반환합니다.
- boolean isDirectory() : 디렉터리인지 알아봅니다. 디렉터리이면 true를 반환합니다.

◦ boolean isAbsolute() : 절대경로를 가졌는지 알아봅니다.

디렉터리 관련 메서드

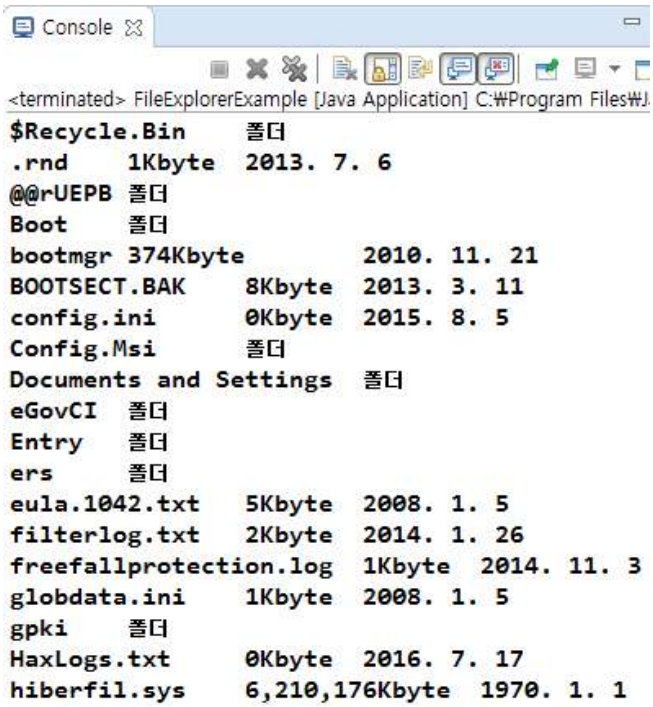
◦ boolean mkdir() : 새로운 디렉터리를 만듭니다. 디렉터리 생성에 실패하면 false를 반환합니다.
◦ String[] list() : 디렉터리 내의 파일 또는 디렉터리를 반환합니다.

다음 프로그램은 C드라이브의 루트 디렉터리 내에 있는 파일 및 디렉터리 목록을 보여주는 예제입니다.

FileExplorerExample.java

```
1:  import java.io.File;
2:  import java.text.DateFormat;
3:  import java.text.NumberFormat;
4:  import java.util.Date;
5:
6:  public class FileExplorerExample {
7:
8:      public static void main(String[] args) {
9:
10:         File myDir = new File("C:/");
11:
12:         File[] listing = myDir.listFiles();
13:
14:         for(int i=0;i<listing.length;++i) {
15:            File file = listing[i];
16:
17:            System.out.print(file.getName() + "\t");
18:
19:            if(file.isFile()) {
20:               NumberFormat f = NumberFormat.getInstance();
21:               DateFormat df = DateFormat.getDateInstance();
22:
23:               System.out.print(f.format(file.length()/1024)+"Kbyte\t");
24:               System.out.print(df.format(new Date(file.lastModified())));
25:            }else {
26:               System.out.print("폴더");
27:            }
28:            System.out.println();
29:         }
30:     }
31: }
```

이 예제는 C드라이브 루트디렉터리의 파일 목록에서 파일일 경우에는 파일의 크기와 마지막 수정된 날짜를 출력하고 디렉터리 일 경우에는 “폴더“라고 출력합니다.



13.3.2. URL 객체

파일클래스가 로컬컴퓨터에 있는 파일을 참조한다면 인터넷상에 있는 주소(URL)를 참조할 수 있는 클래스도 있는데 바로 URL 클래스입니다. URL 클래스는 java.net 패키지에 있습니다.

다음은 URL클래스에서 자주 사용되는 생성자와 메서드입니다.

생성자

생성자와 설명
URL(String spec) String 표현으로부터 URL 객체를 생성합니다.
URL(String protocol, String host, int port, String file) 프로토콜, 호스트이름, 포트번호 그리고 파일이름으로 URL 객체를 생성합니다.
URL(String protocol, String host, int port, String file, URLStreamHandler handler) 프로토콜, 호스트이름, 포트번호, 파일이름 그리고 URL 스트림 핸들러로 URL 객체를 생성합니다.
URL(String protocol, String host, String file) 프로토콜, 호스트이름 그리고 파일이름으로 URL 객체를 생성합니다.

URL(URL context, String spec) 주어진 컨텍스트와 spec으로 URL 객체를 생성합니다.
URL(URL context, String spec, URLStreamHandler handler) 주어진 컨텍스트와 spec, 그리고 URL 스트림 핸들러로 URL 객체를 생성합니다.

메서드

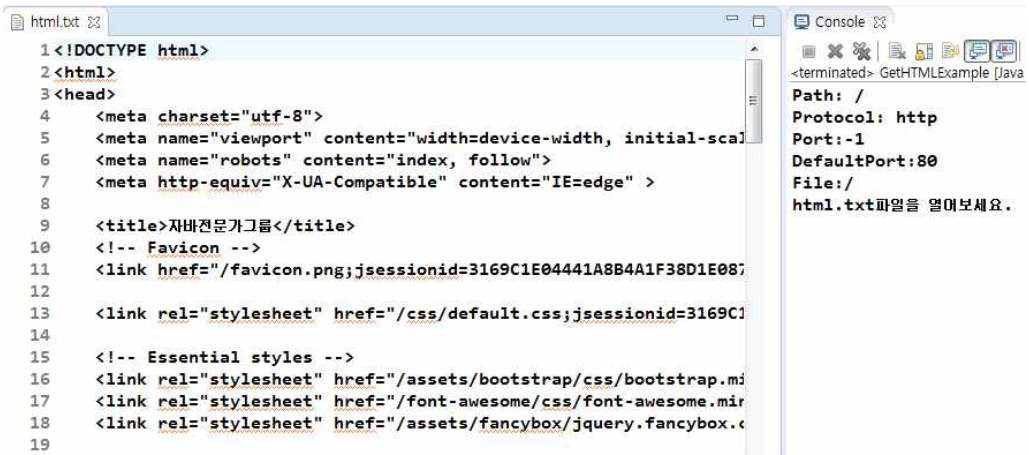
리턴타입	메서드와 설명
Object	getContent() URL이 가진 컨텐츠를 반환합니다.
int	getDefaultPort() URL의 포트번호를 반환합니다. 지정하지 않았다면 -1을 반환합니다. 이 메서드는 jdk1.4에 추가되었습니다. getPort() 메서드와 다른 점은 '기본 포트를 사용할 경우 기본 포트번호를 출력한다'는 것입니다. 예를 들어 http 프로토콜을 이용한 URL에 접속했을 경우 기본 포트를 사용하기 때문에 URL 객체 생성 시 포트번호를 지정하지 않았다면 getPort() 메서드는 -1을 반환하지만 getDefaultPort() 메서드는 80번을 반환합니다.
String	getFile() URL의 파일(폴더)명을 반환합니다.
String	getHost() URL의 호스트 이름을 반환합니다.
String	getPath() URL의 패스를 반환합니다.
int	getPort() URL의 포트번호를 반환합니다. 지정하지 않았다면 -1을 반환합니다.
String	getProtocol() 프로토콜을 반환합니다.
URLConnection	openConnection() URLConnection 객체를 반환합니다.
InputStream	openStream() 입력스트림 객체를 반환합니다.

다음 프로그램은 URL 클래스를 이용하여 인터넷상의 원하는 페이지를 읽어오는 예입니다.

GetHTMLExample.java

```
1:  import java.io.FileNotFoundException;
2:  import java.io.FileOutputStream;
3:  import java.io.IOException;
4:  import java.io.InputStream;
5:  import java.net.MalformedURLException;
6:  import java.net.URL;
7:
8:  public class GetHTMLExample {
```

```
9:     public static void main(String [] args) {
10:       String urlStr = "http://www.javaspecialist.co.kr/";//접속할 사이주 주소
11:
12:       String file = "html.txt";    //접속한 내용을 저장할 파일 명
13:       byte [] inputString = new byte[1024];
14:
15:       URL url = null;
16:       try {
17:          url = new URL(urlStr);
18:       } catch (MalformedURLException e) {
19:          System.out.println("URL 주소가 형식에 맞지 않습니다.");
20:          return;
21:       }
22:
23:       InputStream is = null;
24:       try {
25:          is = url.openStream();
26:       } catch (IOException e) {
27:          System.out.println("주소를 열지 못했습니다.");
28:          return;
29:       }
30:
31:       FileOutputStream fos = null;
32:       try {
33:          fos = new FileOutputStream(file);
34:          while(is.read(inputString, 0, inputString.length) != -1) {
35:             fos.write(inputString);
36:          }
37:
38:          System.out.println("Path: "+url.getPath());
39:          System.out.println("Protocol: "+url.getProtocol());
40:          System.out.println("Port:"+url.getPort());
41:          System.out.println("DefaultPort:"+url.getDefaultPort());
42:          System.out.println("File:"+url.getFile());
43:
44:          System.out.println(file+"파일을 열어보세요.");
45:       } catch (FileNotFoundException e) {
46:          System.out.println("파일이 존재하지 않습니다.");
47:       } catch (IOException e) {
48:          System.out.println(e.getMessage());
49:       } finally {
50:          if(fos != null)
51:             try { fos.close(); } catch (IOException e) { }
52:       }
53:    }
54: }
```




```
        URL url = null;
        try {
           url = new URL(urlStr);
        } catch (MalformedURLException e) {
           System.out.println("URL 주소가 형식에 맞지 않습니다.");
           return;
        }
```

위 코드는 URL을 이용하여 URL 클래스의 인스턴스를 생성합니다.

```
        InputStream is = null;
        try {
           is = url.openStream();
        } catch (IOException e) {
           System.out.println("주소를 열지 못했습니다.");
           return;
        }
```

openStream() 메서드를 호출하면 InputStream객체가 반환됩니다. 이후 InputStream 객체를 사용한다면 인터넷상의 URL을 마치 로컬 컴퓨터의 파일처럼 다룰 수 있습니다.

```
        FileOutputStream fos = null;
        try {
           fos = new FileOutputStream(file);
           while(is.read(inputString, 0, inputString.length) != -1) {
              fos.write(inputString);
           }
```

읽은 데이터를 파일에 쓰기 위해 FileOutputStream 객체를 생성합니다. 이러한 작업을 응용한다면 원하는 사이트의 문서를 브라우저 없이 조회할 수 있을 것입니다.

```
        System.out.println("Path: "+url.getPath());
        System.out.println("Protocol: "+url.getProtocol());
        System.out.println("Port:"+url.getPort());
        System.out.println("DefaultPort:"+url.getDefaultPort());
        System.out.println("File:"+url.getFile());
```

위 코드는 URL클래스의 주요 메서드들을 테스트 한 예입니다.

13.3.3. RandomAccessFile

지금까지 살펴본 입출력 클래스는 순차적으로 데이터를 읽습니다. 즉, 데이터를 읽다가 다시 앞으로 돌아가서 같은 데이터를 읽는 등의 작업을 할 수 없습니다. 그런데 이러한 방법을 제공하는 입출력 클래스가 바로 RandomAccessFile클래스 입니다. 사용법은 다음과 같습니다.

```
RandomAccessFile raf = new RandomAccessFile("c:\config.sys", "rw");
```

생성자의 첫 번째 인자는 오픈하고자하는 파일명입니다. 이때 파일이름은 문자열형태로 사용할 수도 있지만, 앞에서 설명한 파일객체 형태로 사용할 수도 있습니다. 두 번째 인자는 오픈한 파일을 어떻게 접근할 것인가 하는 접근모드를 나타냅니다. "r"과 "rw"를 사용할 수 있으며 "r"은 읽기 전용, "rw"는 읽기와 쓰기가 가능한 모드로 오픈합니다.

RandomAccessFile 클래스가 제공하는 유용한 메서드는 다음과 같습니다.

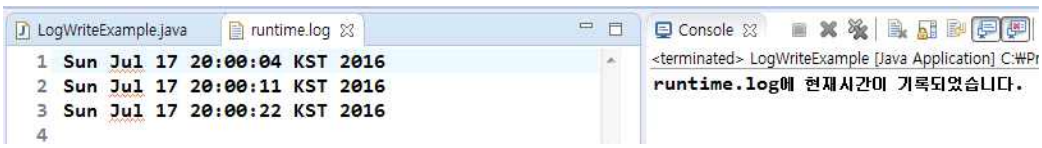
```
long getFilePointer()          // 현재의 file pointer의 위치를 알아냄.
void seek(long position)       // 원하는 위치로 file pointer를 움직임.
long length()                  // file의 size를 알아냄.
```

다음 프로그램은 순차파일이 아닌 임의의 파일관리를 위한 예를 보인 것입니다.

LogWriteExample.java

```
1:  import java.io.FileNotFoundException;
2:  import java.io.IOException;
3:  import java.io.RandomAccessFile;
4:  import java.util.Date;
5:
6:  public class LogWriteExample {
7:     public static void main(String [] args) {
8:        String fileName = "runtime.log";
9:        String mode = "rw";
10:
11:       RandomAccessFile raf = null;
12:       try {
13:          raf = new RandomAccessFile(fileName, mode);
14:
```

```
15:            raf.seek(raf.length());
16:
17:            raf.writeUTF(new Date().toString()+"\n");
18:            System.out.println(fileName + "에 현재시간이 기록되었습니다.");
19:         } catch (FileNotFoundException e) {
20:            System.out.println("파일이 존재하지 않습니다.");
21:         } catch (IOException e) {
22:            System.out.println(e.getMessage());
23:         } finally {
24:            if(raf!=null)
25:               try { raf.close(); } catch (IOException e) { }
26:         }
27:      }
28: }
```



이 프로그램은 실행할 때마다 실행한 시간을 로그기록으로 남깁니다.

```
raf = new RandomAccessFile(fileName, mode);
```

RandomAccessFile 객체를 생성할 때의 접근 모드는 "rw"는 파일을 읽기/쓰기 전용으로 오픈합니다.

```
            raf.seek(raf.length());
```

seek(raf.length())는 파일포인터를 파일의 맨 뒤로 보냅니다. length()메서드는 파일의 길이를 리턴하며, seek()메서드는 원하는 위치로 파일포인터를 보내주는 메서드입니다. 파일 이름에서 알 수 있듯이, 이 파일은 로그(log) 파일을 기록하는 예입니다. 즉, 파일의 뒷부분에 데이터가 계속 기록되는 형태입니다.

```
            raf.writeUTF(new Date().toString()+"\n");
```

이 코드는 현재시간을 알아내서 그 시간을 파일에 저장합니다. writeUTF() 메서드는 16비트 유니코드 문자를 8 비트 문자로 변경하여 저장하는 메서드입니다. 메모장에서 열면 그 내용을 알 수 없습니다. 그러나 이클립스 또는 ㅎ글 등의 문서 편집기를 이용하여 한글 완성형으로 열면 그 내용을 확인할 수 있습니다.

13.4. 객체 직렬화

13.4.1. Serializable 인터페이스

객체 직렬화(Object Serialization)는 파일입출력에 있어서 중요하면서 많이 사용됩니다. 객체 직렬화는 생성된 객체가 스트림을 통해 이동하는 것을 의미합니다. 자바에서는 메모리에서 생성된 객체를 파일에 저장해서 객체에 영속성을 부여하거나, 소켓을 통해 객체를 다른 JVM 환경으로 이동 시키는 것을 객체 직렬화라고 합니다. 객체 직렬화에 사용되는 클래스는 Serializable 이라는 인터페이스를 반드시 implements 해야 합니다. 그런데 이 인터페이스에는 구현할 메서드가 없습니다. Serializable 인터페이스를 implements하는 것은, 이 클래스가 Serialization이 가능함을 나타내는 표시일 뿐입니다. 이러한 인터페이스들을 Marker 인터페이스라고 부릅니다.

다음 프로그램은 Customer 클래스의 객체를 저장하기 위해 Serializable 인터페이스를 구현한 예입니다.

Customer.java

```
1:  import java.io.Serializable;
2:
3:  public class Customer implements Serializable {
4:     private String name;
5:     private char gender;
6:     private String email;
7:     private int birthYear;
8:
9:     public Customer(String name, char gender, String email, int birthYear) {
10:       this.name = name;
11:       this.gender = gender;
12:       this.email = email;
13:       this.birthYear = birthYear;
14:    }
15:
16:    @Override
17:    public String toString() {
18:       return "Customer [name=" + name + ", gender=" + gender + ", email=" + email
    + ", birthYear=" + birthYear + "]";
19:    }
20: }
```

중요한 점은 실제 객체가 Serialization될 때, 데이터부분만 Serialization된다는 것입니다. 즉, 앞의 Person클래스 객체를 파일로 저장할 때, 실제로 저장되는 것은 name, gender, email, birthYear 멤버 변수 데이터만 저장되고, 생성자나 메서드 코드는 저장되지 않습니다.

Customer 클래스의 객체를 파일에 저장 예제를 통해 직렬화에 대해서 알아보겠습니다.

WriteCustomerExample.java

```
1:  import java.io.FileOutputStream;
2:  import java.io.IOException;
3:  import java.io.ObjectOutputStream;
4:
5:  public class WriteCustomerExample {
6:
7:     public static void main(String [] args) {
8:        Customer cust1 = new Customer("허현정", 'M', "heojk2@daum.net", 21);
9:        Customer cust2 = new Customer("허현준", 'M', "loid@gmail.com", 20);
10:
11:       FileOutputStream fos = null;
12:       ObjectOutputStream oos = null;
13:       try {
14:          fos = new FileOutputStream("customer.ser");
15:          oos = new ObjectOutputStream(fos);
16:
17:          oos.writeObject(cust1);
18:          oos.writeObject(cust2);
19:          System.out.println("Customer 데이터가 저장되었습니다.");
20:       } catch (IOException e) {
21:          System.out.println(e.getMessage());
22:       } finally {
23:          if(oos!=null)
24:             try { oos.close(); } catch (IOException e) { }
25:       }
26:
27:    }
28: }
```

파일에 출력하기 위해 FileOutputStream객체를 생성합니다. 그리고 객체를 저장하기 위해 필터스트림인 ObjectOutputStream객체를 생성합니다.

객체를 저장할 수 있는 writeObject() 메서드는 필터스트림인 ObjectOutputStream에 있습니다. 따라서 메서드를 사용하려면 ObjectOutputStream 필터스트림 클래스를 적용할 수밖에 없습니다. (입출력 클래스의 적용은 대부분 이런 방법으로 이루어집니다.) ObjectOutputStream에 Customer 클래스의 인스턴스를 저장합니다.

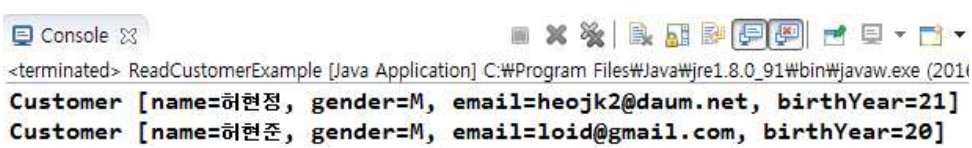
저장된 파일은 직렬화된 파일이기 때문에 텍스트 에디터로는 내용을 알아볼 수 없습니다.

다음 프로그램은 앞의 WriteMyObject 프로그램을 실행했을 때 생성되는 customer.ser파일을 읽어 저장되어 있는 Person 객체의 데이터를 출력하는 예입니다.

ReadCustomerExample.java

```
1:  import java.io.FileInputStream;
2:  import java.io.FileNotFoundException;
3:  import java.io.IOException;
4:  import java.io.ObjectInputStream;
5:
6:  public class ReadCustomerExample {
7:
8:     public static void main(String [] args) {
9:        FileInputStream fis = null;
10:       ObjectInputStream ois = null;
11:       try {
12:          fis = new FileInputStream("customer.ser");
13:          ois = new ObjectInputStream(fis);
14:
15:          Customer cust1 = (Customer)ois.readObject();
16:          Customer cust2 = (Customer)ois.readObject();
17:          System.out.println(cust1.toString());
18:          System.out.println(cust2.toString());
19:
20:       } catch (FileNotFoundException e) {
21:          System.out.println("파일이 존재하지 않습니다.");
22:       } catch (IOException e) {
23:          System.out.println(e.getMessage());
24:       } catch (ClassNotFoundException e) {
25:          System.out.println(e.getMessage());
26:       } finally {
27:          if(ois!=null)
28:             try { ois.close(); } catch (IOException e) { }
29:       }
30:    }
31: }
```

파일로부터 읽기 위해 FileInputStream객체를 생성합니다. 그리고 객체를 읽기 위해 필터스트림인 ObjectInputStream객체를 생성합니다. 객체를 읽을 수 있는 메서드인 readObject() 메서드는 필터스트림인 ObjectInputStream에 존재합니다. readObject() 메서드는 Customer 객체에 customer.ser파일의 객체를 불러들입니다. readObject() 메서드의 결과를 Customer 클래스로 형 변환하고 있는데, 이는 readObject() 메서드의 반환형이 Object형이기 때문에 실제 저장되어 있는 클래스인 Customer 클래스로 형 변환이 필요한 것입니다.



13.4.2. transient

이 키워드는 Serializable과 함께 알아 두면 유용합니다. transient 키워드는 직렬화되지 않는 객체 멤버 변수를 포함한 클래스를 직렬화 시킬 때나 특정변수를 직렬화에서 제외시키고자 할 때 사용하는 제한자입니다.

```
1:  public class MyClass implements Serializable {
2:     public transient Thread myThread;
3:     private String customerID;
4:     private int total;
5:  }
```

이 예에서 스레드 객체는 직렬화가 불가능한 객체이므로 transient키워드로 선언한 것입니다. 만일 transient 키워드가 생략되면 MyClass 클래스는 직렬화 안 됩니다.

```
1:  public class MyClass implements Serializable {
2:     public transient Thread myThread;
3:     private transient String customerID;
4:     private int total;
5:  }
```

이 예에서는 MyClass의 객체를 직렬화 시킬 때 customerID는 직렬화에서 제외한다는 뜻입니다. String클래스는 직렬화 가능하므로 transient 키워드를 생략해도 MyClass는 직렬화 됩니다.

다음 코드는 직렬화 제외되는 변수를 transient 로 지정한 예입니다. Account 클래스의 password 필드는 직렬화에서 제외시키기 위해 transient 로 지정했습니다. 비밀번호 필드 등은 보안상의 이유로도 직렬화에서 제외되어야 할 것입니다.

Account.java

```
1:  import java.io.Serializable;
2:
3:  public class Account implements Serializable {
4:
5:     String accountNo;
6:     String userName;
7:     int balance;
8:     transient String password;
9:
10:    public Account(String accountNo, String userName, int balance, String
    password) {
11:       super();
12:       this.accountNo = accountNo;
13:       this.userName = userName;
```

```
14:         this.balance = balance;
15:         this.password = password;
16:     }
17:     public String toString() {
18:         return accountNo + "[" + userName + ", " + password + "] : " + balance;
19:     }
20: }
```

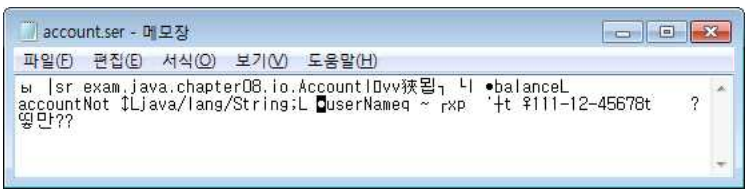
SaveAccountExample.java

```
1:  import java.io.FileOutputStream;
2:  import java.io.ObjectOutputStream;
3:
4:  public class SaveAccountExample {
5:
6:      public static void main(String[] args) throws Exception {
7:          FileOutputStream fos = new FileOutputStream("account.ser");
8:          ObjectOutputStream oos = new ObjectOutputStream(fos);
9:          Account a1 = new Account("111-12-45678", "홍길동", 10000, "1224");
10:         System.out.println(a1);
11:         oos.writeObject(a1);
12:         System.out.println("데이터가 저장되었습니다.");
13:         oos.close();
14:     }
15: }
```

```
Console
<terminated> SaveAccountExample [Java Application]
111-12-45678[홍길동, 1224] : 10000
데이터가 저장되었습니다.
```

SaveAccountExample 클래스는 Account 클래스를 직렬화 시키는 예입니다.

예제를 실행시키면 account.ser 파일이 생성됩니다. 생성된 파일을 메모장에서 열었을 때 아래의 그림처럼 전체 저장된 내용이 무엇인지 정확히 알 수 없습니다.



ReadAccountExample 클래스는 account.ser 파일에 저장된 Account 객체를 불러와서 출력하는 예입니다.

ReadAccountExample.java

```
1:  import java.io.FileInputStream;
2:  import java.io.ObjectInputStream;
3:
4:  public class ReadAccountExample {
5:
6:      public static void main(String[] args) throws Exception {
7:          FileInputStream fis = new FileInputStream("account.ser");
```

```
8:          ObjectInputStream ois = new ObjectInputStream(fis);
9:          Account a2 = (Account)ois.readObject();
10:         System.out.println(a2);
11:         ois.close();
12:     }
13: }
```

Console
<terminated> ReadAccountExample [Java Application] C
111-12-45678[홍길동, null] : 10000

실행결과에서 보는바와 같이 password 필드에 해당하는 부분은 null로 출력된 것을 알 수 있습니다. Account 클래스의 password 변수는 transient 로 선언되었으므로 직렬화시 해당 변수의 정보가 저장되지 않습니다.

13.4.3. serialVersionUID

만일 Account 클래스가 변경되면 어떻게 될까요? Account 클래스에 새로운 필드 또는 메서드가 추가되거나 아니면 기존의 필드 또는 메서드가 삭제되어 클래스가 변경될 수도 있을 것입니다. 직렬화 가능한 클래스의 구조가 변경된다면 클래스 구조가 변경되기 전에 직렬화 시킨 데이터를 불러오지 못할 수도 있습니다.
다음의 실행 결과는 Account 객체를 직렬화 시켜 데이터를 저장한 다음 Account 클래스에 새로운 필드를 추가(String newField;)하고 ReadAccountExample 클래스를 실행시킬 때 발생하는 예외를 보여주고 있습니다.

Console
<terminated> ReadAccountExample [Java Application] C:₩Program Files₩Java₩jre1.8.0_91₩bin₩javaw.exe (2016. 7. 17. 오후 8:31:20)

```
Exception in thread "main" java.io.InvalidClassException: Account; local class incompatible:
        at java.io.ObjectStreamClass.initNonProxy(Unknown Source)
        at java.io.ObjectInputStream.readNonProxyDesc(Unknown Source)
        at java.io.ObjectInputStream.readClassDesc(Unknown Source)
        at java.io.ObjectInputStream.readOrdinaryObject(Unknown Source)
        at java.io.ObjectInputStream.readObject0(Unknown Source)
        at java.io.ObjectInputStream.readObject(Unknown Source)
        at ReadAccountExample.main(ReadAccountExample.java:9)
```

serialVersionUID 필드가 정의되지 않은 상태에서 데이터가 저장되어 있다면 새로운 멤버변수를 추가했을 때 직렬화 된 데이터를 불러올 경우 위와 같은 예외 발생할 것입니다.

나중에 클래스가 변경되더라도 이전에 직렬화 했었던 데이터를 정상적으로 불러오게 하기 위해서는 아래와 같이 멤버변수를 추가해야 합니다. JVM은 클래스의 구조가 다르더라도 serialVersionUID의 값이 같다면 클래스의 구조가 다르더라도 같은 클래스로 간주할 것입니다. 새로운 멤버변수를 추가하고 실행했을 때 위와 같은 예외가 발생한다면 예외 메시지에서 발견할 수 있는 stream classdesc serialVersionUID의 값을 이용해 클래스의 serialVersionUID 값을 설정하면 됩니다.

```
private static final long serialVersionUID = 5004258855763033943L
```

클래스를 처음 만들 때라면 serialVersionUID의 값은 long 형 값의 범위에 포함된다면 어떤 값도 상관없습니다. 다만 다른 클래스의 serialVersionUID 값과 중복되지 않으면 됩니다. 그렇더라도 가능하다면 이클립스에서 생성시켜주는 값을 이용하는 것을 권장합니다. 물론 이미 데이터가 직렬화 되어 있는 상태라면 실행시 발생하는 예외에서 UID값을 찾을 수 있습니다. InvalidClassException 예외 메시지의 오른쪽을 보면 UID 값이 두 개가 있는 것을 확인할 수 있습니다. 두 UID 값 중 왼쪽 UID 값을 이용해 serialVersionUID 값을 지정한다면 이미 직렬화 된 데이터도 클래스의 구조가 바뀌더로 읽을 수 있습니다.

```
stream classdesc serialVersionUID = 5004258855763033943, local class serialVersionUID = 6147737271002642340
```

직렬화 가능한 클래스를 정의할 경우 serialVersionUID 필드를 선언하는 것을 잊지 마세요.

다음은 Account 클래스에 serialVersionUID 필드를 선언한 것입니다. 앞의 코드에 7라인만 추가했습니다. 값은 클래스들끼리 중복되지 않고, long형 범위 안에 포함되는 정수값이면 모두 가능합니다.

Account.java

```
1:  import java.io.Serializable;
2:
3:  public class Account implements Serializable {
4:
5:      private static final long serialVersionUID = 5004258855763033943L;
6:
7:      String accountNo;
8:      String userName;
9:      int balance;
10:     transient String password;
11:     String newField;
12:
13:     public Account(String accountNo, String userName, int balance, String
    password) {
14:         super();
15:         this.accountNo = accountNo;
16:         this.userName = userName;
17:         this.balance = balance;
18:         this.password = password;
19:     }
20:     public String toString() {
21:         return accountNo + "[" + userName + ", " + password + "] : " + balance;
22:     }
23: }
```

13.5. 고객관리 프로그램 III(데이터 저장하기)

고객의 정보를 관리하는 프로그램을 고객의 정보를 저장하도록 수정하겠습니다. 객체 직렬화를 이용하여 ArrayList 객체를 파일에 저장한 다음 프로그램이 시작되면 파일에 저장되어 있는 정보를 로드하여 다시 조회 할 수 있도록 합니다.

13.5.1. Customer 클래스

Customer 클래스는 직렬화 가능한 객체가 되도록 Serializable 인터페이스를 implements 해야 합니다.

Customer.java

```
1:  import java.io.Serializable;
2:
3:  public class Customer implements Serializable {
4:
5:     private static final long serialVersionUID = -4205785232581803172L;
6:
7:     private String name;
8:     private char gender;
9:     private String email;
10:    private int birthYear;
11:
12:    public Customer(String name, char gender, String email, int birthYear) {
13:       super();
14:       this.name = name;
15:       this.gender = gender;
16:       this.email = email;
17:       this.birthYear = birthYear;
18:    }
19:
20:    public String getName() {
21:       return name;
22:    }
23:    public void setName(String name) {
24:       this.name = name;
25:    }
26:    public char getGender() {
27:       return gender;
28:    }
29:    public void setGender(char gender) {
30:       this.gender = gender;
31:    }
32:    public String getEmail() {
```

```
33:         return email;
34:     }
35:     public void setEmail(String email) {
36:         this.email = email;
37:     }
38:     public int getBirthYear() {
39:         return birthYear;
40:     }
41:     public void setBirthYear(int birthYear) {
42:         this.birthYear = birthYear;
43:     }
44:
45:     @Override
46:     public String toString() {
47:         return "Customer [name=" + name + ", gender=" + gender + ", email=" + email
    + ", birthYear=" + birthYear + "]";
48:     }
49: }
```

13.5.2. CustomerManager 클래스

다음 코드는 CustomerManager 클래스입니다. 진하게 작성된 부분이 7장 예제코드에 추가된 부분입니다.

CustomerManager.java

```
1:  import java.io.FileInputStream;
2:  import java.io.FileNotFoundException;
3:  import java.io.FileOutputStream;
4:  import java.io.IOException;
5:  import java.io.ObjectInputStream;
6:  import java.io.ObjectOutputStream;
7:  import java.util.ArrayList;
8:  import java.util.Scanner;
9:
10: public class CustomerManager {
11:
12:     //고객 정보를 저장할 자료구조 선언
13:     static ArrayList<Customer> custList = new ArrayList<>();
14:
15:     //리스트 정보를 조회하기 위해 인덱스를 필요로 함
16:     static int index = -1;
17:
18:     static int count = 0;//custList.size()
19:
20:     //기본 입력장치로부터 데이터를 입력받기 위해 Scanner객체 생성
21:     static Scanner scan = new Scanner(System.in);
22:
```

```
23:    public static void main(String[] args) {
24:       readCustomerData();
25:       while(true) {
26:          count = custList.size();
27:          System.out.printf("\n[INFO] 고객 수 : %d, 인덱스 : %d\n", count, index);
28:          System.out.println("메뉴를 입력하세요.");
29:          System.out.println("(I)nsert, (P)revious, (N)ext, " +
30:                "(C)urrent, (U)pdate, (D)elete, (S)ave, (Q)uit");
31:          System.out.print("메뉴 입력: ");
32:          String menu = scan.next();
33:          menu = menu.toLowerCase();        //입력한 문자열을 모두소문자로 변환
34:          switch(menu.charAt(0)) {
35:          case 'i':
36:             System.out.println("고객정보 입력을 시작합니다.");
37:             insertCustomerData();
38:             System.out.println("고객정보를 입력했습니다.");
39:             break;
40:          case 'p' :
41:             System.out.println("이전 데이터를 출력합니다.");
42:             if(index <= 0) {
43:                System.out.println("이전 데이터가 존재하지 않습니다.");
44:             }else {
45:                index--;
46:                printCustomerInfo(index);
47:             }
48:             break;
49:          case 'n' :
50:             System.out.println("다음 데이터를 출력합니다.");
51:             if(index >= count-1) {
52:                System.out.println("다음 데이터가 존재하지 않습니다.");
53:             }else {
54:                index++;
55:                printCustomerInfo(index);
56:             }
57:             break;
58:          case 'c' :
59:             System.out.println("현재 데이터를 출력합니다.");
60:             if( (index >= 0) && (index < count)) {
61:                printCustomerInfo(index);
62:             }else {
63:                System.out.println("출력할 데이터가 선택되지 않았습니다.");
64:             }
65:             break;
66:          case 'u' :
67:             System.out.println("데이터를 수정합니다.");
68:             if( (index >= 0) && (index < count)) {
69:                System.out.println(index + "번째 데이터를 수정합니다.");
70:                updateCustomerData(index);
71:             }else {
72:                System.out.println("수정할 데이터가 선택되지 않았습니다.");
```

```
73:             }
74:             break;
75:          case 'd' :
76:             System.out.println("데이터를 삭제합니다.");
77:             if( (index >= 0) && (index < count)) {
78:                System.out.println(index + "번째 데이터를 삭제합니다.");
79:                deleteCustomerData(index);
80:             }else {
81:                System.out.println("삭제할 데이터가 선택되지 않았습니다.");
82:             }
83:             break;
84:          case 's' :
85:             saveCustomerData();
86:             break;
87:          case 'q' :
88:             System.out.println("프로그램을 종료합니다.");
89:             scan.close();          //Scanner 객체를 닫아준다.
90:             System.exit(0);        //프로그램을 종료시킨다.
91:             break;
92:          default :
93:             System.out.println("메뉴를 잘 못 입력했습니다.");
94:          }//end switch
95:       }//end while
96:    }//end main
97:
98:    public static void insertCustomerData() {
99:       System.out.print("이름 : ");
100:      String name = scan.next();
101:      System.out.print("성별(M/F) : ");
102:      char gender = scan.next().charAt(0);
103:      System.out.print("이메일 : ");
104:      String email = scan.next();
105:      System.out.print("출생년도 : ");
106:      int birthYear = scan.nextInt();
107:
108:      //입력받은 데이터로 고객 객체를 생성
109:      Customer cust = new Customer(name, gender, email, birthYear);
110:
111:      //고객 객체를 ArrayList에 저장
112:      custList.add(cust);
113:   }
114:
115:   //고객데이터 출력
116:   public static void printCustomerInfo(int index) {
117:      Customer cust = custList.get(index);
118:      System.out.println("==========CUSTOMER INFO================");
119:      System.out.println("이름 : " + cust.getName());
120:      System.out.println("성별 : " + cust.getGender());
121:      System.out.println("이메일 : " + cust.getEmail());
122:      System.out.println("출생년도 : " + cust.getBirthYear());
```

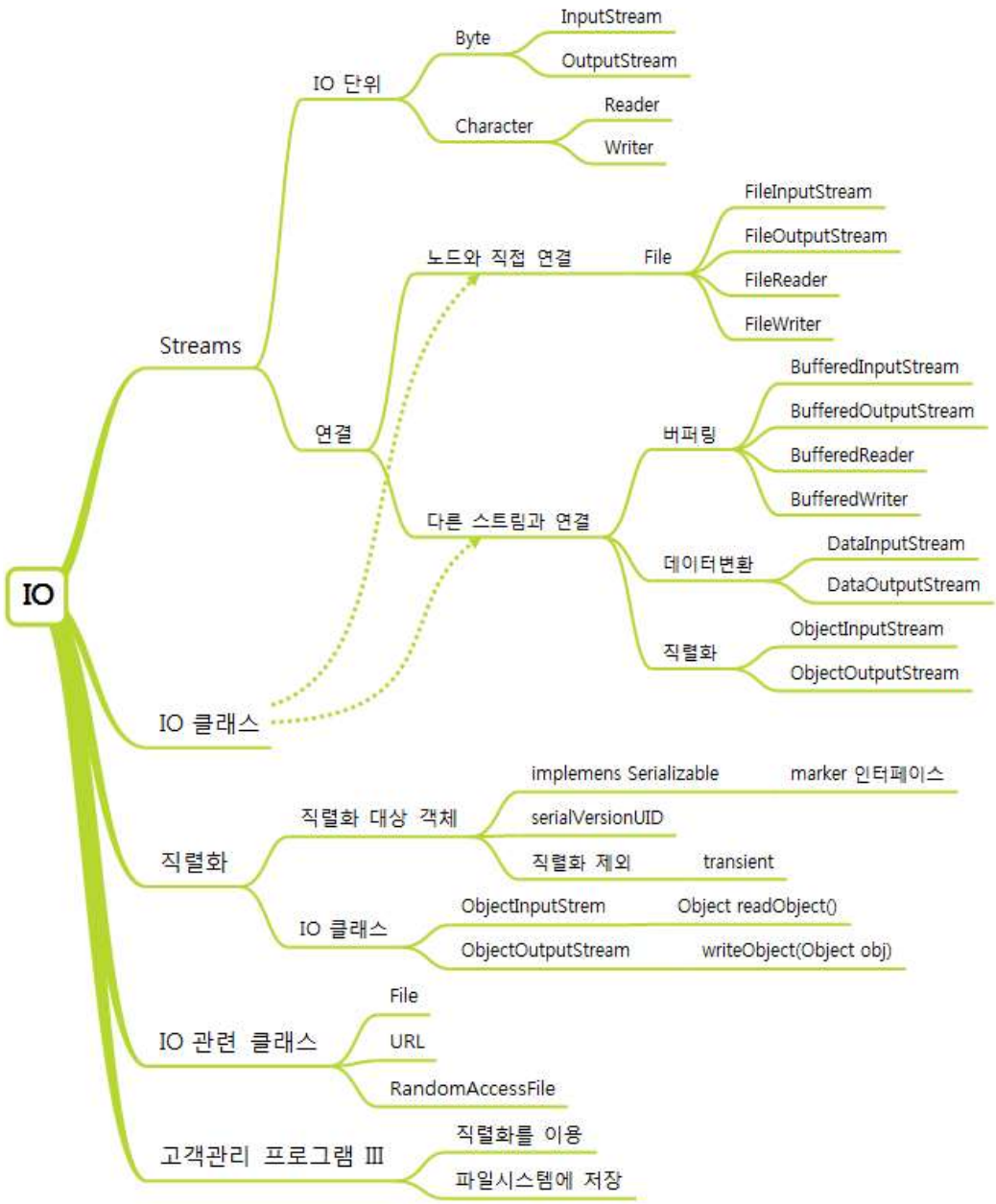
```
123:        System.out.println("=====================================");
124:    }
125:
126:    //index 위치의 고객정보를 삭제합니다.
127:    public static void deleteCustomerData(int index) {
128:        custList.remove(index);
129:    }
130:
131:    //index 위치의 고객 정보를 수정합니다.
132:    public static void updateCustomerData(int index) {
133:        Customer cust = custList.get(index);
134:        System.out.println("---------UPDATE CUSTOMER INFO----------");
135:        System.out.print("이름(" + cust.getName() + ") :");
136:        cust.setName(scan.next());
137:
138:        System.out.print("성별(" + cust.getGender() + ") :");
139:        cust.setGender(scan.next().charAt(0));
140:
141:        System.out.print("이메일(" + cust.getEmail() + ") :");
142:        cust.setEmail(scan.next());
143:
144:        System.out.print("출생년도(" + cust.getBirthYear() + ") :");
145:        cust.setBirthYear(scan.nextInt());
146:    }
147:
148:    @SuppressWarnings("unchecked")
149:    public static void readCustomerData() {
150:        String fileName = "customer.data";
151:
152:        FileInputStream fis = null;
153:        ObjectInputStream ois = null;
154:        try {
155:            fis = new FileInputStream(fileName);
156:            ois = new ObjectInputStream(fis);
157:
158:            custList = (ArrayList<Customer>)ois.readObject();
159:        } catch (FileNotFoundException e) {
160:            System.out.println("파일이 존재하지 않습니다.");
161:        } catch (IOException e) {
162:            System.out.println(e.getMessage());
163:        } catch (ClassNotFoundException e) {
164:            System.out.println(e.getMessage());
165:        } finally {
166:            if(ois != null)
167:                try{ois.close();} catch (IOException e) { }
168:        }
169:    }
170:
171:    public static void saveCustomerData() {
172:        String fileName = "customer.data";
```

```
173:
174:        FileOutputStream fos = null;
175:        ObjectOutputStream oos = null;
176:
177:        try {
178:           fos = new FileOutputStream(fileName);
179:           oos = new ObjectOutputStream(fos);
180:
181:           oos.writeObject(custList);
182:           System.out.println("고객 데이터가 저장됐습니다.");
183:        } catch (FileNotFoundException e) {
184:           System.out.println("파일이 존재하지 않습니다.");
185:        } catch (IOException e) {
186:           System.out.println(e.getMessage());
187:        } finally {
188:           if(oos != null)
189:              try { oos.close(); } catch(Exception e){}
190:        }
191:     }
192:
193:}//end class
```

```
Console
<terminated> CustomerManager (2) [Java Application] C:₩Program Files₩Java₩jre1.8.0_91₩bin₩javaw.exe (2016. 7. 17. 오후 8:4
파일이 존재하지 않습니다.

[INFO] 고객 수 : 0, 인덱스 : -1
메뉴를 입력하세요.
(I)nsert, (P)revious, (N)ext, (C)urrent, (U)pdate, (D)elete, (S)ave, (Q)uit
메뉴 입력: i
고객정보 입력을 시작합니다.
이름 : 허현정
성별(M/F) : M
이메일 : heojk2@javaspecialist.co.kr
출생년도 : 2006
고객정보를 입력했습니다.

[INFO] 고객 수 : 1, 인덱스 : -1
메뉴를 입력하세요.
(I)nsert, (P)revious, (N)ext, (C)urrent, (U)pdate, (D)elete, (S)ave, (Q)uit
메뉴 입력: i
고객정보 입력을 시작합니다.
이름 : 허현준
성별(M/F) : M
이메일 : loid@javaspecialist.co.kr
출생년도 : 2007
고객정보를 입력했습니다.

[INFO] 고객 수 : 2, 인덱스 : -1
메뉴를 입력하세요.
(I)nsert, (P)revious, (N)ext, (C)urrent,
메뉴 입력: i
고객정보 입력을 시작합니다.
이름 : 허현수
성별(M/F) : M
이메일 : hjk7902@gmail.com
출생년도 : 2010
```

```
Console
CustomerManager (2) [Java Application] C:₩Program Files₩Java₩jre1.8.0_91₩bin₩javaw.exe (2016. 7. 17. 오후 8:52:40)

[INFO] 고객 수 : 3, 인덱스 : -1
메뉴를 입력하세요.
(I)nsert, (P)revious, (N)ext, (C)urrent, (U)pdate, (D)elete, (S)ave, (Q)uit
메뉴 입력: n
다음 데이터를 출력합니다.
==========CUSTOMER INFO================
이름 : 허현정
성별 : M
이메일 : heojk2@javaspecialist.co.kr
출생년도 : 2006
========================================

[INFO] 고객 수 : 3, 인덱스 : 0
메뉴를 입력하세요.
(I)nsert, (P)revious, (N)ext, (C)urrent, (U)pdate, (D)elete, (S)ave, (Q)uit
메뉴 입력:
```

13.6. 마인드맵 정리


IO
Streams
IO 단위
Byte
InputStream
OutputStream
Character
Reader
Writer
연결
노드와 직접 연결
File
FileInputStream
FileOutputStream
FileReader
FileWriter
다른 스트림과 연결
버퍼링
BufferedInputStream
BufferedOutputStream
BufferedReader
BufferedWriter
데이터변환
DataInputStream
DataOutputStream
직렬화
ObjectInputStream
ObjectOutputStream
IO 클래스
직렬화
직렬화 대상 객체
implemens Serializable
marker 인터페이스
serialVersionUID
직렬화 제외
transient
IO 클래스
ObjectInputStrem
Object readObject()
ObjectOutputStream
writeObject(Object obj)
IO 관련 클래스
File
URL
RandomAccessFile
고객관리 프로그램 III
직렬화를 이용
파일시스템에 저장