

IO(Input & Output)

1. File 클래스

- ❖ java.io 패키지에 존재하는 시스템에 있는 파일이나 디렉토리를 추상화한 클래스
- ❖ File 클래스를 이용하면 파일의 크기, 생성, 삭제 작업 및 마지막 수정날짜 등 다양한 정보를 알 수 있는 메소드를 제공
- ❖ 경로
 - ✓ 절대 경로: 루트로부터의 위치
 - Windows -> 루트드라이브:\디렉토리경로\파일이름
 - Web -> 프로토콜://자원경로
 - 그 이외의 경우 -> /루트디렉토리/디렉토리경로/파일이름
 - Windows 경우는 디렉토리 기호가 \ 그 이외의 경우는 /
 - ✓ 상대경로: 현재 위치로부터의 경로
 - ./: 현재 디렉토리
 - ../: 상위 디렉토리

1. File 클래스

❖ 생성자

생성자	설명
File(String pathname)	문자열 pathname을 가지고 경로를 생성하여 File 객체를 생성한다.
File(String parent, String child)	Parent와 child 문자열을 연결한 문자열로 경로를 생성하여 File 객체를 생성한다.
File(File parent, String child)	Parent의 파일 객체와 child 문자열로 경로를 생성하여 File 객체를 생성한다.

1. File 클래스

❖ 주요 메소드

반환형	메서드	설명
boolean	canRead()	파일을 읽을 수 있으면 true, 그렇지 않으면 false다.
	canWrite()	파일을 쓸 수 있으면 true, 그렇지 않으면 false다.
	createNewFile()	파일을 새로 생성하면 true, 그렇지 않으면 false다.
	delete()	파일을 지우면 true, 그렇지 않으면 false다.
	exists()	파일이나 디렉토리가 존재하면 true, 그렇지 않으면 false다.
String	getAbsolutePath()	파일의 절대 경로를 반환한다.
	getCanonicalPath()	파일의 정규 경로를 반환한다.
	getName()	파일명을 반환한다.
boolean	isDirectory()	디렉토리면 true, 그렇지 않으면 false다.
	isFile()	파일이면 true, 그렇지 않으면 false다.
long	lastModified()	1970년 1월 1일부터 현재까지의 시간을 밀리세컨드 초로 반환한다.
	length()	파일의 크기를 바이트로 반환한다.
String[]	list()	특정 디렉토리의 모든 파일과 자식 디렉토리를 스트링 배열로 반환한다.
boolean	mkdir()	디렉토리를 생성하면 true, 디렉토리가 있어서 생성하지 못하면 false다.
	renameTo(File dest)	dest 파일 객체로 이름을 바꾸면 true, 그렇지 않으면 false다.

예제-FileInfo.java(파일정보출력)

```
import java.io.File;
public class FileInfo {
    public static void main(String args[]) {
        String str = "";
        // 자신의 컴퓨터에 있는 파일의 경로로 변경해야 합니다.
        File file = new File("./src/chap13/FileInfo.java");

        if (file.exists()) {
            str += "파일명: " + file.getName() + "\n" + "파일의 크기 : "
                + file.length() + "\n" + "마지막 수정일 : " +
            file.lastModified()
                + "\n" + "부모 디렉토리 : " + file.getParent();
        } else {
            str = "해당파일이 존재하지 않습니다.";
        }
        System.out.println(str);
        System.out.println("=====");
        // 프로젝트 디렉토리의 모든 내용 출력
        File dir = new File("./");
        String[] strs = dir.list();
        for (int i = 0; i < strs.length; i++) {
            System.out.println(strs[i]);
        }
    }
}
```

```
<terminated> FileInfo [Java Application]
파일명: autoexec.bat
파일의 크기 : 24
마지막 수정일 : 1244670140125
부모 디렉토리 : c:\=====
.classpath
.project
.settings|
bin
src
```

예제 - DirectoryInfo.java

```
import java.io.File;
import java.util.ArrayList;
import java.util.List;

public class DirectoryInfo {
    static int totalFiles = 0;
    static int totalDirs = 0;

    public static void printFileList(File dir) {
        System.out.println(dir.getAbsolutePath() + " 디렉토리");
        File[] files = dir.listFiles();
        List<String> subDir = new ArrayList<String>();

        for (int i = 0; i < files.length; i++) {
            String filename = files[i].getName();
            if (files[i].isDirectory()) {
                filename = "[" + filename + "]";
                subDir.add(i + "");
            }
            System.out.println(filename);
        }
    }
}
```

예제-DirectoryInfo.java

```
int dirNum = subDir.size();
int fileNum = files.length - dirNum;
totalFiles += fileNum;
totalDirs += dirNum;
System.out.println(fileNum + "개의 파일, " + dirNum + "개의 디렉토리");
System.out.println();
for (int i = 0; i < subDir.size(); i++) {
    int index = Integer.parseInt((String) subDir.get(i));
    printFileList(files[index]);
}
}

public static void main(String[] args) {
    File dir = new File("./");

    if (!dir.exists() || !dir.isDirectory()) {
        System.out.println("유효하지 않은 디렉토리입니다.");
        System.exit(0);
    }
    printFileList(dir);
    System.out.println();
    System.out.println("총 " + totalFiles + "개의 파일");
    System.out.println("총 " + totalDirs + "개의 디렉토리");
}
}
```

예제-FileDelete.java(파일삭제)

```
import java.io.*;  
  
public class FileDelete {  
    public static void delete(String filename) {  
        // 파일이름을 나타내기 위해, File 객체를 생성  
        File f = new File(filename);  
        // 파일이나 디렉토리가 존재하는지와 쓰기 방지가 되어 있는지를 확인  
        if (!f.exists()) {  
            System.out.println("Delete : 파일을 찾을 수 없습니다. : "  
                + filename);  
            System.exit(0);  
        }  
        if (!f.canWrite()) {  
            System.out.println("Delete : 쓰기 방지가 되어서 삭제할 수  
없습니다. : " + filename);  
            System.exit(0);  
        }  
        // 디렉토리이면, 비어있는지 확인  
        if (f.isDirectory()) {  
            String[] files = f.list();  
            if (files.length > 0)  
                System.out  
                    .println("Delete : 디렉토리가 비어 있지 않습니다. : " + filename);  
            System.exit(0);  
        }  
    }  
}
```

예제-FileDelete.java(파일삭제)

```
// 모든 검사를 통과했으면 파일 삭제
boolean success = f.delete();
if (!success)
    System.out.println("Delete : 파일 삭제 실패");
else
    System.out.println("Delete : 파일 삭제 성공");
}

public static void main(String[] args) {
    delete("./pro.txt");
}

}
```

1. File 클래스

- ❖ JDK 1.7에서 파일을 조작하기 위해 사용하는 클래스로 java.nio.file.Path 를 추가
- ❖ 기존 File 클래스의 문제점
 - ✓ 파일의 메타 데이터와 심볼릭 링크를 취급할 수 없는 제약
 - ✓ 디렉토리 밑의 파일의 생성/갱신/삭제를 감시할 수 없음
- ❖ Path 클래스에는 위의 문제 해결을 위한 메소드 뿐 아니라 파일 경로를 조작하는 편리하고 강력한 메소드가 존재하며 Path 객체와 File 객체 사이의 변환도 가능
- ❖ Path 클래스를 이용해서 코드를 작성하고 필요에 따라 File 클래스의 객체로 변환해서 사용하는 것을 권장

1. File 클래스

❖ Path 클래스의 생성

- ✓ Paths.get("파일 경로")
- ✓ URI uri = URI.create("file:///파일 경로");
Paths.get(uri);

❖ Path 클래스의 주요 메소드

- ✓ toString: 경로를 문자열로 리턴
- ✓ toAbsolutePath: 절대 경로를 리턴
- ✓ toFile: File 객체를 리턴
- ✓ toUri: Uri를 리턴
- ✓ getParent: 부모 경로를 리턴
- ✓ register: 파일 감시 서비스를 등록

1. File 클래스

- ❖ jdk1.7 이상에서의 파일 복사
 - ✓ 이전에는 스트림을 이용해서 파일의 내용을 읽어서 다시 기록
 - ✓ 1.7 이상에서는 Files 클래스의 copy 메소드를 이용하면 쉽게 복사
- ❖ 파일 삭제는 Files.delete 메소드 이용
- ❖ 파일 생성은 Files.createFile 메소드 이용 – 이전 API에서는 File 인스턴스를 생성하고 createNewFile()을 호출
- ❖ 디렉토리 생성은 Files.createDirectory 메소드 이용 – 이전 API에서는 File 인스턴스를 생성하고 mkdir()을 호출

1. File 클래스

```
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;

public class SevenFileCopy {

    public static void main(String[] args) {
        Path from = Paths.get("./pro.xml");
        Path to = Paths.get("./copy.xml");

        try {
            Files.copy(from, to);
            System.out.println("파일 복사 성공");
        }catch(Exception e) {
            System.out.println(e.getMessage());
        }
    }
}
```

1. File 클래스

- ❖ 메모리 사용량이 많은 데이터를 일시적으로 파일로 출력하는 목적으로 프로그램상에서 임시 파일을 생성할 수 있음
- ❖ 1.7 이전의 API
 - ✓ `File.createTempFile(String pre, String ext, File directory)`
 - ✓ `pre`는 접두어
 - ✓ `ext` 대신에 `null`을 대입하면 확장자가 `.tmp`
 - ✓ `directory`를 생략할 수 있는데 이 경우에는 OS의 임시 파일 디렉토리 사용
- ❖ 1.7 API
 - ✓ `Files.createTempFile(Path path, String pre, String ext)`
 - ✓ 임시 디렉토리를 만들려면 `createTempDirectory`

1. File 클래스

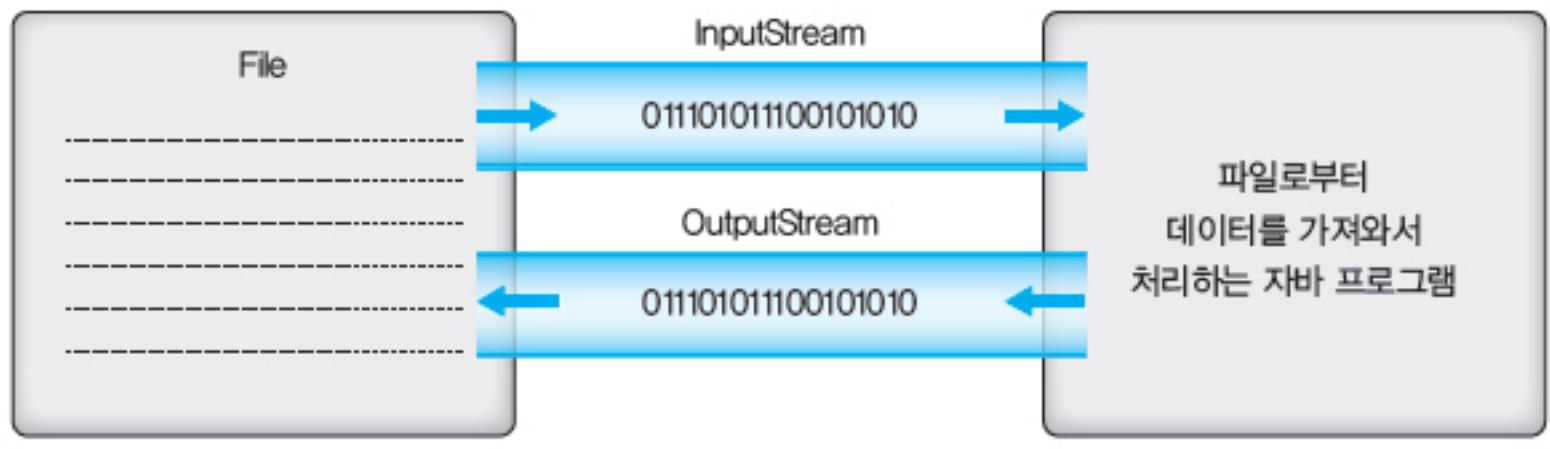
```
import java.io.File;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;

public class TempFileUse {

    public static void main(String[] args) {
        File tempDir = new File("./");
        Path path = Paths.get("./");
        try {
            File.createTempFile("pre", ".tmp", tempDir);
            System.out.println("1.7 이전 API 활용");
            Files.createTempFile(path, "pre", ".tmp");
            Thread.sleep(10000);
            System.out.println("1.7 API 활용");
        } catch(Exception e) {
            System.out.println(e.getMessage());
        }
    }
}
```

2. Stream

- ❖ 입출력을 처리하는 중간 매개체
- ❖ 데이터를 운반하는데 사용하는 연결 통로
- ❖ JVM 외부에 있는 매체(파일, 네트워크 소켓 통신, 데이터베이스 등)에 존재하는 데이터 소스에 접근하기 위해서는 ‘스트림’이라는 매개체가 필요
- ❖ 스트림을 이용하면 자바 프로그램에서는 어떤 데이터 소스에서 데이터를 받아오더라도 공통된 방법을 사용할 수 있기 때문에 보다 편리하게 프로그래밍 가능
- ❖ 분류
 - ✓ 입력 스트림 : 데이터를 가져오는데 사용하는 스트림
 - ✓ 출력 스트림 : 데이터를 출력하는데 사용하는 스트림



2. Stream

❖ 스트림의 특징

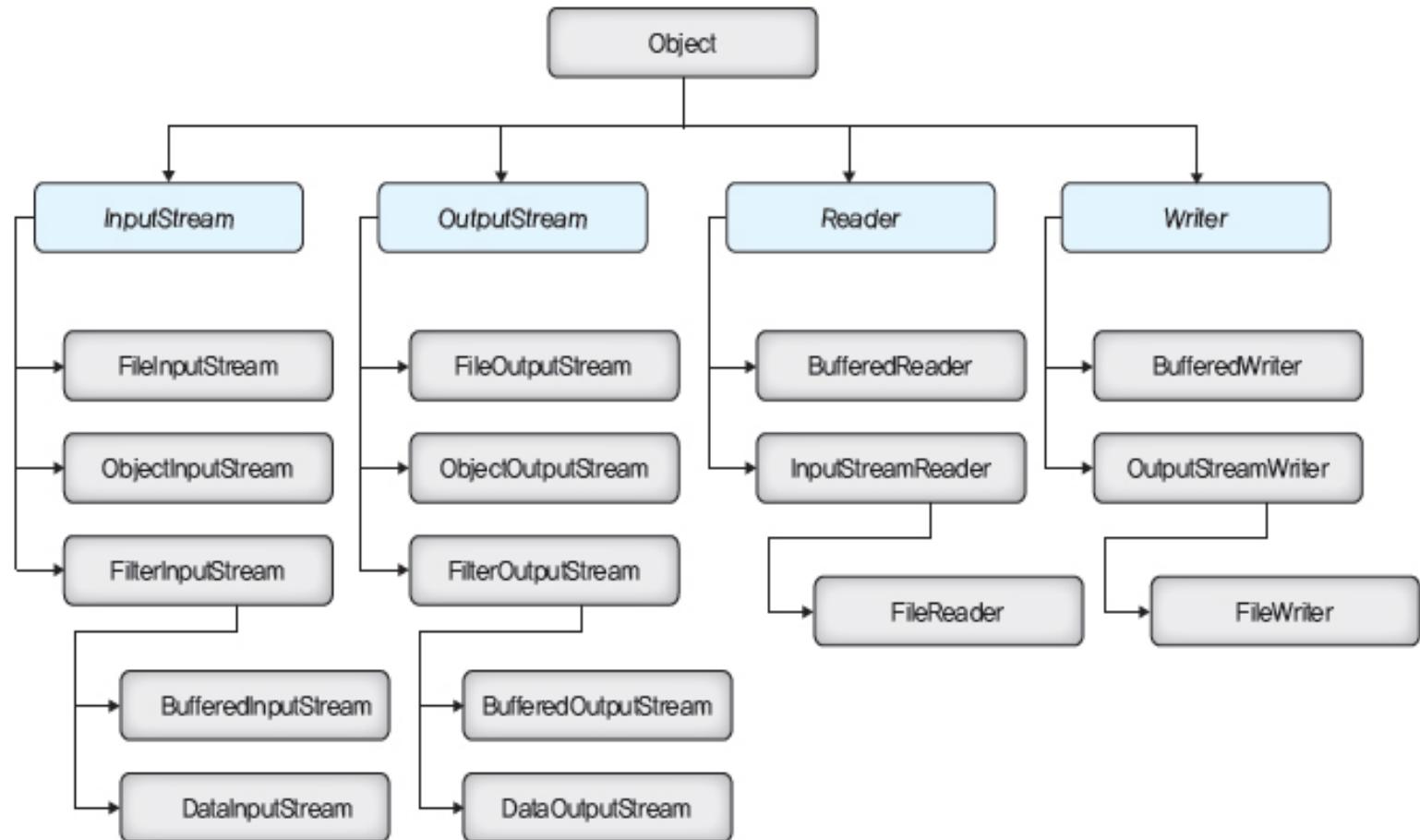
- ✓ FIFO 구조: 순차적 접근만 허용 - 앞뒤 순서가 바뀌는 일은 발생하지 않는 장점
- ✓ 단방향성: 특정 매체에 데이터를 읽고 쓰려면 읽기 전용 스트림과 쓰기 전용 스트림이 필요
- ✓ 스트림이 동작하는 동안에는 해당 프로그램(스트림을 담당하는 스레드)이 블록(blocking)되면서 자연 상태로 빠짐
- ✓ 스트림 동작(혹은 데이터 전송)이 끝나야 블록이 해제되면서 동작을 다시 시작
- ✓ 유연한 구조
- ✓ 여러 스트림 객체를 조합해서 사용할 수 있기 때문에 다양한 데이터 처리가 가능

❖ 스트림 분류

- ✓ 입력 스트림 : 자바 프로그램을 기준으로 데이터가 프로그램으로 유입되는 스트림
- ✓ 출력 스트림 : 자바 프로그램을 기준으로 데이터가 프로그램 외부로 나가는 스트림
- ✓ 문자 스트림 : 데이터를 문자 단위로 읽고 쓰는 스트림
- ✓ 바이트 스트림 : byte 단위 - 8bit 단위로 데이터를 읽고 쓰기 위한 스트림

2. Stream

- ❖ 입출력 처리를 위한 매개체 – 스트림



3. ByteStream

- ❖ InputStream은 파일이나 화면 등의 외부 장치로부터 바이트 단위로 입력을 수행하는 데 필요한 메소드를 정의한 추상 클래스
- ❖ OutputStream은 파일이나 화면 등의 외부 장치로 바이트 단위로 데이터를 내보내는 경우 필요한 메소드를 정의한 추상 클래스
- ❖ 자바 프로그램은 파일이나 화면 및 소켓 객체를 생성하고 생성된 객체와 스트림을 연결한 후 데이터를 입출력
- ❖ 특징
 - ✓ InputStream과 OutputStream은 추상 클래스
 - ✓ 데이터를 가져오는 read() 메소드와 데이터를 보내는 write() 메소드를 제공
 - ✓ IOException 예외를 Throws
 - ✓ 작업을 수행하고 나면 close() 메소드로 스트림을 닫아야 함

3. ByteStream

❖ InputStream의 메소드

- ✓ int available(): 읽을 수 있는 바이트 수 리턴
- ✓ void close(): 스트림 닫기
- ✓ abstract int read(): 입력 스트림로부터 다음의 바이트 데이터를 읽어서 리턴하는데 읽은 데이터가 없으면 -1을 리턴
- ✓ int read(byte[] b): 입력 스트림로부터 읽어들여 그것을 버퍼 배열 b에 추가하는 데 읽는데 성공하면 읽은 위치를 리턴하고 읽지 못했으면 -1을 리턴
- ✓ int read(byte[] b, int off, int len): 입력 스트림로부터 off 부터 len 바이트 만큼의 데이터를 바이트 배열에 읽어옴
- ✓ long skip (long n): 입력 스트림으로부터의 데이터를 n 바이트만 스킵 해 그 범위의 데이터를 파기

3. ByteStream

❖ OutputStream의 메소드

- ✓ void close (): 스트림 닫기
- ✓ void write(byte[] b): b를 출력 스트림에 기록
- ✓ void write(byte[] b, int off, int len): 바이트 배열 b를 off부터 len 만큼 기록
- ✓ void write(int b): b를 문자로 변환해서 출력 스트림에 기록
- ✓ void flush(): 버퍼에 남아있는 데이터를 모두 출력시키는 메소드로 출력이 완료되면 이 메소드를 호출해서 버퍼의 데이터를 전부 출력해야 함
- ✓ 실제 데이터를 전송할 때는 write 메소드를 이용해서 스트림에 기록한 후 flush 메소드를 호출해서 전송해야 함

3. ByteStream

❖ InputStream, OutputStream의 하위 클래스들

- ✓ FileInputStream/FileOutputStream: 파일로부터 바이트 단위로 데이터를 입출력하기 위한 스트림
- ✓ ByteArrayInputStream/ByteArrayOutputStream: 바이트 배열(메모리)을 입출력하기 위한 스트림
- ✓ PipedInputStream/PipedOutputStream: 프로세스 간에 바이트 단위의 데이터를 입출력하기 위한 스트림
- ✓ AudioInputStream/AudioOutputStream: 오디오에 입출력하기 위한 스트림

❖ 위의 스트림을 기반으로 생성되는 보조 스트림

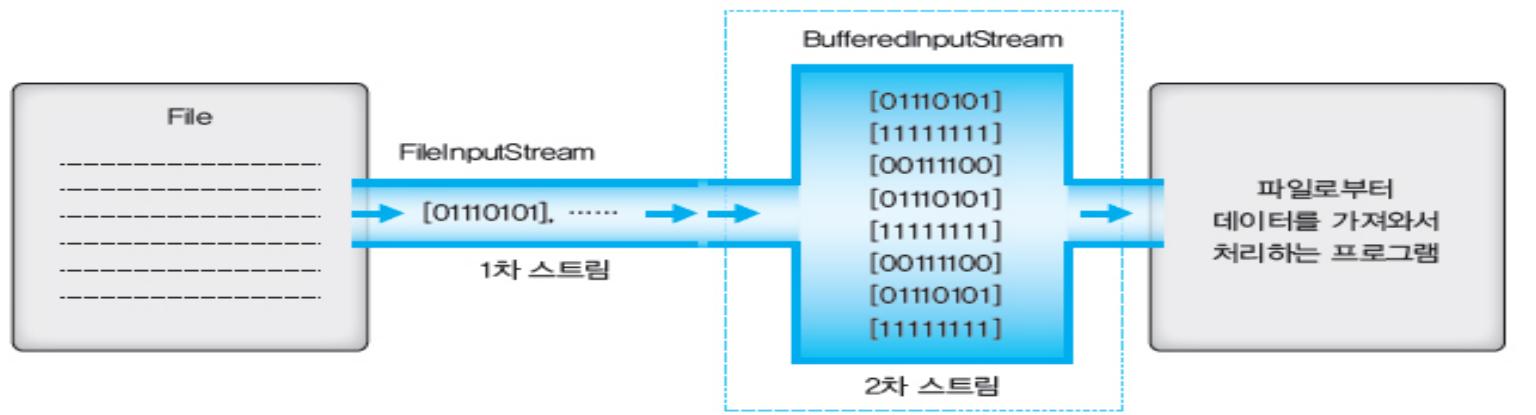
- ✓ FilterInputStream/FilterOutputStream: 필터를 이용한 입출력을 지원하는 추상클래스
- ✓ BufferedInputStream/BufferedOutputStream: 버퍼를 이용한 입출력
- ✓ DataInputStream/DataOutputStream: 기본형 데이터 입출력
- ✓ SequenceInputStream: 2개의 스트림을 연결하기 위한 스트림
- ✓ LineNumberInputStream: 읽어온 데이터의 라인번호를 카운트
- ✓ ObjectInputStream/ObjectOutputStream: 객체 단위의 입출력
- ✓ PushbackInputStream: 버퍼에서 읽어온 데이터를 되돌리는 스트림
- ✓ PrintStream: 버퍼를 이용하여 출력을 위한 메소드를 조금 더 소유(print, println, printf)

3. ByteStream

❖ BufferedInputStream/BufferedOutputStream

- ✓ FilterInputStream, FilterOutputStream 클래스를 상속받은 하위 클래스
- ✓ 내부에 버퍼를 저장하고 있어서 IO 프로그램의 단점인 자연성을 최대한 줄이고자 생성
- ✓ 2차 스트림 클래스 : 데이터 소스에서 직접 데이터를 입력받는 것보다는 다른 입력 스트림을 매개변수로 받은 후 버퍼 스트림으로 변환

```
사용예: FileInputStream fis = new FileInputStream("D:\\test.txt");
        BufferedInputStream bis = new BufferedInputStream(fis);
```



ByteArrayStreamMain.java

```
import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.IOException;

public class ByteArrayOutputMain {

    public static void main(String[] args) {
        String insa = "Hello";
        // 문자열의 바이트 크기만큼 배열 생성
        byte[] outSrc = new byte[insa.getBytes().length];
        // 문자열을 바이트 배열로 변환해서 읽을 수 있는 스트림으로 변환
        ByteArrayInputStream bais = new ByteArrayInputStream(insa.getBytes());
        // 바이트 배열에 데이터를 기록할 수 있는 스트림 생성
        ByteArrayOutputStream baos = new ByteArrayOutputStream();

        int idx = 0;
        try {
            while (true) {
                // bais의 소스에서 outSrc의 크기만큼을 읽어서 outSrc에
                저장하고 읽은 개수를 idx에 저장
                idx = bais.read(outSrc);
                // 읽은 데이터가 없으면 읽기 중단
                if (idx == -1)
                    break;
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

ByteArrayStreamMain.java

```
// 읽은 데이터를 출력
for (byte b : outSrc) {
    System.out.println((char) b);
}
} catch (Exception e) {
    System.out.println(e.getMessage());
} finally {
    // 스트림 닫기
    try {
        bais.close();
        baos.close();
    } catch (IOException e) {
        System.out.println(e.getMessage());
    }
}
```

}

3. ByteStream

❖ FileInputStream

- ✓ FileInputStream은 시스템에 있는 파일을 읽을 수 있는 기능을 제공
- ✓ 파일을 읽을 때는 파일의 경로 또는 파일 객체를 생성자의 매개 변수로 설정
- ✓ 파일이 존재하지 않으면 FileNotFoundException을 발생
- ✓ 생성자
 - FileInputStream(String name)
 - FileInputStream(File file)

❖ FileOutputStream

- ✓ FileOutputStream은 시스템에 있는 파일에 기록할 수 있는 기능을 제공
- ✓ 파일이 존재하지 않으면 파일을 생성하고 파일이 있으면 수정
- ✓ 생성자 – append는 존재하는 경우 추가할 것인지의 여부
 - FileOutputStream(String name)
 - FileOutputStream(String name, boolean append)
 - FileOutputStream(File file)
 - FileOutputStream(File file, boolean append)

FileStreamMain.java

```
import java.io.*;  
  
public class FileStreamMain {  
    public static void main(String[] args) {  
        // 현재 디렉토리의 byte.txt 파일에 바이트 단위로 데이터를 기록할 수 있는  
        // 파일출력스트림 생성 - 이렇게 생성하면 close 하지 않아도 됨  
        try (FileOutputStream fos = new FileOutputStream("./byte.txt")) {  
            // 기록할 바이트 배열 생성  
            byte[] b = { 'H', 'e', 'l', 'l', 'o' };  
            // 파일출력스트림에 b의 내용을 바이트 단위로 기록  
            fos.write(b);  
            // 버퍼의 내용을 전부 출력  
            fos.flush();  
            // 스트림 닫기  
            System.out.println("파일 쓰기에 성공했습니다.");  
        } catch (FileNotFoundException e) {  
            System.out.println("파일이 생성되지 않습니다.₩n 경로를  
확인하세요");  
        } catch (Exception e) {  
            System.out.println("파일에 쓰기를 실패 했습니다.₩n 내용을  
확인하세요");  
        }  
    }  
}
```

FileStreamMain.java

```
// 현재 디렉토리의 byte.txt 파일에서 바이트 단위로 데이터를 읽을 수 있는
// 파일입력스트림 생성 - 이렇게 생성하면 close 하지 않아도 됨
try (FileInputStream fis = new FileInputStream("./byte.txt")) {
    // 파일의 크기만큼의 바이트 배열을 생성
    byte[] b = new byte[(fis.available())];
    // 데이터 읽기
    fis.read(b);
    // 읽은 데이터를 출력
    for (byte temp : b) {
        System.out.print((char) temp + ",");
    }
} catch (FileNotFoundException e) {
    System.out.println("파일이 존재하지 않습니다.\\n 경로를
확인하세요");
} catch (Exception e) {
    System.out.println("파일에서 읽기를 실패 했습니다.\\n 내용을
확인하세요");
}
}
```

3. ByteStream

- ❖ BufferedInputStream/BufferedOutputStream: 바이트 입출력을 위한 클래스
- ❖ 버퍼는 입출력을 향상 시키기 위하여 사용하는 방법
- ❖ 버퍼를 사용하지 않는 클래스는 내부적으로 하부 플랫폼의 네이티브 메소드를 호출하는데 호출이 많아 질수록 I/O 효율이 저하
- ❖ 버퍼를 사용하여 데이터를 모았다가 한꺼번에 처리하면 네이티브 메소드 호출 횟수가 줄어들어서 효율이 향상
- ❖ 생성자는 InputStream과 OutputStream 객체를 매개변수로 받아서 생성

BufferStreamMain.java

```
import java.io.*;

public class BufferStreamMain {
    public static void main(String[] args) {
        // buffer.txt 파일에 데이터를 바이트 단위로 기록할 수 있는 스트림 생성
        try (FileOutputStream fos = new FileOutputStream("./buffer.txt");
             BufferedOutputStream bos = new
BufferedOutputStream(fos)) {
            byte[] b = { 'H', 'e', 'l', 'l', 'o' };
            bos.write(b);
            bos.flush();
            System.out.println("파일 쓰기에 성공했습니다.");
        } catch (FileNotFoundException e) {
            System.out.println("파일이 생성되지 않습니다.₩n 경로를
확인하세요");
        } catch (Exception e) {
            System.out.println("파일에 쓰기를 실패 했습니다.₩n 내용을
확인하세요");
        }
    }
}
```

BufferStreamMain.java

```
// buffer.txt 파일에 데이터를 바이트 단위로 읽을 수 있는 스트림 생성
try (FileInputStream fis = new FileInputStream("./buffer.txt"));
    BufferedInputStream bis = new BufferedInputStream(fis);
{
    byte[] b = new byte[(bis.available())];
    bis.read(b);
    for (byte temp : b) {
        System.out.print((char) temp);
    }
} catch (FileNotFoundException e) {
    System.out.println("파일이 존재하지 않습니다.\\n 경로를
확인하세요");
} catch (Exception e) {
    System.out.println("파일에서 읽기를 실패 했습니다.\\n 내용을
확인하세요");
}
}
```

3. ByteStream

- ❖ DataInputStream & DataOutputStream은 primitive type 데이터를 입출력하기 위한 클래스

- ✓ DataInputStream 클래스

- 생성자

- DataInputStream (InputStream in)

- 메소드

- int read (byte[] b): 바이트 단위로 읽어서 b에 저장
 - int read (byte[] b, int off, int len)
 - boolean readBoolean ()
 - byte readByte ()
 - char readChar ()
 - double readDouble ()
 - float readFloat ()
 - int readInt ()
 - long readLong ()
 - short readShort ()
 - String readUTF ()
 - static String readUTF (DataInput in)
 - int skipBytes (int n)

3. ByteStream

- ✓ DataOutputStream 클래스
 - 생성자
 - DataOutputStream (OutputStream in)
 - 메소드
 - void flush ()
 - int size ()
 - void write (byte[] b, int off, int len)
 - void write (int b)
 - void writeBoolean (boolean v)
 - void writeByte (int v)
 - void writeBytes (String s)
 - void writeChar (int v)
 - void writeChars (String s)
 - void writeDouble (double v)
 - void writeFloat (float v)
 - void writelnt (int v)
 - void writeLong (long v)
 - void writeShort (int v)
 - void writeUTF (String str)

DataOMain.java

```
import java.io.*;
public class DataOMain {
    public static void main(String[] args) {
        try (DataOutputStream dataout = new DataOutputStream(new
FileOutputStream("./test.txt"))) {
            dataout.writeInt(123);
            dataout.writeChar('k');
            dataout.writeDouble(123.4);
        } catch (Exception e) {
            System.out.println("파일에 기록하는 것을 실패했습니다.");
            System.err.println(e.getMessage());
            System.exit(0);
        }
        try (DataInputStream datain = new DataInputStream(new
FileInputStream("./test.txt"))) {
            System.out.println(datain.readInt());
            System.out.println(datain.readChar());
            System.out.println(datain.readDouble());
        } catch (Exception e) {
            System.out.println("파일에서 읽기에 실패했습니다.");
            System.err.println(e.getMessage());
            System.exit(0);
        }
    }
}
```

123

k

123.4

3. ByteStream

- ❖ SequenceInputStream: 여러 개의 입력 스트림을 연속적으로 연결해서 하나의 스트림으로부터 데이터를 읽는 것과 같이 처리하기 위한 스트림
- ❖ 큰 파일을 여러 개의 작은 파일로 나누어 두었다가 하나의 파일로 합치는 것과 같은 작업을 수행할 때 사용할 수 있는 클래스
- ❖ 생성자
 - ✓ SequenceInputStream(Enumeration e)
 - ✓ SequenceInputStream(InputStream s1, InputStream s2)

SequenceMain.java

여러 개의 스트림을 합쳐서 읽기

```
import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.io.SequenceInputStream;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

public class SequenceMain {
    public static void main(String[] args) {
        byte[] ar1 = { 0, 1, 2 };
        byte[] ar2 = { 3, 4, 5 };

        byte[] outSrc = null;

        // 2개의 스트림을 리스트에 저장
        List <ByteArrayInputStream> list = new ArrayList<ByteArrayInputStream>();
        list.add(new ByteArrayInputStream(ar1));
        list.add(new ByteArrayInputStream(ar2));

        Input Source1 :[0, 1, 2]
        Input Source2 :[3, 4, 5]
        Output Source :[0, 1, 2, 3, 4, 5]
```

SequenceMain.java

```
// 리스트의 요소를 이용해서 연결된 스트림 객체 생성  
SequenceInputStream input = new SequenceInputStream(list.get(0),  
list.get(1));  
  
// 바이트 배열 출력 스트림 생성  
ByteArrayOutputStream output = new ByteArrayOutputStream();  
  
int data = 0;  
  
// 연결된 스트림 객체의 모든 내용을 읽어서 출력 스트림에 기록  
try {  
    while ((data = input.read()) != -1) {  
        output.write(data);  
    }  
} catch (IOException e) {  
}  
  
// 출력 스트림에 기록된 내용을 바이트 배열에 저장  
outSrc = output.toByteArray();  
// 데이터 출력  
System.out.println("Input Source1 :" + Arrays.toString(ar1));  
System.out.println("Input Source2 :" + Arrays.toString(ar2));  
System.out.println("Output Source :" + Arrays.toString(outSrc));  
}  
}
```

3. ByteStream

- ❖ PrintStream: 데이터를 기반 스트림에 다양한 형태로 출력할 수 있는 print, println, printf와 같은 메소드를 overriding해서 제공하는 클래스
- ❖ 데이터를 문자로 출력하는 것이기 때문에 문자 기반 스트림의 역할을 수행
- ❖ 향상된 기능의 PrintWriter가 추가되었지만 System.out이 PrintStream으로 만들어져 있어서 아직도 사용
- ❖ 생성자
 - ✓ PrintStream(File f)
 - ✓ PrintStream(File f, String csn)
 - ✓ PrintStream(OutputStream out)
 - ✓ PrintStream(OutputStream out, boolean autoFlush)
 - ✓ PrintStream(OutputStream out, boolean autoFlush, String encoding)
 - ✓ PrintStream(String filename)
 - ✓ PrintStream(String filename, String csn)
 - ✓ autoFlush 옵션은 기본값은 false 인데 이 값을 true로 설정하면 println이 호출되거나 개행 문자가 출력될 때 자동 flush
 - ✓ encoding 과 csn은 문자열 인코딩 방식

3. ByteStream

- ❖ JDK 1.7 이후에서는 Path 클래스와 Files 클래스를 사용하여 스트림을 생성
- ❖ try-with-resources 구문을 사용하여 스트림을 닫는 처리를 생략 가능
- ❖ try 키워드 직후의 괄호 안에 선언한 자원은 try문이 정상적으로 종료하거나 예외를 발생하는 등 갑자기 종료했는지에 관계없이 닫힘
- ❖ 위와 같은 형태로 스트림을 생성하면 finally 절이 불필요

SevenBinaryFile.java

```
import java.io.InputStream;
import java.io.OutputStream;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;

public class SevenBinaryFile {
    public static void main(String[] args) {
        Path path = Paths.get("./seven.txt");
        try(OutputStream os = Files.newOutputStream(path)){
            os.write(97);
            os.flush();
        }catch(Exception e) {
            System.out.println(e.getMessage());
        }
        try(InputStream is = Files.newInputStream(path)){
            System.out.print((char)is.read());
        }catch(Exception e) {
            System.out.println(e.getMessage());
        }
    }
}
```

4.Character Stream

- ❖ java.io 패키지에서 Reader와 Writer 클래스를 제공하는데 2바이트 단위로 입출력 할 수 있는 문자 기반 스트림의 최상위 클래스
- ❖ 바이트 스트림은 1바이트 단위로 입출력하기 때문에 영문자로 구성된 파일, 동영상 파일, 음악 파일의 입출력 등에는 적합하지만 문자는 2바이트로 구성되기 때문에 문자 단위의 입출력에는 부적합해서 이를 보완하기 위해서 만든 스트림
- ❖ Reader의 메소드
 - ✓ void close()
 - ✓ int read()
 - ✓ int read(char [] buf): buf 만큼 읽어서 읽은 문자 수 리턴
 - ✓ int read(char[] buf, int off, int len)
 - ✓ int read(CharBuffer target)
 - ✓ boolean ready()
 - ✓ long skip(long n)
- ❖ Writer의 메소드
 - ✓ void close (): 스트림 닫기
 - ✓ void flush(): 출력 스트림을 비우는데 이 때 데이터를 출력 스트림으로 전송
 - ✓ void write(String str): str를 출력 스트림에 기록
 - ✓ void write(String str, int off, int len): str에서 off부터 len 만큼 기록

4.Character Stream

❖ Reader, Writer의 하위 클래스들

- ✓ FileReader/FileWriter: 파일로부터 문자 단위로 입출력하기 위한 스트림
- ✓ CharArrayReader/CharArrayWriter: char 배열(메모리)을 입출력하기 위한 스트림
- ✓ StringReader/StringWriter: char 배열(메모리)을 입출력하기 위한 스트림
- ✓ PipedReader/PipedWriter: 프로세스 간에 문자 단위의 데이터를 입출력하기 위한 스트림

❖ 위의 스트림을 생성하고 그 스트림을 기반으로 생성되는 보조 스트림

- ✓ FilterReader/FilterWriter: 필터를 이용한 입출력
- ✓ BufferedReader/BufferedWriter: 버퍼를 이용한 입출력
- ✓ DataReader/DataWriter: 기본형 데이터 입출력
- ✓ SequenceReader: 2개의 스트림을 연결하기 위한 스트림
- ✓ LineNumberReader: 읽어온 데이터의 라인번호를 카운트
- ✓ ObjectReader/ObjectWriter: 객체 단위의 입출력
- ✓ PushbackReader: 버퍼에서 읽어온 데이터를 되돌리는 스트림
- ✓ PrintStream: 버퍼를 이용하여 출력을 위한 메소드를 더 소유(print, println, printf)

예제 – StringReaderWriter.java

```
import java.io.IOException;
import java.io.StringReader;
import java.io.StringWriter;
public class StringReaderWriter {
    public static void main(String[] args) {
        String inputData = "안녕하세요 반갑습니다.";
        StringReader input = new StringReader(inputData);
        StringWriter output = new StringWriter();

        int data = 0;

        try {
            while (true) {
                data = input.read();
                if (data == -1)
                    break;
                output.write(data);
            }
        } catch (IOException e) {
            System.out.println(e.getMessage());
        }
        System.out.println("Input Data :" + inputData);
        System.out.println("Output Data :" + output.toString());
    }
}
```

Input Data :안녕하세요 반갑습니다.
Output Data :안녕하세요 반갑습니다.

4.Character Stream

- ❖ InputStreamReader/OutputStreamWriter: 바이트 기반의 스트림을 문자 기반의 스트림으로 연결해주는 클래스
- ❖ 바이트 기반 스트림의 데이터를 지정된 인코딩의 문자열로 변환하는 작업을 수행할 수 있음
- ❖ 생성자
 - ✓ InputStreamReader
 - InputStreamReader(InputStream in)
 - InputStreamReader(InputStream in, Charset cs)
 - InputStreamReader(InputStream in, CharsetDecoder dec)
 - InputStreamReader(InputStream in, String charsetName)
 - ✓ OutputStreamWriter
 - OutputStreamWriter(OutputStream out)
 - OutputStreamWriter(OutputStream out, Charset cs)
 - OutputStreamWriter(OutputStream out, CharsetEncoder enc)
 - OutputStreamWriter(OutputStream out, String charsetName)

4.Character Stream

❖ FileReader

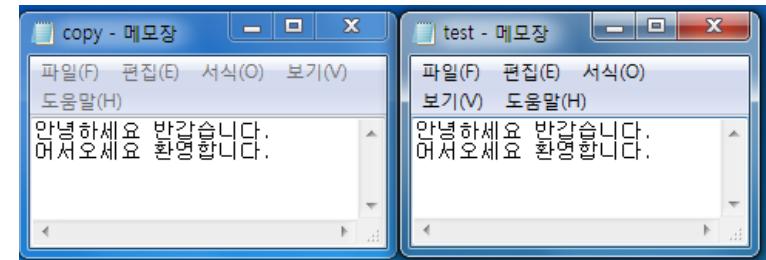
- ✓ FileReader 클래스는 시스템에 있는 파일을 읽을 수 있는 기능을 제공
- ✓ 파일을 읽을 때는 파일의 경로 및 File 객체를 생성자의 매개변수로 지정할 수 있으며 파일이 존재하지 않으면 FileNotFoundException 예외가 발생
- ✓ 문자 스트림으로 한 문자를 읽기 때문에 화면에 출력하더라도 한글이 깨지는 현상이 일어나지 않음

❖ FileWriter

- ✓ FileWriter 클래스는 파일에 문자 단위로 출력할 때 사용하는 클래스
- ✓ FileWriter 클래스의 생성자는 파일의 경로 또는 File 객체를 이용하여 객체를 생성
- ✓ 경로가 존재하지 않으면 IOException를 발생

예제-FileCopy.java(파일 쓰기 및 읽기)

```
import java.io.*;
public class FileCopy {
    public static void main(String args[]) {
        int r=-1;
        FileReader reader = null;
        FileWriter writer = null;
        try {
            reader = new FileReader(new File("./test.txt"));
            writer = new FileWriter(new File("./copy.txt"));
            while ((r = reader.read()) != -1){
                writer.write(r);
            }
            System.out.println("파일쓰기에 성공했습니다. \n");
            System.out.println("두 개의 파일을 열어서 확인해 보시기
바랍니다.");
            reader.close();
            writer.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```



4.Character Stream

- ❖ BufferedReader/BufferedWriter: 버퍼를 이용한 문자 입출력 클래스
 - ✓ BufferedReader 클래스에 줄 단위로 읽어오는 String readLine() 메소드가 존재
 - ✓ BufferedWriter 클래스에 줄 변경을 추가해주는 void newLine() 메소드가 존재

BufferedReaderMain.java

```
import java.io.*;
public class BufferedReaderMain {
    public static void main(String[] args) {
        String str = ""; // 입력 받은 문자열 저장
        // BufferedReader타입의 변수 reader선언
        BufferedReader reader = null;
        try {
            while (true) {
                System.out.print("문자열을 입력(중단은 끝):");
                // BufferedReader객체 reader생성
                reader = new BufferedReader(new
InputStreamReader(System.in));
                // reader객체를 사용해서 라인 단위로 입력
                str = reader.readLine();
                // 입력한 문자열이 "끝"이면 루프 탈출
                if (str.equals("끝"))
                    break;
                // 입력한 문자열 출력
                System.out.println(str);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
        System.out.println("종료");
    }
}
```

```
문자열을 입력(종단은 끝)
Hello
Hello
문자열을 입력(종단은 끝)
반갑습니다
반갑습니다
문자열을 입력(종단은 끝)
Hello
Hello
```

웹 서버 로그 파일

127.0.0.1 -- [30/May/2017:16:59:20 +0900] "GET / HTTP/1.1" 404 994
0:0:0:0:0:0:1 -- [30/May/2017:16:59:23 +0900] "GET /library/ HTTP/1.1" 200 12223

❖ 공백을 기준으로 분할 했을 때

- 1 : 접속한 클라이언트의 IP 주소, 혹은 도메인
- 2 : REMOTE_IDENT (RFC 931 identification (아이덴티피케이션:동일함 확인)
 - 서버가 RFC 931 을 지원하는 경우 이 환경 변수에 클라이언트 시스템에서 CGI프로그램을 실행시킨 사용자 이름이 저장
- 3 : 사용자이름(.htaccess .htpasswd 에 정의된 사용자 id)
- 4 : 클라이언트(사용자 브라우저)의 접속시간정보 (httpd 접속시간)(날짜)
- 5 : 시간
- 6 : 클라이언트(사용자 브라우저) 요청종류 (GET , POST)
- 7 : 클라이언트가 요청한 홈페이지 URL 주소 (요청한 자료 & 자료위치)
- 8 : 프로토콜 버전
- 9 : 상태코드 (예. 200 정상처리)
- 10 : 전송데이터 크기

상태코드 일부 304 은 - (하이픈) 로 표시

hits – 모든 상태 코드 포함

files – 상태코드 200번만

로그 분석

- ❖ log.txt 파일을 프로젝트 디렉토리에 복사 한 후 실습

로그 분석(LogMain1)

❖ Apache Web Server에서 유효한 페이지 접근 횟수 출력(상태 코드가 200번인 경우의 수)

유효한 접속 횟수:327

```
import java.io.*;  
  
public class LogMain1 {  
    public static void main(String[] args) {  
        int cnt = 0;  
        try (BufferedReader br = new BufferedReader(new FileReader("./log.txt")))  
        {  
            while (true) {  
                String x = br.readLine();  
                if (x == null)  
                    break;  
                String[] ar = x.split(" ");  
                if (ar[8].equals("200")) {  
                    cnt = cnt + 1;  
                }  
            }  
        } catch (IOException e) {  
            e.printStackTrace();  
        }  
        System.out.println("유효한 접속 횟수:" + cnt);  
    }  
}
```

로그 분석(LogMain2)

- ❖ Apache Web Server에 접속한 ip 수 출력
접속한 IP 개수:99

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

public class LogMain2 {
    public static void main(String[] args) {
        Set<String> set = new HashSet<String>();
        try (BufferedReader br = new BufferedReader(new FileReader("./log.txt")))
    {
        while (true) {
            String x = br.readLine();
            if (x == null)
                break;
            String[] ar = x.split(" ");
            set.add(ar[0]);
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
    System.out.println("접속한 IP 개수:" + set.size());
}
```

로그 분석(LogMain3)

- ❖ Apache Web Server에서 전체 트래픽 출력

For input string: ""_""

For input string: ""_""

일일 트래픽: 29985511바이트

로그 분석(LogMain3)

```
import java.io.*;
public class LogMain3 {
    public static void main(String[] args) {
        int sum = 0;
        try (BufferedReader br = new BufferedReader(new FileReader("./log.txt")))
{
            while (true) {
                try {
                    String x = br.readLine();
                    if (x == null)
                        break;
                    String[] ar = x.split(" ");
                    if (!ar[9].equals("-"))
                        sum = sum + Integer.parseInt(ar[9]);
                }
                }catch(Exception e) {
                    System.out.println(e.getMessage());
                }
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
        System.out.println("일일 트래픽:" + sum + "바이트");
    }
}
```

4.CharacterStream

- ❖ JDK 1.7 버전 이후에서는 BufferedReader 인스턴스를 사용하기 위해서는 Files 클래스의 newBufferedReader 메소드를 이용하는데 첫번째 매개변수로는 Path 클래스의 인스턴스를 두번째 매개변수로는 Charset 클래스의 인스턴스를 이용해서 인코딩을 설정
- ❖ Charset 클래스의 인스턴스는 StandardCharsets 클래스의 상수를 이용

SevenTextFile

```
public class SevenTextFile {  
    public static void main(String[] args) {  
        Path path = Paths.get("./sample.txt");  
        try(BufferedWriter writer = Files.newBufferedWriter(path,  
StandardCharsets.UTF_8)){  
            writer.append("우리나라");  
            writer.newLine();  
            writer.append("대한민국");  
        }catch(Exception e) {  
            System.out.println(e.getMessage());  
        }  
        try(BufferedReader reader = Files.newBufferedReader(path,  
StandardCharsets.UTF_8)){  
            for(String line; (line = reader.readLine()) != null;) {  
                System.out.println(line);  
            }  
        }catch(Exception e) {  
            System.out.println(e.getMessage());  
        }  
    }  
}
```

5. RandomAccessFile

- ❖ 입력 스트림과 출력 스트림의 두 가지 기능을 가지고 있는 스트림이며 기존의 입력 스트림과 달리 한 번 읽었던 입력 스트림을 다시 읽을 수 있는 스트림
- ❖ 생성자

생성자	설명
RandomAccessFile (File file, String mode)	<p>파일의 경로를 가지고 있는 file과 mode로 RandomAccessFile 클래스의 객체를 생성한다. Mode가 가질 수 있는 값은 아래와 같다.</p> <ul style="list-style-type: none">• r: 읽기 전용• rw: 읽기와 쓰기 전용• rws: read write synchronized write를 한 것은 즉시 실제 파일에 반영되고 파일의 내용뿐만 아니라 파일의 상태정보를 포함하기 때문이다.• rwd: read write data write를 한 것은 즉시 실제 파일에 반영되고 파일의 내용만 포함되고, 파일의 상태정보는 변경되지 않기 때문이다.
RandomAccessFile (String file, String mode)	파일의 경로를 가지고 있는 문자열 file과 mode로 RandomAccessFile 클래스의 객체를 생성한다

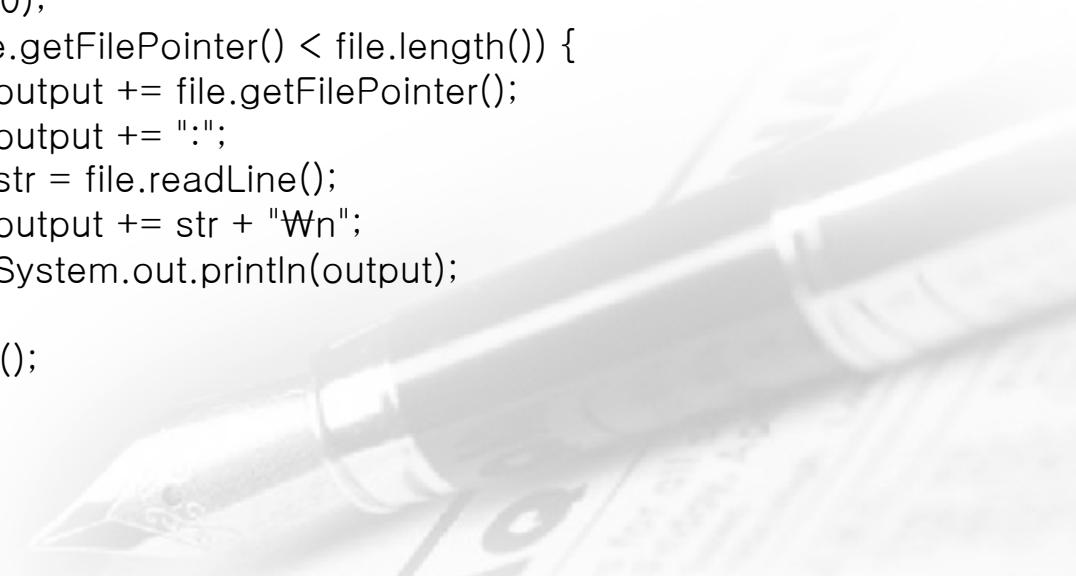
5.RandomAccessFile

❖ 메소드

- ✓ void seek(long pos): 파일 포인터의 위치를 설정하는 메소드
- ✓ void write(배열): 배열을 기록
- ✓ int read(배열): 배열의 크기만큼 데이터를 읽어 옴
- ✓ long getFilePointer(): 현재 파일 포인터의 위치 리턴
- ✓ long length(): 파일의 크기

예제 - RandomFileMain.java

```
public class RandomFileMain {  
    public static void main(String args[]) {  
        String output = "";  
        String str = "";  
        try {  
            RandomAccessFile file = new  
RandomAccessFile("./randomtest.txt", "rw");  
            String k = new String("Bye Bye Java");  
            file.seek(file.length());  
            file.write(k.getBytes());  
            file.writeChar('\n');  
            file.seek(0);  
            while (file.getFilePointer() < file.length()) {  
                output += file.getFilePointer();  
                output += ":";  
                str = file.readLine();  
                output += str + "\n";  
                System.out.println(output);  
            }  
            file.close();  
        }  
    }  
}
```



```
Problems @ Java  
<terminated> Random  
0:Bye Bye Java  
0:Bye Bye Java  
14:Bye Bye Java  
0:Bye Bye Java  
14:Bye Bye Java  
28:Bye Bye Java  
|
```

예제 - RandomFileMain.java

```
        catch (Exception e) {  
            System.out.println("Error: " + e.toString());  
        }  
        System.exit(0);  
    }  
}
```

6. 객체 직렬화

- ❖ 자바 객체를 저장하거나 전송하기 위하여 자바 객체의 코드를 다시 복원 가능한 Stream으로 만들어 주는 것
- ❖ 객체를 직렬화하기 위한 방법은 Serializable 인터페이스 구현
- ❖ Serializable을 이용하면 모든 데이터를 직렬화(변수에 transient가 사용되면 직렬화에서 제외)
- ❖ ObjectOutputStream
 - ✓ ObjectOutputStream 인터페이스를 구현한 클래스로 객체를 파일에 기록 가능한 클래스
 - ✓ ObjectOutputStream 인터페이스는 writeObject(Object obj) 메소드를 포함하는데 이 메소드가 객체의 데이터를 직렬화 시켜주는 메소드(직렬화 메소드)
 - ✓ 만약 obj가 Serializable 인터페이스로 구현되어 있지 않다면 NotSerializableException 예외가 발생
- ❖ ObjectInputStream
 - ✓ ObjectInput 인터페이스를 구현한 클래스로 직렬화 된 객체를 읽어올 수 있는 클래스
 - ✓ ObjectInput 인터페이스는 readObject() 메소드를 포함하는데 이 메소드는 객체의 데이터를 복원 시켜주는 메소드(역 직렬화 메소드)

예제-객체 직렬화(파일 쓰기 및 읽기)

```
Problems @ Javadoc Declaration Console
<terminated> ObjectSerialize [Java Application] C:\Program Files\Java\jdk1.7.0_04\bin\javaw.exe (
Data [number=1, name=박문석, age=44]
Data [number=2, name=이유진, age=43]
Data [number=3, name=김태현, age=42]
=====
직렬화해서 저장한 후 가져와서 출력
=====
Data [number=0, name=박문석, age=44]
Data [number=0, name=이유진, age=43]
Data [number=0, name=김태현, age=42]
```

예제- 객체 직렬화

Data 클래스

```
import java.io.*;  
public class Data implements Serializable {  
    private static final long serialVersionUID = 1L;  
    public static int bunho;  
  
    private transient int number;  
    private String name;  
    private int age;  
  
    static {  
        bunho = 0;  
    }  
  
    public Data() {  
        number = ++bunho;  
        name = "noname";  
        age = 0;  
    }  
    public Data(String name, int n) {  
        number = ++bunho;  
        this.name = name;  
        age = n;  
    }  
}
```

예제- 객체 직렬화

```
public int getNumber() {  
    return number;  
}  
public String getName() {  
    return name;  
}  
public void setName(String name) {  
    this.name = name;  
}  
public int getAge() {  
    return age;  
}  
public void setAge(int age) {  
    this.age = age;  
}  
@Override  
public String toString() {  
    return "Data [number=" + number + ", name=" + name + ", age=" + age +  
"]";  
}
```

예제- 객체 직렬화

ObjectSerialize 클래스

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.util.ArrayList;
import java.util.List;

public class ObjectSerialize {
    public static void main(String[] args) {
        ObjectInputStream ois = null;
        ObjectOutputStream oos = null;
        FileInputStream fis = null;
        FileOutputStream fos = null;

        List<Data> list = new ArrayList<Data>();
        list.add(new Data("박문석", 44));
        list.add(new Data("이유진", 43));
        list.add(new Data("김태현", 42));

        for (Data k : list)
            System.out.println(k);
    }
}
```

예제- 객체 직렬화

```
System.out.println("=====");
System.out.println("직렬화해서 저장한 후 가져와서 출력");
System.out.println("=====");

try {
    fos = new FileOutputStream("./object.dat");
    oos = new ObjectOutputStream(fos);
    oos.writeObject(list);

    fis = new FileInputStream("./object.dat");
    ois = new ObjectInputStream(fis);
    List<Data> result = (List<Data>) ois.readObject();
    System.out.println(result);
} catch (IOException e) {
    e.printStackTrace();
} catch (ClassNotFoundException e) {
    e.printStackTrace();
}
```

예제- 객체 직렬화

```
finally {  
    try {  
        if (fis != null)  
            fis.close();  
        if (ois != null)  
            ois.close();  
        if (fos != null)  
            fos.close();  
        if (oos != null)  
            oos.close();  
    } catch (IOException ioe) {  
        ioe.printStackTrace();  
    }  
}
```

6. autocloseable

- ❖ JDK 1.7 버전에서 추가된 인터페이스
- ❖ 더 이상 사용되지 않는 자원을 자동으로 닫아주는 메소드를 소유
- ❖ try – resource 안에서 생성한 스트림은 close()를 호출하지 않아도 자동으로 닫아줌

7. 표준 입출력

- ❖ 데이터 입력과 출력의 표준
- ❖ Java에서는 표준 입출력을 위해서 3가지 입출력 스트림을 제공(System.in, System.out, System.err)
- ❖ 자바 애플리케이션을 실행하면 바로 생성해주므로 별도로 생성하지 않고 바로 사용

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

- ❖ Eclipse는 콘솔로의 출력을 가로채서 에디터의 화면에 출력하는 것
- ❖ System.out과 System.err은 표준 출력장치에 출력할 수 있는 스트림인데 out은 검정색으로 err은 빨간색으로 텍스트를 출력
- ❖ System.in은 바로 사용하지 않고 BufferedInputStream이나 BufferedReader를 이용해서 사용
- ❖ Console 입력은 버퍼를 가지고 있기 때문에 Backspace를 이용한 편집이 가능하며 입력의 끝은 Enter 또는 control + z(유닉스나 맥에서는 control + d)이며 이 키를 만나면 버퍼의 내용을 읽기 시작

7. 표준 입출력

Enter를 누를 때 까지 입력받아서 출력하기

```
public class EnterMain {  
    public static void main(String[] args) {  
        try {  
            int input = 0;  
            while((input=System.in.read())!= -1) {  
                System.out.println("input :" + input + ", (char)input :" +  
                (char)input);  
            }  
        } catch(Exception e) {  
            e.printStackTrace();  
        }  
    }  
}
```

7. 표준 입출력

- ❖ System.in을 바로 사용하게 되면 Enter도 하나의 입력으로 간주하기 때문에 입력받아서 출력할 때 매번 Enter를 제거해서 출력해야 하는 번거로움이 있음
- ❖ 이러한 문제를 해결하는 방법은 BufferedReader를 이용해서 입력받거나 Scanner를 이용하는 방법이 있음
- ❖ BufferedReader 클래스나 Scanner 클래스의 readLine()이나 nextLine() 이용해서 입력을 받으면 Enter를 누를 때 까지 입력을 받고 Enter를 제거한 곳 까지만 리턴을 함
- ❖ setOut, setIn, setErr 메소드를 이용해서 출력 대상을 변경할 수 있음

7. 표준 입출력

Enter를 누를 때 까지 입력받아서 출력하기

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.util.Scanner;
public class BufferEnterMain {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        try {
            String input = null;
            BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
            System.out.print("입력:");
            input = br.readLine();
            System.out.println(input);

            System.out.print("입력:");
            input = sc.nextLine();
            System.out.println(input);
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            sc.close();
        }
    }
}
```

7. 표준 입출력

출력 대상을 변경해서 출력하기

```
import java.io.FileOutputStream;
import java.io.PrintStream;
public class ChangeMain {
    public static void main(String[] args) {
        PrintStream ps = null;
        FileOutputStream fos = null;
        try {
            fos = new FileOutputStream("./test.txt");
            ps = new PrintStream(fos);
            System.setOut(ps);
        } catch (Exception e) {
            System.err.println("잘못된 파일 경로입니다.");
        }
        System.out.println("Hello by System.out");
        System.err.println("Hello by System.err");
    }
}
```

연습문제

- ❖ log.txt 파일을 읽어서 IP 별 트래픽의 합계를 구하시오.