제6장 : 파일 조작

파일 조작의 기본 순서

1. 파일을 연다
2. 파일을 읽거나 쓴다
3. 파일을 닫는다

FileWriter


```
// 파일 열기, append 모드
FileWriter fw = new FileWriter("c:\save.dat", true);
// 내용 작성
fw.write("A");
fw.flush();   // 파일 쓰기
fw.close();   // 파일 닫기
```

FileReader


```
// 파일 열기
FileReader fr = new FileReader("c:\save.dat");
// 파일 읽기
int i = fr.read();
while (i != -1) {
  char ch = (char) i;
  System.out.print(ch);
  i = fr.read;
}
fr.close();   // 파일 닫기
```

바이너리파일의 조작

파일에는 크게 두 종류의 파일이 있다

1. 텍스트 파일
   a. 문자로 구성
   b. 메모장, 소스코드, HTML 등이 해당
   c. FileReader, FileWriter 사용

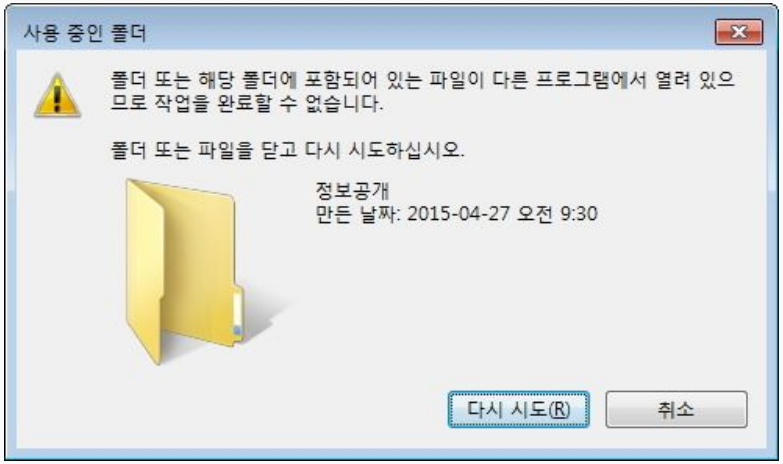
2. 바이너리 파일
   a. 문자와 데이터(바이트 배열)
   b. Excel, Java의 class 파일, 이미지 파일, 동영상 파일, 음악 파일 등
   c. FileInputStream, FileOutputStream 사용

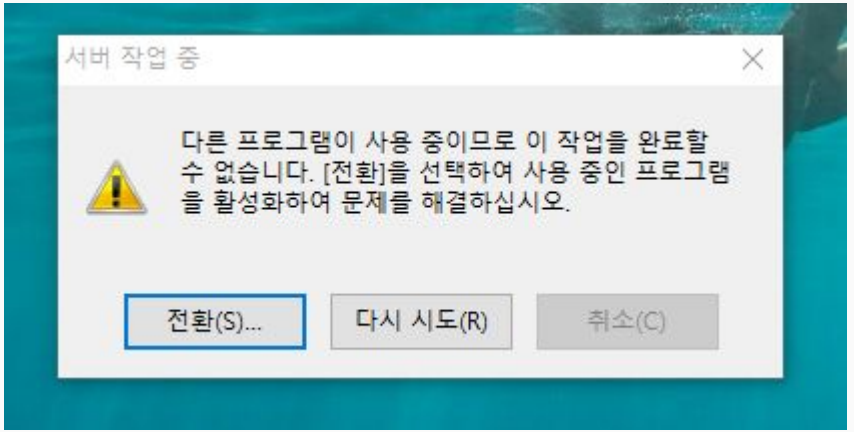
파일에 2진수 01000001 을 추가로 저장하는 프로그램


```
FileOutputStream fos = new FileOutputStream("save.dat", true);
fos.write(65);     // 2진수 01000001
fos.flush();
fos.close();
```

close() 를 안 했을 경우

다른 프로그램에서 해당 파일을 쓸 수 없음





예외처리

올바른 예외 처리를 해서 파일을 반드시 close() 해야 함

```
public class WriteToFile {
    public static void main(String args[]) {

        String message = "This is a sample message.\n";

        File file = new File("test1.txt");
        FileWriter writer = null;

        try {
            // 기존 파일의 내용에 이어서 쓰려면 true를, 기존 내용을 없애고 새로 쓰려면 false를 지정한다.
            writer = new FileWriter(file, true);
            writer.write(message);
            writer.flush();

            System.out.println("DONE");
        } catch(IOException e) {
            e.printStackTrace();
        } finally {
            try {
                if(writer != null) writer.close();
            } catch(IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

Java8 이후의 예외처리

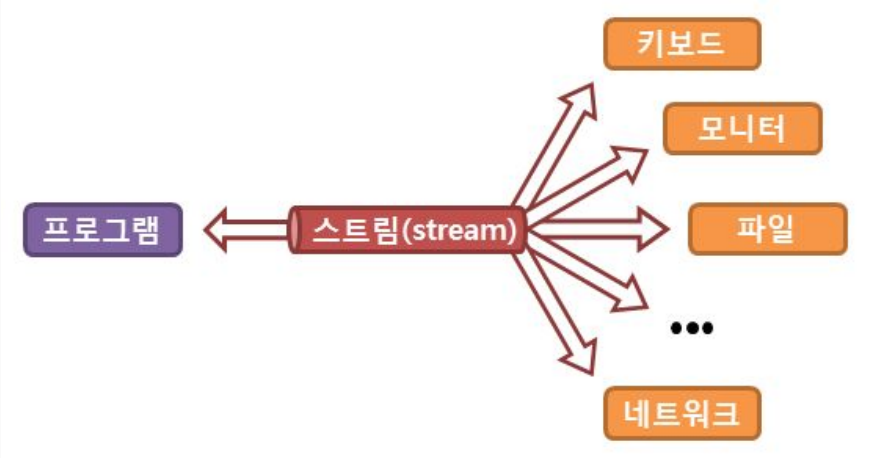
Java7 까지는 try catch finally 를 사용한 지저분한 예외처리를 해야 하지만
Java8 부터는 AutoCloseable을 구현하고 있는 객체는 try () 에서 자동으로 close() 해 줌

```
try (FileWriter fw = new FileWriter("save.dat", true)) {
  fw.write('A');
  fw.flush();
} catch(IOException e) {
  // 예외 처리
}
```

스트림

1. FileReader 는 데이터를 조금씩 읽거나 쓰는 API 임
2. 한번에 10GB의 파일을 읽어버리게 되면 메모리 부족이 발생 함
3. 스트림을 통해 데이터를 조금씩 흘려보내고 처리하면 이를 해결할 수 있음
4. FileReader, FileWriter, FileInputStream, FileOutputStream 는 모두 스트림 계열

여러가지 스트림



http://tcpschool.com/java/java_io_stream

문자열로부터 한 글자씩 읽기


```
String msg = "Hello World";
Reader reader = StringReader(msg);
char ch1 = (char) reader.read();    // H
char ch2 = (char) reader.read();    // e
```

바이트 배열에 값을 쓰기


```
ByteArrayOutputStream baos = new ByteArrayOutputStream();
baos.write(65);
baos.write(66);
byte[] data = baos.toByteArray();
```

표준 입출력

표준 출력 : System.out => 모니터
표준 입력 : System.in => 키보드

표준 출력을 변경할 수 있음

```
› java Main > data.txt
```

출력을 data.txt로 변경하였고 data.txt 입력으로 Main 프로그램의 실행 결과를 받음

필터

FilterReader, FilterWriter, FilterInputStream, FilterOutput 을 상속 받는 클래스들.

스트림을 변경시키는 용도

```
// 파일 출력 스트림 생성
FileOutputStream fos = new FileOutputStream("data.txt");
// 암호화 스트림 연결
Cipher c = Cipher.getInstance("AES/CBC/PKCS5Padding");
CipherOutputStream cos = new ChipherOutputStream(fos, c);
// 암호화하여 파일에 쓰기
cos.write(65);
```

버퍼링을 위한 필터

문자용 필터 : BufferedReader, BufferedWriter
바이트용 필터 : BufferedInputStream, BufferedOutputStream

1. 처리성능 향상
   a. 파일에 쓰거나 읽을때 마다 매번 하드디스크를 조작하는 것은 매우 느림
   b. 한 번에 모든 작업을 수행하여 쓰는 것이 효율적
   c. 버퍼링은 미리 읽어 두거나 한 번에 쓰는 작업
   d. 따라서 파일 조작은 버퍼링을 수행하는 것이 효율적

```
FileReader fr = new FileReader("save.dat");
BufferedReader br = new BufferedREader(fr);
String line = null;
while ((line = br.readLine())) != null) {
  // 한 줄씩 처리
}
```

파일 시스템의 조작


```
File file = new File("save.dat");
```

<파일 조작 메서드>

1. 삭제 : delete()
2. 이름 변경 : renameTo(File dest)
3. 존재 확인 : exists()
4. 파일인지? : isFile()
5. 디렉토리인지? : isDirectory()
6. 용량 : length()
7. 폴더 내 파일들 : listFiles()

복사기능은 제공 안 함

java.nio.file.Files

파일에 관련된 다양한 기능은 Files 클래스에서 제공

<제공되는 태표적인 기능>
복사, 이동, 삭제, 모든 내용 읽기, 모든 행을 읽어서 리스트로 반환 등

https://docs.oracle.com/javase/8/docs/api/java/nio/file/Files.html

Path 객체를 얻는 방법

```
Path path1 = Paths.get("save.dat");
Path path2 = file.toPath();
```

연습문제 6-1

FileInputStream, FileOutputStream 클래스를 사용하여 파일을 복사하는 프로그램을 작성하시오.

원본 파일 경로와 복사할 파일 경로는 프로그램 실행시 파라미터로 전달되는 것으로 하고, 버퍼링이나 에러 처리는 하지 않는다.

연습문제 6-2

연습문제 6-1에서 작성한 프로그램을 다음 조건을 수행하도록 수정하시오.

- java.util.zip.GZIPOutputStream 을 사용하여 압축한다.
- 버퍼링을 수행하시오.
- 예외처리를 하시오.