**File I/O**

= File Input/Output 으로 파일을 읽고 쓰는 방법

**File Input: 파일 읽기**

File System에서 Java로 Input

**File Output: 파일 쓰기**

Java에서 File System으로 Output

Java 1.8 이전까지는 I/O가 매우 어렵고 복잡했으나 Java 1.8 부터 비교적 간편해졌으며, 코드를 분석하기도 쉬워졌습니다.

FileReader reader -> 바이트 단위로 읽어 옴

BufferedReader bufferedReader -> 엔터키를 단위로 읽어 옴

자바 pipe는 오픈 순과 역순으로 닫아야 함

reader먼저 열고, br 열었으면 br먼저 닫고 reader 닫기

File file = new File("C:\\Java Exam", "Java Exam.txt");

System.out.println(file.exists()); // 파일이 존재하는지 여부 확인

System.out.println(file.isFile()); // 파일 객체인지 확인

System.out.println(file.isDirectory()); // 폴더 객체인지 확인

System.out.println(file.getAbsolutePath()); // 파일의 절대경로 확인

* 굉장히 많이 사용 암기해서 나쁠 것 없음 절대 경로

System.out.println(file.getName()); // 파일의 이름 확인

System.out.println(file.length()); // 파일의 크기(byte) 확인

System.out.println(file.lastModified()); // 파일이 마지막으로 수정된 시간

Date date = new Date(file.lastModified());

System.out.println(date);

System.out.println(file.getParent()); // 파일이 존재하는 경로

// 객체가 폴더일 경우, 폴더내에 존재하는 모든 항목의 목록

System.out.println(file.listFiles());

**파일을 쓸 때엔 확인 해야 할 항목 예시**

C:\java\outputs 라는 폴더에 java\_output.txt 라는 파일을 쓰려 한다면,

1. C:\java\outputs 라는 파일이 존재하는지 확인

2. 없다면 폴더를 생성

3. java\_output.txt 파일이 존재하는지 확인

4. 이미 있다면 새로 생성할 파일의 이름을 java\_output (2).txt 로 생성

5. 이 마저도 있다면 java\_output (3).txt로 생성

6. 파일이 없을 때 까지 순번을 증가시키며 찾기

7. 존재하지 않는 파일 순번을 찾았다면 파일 내용을 쓰기

재귀 호출의 경우, call stack이 한도만큼 쌓이면 애플리케이션을 강제로 죽임

멀티쓰레드 환경에서 cpu idle 타임을 모두 독점