2019년 2학기 객체지향프로그래밍
-2차 프로그래밍 과제

파일 관리 시스템

실습반 318호 / 남경진 조교

이름 신지수

학번 201621818

학과/학년 심리학과 / 4

Ⅰ. 서론

본 과제는 Java로 파일 관리 시스템의 일부 기능을 구현하고, 객체 지향 언어의 상속과 다형성 개념을 익히는 것을 목표로 한다. 프로그램이 제공하는 파일 시스템의 구조와 기능은 UNIX나 Microsoft Windows의 계층적인 파일 시스템 구조와 유사하다. 파일 관리 시스템은 총 일곱 가지 기능을 지원한다. 기능의 목록은 다음과 같다.

1. 파일 생성하기
2. 파일/디렉토리 삭제하기
3. 디렉토리 생성하기
4. 파일/디렉토리 정보 보기
5. 상위/하위 디렉토리로 이동하기
6. 파일/디렉토리의 접근 권한 변경하기
7. 현재 작업 디렉토리의 위치 보기

프로그램이 시작됨과 동시에 사용자의 현재 작업 디렉토리(cwd; current working directory)는 '/'라는 이름의 root 디렉토리로 설정된다. 프로그램은 사용자가 입력하는 명령문 중 유효한 것에 대하여만 수행한다. 또한 명령문이 유효하더라도 명령을 수행할 수 없는 경우(삭제하고자 하는 파일이 존재하지 않는 경우 등)에는 명령을 수행하지 않는다.

기능을 구현하기 위해 다섯 개의 클래스를 정의하여 사용하였다. 첫째로, Property 클래스는 파일과 디렉토리의 공통적인 속성을 저장하는 클래스이다. 파일이나 디렉토리의 이름, 접근모드에 대한 정보를 인스턴스 변수로 가지며, 이와 관련한 메소드를 가진다. 또한 Object 클래스의 equals() 메소드를 오버라이드하여 파일과 디렉토리의 이름을 비교하고자 하였다. 둘째로, File 클래스는 Property 클래스를 상속받아 일반 파일을 나타내는 클래스이다. 파일의 내용을 추가로 저장하며, 이를 출력하는 여러 메소드들을 가진다. 셋째로, Directory 클래스 또한 Property 클래스를 상속받는다. 이는 디렉토리를 나타내는 클래스로, 상하위 디렉토리를 참조하는 변수들을 가지며 디렉토리에 파일이나 디렉토리를 추가하거나 삭제하는 메소드, 상하위 디렉토리의 위치를 반환하는 메소드 등 디렉토리와 관련된 메소드들을 갖는다. 넷째로, Command 클래스는 명령문의 유효성을 판단하는 메소드들을 가진 클래스이다. 마지막으로 FileManagingTest 클래스는 main함수를 가지며 프로그램을 운영하는 기준이 된다.

보고서와 함께 제출한 프로그램에서는 과제에서 제시한 내용을 모두 구현하였으며, 클래스 상속, override, overloading 등의 개념을 본 프로그램에서 사용하였다.

Ⅱ. 본론

1. 프로그램 설명

과제에서 요구하는 파일 관리 시스템은 UNIX와 Microsoft Windows의 것과 유사하다. 파일 시스템의 구조는 계층적인 디렉토리 구조를 가지며, 일반 파일과 디렉토리로 이루어져 있다. 최상위 디렉토리를 root 디렉토리라 부르며 '/'로도 표시한다. 이 시스템을 다루는 명령어의 형식은 다음과 같다.

명령어 [접근권한] 파일_이름 [파일_내용]

괄호 친 내역은 옵션으로, 명령어에 따라 필요하거나 그렇지 않을 수도 있다. 시스템은 명령어에 따라 각기 다른 기능을 수행하며, 명령을 수행하기에 필요한 정보들이 명령문에 제공되지 않으면 명령을 수행하지 않는다. 본 프로그램에서 수행할 수 있는 명령어는 new, mkdir, del, show, chdir, chmod, cwd이며, 모두 소문자로 작성해야 한다. 접근 권한에는 두 가지 종류가 있는데, 하나는 읽기 권한(r)이고 다른 하나는 쓰기 권한(w)이다. 읽기 권한은 파일과 디렉토리의 내용을 읽을 수 있는

권한이고 쓰기 권한은 파일이나 디렉토리를 생성하거나 삭제할 수 있는 권한을 말한다. 한 파일은 한 가지 이상의 권한을 가질 수 있으며, 명령문에서 이를 +r, +w, +rw로 표현한다. 파일/디렉토리 명으로는 오직 영문자(대문자와 소문자)와 숫자로만 구성될 수 있으며, 현재 디렉토리는 '.', 상위 디렉토리는 '..'으로도 표현한다.

프로그램에서 제공하는 일곱 가지 기능은 다음과 같다.

1) 일반 파일 생성하기

new [접근_권한] 파일_이름 파일_내용

이는 현재 작업 디렉토리에 일반 파일을 생성하는 명령문이다. 일반 파일은 파일 이름, 파일 내용, 접근 권한을 고유의 정보로 가진다. 접근 권한은 '+rw', '+r', '+w', '+'라고 작성하거나 아예 작성하지 않을 수 있다. '+'와 작성하지 않은 경우에는 일반 파일의 접근 권한을 기본값인 +rw로 설정한다. 파일 이름은 오직 영문자(대소문자)와 숫자로만 이루어질 수 있으며, 한 디렉토리 내에는 동일한 이름의 파일이 함께 존재할 수 없다. 따라서 유효하지 않은 이름, 이미 존재하는 이름으로 파일울 생성할 수 없다. 이러한 경우에는 사용자에게 오류 메시지를 출력한다. 현재 작업 디렉토리에 쓰기 권한이 있어야 파일을 생성할 수 있으며 그렇지 않은 경우에도 오류 메시지를 출력한다. 일반 파일의 내용은 띄어쓰기를 포함하는 하나의 스트링이다.

2) 파일/디렉토리 삭제하기

del 파일_이름

이는 현재 작업 디렉토리에서 입력한 이름의 파일(일반 파일과 디렉토리를 통칭)을 삭제하는 기능이다. 만약 디렉토리를 삭제하는 경우에는 디렉토리에 속한 모든 하위 파일 및 디렉토리를 삭제한다. 입력한 이름의 파일이 존재하지 않을 경우에는 오류 메시지를 출력한다. 이 기능 또한 현재 작업 디렉토리에 쓰기 권한이 있어야 실행 가능하며, 그렇지 않은 경우에는 오류 메시지를 출력한다.

3) 디렉토리 생성하기

mkdir [접근_권한] 디렉토리_이름

이는 현재 작업 디렉토리에 디렉토리를 생성하는 명령문이다. 접근 권한, 디렉토리 이름, 생성 조건 등은 모두 '일반 파일 생성하기'의 것과 동일하다.

4) 파일/디렉토리 보기

show 파일_이름

입력한 파일 이름에 해당하는 파일이 일반 파일일 경우에는 파일 내용을 출력하고, 디렉토리일 경우에는 해당 디렉토리의 아래에 저장된 모든 파일의 이름과 접근 권한을 함께 출력한다. 이 때 한 줄에 하나의 파일 정보만을 출력한다. 이 때 보고자 하는 파일은 현재 작업 디렉토리에 존재해야 하며, 현재 작업 디렉토리는 읽기 권한이 있어야 한다. 만약 조건을 충족하지 못할 경우에는 에러 메시지를 출력한다. 예외적으로 'show .'라고 입력하였을 때에는 현재 작업 디렉토리에 존재하는 파일과 디렉토리의 정보를 출력한다.

5) 상위/하위 디렉토리로 이동하기

chdir 디렉토리_이름

현재 작업 디렉토리를 기준으로 상위 또는 하위 디렉토리로 이동하는 기능이다. 상위 디렉토리로 이동할 경우, 디렉토리 이름을 '..'라고 작성하고, 하위 디렉토리로 이동할 경우, 디렉토리의 이름을 적는다. 디렉토리의 이동은 한 단계씩만 가능하다. 디렉토리의 이동은 목적지 디렉토리에

읽기 권한이 있어야 가능하다. 성공적인 디렉토리 이동 후에는 현재 작업 디렉토리의 절대경로를 출력한다.

6) 파일의 접근 권한 변경하기

chmod 접근_권한 파일_이름

입력한 파일 이름을 가진 파일을 현재 작업 디렉토리에서 찾아서, 해당 파일의 접근 권한을 입력한 값으로 변경한다. 입력한 파일 이름을 가진 파일이 없거나, 접근 권한을 유효한 값으로 입력하지 않은 경우에는 접근 권한을 변경하지 못한다.

7) 현재 작업 디렉토리의 위치 보기

cwd

현재 작업 디렉토리의 절대경로를 출력한다.

2. 자료구조

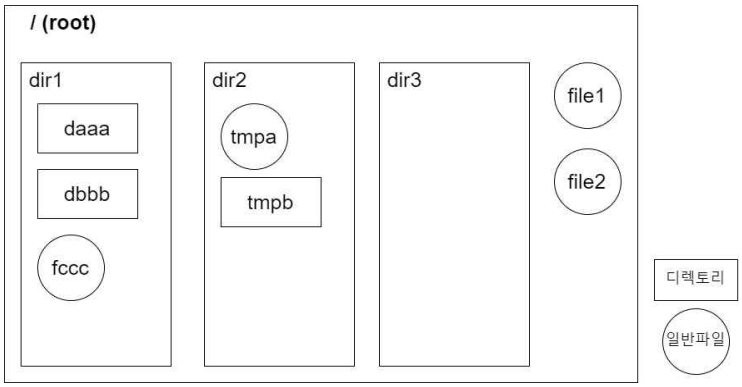

그림 1 디렉토리 구조의 예시

위의 그림은 디렉토리 구조의 예시이다. 사각형은 디렉토리를, 원은 일반 파일을 의미한다. 모든 일반 파일과 디렉토리는 root 디렉토리(/) 아래에 있다. 이러한 디렉토리 구조를 구현할 자료구조는 여러 가지이다.

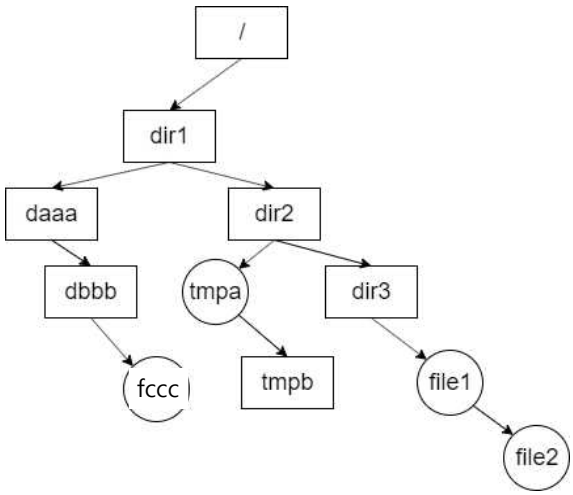

그림 2 Left Child, Right Sibling Tree

프로그램 계획 단계에서 가장 처음에 고려하였던 자료구조는 Left Child, Right Sibling 트리이

다. 이진트리이며 트리의 노드는 일반 파일과 디렉토리이다. 왼쪽 자식으로는 해당 디렉토리 내부의 파일을 연결시키고, 오른쪽 자식은 같은 디렉토리에 있는 파일들을 연결시킨다. 그림 2는 그림 1을 Left Child, Right Sibling 트리로 표현한 것이다. 이는 메모리를 효율적으로 사용할 수 있다는 장점이 있으나 복잡성과 시간 제약으로 인하여 다른 자료구조를 사용하기로 하였다.

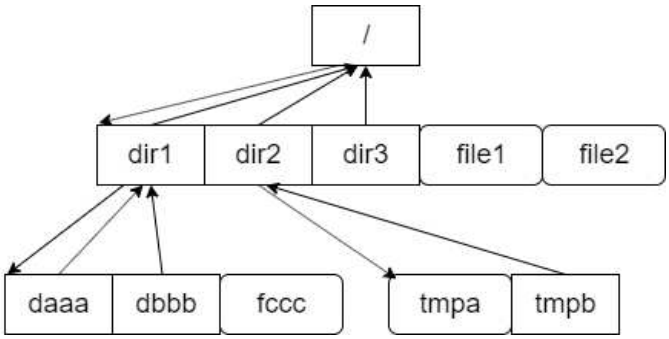

그림 3 배열을 이용한 linked list

둘째로 고려한 자료구조는 배열을 연결한 리스트이다. 디렉토리에 여러 파일들을 저장할 수 있는 배열을 연결하여 디렉토리 구조를 구현하였다. 트리와 다르게 배열은 인덱스를 통하여 데이터에 접근할 수 있고, 자료의 추가와 삭제가 편리하다는 장점이 있다. 또한 과제에서 절대경로를 출력하거나, 상위 디렉토리로 이동하는 기능을 요구하기에 상위 디렉토리에 다시 연결하여 doubly linked list와 유사하게 구현하였다. 그러나 배열은 정적 자료구조이다. 여러 개의 파일을 저장하기 위하여 처음 선언 시 배열의 크기를 크게 잡아야 하며, 이는 메모리 낭비를 유발한다. 따라서 일반 배열이 아닌 java.util.ArrayList에서 제공하는 ArrayList를 사용하였다.

3. 클래스

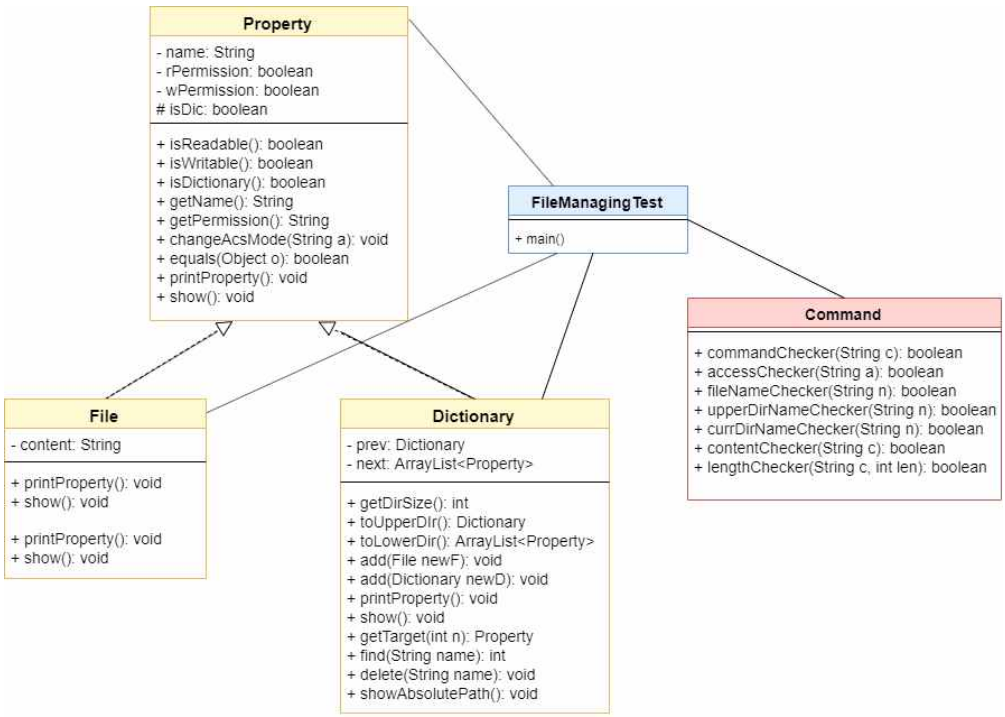

그림 4 클래스 다이어그램

프로그램을 구현하기 위해서 총 다섯 개의 클래스를 정의하여 사용하였다. 다섯 개의 클래스를 정리하여 그림 4로 제시하였다.

1) Property 클래스

Property 클래스는 File 클래스와 Directory 클래스의 super class로서 일반 파일과 디렉토리가 공통적으로 가지고 있는 속성들을 저장한다. 따라서 Property 클래스는 일반 파일과 디렉토리에 공통적으로 가진 파일의 이름, 접근 권한을 변수로 가지며 이와 관련된 메소드들을 가지고 있다.

표 1 Property 클래스 필드 변수: 흰색 배경 / 메소드: 회색 배경

이름	역할/기능	특징
name	파일의 이름	영문자(대소문자), 숫자로만 이루어짐
rPermission	읽기 권한	
wPermission	쓰기 권한	
isDir		true이면 디렉토리, false이면 파일. File, Directory 생성자에서 초기화됨
isReadable()	읽기 권한 여부를 반환	
isWritable()	쓰기 권한 여부를 반환	
isDirectory()	isDir 반환	
getName()	파일 이름 반환	
getPermission()	파일의 접근 권한을 반환	
changeAcsMode()	파일의 접근 권한을 변경	
equals()	파일 이름을 기준으로 Property 객체를 비교	Object클래스의 equals 함수를 override
printProperty()		구현하지 않음 subclass에서 override
show()		구현하지 않음 subclass에서 override

클래스의 인스턴스 변수들은 null 또는 false로 초기화 되어있고, 이는 File 객체나 Directory 객체가 생성될 때 재설정된다.

getPermission() 메소드는 접근권한을 String으로 반환하고, changeAcsMode() 메소드는 접근권한을 매개변수로 받을 때 String으로 받는다. 명령문의 입력과, 파일의 속성 출력 시 접근 권한을 String으로 다루어야 하기에 호환성을 위해 접근 권한을 String으로 처리하였다.

equals() 메소드는 Object 클래스의 equals() 메소드를 override 하였다. 이를 통해 equals()가 단순히 객체를 비교하는 역할이 아닌 객체의 이름을 비교하는 역할을 수행하게 되었다. 자료구조로 사용하는 ArrayList에서 equals() 함수를 이용하여 비교, 삭제 등의 기능을 수행하는데, 이름을 기준으로 파일을 비교하고자 하여 equals() 메소드를 override하였다.

printProperty()와 show() 함수는 super class에서는 선언만 하였고, 구현은 subclass가 하였다. 객체의 클래스에 따라 다른 수행을 해야 하기에 각 subclass가 override하여 구현하게끔 했다.

2) File 클래스

File 클래스는 Property 클래스를 상속받아 일반 파일을 객체로 생성한다. 일반 파일은 디렉토리와 다르게 파일 내용(content)를 갖는다. 따라서 File 클래스는 파일 내용을 String으로 저장하는 변수 content와 이를 출력하는 show() 메소드를 가진다. printProperty() 메소드와 show() 메소드는 Property 클래스의 메소드를 override하였다. 그 이유는 '파일/디렉토리 보기' 기능에서 대상에 따라 하는 일이 다르기 때문이다. 해당 기능을 수행하는 메소드를 show() 메소드로 정의하였는데, File에 대해 show()를 할 경우에는 파일의 content를 출력 하고, Directory에 대하여 show()를 할 경우에는 디렉토리 내 모든 파일의 이름과 접근 권한을 출력한다. 또한 Directory 객체의 show()를

호출하면 내부 파일들의 printProperty() 메소드가 호출되는데, 디렉토리일 경우 '이름/+접근권한'의 형식으로 출력되고 파일의 경우 '이름+접근권한'의 형태로 출력된다. 두 클래스에 모두 앞서 말한 기능들이 필요하나 수행하는 일이 다르기에 이름은 같이 하고 내용은 subclass에서 override하는 형식을 사용하였다.

표 2 File 클래스 필드 변수: 흰색 배경 / 메소드: 회색 배경

이름	역할/기능	특징
content	파일의 내용	여러 개의 띄어쓰기도 포함한다
printProperty()	파일 이름과 접근 권한을 출력	ex) filename+rw
show()	파일의 내용을 출력	

3) Directory 클래스

Directory 클래스는 Property 클래스의 subclass로서, 디렉토리 객체를 생성한다. 대부분의 기능들이 현재 작업 디렉토리에 대하여 수행되기에 기능을 수행하는 메소드들은 모두 Directory 클래스에 속한다.

표 3 Directory 클래스 필드 변수: 흰색 배경 / 메소드: 회색 배경

이름	역할/기능	특징
prev	상위 디렉토리를 참조	
next	하위 디렉토리 (ArrayList<Property>) 참조	
getDirSize()	하위 디렉토리의 크기 (파일 개수)를 반환	
toUpperDir()	prev를 반환	
toLowerDir()	next를 반환	
add(File f)	하위 디렉토리에 일반 파일을 추가	
add(Directory d)	하위 디렉토리에 디렉토리를 추가	
printProperty()	파일 이름과 접근 권한을 출력	
show()	하위 디렉토리의 모든 파일들의 이름과 접근 권한을 출력	하위 디렉토리(next) 내 객체들의 printProperty()를 호출
getTarget()	찾고자 하는 이름과 같은 이름을 가진 파일을 반환	Property 객체를 반환
find()	찾고자 하는 이름과 같은 이름을 가진 파일을 찾음	있다면 파일의 인덱스를 반환 없다면 -1을 반환
delete()	주어진 이름과 같은 이름을 가진 파일을 삭제	
showAbsolutePath()	현 작업 디렉토리의 절대경로를 출력	stack을 이용

상하 디렉토리 간 접근이 가능해야 하기 때문에 서로를 참조하는 변수를 정의하였다. 차이점이 있다면 상위 파일은 무조건 디렉토리이기 때문에 prev는 Directory 타입의 변수로 설정하였다. 그러나 디렉토리의 하위 파일로는 일반 파일과 디렉토리가 모두 올 수 있다. 따라서 둘 모두를 담기 위해 super type인 Property 타입으로 ArrayList를 선언하였고, next 변수는 이 배열을 참조한다. 이러한 방식으로 doubly linked list와 유사한 자료구조를 구현하였다.

add() 메소드는 add(File f), add(Directory d)로 overloading하였다. 디렉토리의 내부에는 일반

파일과 디렉토리가 모두 생성될 수 있기 때문에 두 타입에 대하여 모두 기능을 수행할 수 있도록 하였다. add(Property p)로 선언하지 않은 이유는, 디렉토리 안에 일반 파일도, 디렉토리도 아닌 단순 속성을 나타내는 무언가가 생성될 수는 없기 때문이다.

printProperty(), show() 메소드는 File 클래스 항목에서 설명한 바와 같이 super class의 함수를 override하여 Directory 클래스에 맞게 재정의하였다.

하위 디렉토리를 ArrayList<Property> 타입으로 구현하였기에, ArrayList 클래스에 내장된 size(), get(), remove(), indexOf(), add() 등의 메소드들을 사용하여 위의 메소드들을 정의하였다.

또한 getTarget() 메소드가 Property 객체를 반환하는 이유는 찾는 파일이 File type과 Directory type 중 무엇이 될지 모르기 때문에 그렇게 하였다. 또한 이름과 인자가 같으나 반환타입이 다른 두 메소드가 있을 경우, 오류가 발생하기 때문에 이를 피하기 위하여 이와 같이 메소드를 정의하였다.

showAbsolutePath()는 stack을 사용하였는데, 현재 위치한 디렉토리부터 시작하여 상위 디렉토리로 이동하면서 그 이름을 stack에 계속 저장하였다. root directory에 다다른 뒤에는 stack에서 하나씩 꺼내며 출력함으로 절대경로 출력을 구현하였다.

4) Command 클래스

Command 클래스는 사용자가 입력한 명령문을 검토하고 그 유효성을 판단하는 메소드들로 구성되어 있다. 메소드와 관련된 설명은 다음과 같다.

표 4 Command 클래스 메소드: 회색 배경

이름	역할/기능	특징
commandChecker()	명령어의 유효성 검사	제시된 일곱 개의 명령어만 유효
accessChecker()	접근 권한의 유효성 검사	+rw, +r, +w, + 입력 시 유효 그 외는 무효
fileNameChecker()	파일 이름의 유효성 검사	파일 이름이 영문자, 숫자로만 구성되어야 유효
upperDirNameChecker()	파일 이름이 '..'인지 검사	.. 만 유효
currDirNameChecker()	파일 이름이 '.'인지 검사	. 만 유효
contentChecker()	파일 내용의 입력 여부 검사	파일 이름 뒤 내용이 입력되었으면 true
lengthChecher()	명령문의 길이를 검사	

lengthChecker() 명령문은 입력한 명령문의 최소 길이를 기준으로 해당 명령문이 유효한 명령문인지 판단하는 메소드이다. 예를 들면, 새로운 디렉토리를 생성하는 명령문은 mkdir dir1 또는 mkdir +rw dir2이라고 입력된다. 따라서 명령문은 최소 두 글자 이상이 되어야 한다. 또한 명령문의 워드 사이에 띄어쓰기가 여러 개 와도 되기 때문에 길이의 상한은 두지 않고 오직 하한으로만 유효성을 판단한다.

또한 위의 메소드들은 모두 static 메소드이다, 그 이유는 위의 메소드들은 사용자의 입력마다 호출되어야 하기 때문에, 프로그램의 시작과 동시에 위 메소드들을 모두 메모리에 올리고 모든 입력에 대하여 같은 메소드를 공유하는 것이 더 효율적이라고 판단했기 때문이다.

5) FileManagingTest 클래스

FileManagingTest 클래스는 파일 관리 시스템을 운영하는 main 함수가 존재한다. 명령문을 입력받고, 여러 클래스를 활용하여 명령문을 검토하고, 명령문대로 기능을 수행한다.

4. 기능

주어진 명령문을 처리하고, 유효한 명령어에 한하여 그에 맞는 기능을 구현하기까지의 흐름도를 그림 5에 제시하였다. 파일 관리 시스템은 무한 반복된다.

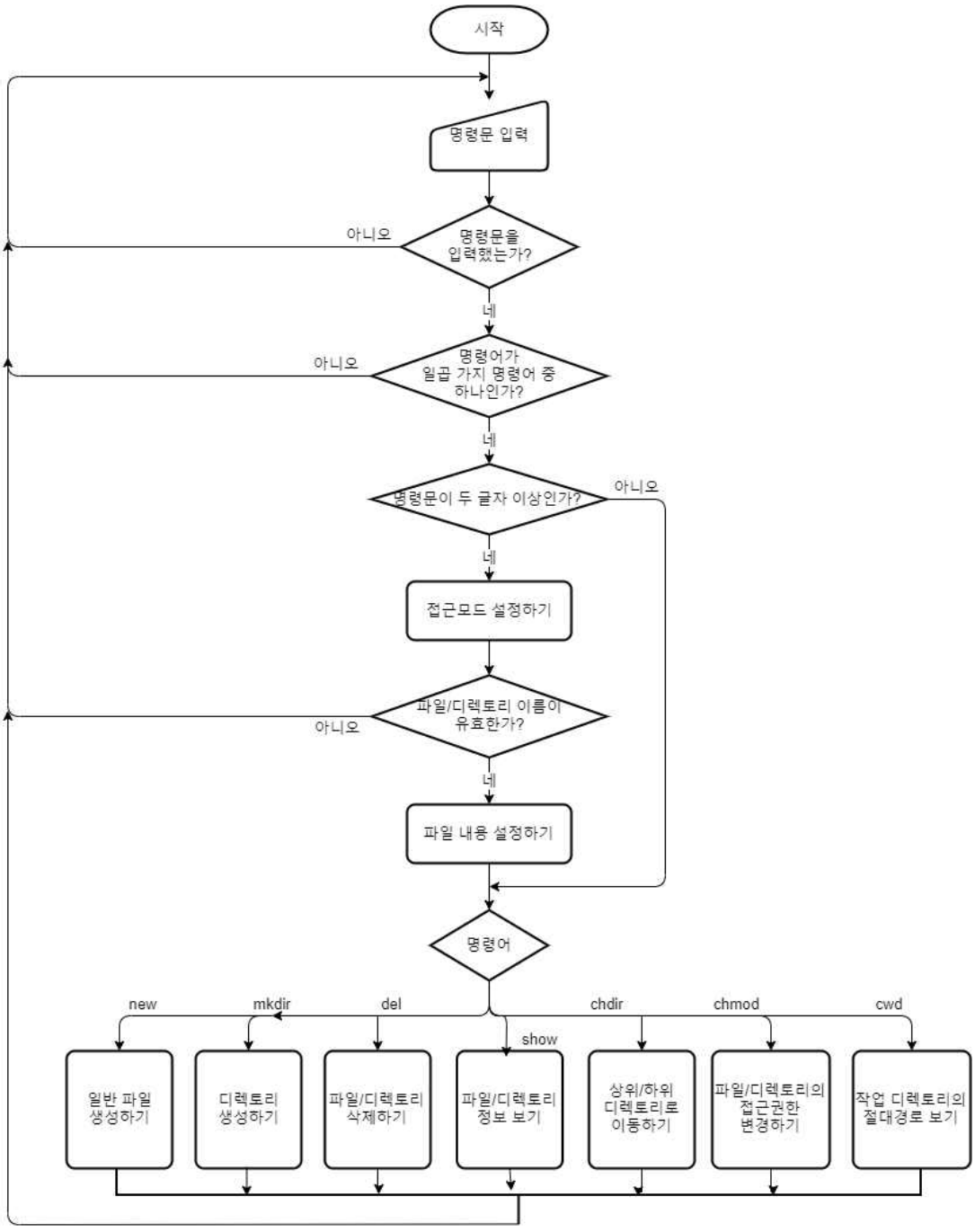

그림 5 파일 관리 시스템의 Flow chart

기능과 관련한 모든 조건은 <본론 1. 프로그램 설명>에 적힌 바와 같다.

1) 일반 파일 생성하기

일반 파일을 생성하는 과정을 그림 6의 흐름도로 제시하였다.

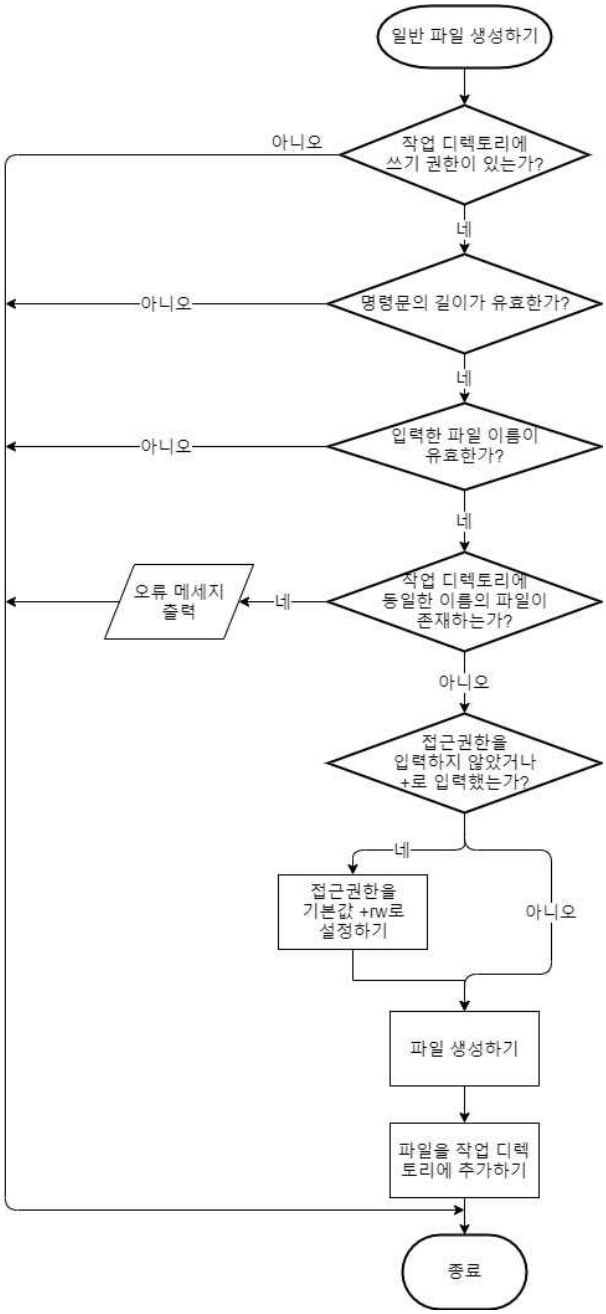

그림 6 일반 파일 생성하기(new)의 Flow chart

```
% new +rw file.txt this is file.
% show .
% new file1 This is    file1.
% new +w file2 This is file2.
% show .
file1+rw
file2+w
% new +r file1 Is file1 exist?
중복된 이름의 파일/디렉토리가 존재합니다.
%
```

◀ 일반 파일을 생성하는 예시이다. 현재 위치는 /(root directory)이고, 쓰기 권한이 있기 때문에 일반 파일을 생성할 수 있다. Line #1의 명령문은 파일 이름에 특수 기호인 '.'이 들어가서 파일이 생성되지 않은 것을 볼 수 있다. (Line #2의 명령문 show .로 현재 디렉토리의 파일들을 출력하였지만 아무것도 출력되지 않았다.) Line #3, 4의 명령문으로 파일을 생성하였고 이를 show .로 확인할 수 있었다. 또한 마지막 명령문에서 이미 존재하는 이름의 파일을 생성하려고 하자 오류메세지가 출력되고 생성이 되지 않는 것을 볼 수 있다.

2) 디렉토리 생성하기

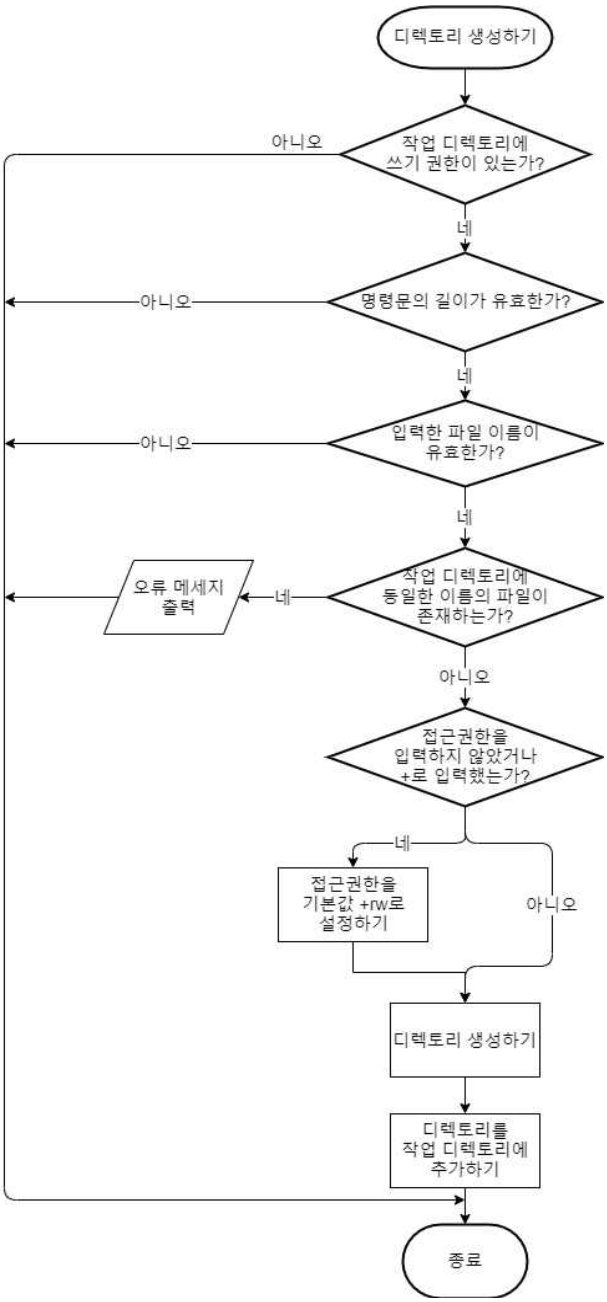

그림 9 디렉토리 생성하기의 flow chart

```
% mkdir +rw dir1
% mkdir +w dir2
% mkdir +r dir3
% show .
file1+rw
file2+w
dir1/+rw
dir2/+w
dir3/+r
```

◀ 디렉토리를 생성하는 예시이다. 현재 위치는 /(root directory)이며, 쓰기 권한이 있기 때문에 디렉토리를 생성할 수 있다. / 내부에 dir1, dir2, dir3 디렉토리를 생성하였으며 세 디렉토리의 접근 권한은 순서대로 +rw, +w, +r이다. 정상적으로 디렉토리가 생성되었는지 확인하기 위하여 show . 명령문으로 확인하였다.

```
% chdir dir3
/dir3
% mkdir newDirIndir3
% show .
%
```

◀ dir3 파일로 이동하여 디렉토리를 새롭게 생성하고자 하였다. 그러나 dir3 파일에 쓰기 권한을 주지 않았기에 내부에 디렉토리를 생성할 수 없었다.

3) 파일 삭제하기

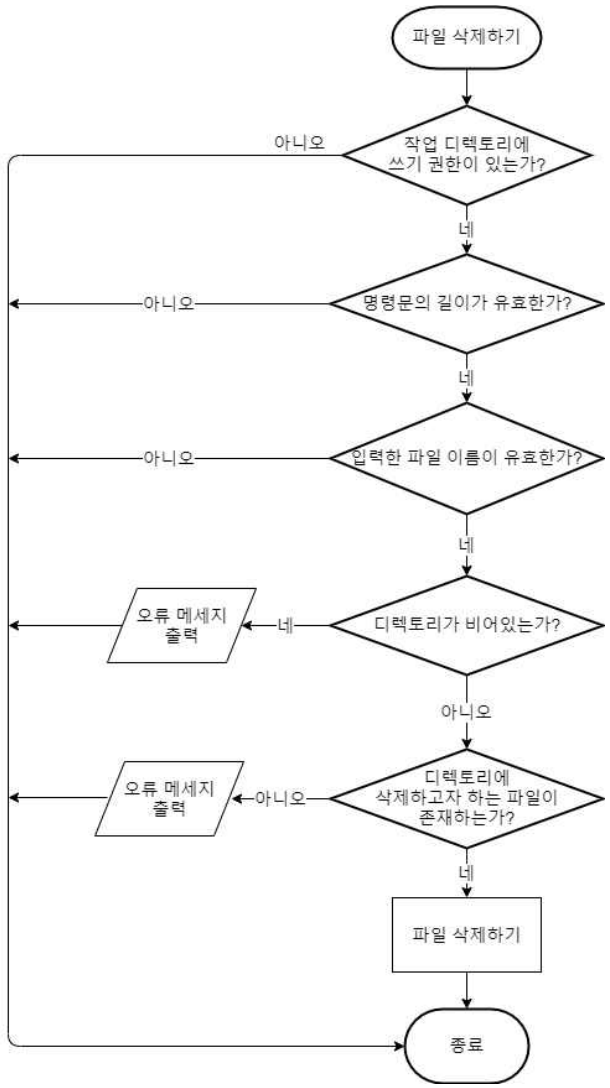

그림 11 파일 삭제하기의 Flow chart

```
% del file3
삭제하고자 하는 파일/디렉토리가 존재하지 않습니다.
% del file2
% show .
file1+rw
dir1/+rw
dir2/+w
dir3/+r
%
```

◀ 일반 파일을 삭제한 예시이다. root directory에 쓰기 권한이 있기 때문에 내부 파일을 삭제할 수 있다. Line #1은 파일 file3을 삭제하는 명령문이다. 해당 파일이 없기에 오류 메시지를 출력한다. Line #3에는 존재하는 파일인 file2를 삭제하였다. 그러자 디렉토리 내 file2가 삭제되었다.

4) 파일 보기

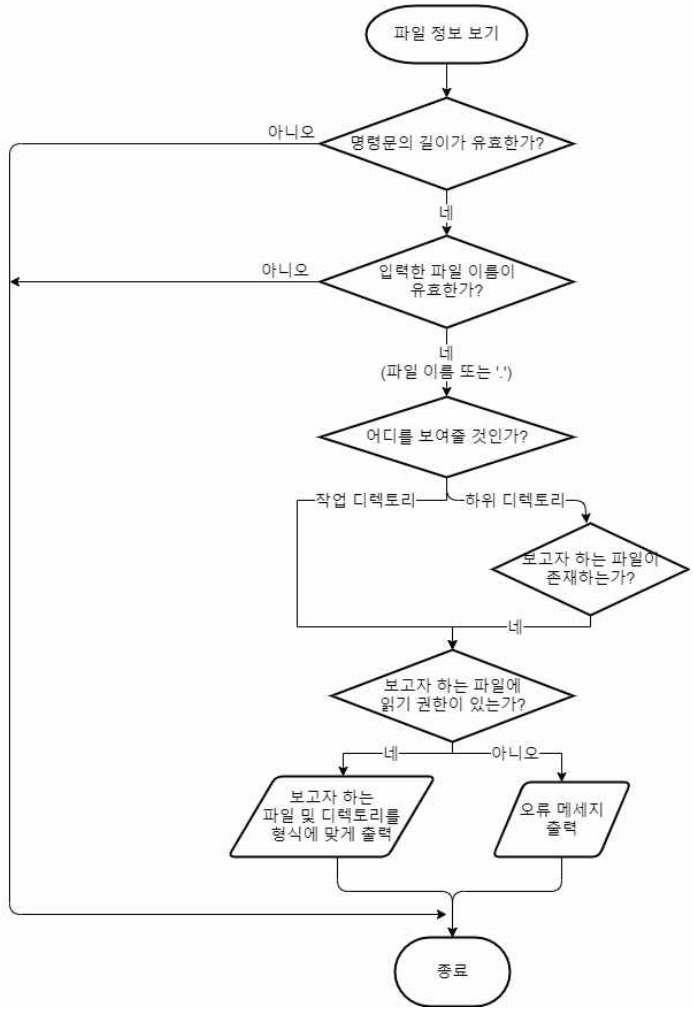

그림 13 파일 정보 보기의 Flow chart

```
% show
file1+rw
dir1/+rw
dir2/+w
dir3/+r
```

◀ 현재 디렉토리(cwd, '.') 내 파일들을 보여준다. 명령문의 형식은 'show .' 이나 이는 타 기능의 예시에서 여러 차례 보여주었다. 옆에 제시한 예시는 명령문 사이에 공백을 둘 이상 넣은 경우이다. show . 와 같이 root directory 내의 파일의 이름과 접근 권한을 출력한다.

```
% show dir1
daaa/+rw
dbbb/+w
fccc+rw
% show dir2
dir2에 대한 읽기 권한이 없습니다.
```

◀ /의 하위 디렉토리인 dir1을 보고자 하니 dir1 내에 있는 파일들의 이름과 접근 권한이 출력되었다. 그러나 dir2를 보려는 명령문에는 오류 메시지가 출력되었다. 디렉토리를 보려면 대상 디렉토리에 읽기 권한이 있어야 한다. 그러나 dir2는 쓰기 권한만 가지고 있기 때문에 내부를 볼 수 없었다.

```
% show file1
This is    file1.
```

◀ 디렉토리가 아닌 파일을 보았다. 그러자 파일 내용이 출력되었다.

```
% show ffff
해당 파일/디렉토리가 존재하지 않습니다.
```

◀ 존재하지 않는 파일을 보려하자 오류 메시지가 출력됐다.

5) 디렉토리 이동하기

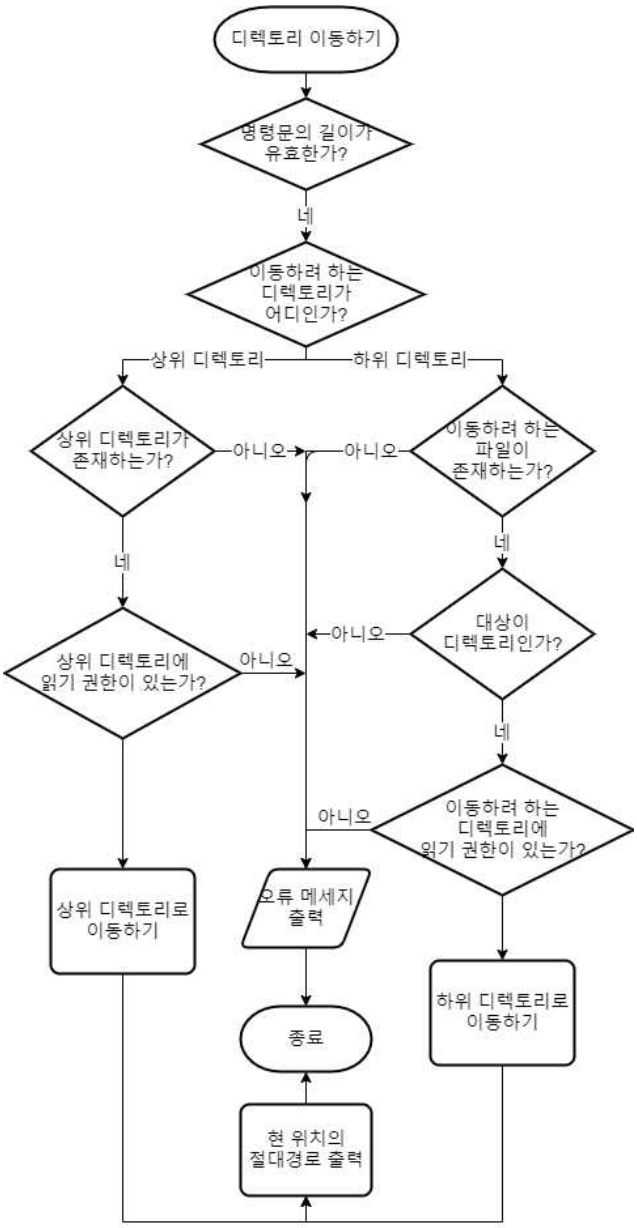

그림 18 디렉토리 이동하기의 Flow chart

```
% chdir ..
상위 디렉토리로 이동이 불가능합니다.
% chdir dir1
/dir1
% chdir dbbb
하위 디렉토리로 이동이 불가능합니다.
% chdir ..
/
% chdir abcde
해당 디렉토리가 존재하지 않습니다.
% chdir file1
해당 디렉토리가 존재하지 않습니다.
```

◀ 현 작업 디렉토리는 root directory이다. Line #1의 명령에 의해서는 이동이 불가능하다. root directory의 상위에 디렉토리가 존재하지 않기 때문이다. 따라서 오류 메시지를 출력한다. Line #3을 통해 현 디렉토리의 하위 디렉토리인 dir1으로 이동하자 dir1의 절대경로가 출력된다. Line #5 명령문으로 dir1의 하위 디렉토리인 dbbb로 이동하려고 하였으니 해당 디렉토리는 읽기 권한이 없기에 이동이 불가능하다. 또한 Line #9으로 존재하지 않는 디렉토리로 이동하려고 하자 이동이 불가능하다. Line #11은 이동하려는 대상이 존재하는 파일지만 대상이 일반 파일일 경우이다. 시스템은 해당 디렉토리가 없고 이동이 불가능하다는 메시지를 출력한다.

6) 파일의 접근권한 변경하기

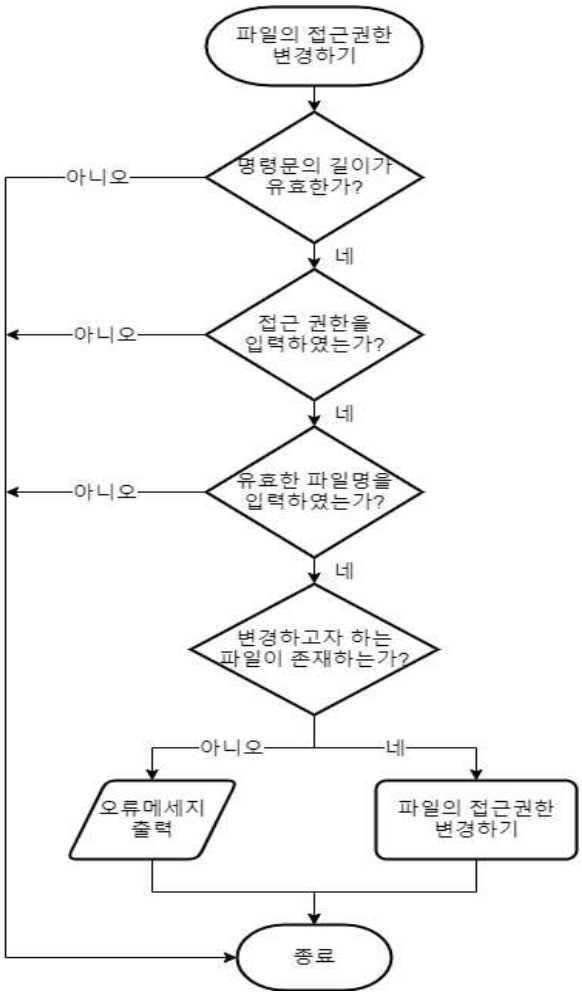

그림 21 파일의 접근 권한 변경의 Flow chart

```
% show .
file1+rw
dir1/+rw
dir2/+w
dir3/+r
% chmod dir2
% show .
file1+rw
dir1/+rw
dir2/+w
dir3/+r
% chmod +rw dir4
해당 파일/디렉토리가 존재하지 않습니다.
% chmod +rw dir3
% chmod +r file1
% show .
file1+r
dir1/+rw
dir2/+w
dir3/+rw
```

◀ chmod dir2는 접근 권한을 입력하지 않았기에 dir2의 접근 권한에는 변화가 없었다.

◀ 존재하지 않는 파일에 대하여 접근 권한을 변경하려 하자 오류 메시지를 출력한다.

◀ dir3의 접근 권한을 r에서 rw로, file1의 접근 권한을 rw에서 r로 변경하였다. show .를 통해 현재 파일들의 상태를 출력하자 두 파일의 접근 권한이 변경된 것을 확인할 수 있다.

7) 현재 작업 디렉토리의 절대경로 출력하기

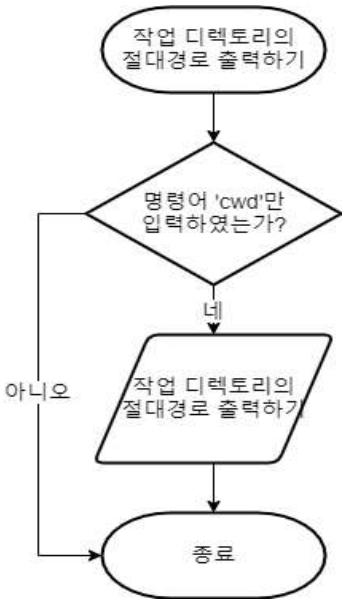

그림 23 cwd의 Flow chart

```
% cwd
/
% chdir dir3
/dir3
% cwd
/dir3
%
```

◀ cwd를 통해 현재 위치를 출력하였다.

```
% cwd file1
```

◀ 'cwd'가 아닌 다른 명령문을 입력하였을 때는 현재 위치를 출력하지 않는다.

Ⅲ 결론

이번 과제를 통하여 스스로 코드를 작성해가며 상속과 다형성의 개념을 익혔다. 상속 관계의 클래스 간 메소드의 overriding과 sub type의 객체를 super type에 대입하여 사용하는 개념이 코드를 간결하게 하였다. 또한 메소드의 이름을 공유하는 overloading의 개념 또한 코드를 간결하게 하였다.

아쉬운 점은 main 함수를 간결하게 작성하지 못한 점이다. 처음 클래스를 구상하고 처리 과정에 대해 고려할 때는 여러 예외사항과 처리 방법들을 구체적이게 고려하지 못했다. 바로 main에 명령문의 입력과 처리 등에 대하여 코드를 작성했는데, 코드가 길어지고 복잡해진 뒤에는 정리하기가 곤란하게 되었다. 명령문의 처리와 명령의 수행 등을 모두 함수로 모듈화하는 것의 장점과 방식에 대하여 고려해보는 시간이 되었다.

또한 수업시간에 배운 exception, try-catch를 이용하여 예외들을 처리하고 싶었으나 아직 사용법이 익숙하지 않아 if문으로 예외들을 처리하였다.