과제 3

사진, 사물 검색 프로그램

20232907 정현승

목차

1. 클래스 설계 방법

먼저 클래스를 설계하고, 이후 메소드의 코드를 채우는 방식으로 진행하였다. **설계 자체는** 수업 시간에 한 번 했던 것처럼 **"Picture 하면 있어야 하는 정보와 메서드"를 쭉 적고 이대로 코딩을 진행**하였으며, 코드를 작성하다 필요 없는 부분이나 추가로 필요한 부분이 생기면 그때그때 설계를 수정하는 방식으로 진행하였다. 설계는 Picture, PictureList, Stuff, StuffList, Tag, Image 순서로 진행했고, Picture 클래스를 설계하면서 아래의 Stuff, Tag 등의 클래스에서 어떤 메서드가 필요한지를 대강 파악하여 이를 Stuff, Tag 등의 안쪽 클래스 (Picture 등의 다른 클래스가 변수로 가지는 클래스) 설계 시에 반영했다. 코드를 짜는 건 반대로 Tag, Image, Stuff, StuffList, Picture, PictureList 순서대로, 즉 안쪽을 먼저 완성한 후 바깥쪽을 완성하는 방식으로 진행했다. 안쪽을 먼저 완성한 이유는, 예를 들어 Picture, Stuff, Image 셋 전부 Tag를 사용하는데 Tag 코드를 먼저 짜지 않고 다른 것 먼저 코딩을 진행하면 코딩을 정확히 한 건지 알기 어렵고, 내부 구조가 완성되지 않은 메서드를 사용하면 Eclipse가 매우 많은 에러를 뿜어낼 것을 염려했으며, Tag를 짜다 그 클래스의 구조가 변경되면 세 클래스를 모두 수정해야 하기 때문이다. 특히 설계 시에 메서드별로 입출력 값을 정확하게 정의하지는 않았기에, 코드를 짜면서 어떤 작업은 Tag 내부에서 해결하는 게 편한지, 어떤 작업은 Tag 외부에서 처리하는 게 편한지도 잘 모르는 상태이고, 따라서 클래스의 구조가 조금씩 변경될 가능성이 매우 높았기도 했다. 설계의 대부분이 엎어지는 게 아닌 메서드의 입출력 형식만 조금씩 수정하는 것이지만, 어쨌든 세 클래스 모두를 수정해야 한다는 점은 변하지 않기 때문에 이는 매우 비효율적인 방법이라고 판단하였다. 어차피 팀 프로젝트도 아니기 때문에 여러 클래스의 코드를 동시에 짜는 일도 발생할 수 없으므로, 위와 같은 방식을 택하였다. 결과적으로 과제 3 추가 공지가 계속 올라오고 코딩 중 뒤늦게 문제를 잘못 읽은 것도 발견하면서, 재잘재잘한 설계변경이 생각보다 많아지며 바른 판단이 되었다고 생각한다.

이하 설명은 6개의 클래스 전체 또는 대부분에 해당하는 사항을 설명한 것이다. 따로 모아 설명할 만한 마땅한 부분이 여기밖에 없어서 여기에 작성한다.

문자열로 바꾸어 출력하는 모든 과정은 toString() 메서드를 이용했다. 이를 위해 모든 인스턴스가 static인 StuffList 메서드를 제외한 5개의 클래스 모두 toString() 메서드를 정의하였다. toString() 메서드는 Object 클래스에 이미 정의되어 있는 메서드로써 클래스 변수를 문자열과 더하거나 문자열로 형 변환하는 등 문자열로 취급하면 자동으로 실행되어, 이 메서드에 정의된 대로 클래스 변수를 문자열로 변경한다. 각 클래스에서 이 메서드를 @Override 함으로써 각 클래스를 문자열로 바꾸는 방법을 변경할 수 있고, 이를 활용했다. 단 이렇게 활용하기 위해서는 함수의 modifier를 public으로 설정해야 하는데, 이러면 내부 구현 사항이 보이는 문제가 있다. 그렇다고

이 메서드를 default나 private로 변경하면 @Override가 적용되지 않아 정상적인 작동이 되지 않을 것을 우려해 불가피하게 public으로 설정하였다.

필터링이나 데이터 삭제 등에서 **한 객체가 다른 객체와 같은지 확인하는 과정은 equals(Object) 메서드를 이용**했다. 이 equals(Object) 메서드도 toString() 메서드처럼 Object 클래스에 정의되어 있어, '==' 연산자를 호출할 때 자동으로 호출되어 두 객체가 같은지 아닌지 확인하고 그 결과를 반환한다. 위 toString()처럼 이 메서드를 @Override 함으로써, 우리가 같은 객체라고 취급하는 것을 다른 객체라고, 다른 객체라고 취급하는 것을 같은 객체라고 판정하는 것을 막을 수 있고, '같음'의 기준도 재정의할 수 있다. 이때 '부분적으로 같음'과 같이 '같음'의 기준이 2개 이상 필요한 경우 보편적으로 사용될 만한 기준만 equals(Object)에 넣었고, 다른 기준은 별도의 메서드를 만들었다. 이 메서드도 default로 선언하고 싶었으나 toString()과 같은 이유로 public으로 선언하였다.

toString() 메서드와 equals(Object), 그리고 가끔 쓰이는 getter와 setter는 **Eclipse의 코드 자동 생성 기능을 사용**하여 생성한 뒤 필요한 부분을 수정하는 방식으로 진행하였다. @Override가 메서드 위에 붙은 것도 자동 생성으로 만들어진 것이고, equals(Object) 코드 초반에 if 문 3개가 나란히 있는 것도 자동으로 생성된 것을 수정하지 않은 것이다. 다만 같이 생성되었던 hashcode() 메서드는 용도를 알 수 없어 삭제했으며, 자동 생성 시에 들어있던 Object.equals(Object, Object) 메서드와 Arrays.equals(Object[], Object[]) 메서드도 이 함수의 내부 구현을 배우지 않아, 비정상적인 작동 가능성을 고려하여 전부 지우고 int, String 등의 클래스에 선언되어 있던 equals 메서드로 교체하였다.

Picture 객체를 생성할 때 세미콜론이나 괄호 개수가 맞지 않거나, 문자열을 LocalDateTime으로 변경할 수 없는 경우, 즉 **입력된 문자열이 비정상적이라면 그때그때 예외를 던졌다.** 즉 Picture 객체 생성 중에 비정상을 발견하면 Picture 생성자에서, Stuff[] 객체 생성 중 비정상을 발견하면 Stuff 생성자에서 예외를 던졌다. 이렇게 던져진 예외는 PictureList에서 일괄적으로 처리하여, 제대로 생성되지 않은 Picture 객체가 PictureList에 추가되는 것을 막았다. 생성되다 만 객체는 어떤 값은 들어있고, 어떤 값은 아직 어떤 값도 대입되지 않은 상태로 나타나게 되지만, 어차피 PictureList에 추가되어 있지 않아 이러한 객체는 사용되지 않고 Java의 Garbage collector가 알아서 처리하게 되기 때문에 신경 쓰지 않아도 된다. StuffList에도 제대로 생성되지 않은 Stuff 객체는 추가되지 않고, Stuff 생성자에서 던져진 예외가 PictureList까지 올라가서야 처리되므로 제대로 생성되지 않은 Stuff 객체가 StuffList에 들어갈 일도 없다(Stuff 객체를 StuffList에 추가하는 작업이 Stuff 객체를 완전히 생성한 뒤에 진행되므로, 중간에 문제가 생겨 예외가 던져지면 그 뒤 코드를 전혀 실행하지 않게 되어 Stuff 객체를 StuffList에 추가하는 작업도 건너뛰게 된다).

생략된 변수는 null로 두지 않고, 빈 문자열(String 형이면) 또는 **길이가 0인 배열**(배열 형태의 변수이면)을 넣었다. 이는 비록 저장 공간 면에서는 매우 비효율적이지만, 불필요한 예외 처리를

줄이는 효과가 있다. 예를 들어 for each 문을 실행할 때 그 대상으로 null을 넣으면 NullPointerException이 던져진다. 그러나 길이가 0인 문자열을 넣으면 예외도 던져지지 않고, for 문도 실행되지 않는다. 따라서 for 문 이전에 배열이 null인지 확인하는 조건문을 넣을 필요가 없다. 문자열로 변경할 때도 마찬가지로, null을 문자열로 바꾸면 에러가 발생하거나 'null'로 바뀌어, 생략된 정보를 출력할 때 null을 빈 문자열로 바꾸는 수고를 덜 수 있다. 또 다른 메서드를 사용할 때도 예외가 던져지는 상황을 막을 수 있다. 생략 불가능한 요소가 빈 문자열로 저장되었는지 확인하는 등 문자열이 비었는지 확인이 필요하면 String.isBlank() 메서드를 활용했다. (단 이 메서드는 java 8버전에는 존재하지 않는 함수이기에, 이 메서드를 시용하려면 java 11버전으로 컴파일을 해야 한다.)

또한 배열에 항목을 추가할 때, **배열의 길이를 한 번에 하나씩 늘리는 방안을 선택**했다. 미리 100개의 데이터를 저장할 수 있는 공간을 받아둔 뒤 맨 앞부터 차례대로 저장하는 방법을 선택해도 되지만, 이번에는 딱 필요한 공간만을 받기 위해, 즉 시간을 희생하여 공간을 최적화하는 방법을 택하였다. 다만 이 방법을 쓴 이유는 단순히 공간 최적화를 위한 것은 아니고, 배열을 생성할 때 배열의 길이(예를 들어 PictureList.length) 값을 활용하기 위해서(배열과 배열의 길이를 다른 변수로 두고 싶지 않고, 하나로 묶어 취급하고 싶었다), 그리고 배열에 저장할 공간이 부족할 때만 배열을 다시 생성하는 게 약간 귀찮아 이렇게 하였다. 물론 다음에 배열을 ArrayList로 변경하면 이 과정은 전부 의미 없는 과정이 된다.

저번 과제와 달리 **생성자에서 직접 문자열을 분리해, 각 변수에 값을 넣었다.** 저번 과제처럼 문자열을 분리해 변수에 넣는 과정을 별도의 메서드로 분리하지 않은 것은, 생성자가 이 일을 하는 게 훨씬 직관적이기도 하지만, 우선 설계 당시에는 이를 생각 못 했고, 별도의 메서드로 분리했을 때의 장점인 코드 재활용을 여기서는 활용할 여지가 없다고 판단했기 때문이다. 물론 생성자 내의 코드를 전부 잘라내어 다른 메서드로 옮기는 작업만 하면 지금 당장에라도 둘을 분리할 수 있다.

문자열에서 각 변수에 들어갈 **값을 분리하는 작업**은, Picture 객체를 만들 때는 ">", Stuff 여러 개를 만들 때(StuffList.newStuffs(String) 메서드)는 "]"를, Image나 Stuff 등은 ";" 기준으로 문자열을 분리하였다. 이후 분리된 문자열의 맨 앞에 각각 "<", "["이 있는지 확인 후, 이 문자를 지우고 이후 처리를 진행했다. (Image나 Stuff 등은 이 과정을 생략한다.) 이때 문자열을 분리한 결과로 문자열의 개수가 특정 값이 나와야 하는 상황(Picture, Image, Stuff 객체를 만들 때)에서 나와야 하는 값과 다른 값이 나오면 예외를 던졌다. 처음에는 여는 괄호와 닫는 괄호의 쌍을 찾아, 쌍이 완성되는 것을 기준으로 분리하려 하였으나(이는 자료구조 Stack의 대표적인 활용 예시이기도 하다) 코드도 복잡해지고(이 설명도 간단하지는 않다) C Style이 될 것을 우려해 사용하지 않았다.

배열에서 어떤 객체를 지울 때는, **우선 삭제의 대상이 되는 객체의 index를 찾는다.** 만약 그 대상이 되는 **index를 찾지 못한 경우 예외**를 던지고, 찾았으면 현재의 배열보다 길이가 하나 적

은 배열을 새로 만들고, **그 객체를 제외한 모든 객체를 새 배열에 복사한 뒤 기존 배열을 새 배열로 바꾸는 방법**으로 진행했다. 이때 객체의 index를 찾아 저장하는 변수의 클래스는 Integer를 사용했는데, 이 클래스는 int 형 변수 하나를 저장하며 int 형과 관련한 다양한 메서드를 지원하는, int 대신 사용할 수 있는 매우 편리한 클래스이다. 그리고 int 형이 아닌 별도의 클래스이기 때문에, Integer 클래스는 int형 변수가 가질 수 있는 모든 값은 물론, **null** 값도 가질 수 있다. 여기서 활용하려 하는 것은 Integer 클래스의 다른 유용한 메서드가 아니라, null 값을 가질 수 있다는 점이다. 삭제할 대상의 index를 찾지 못했을 때 이 정보를 담아야 하는데, 0도 index가 될 수 있으므로 이걸로 처리할 수도 없고 적당히 큰 수를 넣기에는 배열의 크기가 커지면 그 수도 index로 활용하게 될 수 있으므로 이걸로 처리하기도 어렵다. 이때 Integer 클래스를 이용하면, index를 찾았다면 그 index 값을 클래스에 넣어주면 되고, 못 찾았다면 이 자료형은 정수가 아닌 **null이라는 또 다른 값을 넣을 수 있으**므로 이를 이용해 삭제할 대상을 찾지 못했다는 점을 어떤 부작용 없이 처리할 수 있다. 이것 대신 index 값에 -1을 넣는 것도 가능했으나, 이 방안을 코드 완성 이후에 떠올리는 바람에 사용하지 않았다.

A. Picture 클래스 설계

Picture 클래스는 다음과 같이 설계하였다.

Divisors			
modifier	자료형	인스턴스명	설명
변수			
private	String	id	id(String 형)
private	LocalDateTime	timeStamp	사진 촬영 시각(DateTime 형)
private	Image	pictureInfo	사진 정보(클래스로 분리)
private	Stuff[]	stuffList	사물 정보(사물 하나를 클래스로 분리했으며, 그 클래스의 배열 형태)
private	Tag	tags	태그('#'으로 시작하는 String의 배열이나 클래스화함)
private	String	comment	코멘트(String 형)
private static	DateTimeFormatter	dateTimeFormat	문자열을 DateTime으로 바꾸거나, 저장된 DateTime을 문자열로 바꿀 때 사용할 형식 저장, 객체마다 다른 게 아니라 모든 객체가 같은 형식을 공유하며, 문자열과 DateTime 간 저장 방식 변환에만 관여하기 때문에 static으로 선언

생성자			
default		Picture(String)	사진 정보를 하나의 문자열로 받아, 이를 id, timeStamp, pictureInfo, stuffList, tags, comment로 각각 나누어 저장하는 생성자
default (미사용)		Picture(String,LocalDateTime,Image,Stuff[],Tag,String)	Picture의 각 변수를 하나하나 받아 객체를 생성하는 생성자, 원래 image 파일 이름을 주면 현재 시각으로 timeStamp 및 id를 만드는 생성자를 요구하였을 때 바로 위의 생성자처럼 구조를 짜면 메서드의 이름과 인자가 같아 문제가 발생하여, 문자열을 각 변수로 나누는 작업을 PictureList 클래스에서 시행하고 Picture 클래스에서는 이 생성자를 이용하여 작업이 완료된 결과물만으로 객체를 생성할 예정이었으나, 이후 image 파일 이름만 주는 생성자 요구가 사라져 사용하지 않게 됨
메소드			
public	void	print()	필수 요구사항으로, 사진 정보를 console에 출력, 사진 정보를 문자열로 만드는 것은 toString 메서드 이용
private (미사용)	boolean	ifStuffExist(Stuff)	이 Picture 객체에 인자로 주어진 Stuff가 있는지 확인하는 메서드, 아직은 사용하지 않으나 검색 기능 추가 시 활용 예정
private (미사용)	void	addStuff(String)	Picture의 Stuff를 {String 형태로 받아 Stuff로 바꾸어} 추가하는 메서드, 아직은 Stuff 추가 및 삭제를 하지 않으니 사용하지 않음
private (미사용)	void	addStuff(Stuff)	Picture의 Stuff를 추가하는 메서드, 아직은 Stuff 추가 및 삭제를 하지 않으니 사용하지 않음
private (미사용)	void	deleteStuff(String)	Picture에서 Stuff를 {String 형태로 받아} 제거하는 메서드, String을 Stuff로 바꾸는 것만 여기서 시행하고 나머지 작업은 아래 deleteStuff(Stuff)에서 시행
private (미사용)	void	deleteStuff(Stuff)	Picture에서 Stuff를 제거하는 메서드, StuffList에서는 지워지지 않으며, Picture에 없는 Stuff를 지우려 하면 Exception을 던짐, 아직은 Stuff 추가 및 삭제를 하지 않으니 사용하지 않음
private (미사용)	boolean	ifTagExist(String)	Picture의 Tag에 인자로 받은 String이 있는지 확인하는 메서드, 아직 검색 기능이 없으니 사용하지 않음
private (미사용)	boolean	ifTagExist(Tag)	Picture의 Tag에 인자로 받은 Tag가 모두 있는지 확인하는 메서드, 아직 검색 기능이 없으니 사용하지 않음

private (미사용)	void	addTag(String)	Picture의 Tag를 추가하는 메서드, 아직은 Tag 추가 및 삭제를 하지 않으니 사용하지 않음
private (미사용)	void	deleteTag(String)	Picture의 Tag를 지우는 메서드, 없는 Tag를 지우려 하면 Exception을 던지며, 아직은 Tag 추가 및 삭제를 하지 않으니 사용하지 않음
default	int	compareToByTime(Picture)	이 Picture 객체와 다른 Picture 객체 중 timeStamp를 기준으로 뭐가 크고 뭐가 작은지 비교하는 메서드, 이 객체에 비해 비교 대상이 크면(시간이 나중이면) 음수, 작으면 양수, 같으면 0을 반환하며, 이는 LocalDateTime.compareTo() 함수의 반환값임. PictureList를 시간순으로 정렬할 때 사용함
default	int	compareToById(Picture)	이 Picture 객체와 다른 Picture 객체 중 id를 기준으로 뭐가 크고 뭐가 작은지 비교하는 메서드, 이 객체에 비해 비교 대상이 크면(처음으로 다른 문자가 나타날 때, 뒤 문자의 ASCII 코드 값이 크면) 음수, 작으면 양수, 같으면 0을 반환하며, 이는 String.compareTo() 함수의 반환값임. PictureList를 id 순으로 정렬할 때 사용하나, 아직 PictureList의 id 정렬은 사용하지 않고 있는 상태
private (미사용)	int	tagLength()	Picture 객체의 tag 길이(개수) 반환, 설계할 때는 필요할 것 같아 넣었으나 아직 사용하지 않음
default	boolean	equalById(Picture)	이 객체와 인자로 주어진 Picture 객체의 id가 같은지 확인하는 메서드, PictureList에서 같은 id를 가진 Picture 객체가 이미 있으면 예외로 처리하기 위해 사용함
public	String	@Override toString()	Picture 객체를 문자열로 바꾸는 메서드로, print() 메서드나 Picture 객체를 문자열로 바꾸는 과정이 필요한 모든 상황에서 사용하는 메서드. @Override 메서드로, 다른 문자열과 Picture 객체를 더하는 등 Picture 객체를 문자열로 처리할 경우 자동으로 이 메서드가 실행되어, 이 메서드에서 정의한 방식으로 Picture 객체를 String으로 변경함
default static	void	setDateTimeFormat(DateTimeFormatter)	static 변수 dateTimeFormat의 setter 문자열을 Picture 객체로 변경하거나 그 반대로 변경할 때 이 메서드를 먼저 실행해 dateTimeFormat을 지정함 이 변수를 읽는 일은 없을 것으로 예상되어 getter는 만들지 않음
public	boolean	@Override equals(Obj	이 객체가 인자로 주어진 객체와 같은지 확인해 그 결과를 반환하는 메서드로, id, timeStamp, pictureInfo, tags, comment

		ect)	가 모두 같고, stuffList는 순서 없이 들어있는 것만 일치하면 같은 것으로 보고 true 반환, 그렇지 않으면 false 반환, 각각의 변수가 일치하는지는 각 자료형의 equals 메서드를 이용함

표 1.1

compareTo 메서드는 String과 LocalDateTime 클래스 등에 이미 선언된 메서드로, 이 객체 대비 비교 대상 객체가 크면 음수, 작으면 양수, 같으면 0을 반환하는 메서드라는 설명을 Eclipse에서 볼 수 있었다. 즉 C 언어에서 내장 qsort 함수를 사용할 때 같이 인자로 넘겨주는, 두 객체의 대소를 비교하는 함수와 같은 역할을 하는 것이다. compareToByTime(Picture) 메서드와 compareToById(Picture) 메서드는 이러한 메서드를 따라 만든 것이다. 아래 그림 1.2는 그 compareTo 메서드를 확인한 방법을 재현한 것이다.

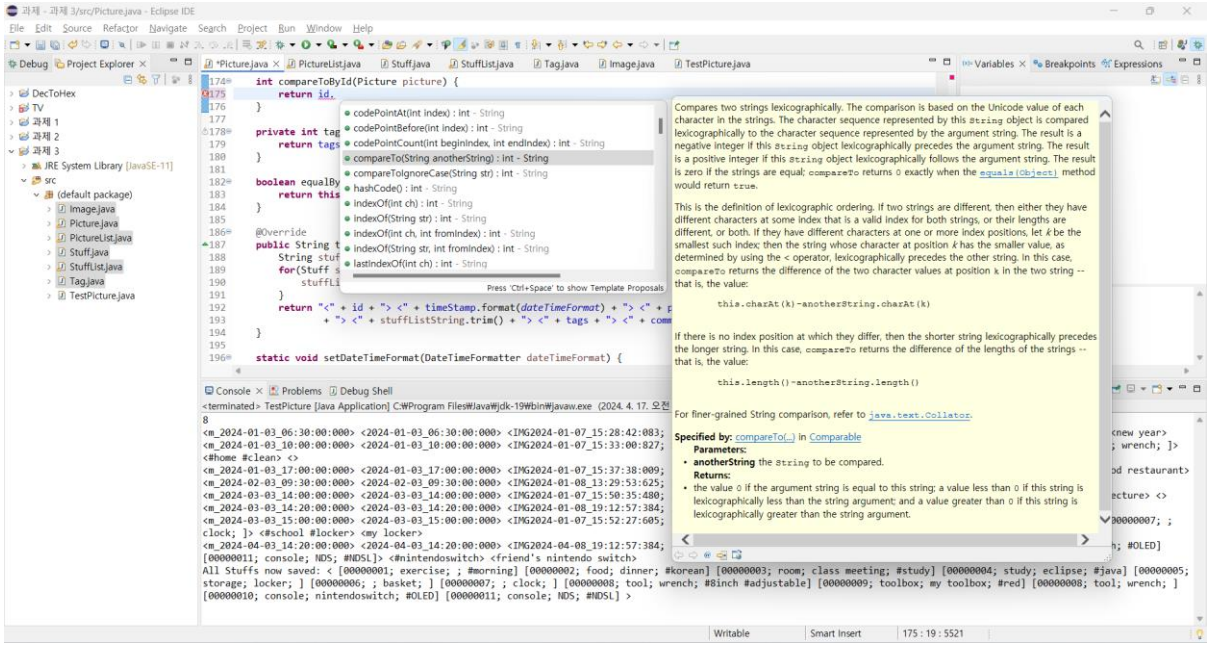

그림 1.2

equals 메서드 후반부의 이중 for 문 2개는 각각 비교 대상인 객체의 Stuff가 이 객체에 있는지, 이 객체의 Stuff가 비교 대상인 객체의 Stuff에 있는지 비교하는 것이다. Stuff 메서드는 순서 없이 비교하기 때문에 이렇게 긴 코드를 사용하였다. 이 긴 코드를 간략히 설명하자면 $A \subset B$, $B \subset A$ 이면 $A = B$ 이므로, $A \subset B$, $B \subset A$ 인지 확인하는 것이다.

미사용 메서드는 설계 시에 필요할 것 같아 넣었지만, 막상 사용하지 않은 메서드이다. 이 메서드를 사용하지는 않았지만, 내부 구현은 전부 되어 있다. 따라서 다음에 Picture 필터링같이 기능 추가가 필요하면, 별도의 코딩 없이 modifier를 public 또는 default로 바꿔 주기만 하면 이를 바로 사용할 수 있다. 이 메서드를 private로 선언한 이유도 다른 게 아니라 아직 이를 활용하지 않아서 우선 설정한 것일 뿐, 나중에 이 메서드를 쓸 일이 있다면 modifier를 변경해도 문제가 없다.

(수업 시간에 modifier를 우선 private로 설정한 후 상황에 따라 modifier를 변경하라고 하셨는데, 아직 변경할 상황이 오지 않았을 뿐이다.) 이미 Picture 클래스를 코딩하는 시점에서 미사용 메서드는 필요가 없을 것이라는 예측도 하긴 했지만, 어차피 다음 과제에서 기능을 추가하다 보면 필요해지게 되므로, 굳이 지우지 않고 코드를 완성하였다. 밑에서 설명하겠지만 테스트 시에 미사용 메서드까지 정상적으로 동작하는지 확인하기 위해 한정자를 임시로 바꾸어 테스트도 진행하였다. 단 미사용 생성자는 앞으로 사용할 일이 있을지는 의문이며, 해당 생성자 설명에도 써 놓았듯 처음에는 그 생성자를 사용할 예정이었다가 계획이 변경된 것이다. 메서드 오버로딩의 장점 중 하나인, 나중에 계획이 변경되어 미사용 메서드로 남겨 두어도 문제없다는 점을 활용하기 위해, 그리고 만에 하나 계획이 다시 변경되어 이 생성자가 쓸 일이 필요할 수도 있기에, 굳이 신경 쓰지 않고 놔두었다. (다만 이 생성자도 Eclipse의 코드 자동 생성 기능을 이용해 제작하였으므로 삭제한 후 필요할 때 다시 만드는 건 아주 어렵지 않다.)

B. PictureList 클래스 설계

PictureList 클래스는 다음과 같이 설계하였다.

PictureList			
modifier	자료형	인스턴스명	설명
변수			
private	Picture[]	pictureList	Picture 객체의 배열
생성자			
default		PictureList()	아직 Picture 객체에 아무것도 없이 배열만 생성할 때 사용 (이후 클래스 내 다른 메서드를 이용해 Picture 추가 가능), 다만 오류를 막기 위해 PictureList에 0개짜리 배열을 넣어주어 다른 메서드 실행 시 NullPointerException을 막음
default		PictureList(String)	필수 요구사항으로, 파일 이름을 받으면 그 파일에서 Picture 객체들의 정보를 읽어 pictureList에 저장하는 생성자, 다만 실제 작업은 savePictureByReadingFile(String) 메서드를 이용함
default (미사용)		PictureList(Picture[])	pictureList 변수에 저장할 값을 주면 그 값을 pictureList에 저장하는 생성자, 깊은 복사 시에 사용이 목적이나 아직은 사용하지 않음
메소드			
private	void	savePictureByReadingFile(String)	파일 이름을 받으면 그 파일에서 Picture 객체들의 정보를 한 줄씩 읽어 pictureList에 저장, 다만 파일 열기와 한 줄씩 읽기, FileNotFoundException 처리만 여기서 하고 읽은 문자열은 savePictureByString(String) 메서드에서 시행

private	void	savePictureByString(String)	Picture 객체 하나의 정보를 String으로 받아 Picture 객체를 생성해 pictureList에 저장함. 다만 문자열을 분리해 Picture의 변수로 만드는 과정은 Picture 생성자 내에서, pictureList에 저장하는 것은 addPicture(Picture) 메서드에서 처리하며, 여기서는 Picture 생성 및 pictureList에 추가하는 중 발생하는 각종 예외를 처리함
public	void	savePictureToFile(String)	권장 요구사항으로, 위 두 메서드와는 반대로 pictureList에 저장된 Picture 정보를 파일에 저장하는 메서드, 다만 Picture 객체를 문자열로 바꾸는 과정은 Picture.toString() 메서드를 이용함
public	int	size()	필수 요구사항으로, pictureList에 저장된 Picture의 개수를 반환
public	Picture	get(int)	필수 요구사항으로, pictureList의 i index의 객체를 반환(i번째가 아니므로 -1을 해 주지 않음)
public	void	add(Picture)	필수 요구사항이나 아래의 addPicture(Picture)와 인자, 성능이 모두 같아 addPicture(Picture)에서 모든 것을 처리함
private	void	addPicture(Picture)	인자로 받은 Picture 객체를 pictureList에 저장하는 메서드, 다만 이미 id가 같은 Picture 객체가 있다면 저장하지 않고 예외를 던짐
private (미사용)	void	deletePicture(Picture)	인자로 받은 Picture 객체를 pictureList에서 지우는 메서드. 없는 Picture 객체를 지우려 하면 예외를 던지고, 해당 Picture의 Stuff는 StuffList에서 삭제되지 않음. 아직은 Picture 삭제를 하지 않으니 사용하지 않음
public	void	sortByDate()	필수 요구사항으로, pictureList에 저장된 객체를 날짜순으로 정렬함, 이때 선택 정렬을 이용했고, 정렬 순서는 Picture.compareToByTime(Picture) 메서드 이용
private (미사용)	void	sortById()	pictureList에 저장된 객체를 id 순으로 정렬함, 이때 선택 정렬을 이용했고, 정렬 순서는 Picture.compareToById(Picture) 메서드 이용, 아직 정렬은 시간순만 요구하였으므로 사용하지 않음
public	String	@Override toString()	PictureList 객체를 문자열로 바꾸는 메서드로, pictureList에 저장된 Picture 객체 전부를 문자열로 바꾸는 과정이 필요한 모든 상황에서 사용하는 메서드.

표 1.3

savePictureByString(String) 메서드는 문자열로부터 Picture 객체 하나를 만들어 pictureList에 추가하는 메서드로 Pictrue 생성자를 호출하는 역할도 하지만, 앞에서 언급했던, 생성자에서 던져지는 각종 예외를 처리하는 역할도 한다. 처음에는 문제상황을 콘솔에 보고할 때 e.printSlackTrace() 메서드를 사용하려 했으나 결과를 보니 어떤 줄에서 에러가 발생했는지 확인하기도 어렵고, 콘솔에 출력된 것만 보면 에러를 콘솔에 보고한 건지, 에러가 나서 프로그램이 종료되었는지 결과를 보는 사람 입장에서는 구분할 수 없다는 문제를 발견하였다. 따라서 catch 문에서 콘솔에 출력하는 코드를 변경하였다. 이는 savePictureByReadingFile(String) 메서드에서 FileNotFoundException 도 같은 것이다.

여기서도 미사용 메서드가 있으나, Picture 클래스와는 달리 코드도 짜지 않은 메서드도 있다. 이는 위에서도 설명했듯 그 부분은 다음에 ArrayList를 도입하면 전부 뜯어고쳐야 하기 때문이다. 이번에 사용하지도 않고 나중에 이 메서드를 사용할 일이 있을 때도 활용하지 못하는 무용지물 메서드는, 구현한 다른 미사용 메서드와 달리 추후 기능 추가에 유용하다는 장점이 없다고 봐도 무방하다. 의미가 전혀 없는 허튼짓을 여기서 할 이유는 없다고 판단하여, 코드를 구현하지 않고 메서드 선언도 주석처리했다. (구현하지 않았기에 코드만 남기고, 위의 설계 표에서도 제외했다.) 이외에 나중에 유용하게 쓸 것으로 보이는 메서드는 전부 구현하였으며, 다만 아직 외부에서 사용하지 않기에 우선 private로 지정하였다.

C. Stuff 클래스 설계

Stuff 클래스는 다음과 같이 설계하였다.

Stuff			
modifier	자료형	인스턴스명	설명
변수			
private	String	id	id(String 형)
private	String	type	type(String 형)
private	String	name	name(String 형)
private	Tag	tags	태그('#'으로 시작하는 String의 배열이나 클래스화함)
생성자			
default		Stuff(String)	Stuff의 내용을 하나의 문자열로 받아, Stuff를 구성하는 각 변수로 나누어 Stuff 객체를 생성하는 생성자, 이때 id는 생성하지 않음
default (미사용)		Stuff(String, String, String)	Stuff의 type, name, tags를 String 형으로 받아, id를 제외한 모든 변수로 나누어 Stuff 객체를 생성하는 생성자

default (미사용)		Stuff(String, String, String, Tag)	Stuff의 각 변수를 하나하나 받아 객체를 생성하는 생성자, 원래 StuffList에서 분리 작업을 모두 끝낼 예정이었으나 코드 작성 중 계획이 바뀌어 사용하지 않게 됨
메소드			
default	boolean	equalByTypeAndName(Stuff)	이 객체와 Stuff의 type, name이 모두 같은지 비교하는 메서드로, 새로 만든 Stuff를 기존과 같은 id를 부여할지 다른 id를 부여할지 결정할 때 사용
default	boolean	equalExceptId(Stuff)	이 객체와 인자로 받은 객체가 id를 제외하고 같은지 확인하는 메서드로, id 외 3개가 모두 같으면 StuffList에도 추가하지 않는데 이때 확인을 위해 사용
default	int	compareTo(Stuff)	이 Stuff 객체와 다른 Stuff 객체 중 id를 기준으로 뭐가 크고 뭐가 작은 지 비교하는 메서드, 이 객체에 비해 비교 대상이 크면(시간이 나중이면) 음수, 작으면 양수, 같으면 두 tags의 비교 결과를 반환하며(이 메서드는 Tag.compareTo(Tag)에서 자세히 설명한다), 이는 String.compareTo() 함수의 반환값임. StuffList를 정렬할 때 사용함
public	String	@Override toString()	Stuff 객체를 문자열로 바꾸는 메서드로, Picture 객체, StuffList 등을 문자열로 바꾸는 과정이 필요한 모든 상황에서 사용하는 메서드
public	boolean	@Override equals(Object)	이 객체가 인자로 주어진 객체와 같은지 확인해 그 결과를 반환하는 메서드로, id만 일치하면 같은 것으로 보고 true 반환, 그렇지 않으면 false 반환. (이미 equalByTypeAndName(Stuff) 메서드와 setId 메서드로 같은 객체로 취급하는 것은 같은 id를, 다른 객체로 보는 것은 다른 id를 부여받았다는 점을 이용하면, id만 비교해도 같은지, 다른지 확인할 수 있다. id를 부여받기 전 객체와 같은 게 있는지 비교하는 것은 이 메서드가 아닌 위 equalByTypeAndName(Stuff) 메서드를 이용한다.) 각각의 변수가 일치하는지는 각 자료형의 equals 메서드를 이용함
default	void	setId(String)	생성자에서 설정하지 않은 id를 설정하는 setter로, id 값이 없는 경우에만 id를 설정하고 그렇지 않으면 id를 설정하지 않음
default	void	setId(Stuff)	생성자에서 설정하지 않은 id를 주어진 Stuff 객체와 동일하게 설정하는 setter로, id 값이 없는 경우에만 id를 설정하고

			그렇지 않으면 id를 설정하지 않음

표 1.4

생성자에서 문자열을 분리하기 전에 공백을 더하는데, 마지막 tag가 생략된 상태에서 split을 진행하면, tag에 해당하는 부분이 아예 나오지 않아(빈 문자열로 나오는 게 아니라, 아예 문자열의 개수가 하나 적게 나온다) 잘못된 입력으로 처리하는 상황을 막기 위함이다. 테스트 중에 이런 문제가 나오는 것을 발견해 수정했다. 분리된 문자열이 하나 적게 나오는 상황에 대해 처리하는 것보다 공백문자 하나를 더해주고 split 하는 게 훨씬 더 간단하여 이렇게 진행하였다.

Stuff 객체는 특성상 기존과 type, name이 같은, 다른 Stuff 객체가 있다면 그 객체와 동일한 id를 받게 된다. 따라서 미사용 생성자를 제외한 다른 생성자에서는 id 값을 넣어주지 않고 null이 들어가게 한다. (즉 들어오는 문자열에서 id 부분은 사용하지 않고 버린다.) 다만 Stuff 객체를 만드는 다른 메서드에서 Stuff를 생성한 뒤 그 객체를 대상으로 반드시 setId 메서드를 실행하도록 코드를 짜 NullPointerException을 막았다. (id는 생략할 수 있는 부분도 아닌 데다가, 생략할 수 있다 해도 앞에서 언급했듯 null을 넣어주는 게 아니라 빈 문자열을 만들어서 넣어준다. 따라서 id 값에 null을 넣고 그대로 내버려두는 행위는 절대 일어날 수 없다.)

D. StuffList 클래스 설계

StuffList 클래스는 다음과 같이 설계하였다. 프로그램 내에서 하나의 StuffList가 공유될 것을 요구하였으므로, 클래스의 모든 변수와 메서드에 static을 붙였다. 따라서 다른 클래스에는 다 있는 toString(), equals(Object) 메서드는 둘 다 없다. 또한 생성자도 없고, 변수는 변수 선언문에 직접 값을 대입하여 초기화하였다. 다른 클래스와의 설명 통일을 위해 아래 표에서 변수와 메서드를 인스턴스라고 부르나 사실 인스턴스는 아니다.

StuffList			
modifier	자료형	인스턴스명	설명
변수			
Private static	Stuff[]	stuffList	현재 저장된 모든 Stuff 객체가 저장된 배열
Private static	int	stuffId	새로운 Stuff 객체가 생길 때 부여할 id 저장(첫 번째 Stuff의 id를 1부터 차례대로 부여하며, 이번에 부여할 id를 저장하는 변수)
모든 인스턴스에 static이 붙었기 때문에 생성자는 없음			
메소드			
default static	Stuff[]	newStuffs(Stirng)	Stuff 여러 개를 하나의 문자열로 받으면, 이를 여러 개의 Stuff로 분리하여 Stuff 객체 여러 개를 생성해 stuffList 배

			열에 저장하고, 이번에 새로 저장한 Stuff 객체의 배열을 반환하는 메서드
private static	Stuff	ifStuffExistByNameAndType(Stuff)	stuffList에 name과 type이 같은 Stuff가 있는지 확인하는 메서드로, 아직 id가 부여되지 않은 Stuff가 기존과 같은 id를 받아야 하는지, 받는다면 무슨 id를 받아야 하는지 알아내기 위해 사용
private static	Stuff	ifStuffExistExceptId(Stuff)	stuffList에 name과 type은 물론 tag까지 같은 Stuff가 있는지 확인하는 메서드로, stuffList 배열에 완전히 같은 객체가 2개 이상 들어가는 것을 막기 위해(2개 이상 들어가면 stuffList 클래스에 쓸데없는 중복이 많아져, print() 메서드를 사용할 때 문자열의 길이가 너무 길어져 한눈에 보기가 힘들다) 사용하는 메서드
default static	Stuff	addStuff(String)	stuffList에 추가할 새로운 Stuff 객체를 문자열로 주면, 이를 Stuff 객체로 만들어 배열에 추가한 뒤, 이렇게 만들어진 Stuff 객체를 반환하는 메서드
private (미사용) static	void	sortById()	stuffList에 저장된 Stuff를 정렬하는 함수, 정렬 순서는 Stuff.compareTo(Stuff) 메서드를 이용함
public static	void	print()	필수 요구사항으로, 저장된 모든 Stuff를 콘솔에 출력함, 단 여기서는 콘솔 출력만 하고, 문자열로 바꾸는 과정은 allStuffToString()에서 진행함
private static	String	allStuffToString()	다른 클래스의 toString()에 해당하는 메서드로, stuffList에 저장된 모든 Stuff를 하나의 문자열로 만들어 반환

표 1.5

원래 Picture의 stuffList를 만드는 과정은 이 클래스와는 상관없는 일이므로 이 클래스에서 취급하는 게 옳지는 않다. 그러나 어차피 Picture에 들어간 모든 Stuff 객체는 StuffList에도 추가하여야 하므로, 편의상 여기서 시행하였다. 그러니까 newStuffs(String) 메서드는 문자열 하나에 입력된 Stuff 여러 개를 클래스로 만들어 StuffList에 저장하고, 이때 생성한 Stuff 객체의 배열을 반환하는데, 이 반환값은 C 언어의 scanf 함수의 반환값처럼, StuffList 클래스 입장에서는 별 의미가 없는 반환값이라 생각하면 된다. 이 의미 없는 반환값을 Picture 클래스가 매우 잘 활용하고 있는 것 정도로 이해하면 된다. Picture 클래스에서 StuffList 생성까지 전부 하지 않는 건 위의 코드 중복 문제도 있고, Stuff를 다루는 것은 Stuff 전체를 모아 놓은 클래스에 묶어 놓는 게 더 좋을 것이라는 판단, 그리고 어차피 각종 작업에서 StuffList 클래스 내 메서드를 많이 쓸 수밖에 없기 때문에, 여러 메서드를 이 과정 때문에 public, default로 돌리는 일 없이 사용하기 위함이다.

newStuffs(String)에서는 Stuff 객체 여러 개를 생성하는데, 이때 기존에 저장된 Stuff 객체와 type, name이 겹치면 기존과 같은 id를, 그렇지 않으면 다른 id를 할당해야 한다. 따라서 Stuff 객체 하나를 생성한 후, id가 기존과 겹치는지 확인하여 겹치면 기존 Stuff와 같은 id를, 그렇지 않으면 다른 id를 넣었다. (Stuff 클래스에서 설명했듯 이 과정을 거치지 않으면 문제가 생긴다.) 다만 기존 Stuff 객체와 tag까지 완전히 겹치면, 새로 생성된 Stuff 객체를 버리고 기존 Stuff 객체(정확히는 그 포인터)를 가져왔다. 이렇게 하는 이유는 StuffList에 tag까지 같은 Stuff가 들어가게 되면, StuffList를 출력할 때 Stuff의 내용이 쓸데없이 많아져 한눈에 보기 힘들다는 점을 고려하였다. 이때는 새로 만들어진 Stuff 객체는 버려지며, Picture 클래스처럼 Java의 Garbage Collector가 알아서 처리할 테니 우리는 신경 쓰지 않아도 된다. (id도 설정하지 않았지만, 어차피 쓰지 않기 때문에 상관없다.)

이전에 들어오지 않은 새로운 Stuff 객체의 id는 1부터 순서대로 할당하였다. 이때 Stuff 클래스의 id는 String 형이고 요구사항의 입출력 예시를 보면 id는 0을 이용해 8자리를 반드시 채웠다는 것 확인할 수 있다. 따라서 Stuff 객체의 id를 부여할 때 1부터 부여하되, String.format 메서드를 이용하여, C언어의 sprintf처럼 정수를 문자열로 변경하였다. 이때 다음에 부여할 id 번호는 stuffId 변수에 저장하였다.

E. Tag 클래스 설계

Tag 클래스는 다음과 같이 설계하였다.

Tag			
modifier	자료형	인스턴스명	설명
변수			
private	String[]	tags	태그(String 형)의 배열, '#'은 빼고 저장함
생성자			
default		Tag(String)	전체 태그를 하나의 문자열로 받아, 이를 각각의 tag로 나누어 저장하는 생성자, 이때 빈 문자열이 들어오면 빈 문자열의 tag 1개가 저장되므로, 빈 문자열이 아닌 경우에만 배열에 저장함
메소드			
default	boolean	ifTagExist(String)	인자로 받은 String이 tags 중에 있는지 확인하는 메서드
default	boolean	ifTagExist(Tag)	인자로 받은 Tag 객체에 있는 String 모두가 이 객체의 tags에 속해 있는지 확인하는 메서드
default	int	getLength()	tags에 저장된 문자열의 개수를 반환하는 메서드
default	void	addTag(Stri	인자로 받은 String을 tags에 추가하는 메서드

		ng)	
default	void	deleteTag(String)	인자로 받은 String을 tags에서 삭제하는 메서드, 인자로 받은 String이 tags에 없다면 예외를 던짐
default	int	compareTo(Tag)	이 Tag 객체와 다른 Tag 객체 중 뭐가 크고 뭐가 작은지 비교하는 메서드, 기준은 태그 길이, 이게 같으면 각 문자열을 앞에서부터 비교해 String.compareTo(String) 메서드의 결괏값을 따름. 이 객체에 비해 비교 대상이 크면(시간이 나중이면) 음수, 작으면 양수, 같으면 0을 반환함. StuffList를 정렬할 때 Stuff 객체의 id가 같은 경우 사용함
public	String	@Override toString()	tag 전체를 문자열로 바꾸는 메서드로, Picture 객체, Stuff 객체, Image 객체를 문자열로 바꾸는 상황에서 사용하는 메서드
public	boolean	@Override equals(Object)	이 객체가 인자로 주어진 객체와 같은지 확인해 그 결과를 반환하는 메서드로, 저장된 문자열이 순서 없이 들어있는 것만 일치하면 같은 것으로 보고 true 반환, 그렇지 않으면 false 반환, 순서에 상관없이 비교하기 위해 주어진 객체에 있는 문자열이 이 객체에 있는지 먼저 비교한 후, 이 객체에 있는 문자열이 다른 객체에 있는지 비교함. String이 일치하는지는 String.equals() 메서드를 이용함

표 1.6

말 그대로 tag를 모아놓은 것이다. 추가, 삭제, 검색 기능을 제외하면 단순한 문자열의 배열이다. Picture, Image, Stuff 셋 모두 tag를 사용하는데, 이 세 클래스 모두 이 클래스를 이용한다. 이 클래스는 이 외 특별한 건 없다.

F. Image 클래스 설계

Image 클래스는 다음과 같이 설계하였다.

Divisors			
modifier	자료형	인스턴스명	설명
변수			
private	String	imageId	Image Id(String 형)
private	String	imageFileName	Image 파일 이름(String 형)
private	String	imageName	Image name(String 형)
private	Tag	tags	태그('#'으로 시작하는 String의 배열이나 클래스화함)

생성자			
default		Image(String)	사진 정보를 하나의 문자열로 받아, 이를 imageId, imageFileName, imageName, tags 로 각각 나누어 저장하는 생성자
메소드			
public	String	@Override toString()	Image 정보 전체를 문자열로 바꾸는 메서드로, Picture 객체를 문자열로 바꾸는 상황에서 사용하는 메서드
public	boolean	@Override equals(Object)	이 객체가 인자로 주어진 객체와 같은지 확인해 그 결과를 반환하는 메서드로, imageId, imageFileName, imageName, tags가 모두 같으면 true 반환, 그렇지 않으면 false 반환, 각각의 변수가 일치하는지는 각 자료형의 equals 메서드를 이용함

표 1.7

생성자에서 문자열을 분리하기 전에 공백을 더하는 것은 위 Stuff 클래스에서 설명한 것과 같은 이유이다.

아직은 이 클래스의 자세한 용도를 이해하지 못해 toString()과 equals(Object) 메서드만 넣었다. 나중에 설계상으로 필요하다면 그때 추가할 계획이다.

2. 테스트 결과

이렇게 만들고 테스트를 진행하였다. 각각 테스트를 진행한 때가 전부 다르기 때문에, 그림의 시간 순서는 뒤죽박죽이 될 수 있다. 또한 autotest에서 '<'과 '>'이 html의 요소로 처리되어 제대로 보이지 않는 문제가 있어, 자체 테스트 진행 후 autotest 진행 전 소스코드를 약간 수정했다. (autotest 결과 화면의 html을 확인해 보면 정상 출력되었지만, 이게 html을 해석하는 방법에 따라 표시되는 화면에서는 보이지 않게 된다. 참고를 위해 autotest가 제대로 보이지 않을 때, 즉 소스코드를 수정하지 않았을 때의 html도 첨부한다.) 따라서 자체 테스트와 autotest의 결과가 약간 다를 수 있다. (다만 수정한 게 큰 게 아니라 단순히 공백 문자를 몇 개 추가한 것뿐이다.)

A. 자체 테스트 결과

우선 테스트에 사용한 picture-normal.data에 저장된 텍스트 파일은 다음과 같다.

```
// Picture information
< m_2024-01-03_06:30:00:000        > < 2024-01-03_06:30:00:000 > < IMG2024-01-07_15:28:42:083; images/morning-excercise.jpg; ;    > <    [00000002; exercise; ; #morning ]> < #happy > < new year >
```

```
<m_2024-01-03_17:00:00:000 > < 2024-01-03_17:00:00:000 > < IMG2024-01-07_15:37:38:009; images/dinner.jpg; ; > < [00000005; food; dinner; #korean ]> < #friend #meet > < good restaurant >
< m_2024-03-03_14:00:00:000 > < 2024-03-03_14:00:00:000 > < IMG2024-01-07_15:50:35:480; images/classroom.jpg; ; > < [00000007; room; class meeting; #study ]> < #class #lecture > <>
< m_2024-03-03_14:20:00:000 > < 2024-03-03_14:20:00:000 > < IMG2024-01-08_19:12:57:384; images/eclipse.jpg; ; > < [00000012; study; eclipse; #java ]> <> <>
< m_2024-03-03_15:00:00:000 > < 2024-03-03_15:00:00:000 > < IMG2024-01-07_15:52:27:605; images/locker.jpg; ; > < [00000008; storage; locker; ] [00000009; ; basket; ] [00000010; ; clock; ]> < #school #locker > < my locker >
< m_2024-02-03_09:30:00:000 > < 2024-02-03_09:30:00:000 > < IMG2024-01-08_13:29:53:625; images/wrench.jpg; ; > < [00000004; tool; wrench; #8inch #adjustable ]> <> <>
< m_2024-01-03_10:00:00:000 > < 2024-01-03_10:00:00:000 > < IMG2024-01-07_15:33:00:827; images/toolbox.jpg; ; > < [00000003; toolbox; my toolbox; #red ] [00000004; tool; wrench; ]> < #home #clean > <>
// end of Picture information
```

앞에서도 간략하게 설명했듯 전체적인 테스트 후 미사용 메서드의 테스트를 진행했다. (단 미사용 생성자는 테스트를 진행하지 않았다.) 요구사항 테스트만 확인하려는 경우 맨 앞의 그림만 확인하면 된다.

사진을 여럿 붙여놓으면 가독성만 떨어뜨릴 것 같아, 테스트 결과는 하나만 넣고 다른 테스트 결과와의 차이점 위주로 설명하였다. 어차피 입력 데이터는 같으므로 각 테스트의 결과가 크게 다르지 않아 차이점을 비교하는 것만으로 메서드 동작 입증은 충분히 가능하다.

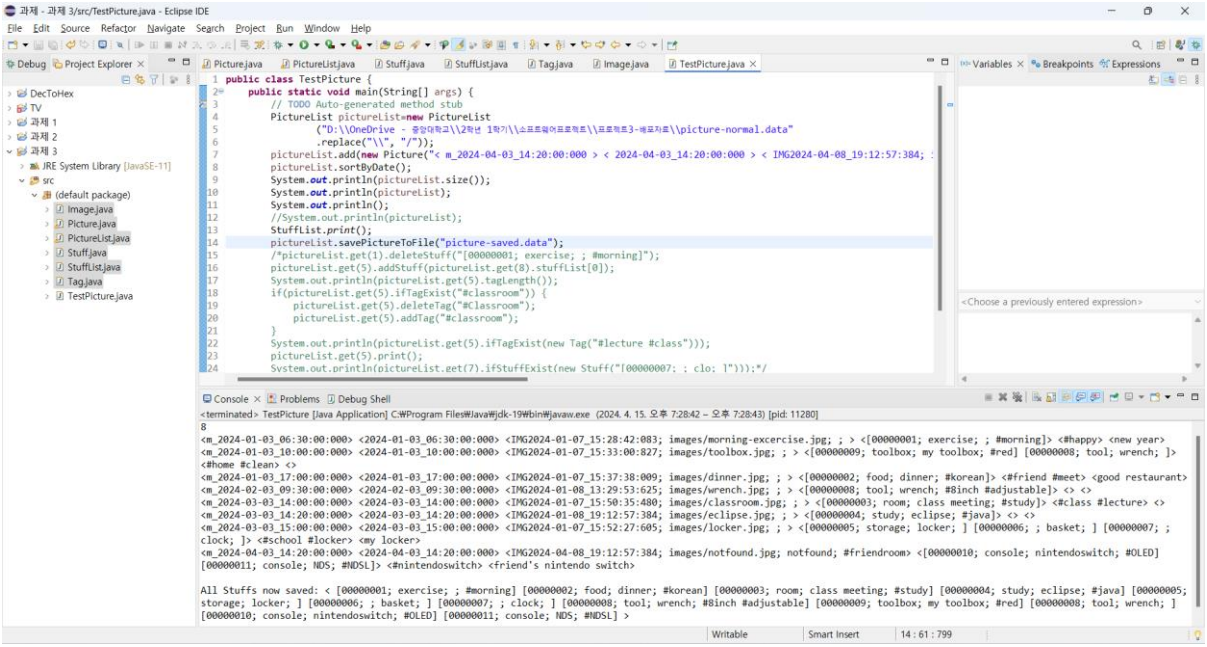

그림 2.1

위 그림 2.1은 전체적인 테스트로, 요구사항에 대한 것만 다룬 결과이다. 순서대로 pictureList.size(), pictureList 전체, StuffList.print()를 출력한 것이다. PictureList.add(Picture) 메서드를 포함해 **전체적으로 정상 작동**함을 알 수 있다. 이때 파일에 데이터를 저장한 결과는 그림 2.2와 같으며, 콘솔에 출력된 것과 결과가 같다.

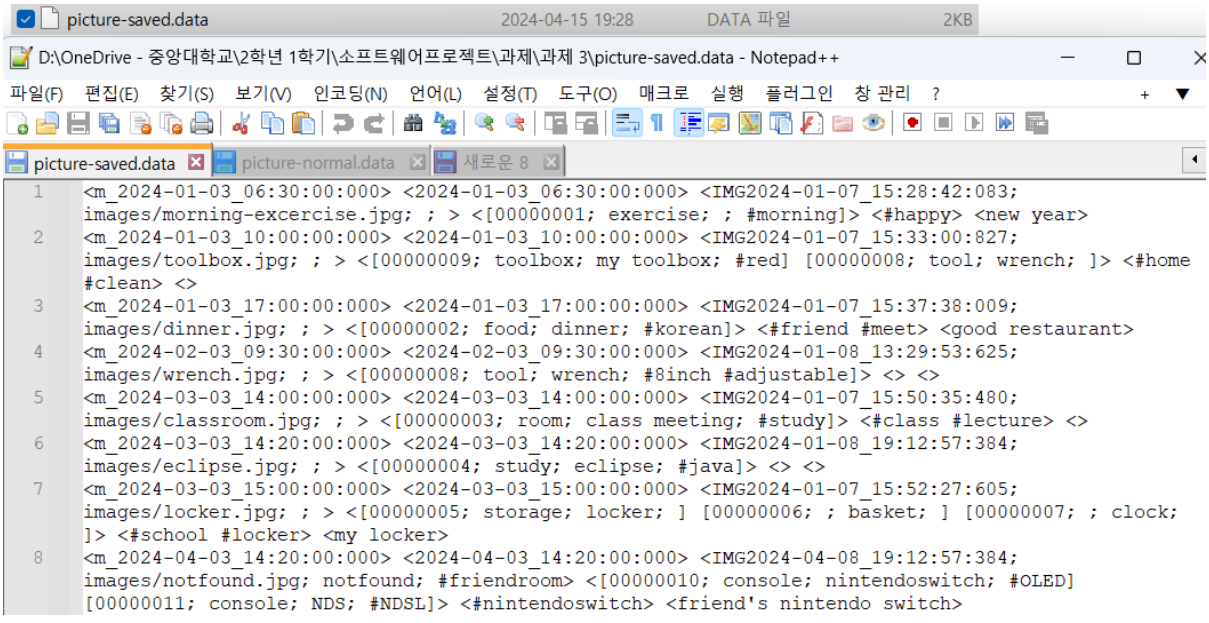

그림 2.2

아래 그림 2.3은 Picture 클래스를, 미사용 메서드를 포함해서 테스트한 결과이다. 먼저 정상 동작이다.

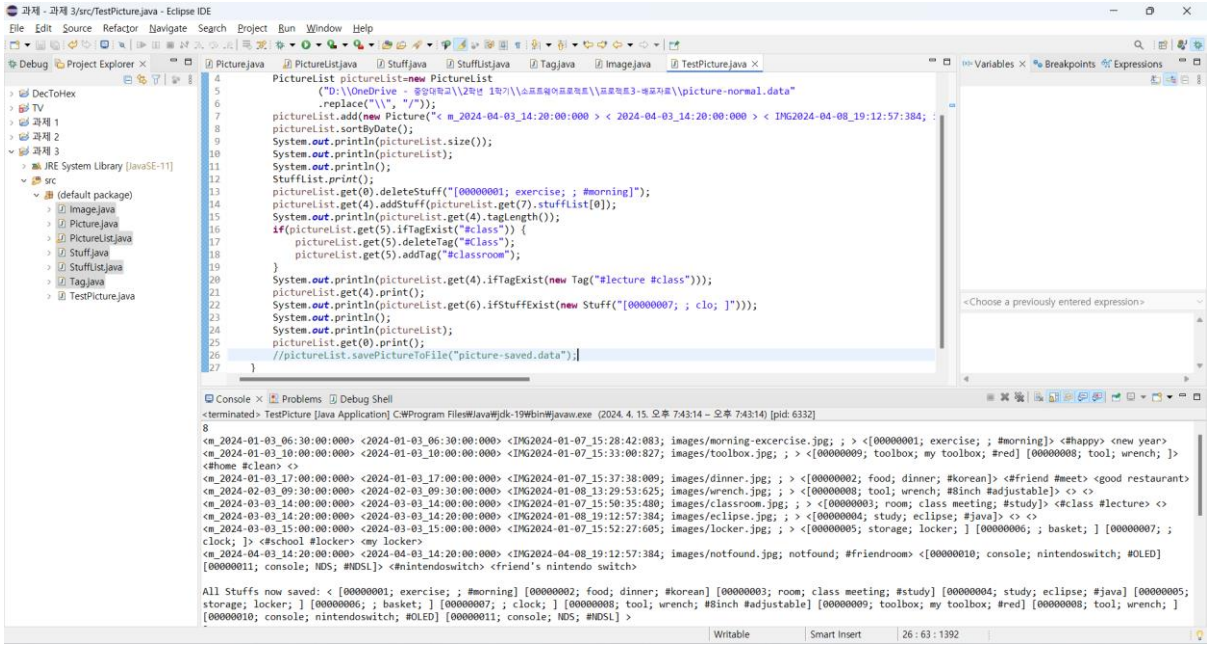

그림 2.3

코드는 위 그림과 같으며, 코드와 결과를 한 화면에 전부 담기 힘들어 결과는 아래 그림 2.4에 담았다.

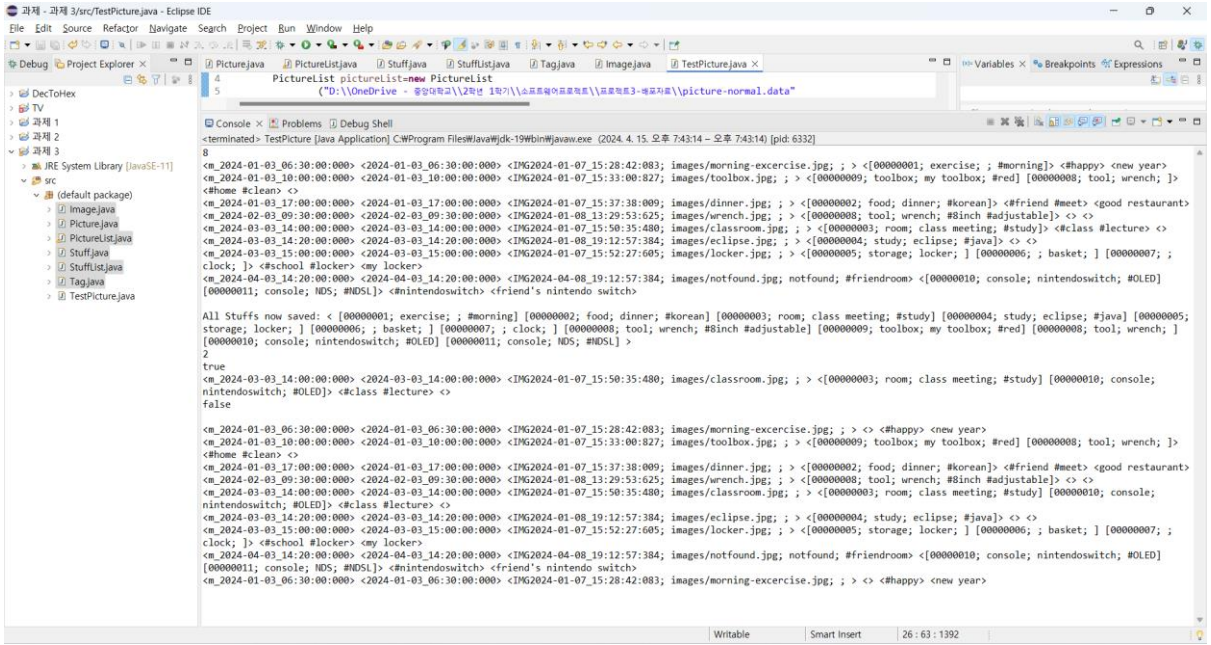

그림 2.4

맨 앞 숫자 8은 pictureList.size()의 결과이다. 그다음은 중간 부분의 코드 실행 전 현재의 pictureList를 출력한 것이고(단 정렬이 앞 코드에 있기에 정렬은 되어 있다), 이후 현재의 StuffList를 출력한 것이다. 이후 pictureList[0]의 Stuff 한 개를 지우고, pictureList[4]의 Stuff에 pictureList[7]의 첫 번째 Stuff를 추가하였다. 이후 pictureList[4]의 태그 개수를 출력하여, 콘솔에 2가 출력되었

다. (이 시점에서 pictureList[4]의 태그는 #class #lecture로 2개이다.) 다음으로 PictureList[5]에 class 태그가 존재하면 class 태그를 지우고 classroom 태그를 넣었다. (다만 조건이 틀렸으므로 태그 변환은 일어나지 않는다.) 그리고 pictureList[4]에 #lecture #class 태그 2개가 모두 존재하는지 확인 결과를 출력해 true가 출력되었고 (아직 pictureList[4]의 태그는 #class #lecture이다. IfTagExist(Tag) 코드가 tag 순서를 바꿔도 정상 작동하는지 확인하기 위한 코드이다) pictureList[6]에 어떤 Stuff 객체 하나가 있는지 없는지 결과를 출력하였다. (없으므로 false가 출력된다.) 이후 이렇게 코드를 거친 후 pictureList 전체를 출력한 뒤 프로그램을 종료했다. 위와 같이 **정상 동작**함을 알 수 있다.

이후는 없는 객체를 delete 하려 할 때 예외가 잘 발생하는지 테스트한 것이다.

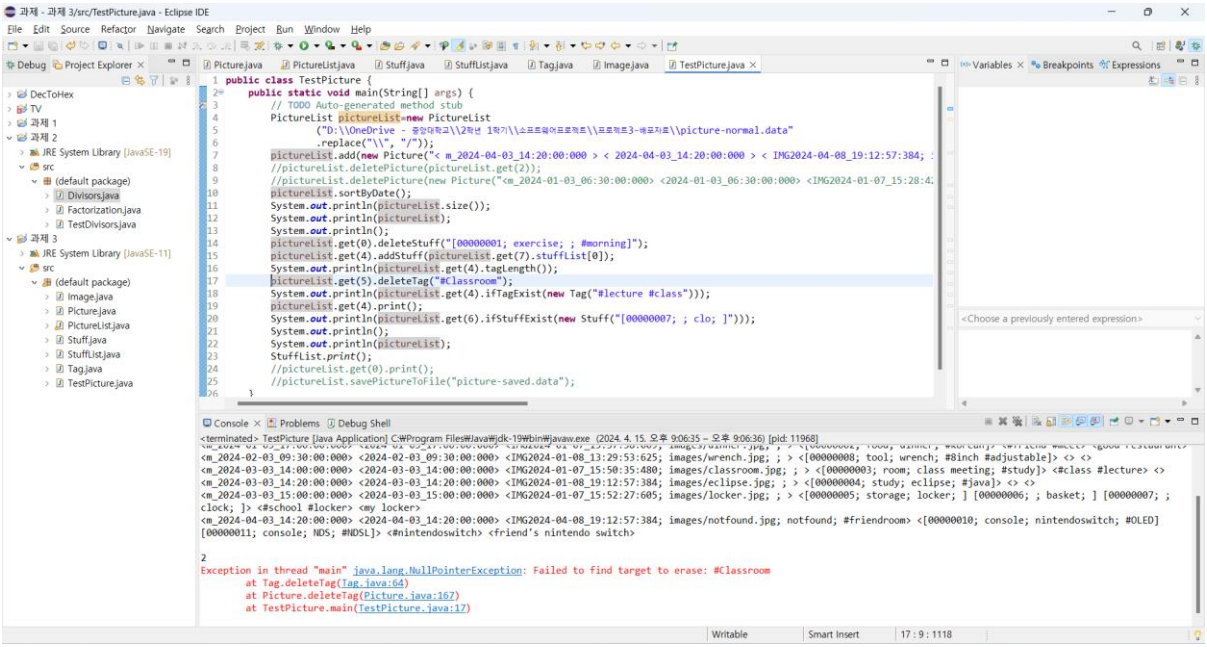

그림 2.5

위 그림 2.5는 없는 태그를 지우려 할 때 예외가 발생하는지 테스트한 것이다. 정상적으로 예외가 출력됨을 확인할 수 있다. 별도의 예외처리는 진행하지 않았으므로 프로그램이 죽는 것 역시 **정상 동작**이다.

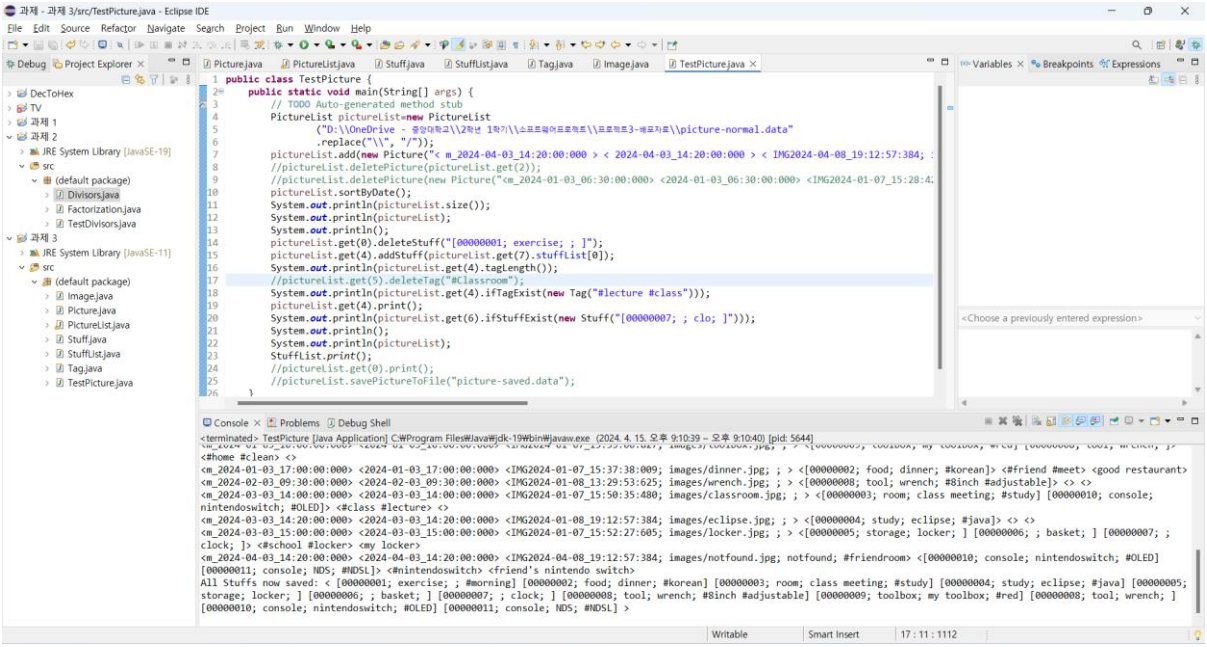

그림 2.6

위 그림 2.6은 위 그림에서 태그 삭제를 지운 것이다. 예외가 던져지지 않고 프로그램이 실행되는 것을 보아 위의 예외는 정확히 없는 태그를 지우려 했기 때문에 발생했음을 알 수 있다.

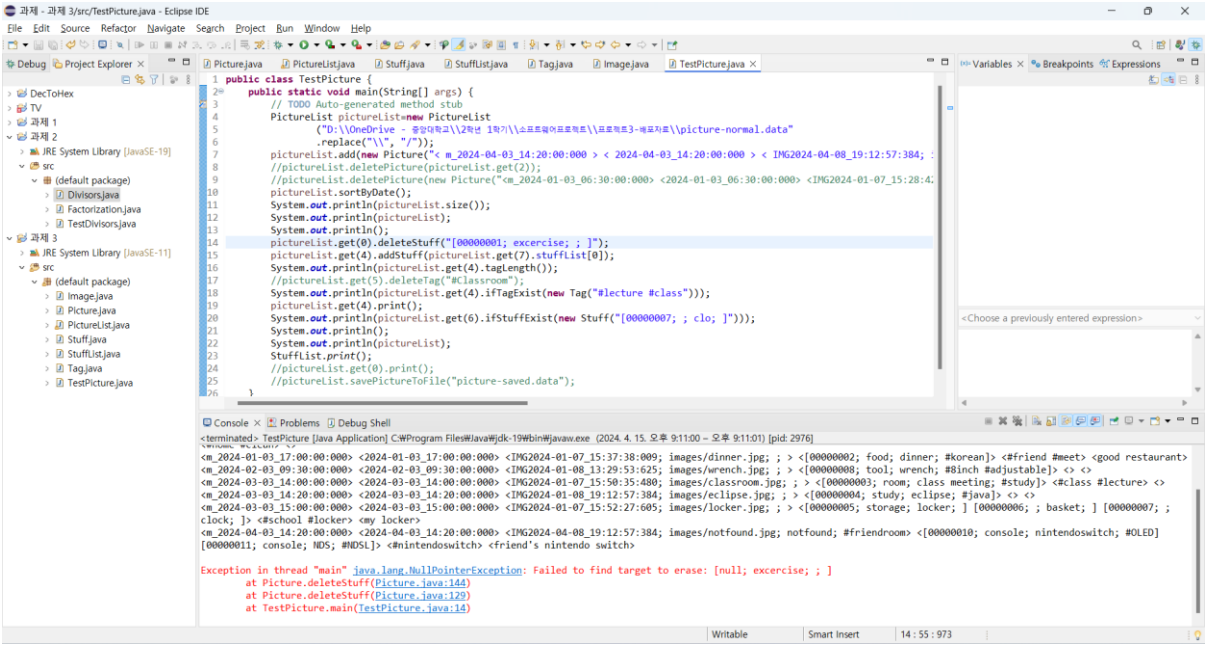

그림 2.7

위 그림 2.7은 없는 Stuff를 지우려 할 때 예외가 발생하는지 테스트한 것으로, 위 그림에서 Stuff의 철자 하나만 변경한 것이다. 정상적으로 예외가 던져짐을 알 수 있다.

아래 그림 2.8은 PictureList 클래스의 미사용 메서드를 테스트한 결과이다.

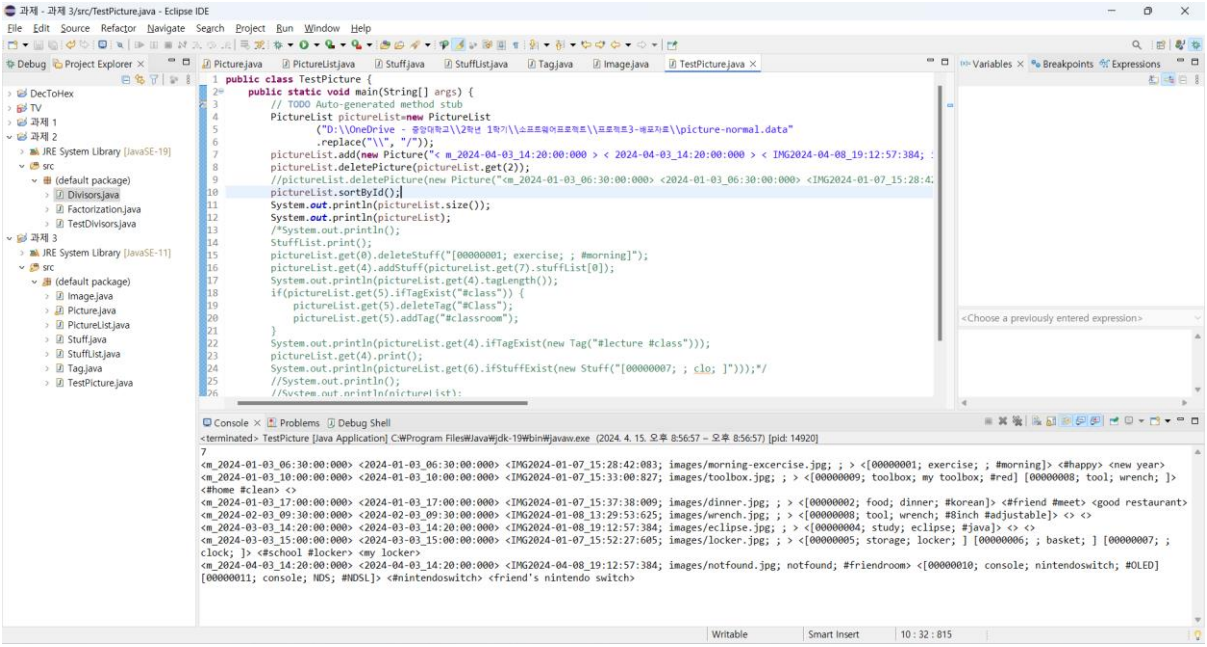

그림 2.8

위 그림 2.8은 sortById() 메서드와 deletePicture(Picture) 메서드를 테스트한 결과이다. 아직은 id 순서와 timeStamp의 순서가 같으니 정렬 결과는 날짜순 정렬과 같다. 또 deletePicture(Picture) 메서드에서 index 2의 사진 정보가 삭제되었다는 점도 확인할 수 있다. 정렬 전 index 2의 사진 정보는 "< m_2024-03-03_14:00:00:000 > < 2024-03-03_14:00:00:000 > < IMG2024-01-07_15:50:35:480; images/classroom.jpg; ; > < [00000007; room; class meeting; #study]> < #class #lecture > <>"인데, 앞의 파일 정보나 앞의 다른 그림과 비교해 이게 없다는 점을 확인할 수 있다.

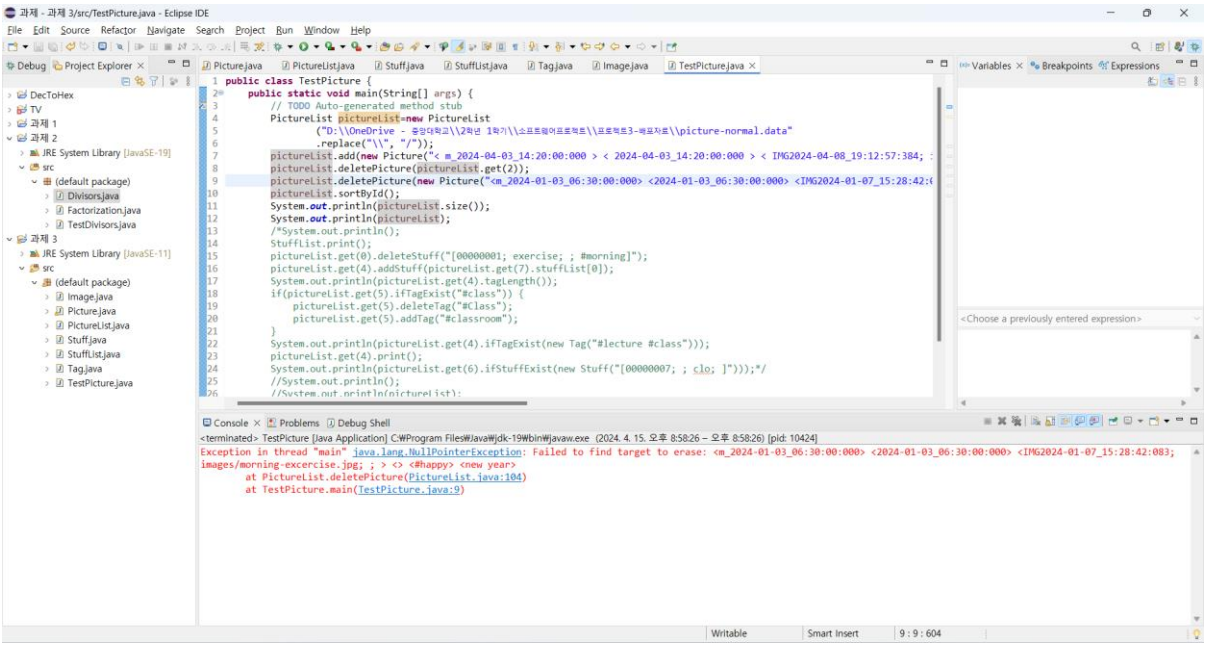

그림 2.9

위 그림 2.9는 deletePicture(Picture) 메서드에서 존재하지 않는 Picture 객체를 삭제하려 할 때 예외를 던지는지 테스트한 것이다. 정상적으로 예외를 던진다는 점을 확인할 수 있다.

마지막으로 StuffList 클래스의 미사용 메서드 테스트이다.

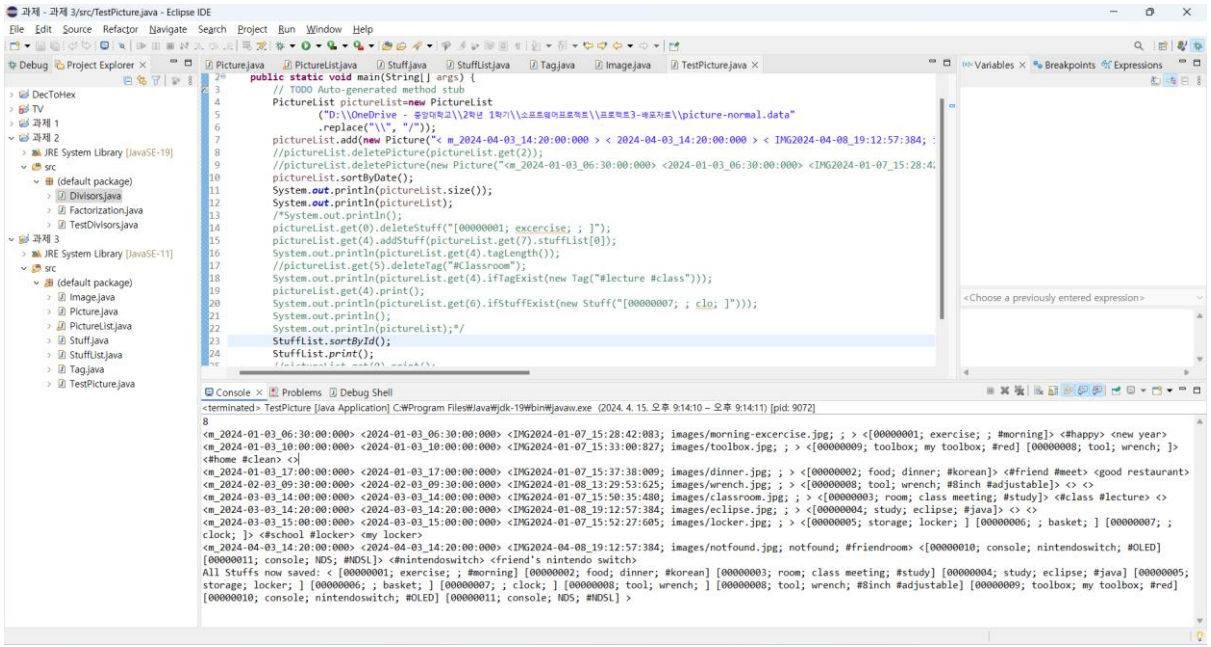

그림 2.10

StuffList 출력이 id 순, id가 같으면 태그 개수가 적은 순서대로 정렬되었음을 확인할 수 있다. 첫 그림과 비교해 id 8인 Stuff 2개 순서 배치가 변경되었음을 알 수 있다.

B. AutoTest 결과

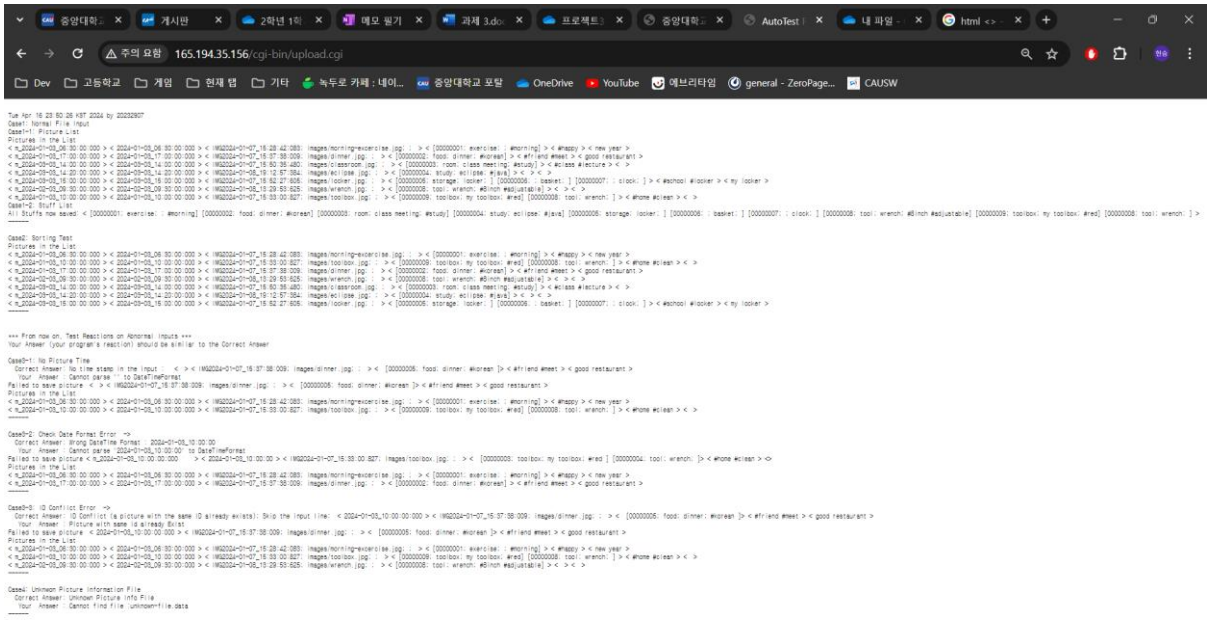

그림 2.11

위 그림 2.11은 **autotest 결과**이다. 앞서 말했듯 html 관련 문제로 공백을 수정한 상태이다. 보다시피 정상적으로 동작한다는 것을 알 수 있다. 위 그림 2.11이 모든 결과를 한 번에 담느라 글씨가 잘 알 보일 것을 우려해 그림 일부를 잘라 확대했다. (그림 2.12)

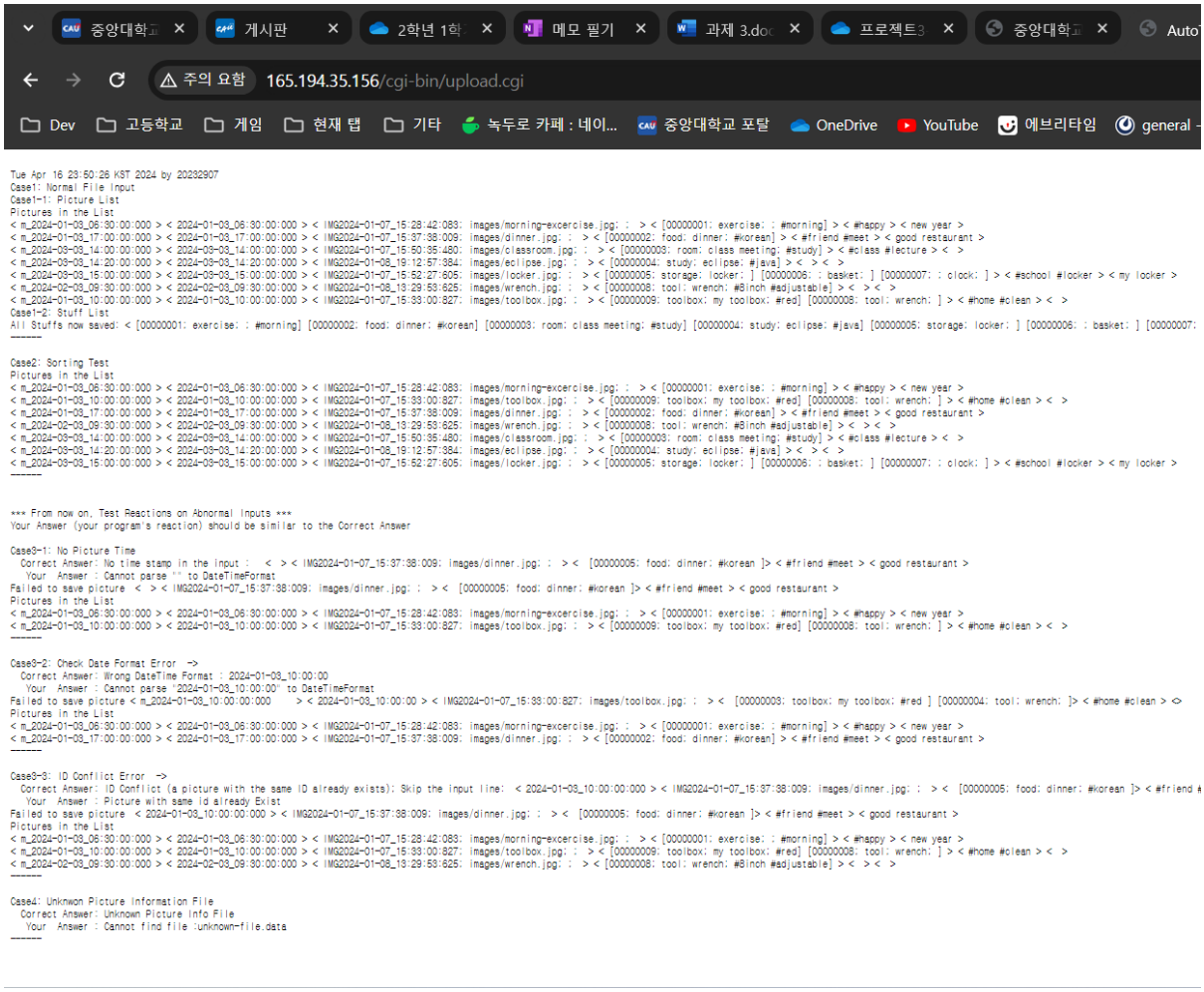

그림 2.12

아래는 공백 수정 전(자체테스트할 때의 코드) 화면에 보이는 autotest 결과와 html 데이터를 그대로 붙여넣은 것이다. 필요 없으면 건너뛰어도 괜찮다. (아래 그림 2.13은 문제가 발생했던 당시가 아니라 이를 나중에 재현한 것이다.)

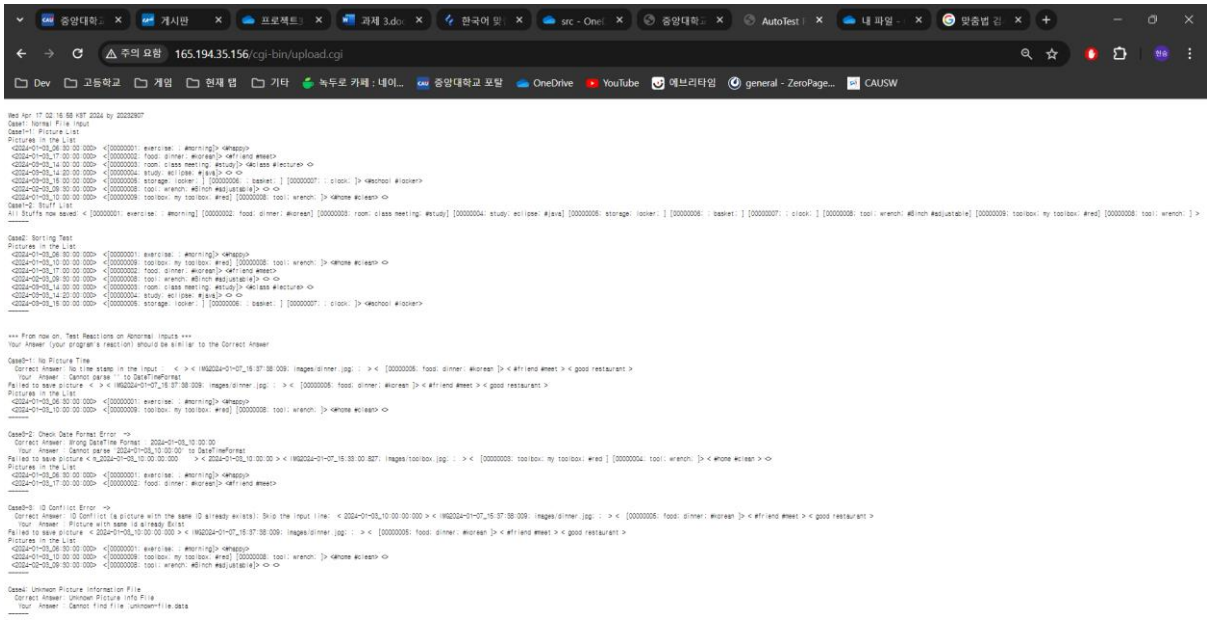

그림 2.13

<pre>
Mon Apr 15 17:31:28 KST 2024 by 20232907
Case1: Normal File Input
Case1-1: Picture List
Pictures in the List
<m_2024-01-03_06:30:00:000> <2024-01-03_06:30:00:000> <img2024-01-07_15:28:42:083; images="" morning-excercise.jpg;="" ;=""> <[00000001; exercise; ; #morning]> <#happy> <new year="">
<m_2024-01-03_17:00:00:000> <2024-01-03_17:00:00:000> <img2024-01-07_15:37:38:009; images="" dinner.jpg;="" ;=""> <[00000002; food; dinner; #korean]> <#friend #meet> <good restaurant="">
<m_2024-03-03_14:00:00:000> <2024-03-03_14:00:00:000> <img2024-01-07_15:50:35:480; images="" classroom.jpg;="" ;=""> <[00000003; room; class meeting; #study]> <#class #lecture> <>
<m_2024-03-03_14:20:00:000> <2024-03-03_14:20:00:000> <img2024-01-08_19:12:57:384; images="" eclipse.jpg;="" ;=""> <[00000004; study; eclipse; #java]> <> <>
<m_2024-03-03_15:00:00:000> <2024-03-03_15:00:00:000> <img2024-01-07_15:52:27:605; images="" locker.jpg;="" ;=""> <[00000005; storage; locker;] [00000006; ; basket;] [00000007; ; clock;]> <#school #locker> <my locker="">
<m_2024-02-03_09:30:00:000> <2024-02-03_09:30:00:000> <img2024-01-08_13:29:53:625; images="" wrench.jpg;="" ;=""> <[00000008; tool; wrench; #8inch #adjustable]> <> <>

<m_2024-01-03_10:00:00:000> <2024-01-03_10:00:00:000> <img2024-01-07_15:33:00:827; images="" toolbox.jpg;="" ;=""> <[00000009; toolbox; my toolbox; #red] [00000008; tool; wrench;]> <#home #clean> <>
Case1-2: Stuff List
All Stuffs now saved: < [00000001; exercise; ; #morning] [00000002; food; dinner; #korean] [00000003; room; class meeting; #study] [00000004; study; eclipse; #java] [00000005; storage; locker;] [00000006; ; basket;] [00000007; ; clock;] [00000008; tool; wrench; #8inch #adjustable] [00000009; toolbox; my toolbox; #red] [00000008; tool; wrench;] >

Case2: Sorting Test
Pictures in the List
<m_2024-01-03_06:30:00:000> <2024-01-03_06:30:00:000> <img2024-01-07_15:28:42:083; images="" morning-excercise.jpg;="" ;=""> <[00000001; exercise; ; #morning]> <#happy> <new year="">
<m_2024-01-03_10:00:00:000> <2024-01-03_10:00:00:000> <img2024-01-07_15:33:00:827; images="" toolbox.jpg;="" ;=""> <[00000009; toolbox; my toolbox; #red] [00000008; tool; wrench;]> <#home #clean> <>
<m_2024-01-03_17:00:00:000> <2024-01-03_17:00:00:000> <img2024-01-07_15:37:38:009; images="" dinner.jpg;="" ;=""> <[00000002; food; dinner; #korean]> <#friend #meet> <good restaurant="">
<m_2024-02-03_09:30:00:000> <2024-02-03_09:30:00:000> <img2024-01-08_13:29:53:625; images="" wrench.jpg;="" ;=""> <[00000008; tool; wrench; #8inch #adjustable]> <> <>
<m_2024-03-03_14:00:00:000> <2024-03-03_14:00:00:000> <img2024-01-07_15:50:35:480; images="" classroom.jpg;="" ;=""> <[00000003; room; class meeting; #study]> <#class #lecture> <>
<m_2024-03-03_14:20:00:000> <2024-03-03_14:20:00:000> <img2024-01-08_19:12:57:384; images="" eclipse.jpg;="" ;=""> <[00000004; study; eclipse; #java]> <> <>
<m_2024-03-03_15:00:00:000> <2024-03-03_15:00:00:000> <img2024-01-07_15:52:27:605; images="" locker.jpg;="" ;=""> <[00000005; storage; locker;] [00000006; ; basket;] [00000007; ; clock;]> <#school #locker> <my locker="">

*** From now on, Test Reactions on Abnormal Inputs ***
Your Answer (your program's reaction) should be similar to the Correct Answer

Case3-1: No Picture Time
Correct Answer: No time stamp in the input : <m_2024-01-03_17:00:00:000> < > < IMG2024-01-07_15:37:38:009; images/dinner.jpg; ; > < [00000005; food; dinner; #korean]> < #friend #meet > < good restaurant >
Your Answer : Cannot parse "" to DateTimeFormat
Failed to save picture <m_2024-01-03_17:00:00:000> < > < IMG2024-01-07_15:37:38:009; images/dinner.jpg; ; > < [00000005; food; dinner; #korean]> < #friend #meet > < good restaurant >
Pictures in the List
<m_2024-01-03_06:30:00:000> <2024-01-03_06:30:00:000> <img2024-01-07_15:28:42:083; images="" morning-excercise.jpg;="" ;=""> <[00000001; exercise; ; #morning]> <#happy> <new year="">
<m_2024-01-03_10:00:00:000> <2024-01-03_10:00:00:000> <img2024-01-07_15:33:00:827; images="" toolbox.jpg;="" ;=""> <[00000009; toolbox; my toolbox; #red] [00000008; tool; wrench;]> <#home #clean> <>

Case3-2: Check Date Format Error ->
Correct Answer: Wrong DateTime Format : 2024-01-03_10:00:00
Your Answer : Cannot parse "2024-01-03_10:00:00" to DateTimeFormat
Failed to save picture < m_2024-01-03_10:00:00:000 > < 2024-01-03_10:00:00 > < IMG2024-01-07_15:33:00:827; images/toolbox.jpg; ; > < [00000003; toolbox; my toolbox; #red] [00000004; tool; wrench;]> < #home #clean > <>
Pictures in the List
<m_2024-01-03_06:30:00:000> <2024-01-03_06:30:00:000> <img2024-01-07_15:28:42:083; images="" morning-excercise.jpg;="" ;=""> <[00000001; exercise; ; #morning]> <#happy> <new year="">
<m_2024-01-03_17:00:00:000> <2024-01-03_17:00:00:000> <img2024-01-07_15:37:38:009; images="" dinner.jpg;="" ;=""> <[00000002; food; dinner; #korean]> <#friend #meet> <good restaurant="">

Case3-3: ID Conflict Error ->
Correct Answer: ID Conflict (a picture with the same ID already exists); Skip the input line: <m_2024-01-03_10:00:00:000> < 2024-01-03_10:00:00:000 > < IMG2024-01-07_15:37:38:009; images/dinner.jpg; ; > < [00000005; food; dinner; #korean]> < #friend #meet > < good restaurant >
Your Answer : Picture with same id already Exist

Failed to save picture <m_2024-01-03_10:00:00:000> < 2024-01-03_10:00:00:000 > < IMG2024-01-07_15:37:38:009; images/dinner.jpg; ; > < [00000005; food; dinner; #korean]> < #friend #meet > < good restaurant >
Pictures in the List
<m_2024-01-03_06:30:00:000> <2024-01-03_06:30:00:000> <img2024-01-07_15:28:42:083; images="" morning-excercise.jpg;="" ;=""> <[00000001; exercise; ; #morning]> <#happy> <new year="">
<m_2024-01-03_10:00:00:000> <2024-01-03_10:00:00:000> <img2024-01-07_15:33:00:827; images="" toolbox.jpg;="" ;=""> <[00000009; toolbox; my toolbox; #red] [00000008; tool; wrench;]> <#home #clean> <>
<m_2024-02-03_09:30:00:000> <2024-02-03_09:30:00:000> <img2024-01-08_13:29:53:625; images="" wrench.jpg;="" ;=""> <[00000008; tool; wrench; #8inch #adjustable]> <> <>

Case4: Unknwon Picture Information File
Correct Answer: Unknown Picture Info File
Your Answer : Cannot find file :unknown-file.data

</img2024-01-08_13:29:53:625;></m_2024-02-03_09:30:00:000></img2024-01-07_15:33:00:827;></m_2024-01-03_10:00:00:000></new></img2024-01-07_15:28:42:083;></m_2024-01-03_06:30:00:000></m_2024-01-03_10:00:00:000></m_2024-01-03_10:00:00:000></good></img2024-01-07_15:37:38:009;></m_2024-01-03_17:00:00:000></new></img2024-01-07_15:28:42:083;></m_2024-01-03_06:30:00:000></img2024-01-07_15:33:00:827;></m_2024-01-03_10:00:00:000></new></img2024-01-07_15:28:42:083;></m_2024-01-03_06:30:00:000></m_2024-01-03_17:00:00:000></m_2024-01-03_17:00:00:000></my></img2024-01-07_15:52:27:605;></m_2024-03-03_15:00:00:000></img2024-01-08_19:12:57:384;></m_2024-03-03_14:20:00:000></img2024-01-07_15:50:35:480;></m_2024-03-03_14:00:00:000></img2024-01-08_13:29:53:625;></m_2024-02-03_09:30:00:000></good></img2024-01-07_15:37:38:009;></m_2024-01-03_17:00:00:000></img2024-01-07_15:33:00:827;></m_2024-01-03_10:00:00:000></new></img2024-01-07_15:28:42:083;></m_2024-01-03_06:30:00:000></img2024-01-07_15:33:00:827;></m_2024-01-03_10:00:00:000></img2024-01-08_13:29:53:625;></m_2024-02-03_09:30:00:000></my></img2024-01-07_15:52:27:605;></m_2024-03-03_15:00:00:000></img2024-01-08_19:12:57:384;></m_2024-03-03_14:20:00:000></img2024-01-07_15:50:35:480;></m_2024-03-

03_14:00:00:000></good></img2024-01-07_15:37:38:009;></m_2024-01-03_17:00:00:000></new></img2024-01-07_15:28:42:083;></m_2024-01-03_06:30:00:000></pre>

3. 소스코드

클래스 6개의 소스코드를 첨부한다. Main 메서드가 들어간 클래스는 첨부하지 않았으며, autotest용으로 수정한 버전이 아닌 원래 버전(즉, autotest의 html 처리 과정에서 문제가 생겼던 버전)을 첨부한다.

A. Picture.java 소스코드

```
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
import java.time.format.DateTimeParseException;

public class Picture {
	private String id;
	private LocalDateTime timeStamp;
	private Image pictureInfo;
	private Stuff[] stuffList;
	private Tag tags;
	private String comment;

	private static DateTimeFormatter dateTimeFormat=
			DateTimeFormatter.ofPattern("yyyy-MM-dd_HH:mm:ss:SSS");

	Picture(String imageFile){
		String[] pictureByString=imageFile.trim().split(">");
		if(pictureByString[pictureByString.length-1].isBlank()){
			String[] tempPictureByString=new String[pictureByString.length-1];
			System.arraycopy(pictureByString, 0, tempPictureByString, 0,
pictureByString.length-1);
			pictureByString=tempPictureByString;
		}
		if(pictureByString.length!=6) {
			throw new IllegalStateException
			("Picture ₩""+ imageFile +"₩" has something wrong with number of
```

```
₩"<₩" and ₩">₩".");
                }

                pictureByString[0]=pictureByString[0].trim();
                if(pictureByString[0].startsWith("<")) {
                        id=pictureByString[0].substring(1).trim();
                } else {
                        throw new IllegalStateException
                        ("Picture ₩""+ imageFile +"₩" has something wrong with number of
₩"<₩" and ₩">₩".");
                }
                if(id.isBlank()) {
                        throw new IllegalStateException("id of Picture ₩""+ imageFile +"₩"
cannot be empty");
                }

                pictureByString[1]=pictureByString[1].trim();
                if(pictureByString[1].startsWith("<")) {
                        pictureByString[1]=pictureByString[1].substring(1).trim();
                        try {
                        timeStamp=LocalDateTime.parse(pictureByString[1],dateTimeFormat);
                        } catch(DateTimeParseException e) {
                                throw new IllegalStateException
                                ("Cannot parse ₩"" + pictureByString[1] + "₩" to
DateTimeFormat");
                        }
                } else {
                        throw new IllegalStateException
                        ("Picture ₩""+ imageFile +"₩" has something wrong with number of
₩"<₩" and ₩">₩".");
                }

                pictureByString[2]=pictureByString[2].trim();
                if(pictureByString[2].startsWith("<")) {
                        pictureInfo=new Image(pictureByString[2].substring(1).trim());
                } else {
                        throw new IllegalStateException
                        ("Picture ₩""+ imageFile +"₩" has something wrong with number of
```

```
₩"<₩" and ₩">₩".");
                }
                if(id.isBlank()) {
                        throw new IllegalStateException
                        ("pictureInfo of Picture ₩""+ imageFile +"₩" cannot be empty");
                }

                pictureByString[3]=pictureByString[3].trim();
                if(pictureByString[3].startsWith("<")) {
                        stuffList=StuffList.newStuffs(pictureByString[3].substring(1).trim());
                } else {
                        throw new IllegalStateException
                        ("Picture ₩""+ imageFile +"₩" has something wrong with number of
₩"<₩" and ₩">₩".");
                }

                pictureByString[4]=pictureByString[4].trim();
                if(pictureByString[4].startsWith("<")) {
                        tags=new Tag(pictureByString[4].substring(1).trim());
                } else {
                        throw new IllegalStateException
                        ("Picture ₩""+ imageFile +"₩" has something wrong with number of
₩"<₩" and ₩">₩".");
                }

                pictureByString[5]=pictureByString[5].trim();
                if(pictureByString[5].startsWith("<")) {
                        comment=pictureByString[5].substring(1).trim();
                } else {
                        throw new IllegalStateException
                        ("Picture ₩""+ imageFile +"₩" has something wrong with number of
₩"<₩" and ₩">₩".");
                }
        }

        //Eclipse의 자동 생성자 생성 이용//

        Picture(String id, LocalDateTime timeStamp, Image pictureInfo, Stuff[] stuffList,
```

```
                    Tag tags, String comment) {
            this.id = id;
            this.timeStamp = timeStamp;
            this.pictureInfo = pictureInfo;
            this.stuffList = stuffList;
            this.tags = tags;
            this.comment = comment;
}

public void print() {
            System.out.println(this.toString());
}

private boolean ifStuffExist(Stuff stuff) {
            for(Stuff savedStuff:stuffList) {
                        if(stuff.equalByTypeAndName(savedStuff))
                                    return true;
            }
            return false;
}

private void addStuff(String stuff) {
            Stuff[] tempList=new Stuff[stuffList.length+1];
            System.arraycopy(stuffList, 0, tempList, 0, stuffList.length);
            tempList[stuffList.length]=StuffList.addStuff(stuff);
            stuffList=tempList;
}

private void addStuff(Stuff stuff) {
            Stuff[] tempList=new Stuff[stuffList.length+1];
            System.arraycopy(stuffList, 0, tempList, 0, stuffList.length);
            tempList[stuffList.length]=stuff;
            stuffList=tempList;
}

private void deleteStuff(String stuff) {
            deleteStuff(new Stuff(stuff));
            //stuffList 등록을 피하기 위해 Stuff를 바로 씀
```

```
}

private void deleteStuff(Stuff stuff) {
	//StuffList 클래스에서는 삭제되지 않음
	Integer positionToDelete=null;
	//삭제대상찾기
	for(int i=0;i<stuffList.length;i++) {
		if(stuffList[i].equalByTypeAndName(stuff)) {
			positionToDelete=i;
			break;
		}
	}
	if(positionToDelete==null) {
		throw new NullPointerException("Failed to find target to erase: "+stuff);
	}
	//삭제 대상 빼고 새 배열에 옮기기
	Stuff[] tempList=new Stuff[stuffList.length-1];
	System.arraycopy(stuffList, 0, tempList, 0, positionToDelete);
	System.arraycopy(stuffList, positionToDelete+1, tempList, positionToDelete,
			stuffList.length-positionToDelete-1);
	stuffList=tempList;
}

private boolean ifTagExist(String tag) {
	return tags.ifTagExist(tag);
}

private boolean ifTagExist(Tag tag) {
	return tags.ifTagExist(tag);
}

private void addTag(String tag) {
	tags.addTag(tag);
}

private void deleteTag(String tag) {
	tags.deleteTag(tag);
}
```

```
	//비교대상이 greater->negative
	int compareToByTime(Picture picture) {
		return this.timeStamp.compareTo(picture.timeStamp);
	}

	int compareToById(Picture picture) {
		return id.compareTo(picture.id);
	}

	private int tagLength() {
		return tags.getLength();
	}

	boolean equalById(Picture picture) {
		return this.id.equals(picture.id);
	}

	@Override
	public String toString() {
		String stuffListString="";
		for(Stuff stuff:stuffList) {
			stuffListString+=stuff+" ";
		}
		return "<" + id + "> <" + timeStamp.format(dateTimeFormat) + "> <" +
pictureInfo
				+ "> <" + stuffListString.trim() + "> <" + tags + "> <" +
comment + ">";
	}

	static void setDateTimeFormat(DateTimeFormatter dateTimeFormat) {
		Picture.dateTimeFormat = dateTimeFormat;
	}

	@Override
	public boolean equals(Object obj) {
		if (this == obj)
			return true;
		if (obj == null)
```

```
                return false;
        if (getClass() != obj.getClass())
                return false;
        Picture other = (Picture) obj;
        boolean result = id.equals(other.id)&&timeStamp.equals(other.timeStamp)
                        &&pictureInfo.equals(other.pictureInfo)
                        &&tags.equals(other.tags)&&comment.equals(other.comment);
        if(result==false) return false;
        //이하는 stuffList 비교, 둘 중 한 쪽만 null이거나, 개수가 다르면 false
        //else if(stuffList==null&&other.stuffList==null) return true;
        //else if(stuffList==null||other.stuffList==null) return false;
        else if(stuffList.length!=other.stuffList.length) return false;
        //하나하나 비교
        //other의 중점에서 비교
        boolean flag=false;
        for(Stuff compareStuff:other.stuffList) {
                flag=false;
                for(Stuff savedStuff:stuffList)
                        if(compareStuff.equals(savedStuff)) {
                                flag=true;
                                break;
                        }
                if(flag==false) return false;
        }
        //this의 중점에서 비교
        for(Stuff savedStuff:stuffList) {
                flag=false;
                for(Stuff compareStuff:other.stuffList)
                        if(savedStuff.equals(compareStuff)) {
                                flag=true;
                                break;
                        }
                if(flag==false) return false;
        }
        return true;
    }
}
```

B. PictureList.java 소스코드

```
import java.io.File;
import java.io.FileNotFoundException;
import java.io.PrintWriter;
import java.time.format.DateTimeFormatter;
import java.util.Scanner;

public class PictureList {
	private Picture[] pictureList;

	PictureList(){
		pictureList=new Picture[0];
	}

	PictureList(String infoFileName){
		pictureList=new Picture[0];
		savePictureByReadingFile(infoFileName);
	}

	PictureList(Picture[] pictureList){
		this.pictureList=pictureList;
	}

	//중간에 추가할 수 있기에 메소드 분리, 다만 아직 쓸 일 없으니 private(프로그램 확장 시 변경)
	private void savePictureByReadingFile(String infoFileName) {
		File file=new File(infoFileName);
		Scanner input;
		try {
			input=new Scanner(file);
		} catch (FileNotFoundException e) {
			// TODO Auto-generated catch block
			System.out.println("Cannot find file :" + infoFileName);
			return;
		}
		Picture.setDateTimeFormat(DateTimeFormatter.ofPattern("yyyy-MM-dd_HH:mm:ss:SSS"));
```

```
		while(input.hasNext()) {
			savePictureByString(input.nextLine());
		}
		input.close();
	}

	private void savePictureByString(String imageInfo) {
		String infoPicture=imageInfo.trim();
		if(infoPicture.startsWith("//")||infoPicture.isBlank())
			return;
		try {
		Picture picture=new Picture(infoPicture);
		addPicture(picture);
		} catch(IllegalStateException e) {
			System.out.println(e.getMessage());
			System.out.println("Failed to save picture "+ imageInfo);
		}
	}

	public void savePictureToFile(String infoFileName) {
		File file=new File(infoFileName);
		PrintWriter output;
		try {
			output = new PrintWriter(file);
		} catch (FileNotFoundException e) {
			// TODO Auto-generated catch block
			System.out.println("Cannot find file :" + infoFileName);
			return;
		}
		Picture.setDateTimeFormat(DateTimeFormatter.ofPattern("yyyy-MM-
dd_HH:mm:ss:SSS"));
		for(Picture picture:pictureList) {
			output.println(picture);
		}
		output.close();
	}

	public int size() {
```

```
		return this.pictureList.length;
	}

	public Picture get(int i) {
		return pictureList[i]; //0을 포함하는 것인가? (autotest에서 -1 넣었다 에러 나옴)
	}

	public void add(Picture pic) {
		addPicture(pic);
	}

	private void addPicture(Picture picture) {
		for(Picture savedPicture:pictureList) { //이미 존재하는 id에 대한 처리를 autotest
에서 요구하여 추가
			if(picture.equalById(savedPicture))
				throw new IllegalStateException("Picture with same id already
Exist");
		}
		Picture[] tempList=new Picture[pictureList.length+1];
		System.arraycopy(pictureList, 0, tempList, 0, pictureList.length);
		tempList[pictureList.length]=picture;
		pictureList=tempList;
	}

	private void deletePicture(Picture picture) {
		Integer positionToDelete=null;
		//삭제대상찾기
		for(int i=0;i<pictureList.length;i++) {
			if(pictureList[i].equals(picture)) {
				positionToDelete=i;
				break;
			}
		}
		if(positionToDelete==null) {
			throw new NullPointerException("Failed to find target to erase:
"+picture);
		}
		//삭제 대상 빼고 새 배열에 옮기기
```

```
		Picture[] tempList=new Picture[pictureList.length-1];
		System.arraycopy(pictureList, 0, tempList, 0, positionToDelete);
		System.arraycopy(pictureList, positionToDelete+1, tempList, positionToDelete,
					pictureList.length-positionToDelete-1);
		pictureList=tempList;
}
/* 추가, 수정, 삭제, 검색은 이번에 요구하지 않았으므로 구현 없음
private PictureList selectTags(Tag tag) {

}

private PictureList selectStuff(Stuff stuff) {

}
*/
public void sortByDate() {
		Picture temp=null;
		for(int i=0;i<pictureList.length-1;i++)
			for(int j=i+1;j<pictureList.length;j++)
				if(pictureList[i].compareToByTime(pictureList[j])>0) {
					temp=pictureList[i];
					pictureList[i]=pictureList[j];
					pictureList[j]=temp;
				}
}

private void sortById() {
		Picture temp=null;
		for(int i=0;i<pictureList.length-1;i++)
			for(int j=i+1;j<pictureList.length;j++)
				if(pictureList[i].compareToById(pictureList[j])>0) {
					temp=pictureList[i];
					pictureList[i]=pictureList[j];
					pictureList[j]=temp;
				}
}

@Override
```

```
	public String toString() {
		String result="";
		for(Picture picture:pictureList)
			result+=picture+"₩n";
		return result.trim();
	}
}
```

C. Stuff.java 소스코드

```
public class Stuff {
	private String id;
	private String type;
	private String name;
	private Tag tags;

	Stuff(String stuff){
		stuff=stuff.trim();
		if(stuff.startsWith("[")&&stuff.endsWith("]")) { //괄호가 섞여 들어오면 제거
			stuff=stuff.substring(1, stuff.length()-1);
		}
		String stuffs[]=(stuff+" ").split(";");
		if(stuffs.length!=4) {
			throw new IllegalStateException
			("Stuff ₩""+ stuff +"₩" has something wrong with number of ₩";₩".");
		}
		id=null;
		type=stuffs[1].trim();
		name=stuffs[2].trim();
		if(type.isBlank()&&name.isBlank()) {
			throw new IllegalStateException
			("both type and name of Stuff ₩""+ stuff +"₩" cannot be empty");
		}
		tags=new Tag(stuffs[3].trim());
	}

	Stuff(String type, String name, String tags) {
		this.id = null;
		this.type = type;
```

```
                this.name = name;
                this.tags = new Tag(tags);
                if(type.isBlank()&&id.isBlank()) {
                        throw new IllegalStateException("both type and name of Stuff cannot be
empty");
                }
        }

        Stuff(String id, String type, String name, Tag tags) {
                this.id = id;
                this.type = type;
                this.name = name;
                this.tags = tags;
                if(type.isBlank()&&id.isBlank()) {
                        throw new IllegalStateException("both type and name of Stuff cannot be
empty");
                }
        }

        boolean equalByTypeAndName(Stuff stuff) {
                if(stuff.type.equals(type)&&stuff.name.equals(name)) {
                        return true;
                }
                return false;
        }

        boolean equalExceptId(Stuff stuff) {
                if(stuff.type.equals(type)&&stuff.name.equals(name)&&stuff.tags.equals(tags)) {
                        return true;
                }
                return false;
        }

        int compareTo(Stuff stuff) {
                int result = id.compareTo(stuff.id);
                if(result==0) {
                        result = tags.compareTo(stuff.tags);
                }
```

```
		return result;
	}

	@Override
	public String toString() {
		return "[" + id + "; " + type + "; " + name + "; " + tags + "]";
	}

	@Override
	public boolean equals(Object obj) {
		if (this == obj)
			return true;
		if (obj == null)
			return false;
		if (getClass() != obj.getClass())
			return false;
		Stuff other = (Stuff) obj;
		return id.equalsIgnoreCase(other.id);
	}

	void setId(String id) {
		if(this.id==null)
			this.id = id;
	}

	void setId(Stuff stuff) {
		if(this.id==null)
			this.id = stuff.id;
	}
}
```

D. StuffList.java 소스코드

```
public class StuffList {
	private static Stuff[] stuffList=new Stuff[0];
	private static int stuffId=1;

	static Stuff[] newStuffs(String stuff) {
		String[] stuffListByString=stuff.split("]");
```

```
                if(stuffListByString[stuffListByString.length-1].isBlank()) {
                        String[] tempStuffList=new String[stuffListByString.length-1];
                        System.arraycopy(stuffListByString, 0, tempStuffList, 0,
tempStuffList.length);
                        stuffListByString=tempStuffList;
                }

                Stuff[] resultStuffList=new Stuff[stuffListByString.length];
                for(int i=0;i<stuffListByString.length;i++) {
                        stuffListByString[i]=stuffListByString[i].trim();
                        if(stuffListByString[i].startsWith("[")) {
                                stuffListByString[i]=stuffListByString[i].substring(1);
                                resultStuffList[i]=addStuff(stuffListByString[i]);
                        }
                        else {
                                throw new IllegalStateException
                                ("Stuffs ₩""+ stuff +"₩" has something wrong with ₩"[₩" and
₩"]₩".");
                        }
                }
                return resultStuffList;
        }

        private static Stuff ifStuffExistByNameAndType(Stuff stuffToCompare) {
                for(Stuff stuff: stuffList) {
                        if(stuffToCompare.equalByTypeAndName(stuff))
                                return stuff;
                }
                return null;
        }

        private static Stuff ifStuffExistExceptId(Stuff stuffToCompare) {
                for(Stuff stuff: stuffList) {
                        if(stuffToCompare.equalExceptId(stuff))
                                return stuff;
                }
                return null;
        }
```

```
//Id 할당은 필요함
static Stuff addStuff(String stuffInfo) {
	stuffInfo=stuffInfo.trim();
	if(stuffInfo.startsWith("[")&&stuffInfo.endsWith("]")) {
		stuffInfo=stuffInfo.substring(1,stuffInfo.length()-1).trim();
	}
	Stuff newStuff=new Stuff(stuffInfo);
	Stuff existingStuff=ifStuffExistByNameAndType(newStuff);
	if(existingStuff!=null) {
		Stuff existingSameStuff=ifStuffExistExceptId(newStuff);
		if(existingSameStuff!=null) {
			newStuff=existingSameStuff;
			//전체출력시 같은 거 여러 번 나오는 불편한 상황 방지
		} else {
			newStuff.setId(existingStuff);
			Stuff[] tempList=new Stuff[stuffList.length+1];
			System.arraycopy(stuffList, 0, tempList, 0, stuffList.length);
			tempList[stuffList.length]=newStuff;
			stuffList=tempList;
		}
	} else {
		newStuff.setId(String.format("%08d", stuffId));
		//00000001부터 차례대로 ID를 주기 위한 유일한 방법
		stuffId++;
		Stuff[] tempList=new Stuff[stuffList.length+1];
		System.arraycopy(stuffList, 0, tempList, 0, stuffList.length);
		tempList[stuffList.length]=newStuff;
		stuffList=tempList;
	}
	return newStuff;
}

private static void sortById() {
	Stuff temp=null;
	for(int i=0;i<stuffList.length-1;i++)
		for(int j=i+1;j<stuffList.length;j++)
			if(stuffList[i].compareTo(stuffList[j])>0) {
```

```
                                        temp=stuffList[i];
                                        stuffList[i]=stuffList[j];
                                        stuffList[j]=temp;
                                }
        }

        public static void print() {
                System.out.println(allStuffToString());
        }

        private static String allStuffToString() {
                String stuffListByString="All Stuffs now saved: < ";
                for(Stuff stuff: stuffList) {
                        stuffListByString+=stuff+" ";
                }
                return stuffListByString+">";
        }
}
```

E. Tag.java 소스코드

```
public class Tag {
        private String[] tags; //절대 null이 되지 않음

        Tag(String tagInfo){
                tags=new String[0];
                String[] splitResult=tagInfo.replace(" ", "").split("#");
                //중간의 공백도 삭제하기 위해 trim이 아닌 replace 사용
                //바로 넣지 않는 이유는 맨 앞에 빈 문자열이 생기기 때문
                for(String tag:splitResult) {
                        if(!tag.isBlank())
                                addTag(tag);
                }
        }

        boolean ifTagExist(String tagToCompare) {
                tagToCompare=tagToCompare.replace("#", "").trim();
                for(String tag:tags) {
                        if(tagToCompare.equalsIgnoreCase(tag))
```

```
                        return true;
            }
            return false;
}

boolean ifTagExist(Tag tagToCompare) {
            boolean flag=false;
            for(String compareTag:tagToCompare.tags) {
                        flag=false;
                        for(String savedTag:tags)
                                    if(compareTag.equalsIgnoreCase(savedTag)) {
                                                flag=true;
                                                break;
                                    }
                        if(flag==false) return false;
            }
            return true;
}

int getLength() {
            return tags.length;
}

void addTag(String tag) {
            String[] tempList;
            tempList=new String[getLength()+1];
            System.arraycopy(tags, 0, tempList, 0, getLength());
            tempList[getLength()]=tag.replace("#", "").trim();
            tags=tempList;
}

void deleteTag(String tag) {
            tag=tag.trim();
            if(tag.startsWith("#"))
                        tag=tag.substring(1).trim();

            Integer positionToDelete=null;
            //삭제대상찾기
```

```
		for(int i=0;i<tags.length;i++) {
			if(tags[i].equalsIgnoreCase(tag)) {
				positionToDelete=i;
				break;
			}
		}
		if(positionToDelete==null) {
			throw new NullPointerException("Failed to find target to erase: #"+tag);
		}
		//삭제 대상 빼고 새 배열에 옮기기
		String[] tempList=new String[getLength()-1];
		System.arraycopy(tags, 0, tempList, 0, positionToDelete);
		System.arraycopy(tags, positionToDelete+1, tempList, positionToDelete,
				tags.length-positionToDelete-1);
		tags=tempList;
	}

	int compareTo(Tag tag) {
		int result=this.getLength()-tag.getLength();
		for(int i=0;result==0&&i<this.getLength();i++) {
			result=this.tags[i].compareTo(tag.tags[i]);
		} //길이가 다르면 for문 진입조차 하지 않음
		return result;
	}

	@Override
	public String toString() {
		String result="";
		for(String tag:tags) {
			result+="#"+tag+" ";
		}
		return result.trim();
	}

	//이하는 Eclipse의 자동 생성
	@Override
	public boolean equals(Object obj) {
		if (this == obj)
```

```
                return true;
        if (obj == null)
                return false;
        if (getClass() != obj.getClass())
                return false;
        Tag other = (Tag) obj;
        //하나하나 비교
        //other의 중점에서 비교
        boolean flag=false;
        for(String compareTag:other.tags) {
                flag=false;
                for(String savedTag:this.tags)
                        if(compareTag.equals(savedTag)) {
                                flag=true;
                                break;
                        }
                if(flag==false) return false;
        }
        //this의 중점에서 비교
        for(String savedTag:this.tags) {
                flag=false;
                for(String compareTag:other.tags)
                        if(savedTag.equals(compareTag)) {
                                flag=true;
                                break;
                        }
                if(flag==false) return false;
        }
        return true;
    }
}
```

F. Image.java 소스코드

```
public class Image {
    private String imageId;
    private String imageFileName;
    private String imageName;
    private Tag tags;
```

```
	Image(String imageInfo){
		String[] imageByString=(imageInfo+" ").split(";");
		if(imageByString.length!=4) {
			throw new IllegalStateException
			("Image ₩""+ imageInfo +"₩" has something wrong with ₩";₩".");
		}
		imageId=imageByString[0].trim();
		imageFileName=imageByString[1].trim();
		imageName=imageByString[2].trim();
		tags=new Tag(imageByString[3]);
	}

	@Override
	public String toString() {
		return imageId + "; " + imageFileName + "; " + imageName + "; " + tags;
	}

	@Override
	public boolean equals(Object obj) {
		if (this == obj)
			return true;
		if (obj == null)
			return false;
		if (getClass() != obj.getClass())
			return false;
		Image other = (Image) obj;
		return
imageId.equals(other.imageId)&&imageFileName.equals(other.imageFileName)

	&&imageName.equals(other.imageName)&&tags.equals(other.tags);
	}
}
```

4. 본인 인증

본인 인증을 위해 별도의 사진은 찍어놓지 않았으나, 프로젝트를 Onedrive에 저장하여 유일하게 Onedrive 파일 수정 기록이 남아 있다. 따라서 이것만이라도 첨부한다.

A. Picture.java 버전 기록

버전 기록

버전	수정한 날짜	수정한 사람	크기
26.0	2024. 4. 15. 오후 09:05	정현승	7.40KB
25.0	2024. 4. 15. 오후 08:49	정현승	7.36KB
24.0	2024. 4. 15. 오후 07:43	정현승	7.40KB
23.0	2024. 4. 15. 오후 05:37	정현승	7.36KB
22.0	2024. 4. 15. 오후 05:24	정현승	7.86KB
21.0	2024. 4. 15. 오후 03:39	정현승	7.37KB
20.0	2024. 4. 15. 오후 03:21	정현승	7.36KB
19.0	2024. 4. 14. 오전 02:32	정현승	7.37KB
18.0	2024. 4. 13. 오후 11:05	정현승	7.04KB
17.0	2024. 4. 13. 오후 10:37	정현승	7.09KB
16.0	2024. 4. 13. 오후 10:34	정현승	7.07KB
15.0	2024. 4. 13. 오후 10:28	정현승	7.06KB
14.0	2024. 4. 13. 오후 10:08	정현승	7.09KB
13.0	2024. 4. 13. 오후 10:06	정현승	6.94KB
12.0	2024. 4. 13. 오후 09:48	정현승	6.95KB
11.0	2024. 4. 13. 오후 09:23	정현승	6.15KB
10.0	2024. 4. 13. 오후 08:52	정현승	5.61KB
9.0	2024. 4. 13. 오후 08:26	정현승	5.60KB
8.0	2024. 4. 13. 오후 08:13	정현승	5.58KB
7.0	2024. 4. 13. 오후 07:43	정현승	5.67KB
6.0	2024. 4. 13. 오후 06:09	정현승	5.36KB
5.0	2024. 4. 13. 오후 05:21	정현승	5.41KB
4.0	2024. 4. 11. 오후 10:19	정현승	5.34KB
3.0	2024. 4. 11. 오전 02:16	정현승	1.83KB
2.0	2024. 4. 10. 오후 10:09	정현승	1.83KB
1.0	2024. 4. 9. 오후 10:32	정현승	1.83KB

그림 4.1

B. PictureList.java 버전 기록

버전 기록

버전	수정한 날짜	수정한 사람	크기
22.0	2024. 4. 15. 오후 09:05	정현승	4.30KB
21.0	2024. 4. 15. 오후 08:49	정현승	4.31KB
20.0	2024. 4. 14. 오전 02:32	정현승	4.30KB
19.0	2024. 4. 14. 오전 01:41	정현승	4.09KB
18.0	2024. 4. 13. 오후 10:26	정현승	4.02KB
17.0	2024. 4. 13. 오후 10:08	정현승	4.01KB
16.0	2024. 4. 13. 오후 09:57	정현승	3.89KB
15.0	2024. 4. 13. 오후 09:48	정현승	3.89KB
14.0	2024. 4. 13. 오후 09:32	정현승	3.88KB
13.0	2024. 4. 13. 오후 09:04	정현승	3.88KB
12.0	2024. 4. 13. 오후 08:26	정현승	3.88KB
11.0	2024. 4. 13. 오후 08:09	정현승	3.88KB
10.0	2024. 4. 13. 오후 07:57	정현승	3.88KB
9.0	2024. 4. 13. 오후 07:43	정현승	3.88KB
8.0	2024. 4. 13. 오후 06:13	정현승	3.70KB
7.0	2024. 4. 13. 오후 06:09	정현승	3.67KB
6.0	2024. 4. 13. 오후 05:51	정현승	3.49KB
5.0	2024. 4. 13. 오후 05:10	정현승	3.34KB
4.0	2024. 4. 13. 오후 05:10	정현승	3.34KB
3.0	2024. 4. 13. 오후 04:47	정현승	3.46KB
2.0	2024. 4. 11. 오후 10:30	정현승	2.71KB
1.0	2024. 4. 9. 오후 10:32	정현승	607바이트

그림 4.2

C. Stuff.java 버전 기록

버전 기록

버전	수정한 날짜	수정한 사람	크기
16.0	2024. 4. 13. 오후 10:14	정현승	2.23KB
15.0	2024. 4. 13. 오후 10:04	정현승	2.06KB
14.0	2024. 4. 13. 오후 10:04	정현승	2.09KB
13.0	2024. 4. 13. 오후 08:26	정현승	2.09KB
12.0	2024. 4. 13. 오후 08:09	정현승	2.01KB
11.0	2024. 4. 13. 오후 06:09	정현승	1.98KB
10.0	2024. 4. 13. 오후 05:30	정현승	1.98KB
9.0	2024. 4. 13. 오후 04:24	정현승	1.98KB
8.0	2024. 4. 13. 오후 04:20	정현승	1.90KB
7.0	2024. 4. 13. 오후 04:18	정현승	1.91KB
6.0	2024. 4. 13. 오후 04:17	정현승	1.92KB
5.0	2024. 4. 11. 오후 09:42	정현승	1.63KB
4.0	2024. 4. 11. 오전 02:16	정현승	1.46KB
3.0	2024. 4. 10. 오후 10:40	정현승	1.40KB
2.0	2024. 4. 10. 오후 10:09	정현승	848바이트
1.0	2024. 4. 9. 오후 10:32	정현승	842바이트

그림 4.3

D. StuffList.java 버전 기록

버전 기록

버전	수정한 날짜	수정한 사람	크기
13.0	2024. 4. 15. 오후 03:21	정현승	3.05KB
12.0	2024. 4. 13. 오후 08:26	정현승	3.15KB
11.0	2024. 4. 13. 오후 08:09	정현승	3.15KB
10.0	2024. 4. 13. 오후 06:11	정현승	2.87KB
9.0	2024. 4. 13. 오후 06:11	정현승	2.87KB
8.0	2024. 4. 13. 오후 06:10	정현승	2.87KB
7.0	2024. 4. 13. 오후 06:09	정현승	2.86KB
6.0	2024. 4. 13. 오후 04:29	정현승	2.86KB
5.0	2024. 4. 13. 오후 04:28	정현승	3.58KB
4.0	2024. 4. 13. 오후 04:27	정현승	3.43KB
3.0	2024. 4. 11. 오후 09:11	정현승	3.01KB
2.0	2024. 4. 10. 오후 11:44	정현승	2.45KB
1.0	2024. 4. 9. 오후 10:32	정현승	882바이트

그림 4.4

E. Tag.java 버전 기록

버전 기록

버전	수정한 날짜	수정한 사람	크기
20.0	2024. 4. 15. 오후 03:21	정현승	3.05KB
19.0	2024. 4. 13. 오후 10:41	정현승	2.66KB
18.0	2024. 4. 13. 오후 10:05	정현승	2.65KB
17.0	2024. 4. 13. 오후 10:04	정현승	2.66KB
16.0	2024. 4. 13. 오후 09:48	정현승	2.71KB
15.0	2024. 4. 13. 오후 09:15	정현승	2.70KB
14.0	2024. 4. 13. 오후 08:52	정현승	2.89KB
13.0	2024. 4. 13. 오후 08:45	정현승	2.89KB
12.0	2024. 4. 13. 오후 08:44	정현승	2.86KB
11.0	2024. 4. 13. 오후 08:30	정현승	2.55KB
10.0	2024. 4. 13. 오후 08:26	정현승	2.55KB
9.0	2024. 4. 13. 오후 08:09	정현승	2.34KB
8.0	2024. 4. 13. 오후 06:09	정현승	2.29KB
7.0	2024. 4. 13. 오후 05:21	정현승	2.27KB
6.0	2024. 4. 11. 오후 09:06	정현승	1.91KB
5.0	2024. 4. 10. 오후 11:33	정현승	1.17KB
4.0	2024. 4. 10. 오후 10:22	정현승	1.13KB
3.0	2024. 4. 10. 오후 10:21	정현승	1.13KB
2.0	2024. 4. 10. 오후 10:08	정현승	377바이트
1.0	2024. 4. 9. 오후 10:32	정현승	547바이트

그림 4.5

F. Image.java 버전 기록

버전 기록

버전	수정한 날짜	수정한 사람	크기
10.0	2024. 4. 13. 오후 09:17	정현승	997바이트
9.0	2024. 4. 13. 오후 09:16	정현승	997바이트
8.0	2024. 4. 13. 오후 09:15	정현승	1.00KB
7.0	2024. 4. 13. 오후 05:57	정현승	637바이트
6.0	2024. 4. 13. 오후 05:53	정현승	638바이트
5.0	2024. 4. 13. 오후 05:30	정현승	639바이트
4.0	2024. 4. 11. 오후 09:22	정현승	641바이트
3.0	2024. 4. 11. 오전 02:16	정현승	628바이트
2.0	2024. 4. 10. 오후 10:08	정현승	783바이트
1.0	2024. 4. 9. 오후 10:32	정현승	787바이트

그림 4.6

평 가 표

평가 항목	학생 자체 평가 (리포트 해당 부분 표시 및 간단한 의견)	평가 (빈칸)	점수 (빈칸)
기본 동작 - autotest 결과화면	기본 동작: Chapter 2 Autotest: Chapter 2B		
설계 사항 - 설계 착안점 - 사용한 클래스 설명 - 시행착오 및 해결책	설계 사항: Chapter 1 설계 착안점: Chapter 1(일단 뭐가 필요한지 다 써보고 그대로 설계, 전부 맞았다면 미사용 메서드가 없었을 것) 사용한 클래스 설명: Chapter 1A~1F (공통부분 Chapter 1) 시행착오 및 해결책: 클래스 설명 중 같이 설명(에러를 이렇게 해결함 등, 애초에 큰 에러는 별로 발생하지 않아 착오를 별로 겪지 않음(대신 삽질을 했을 뿐))		
본인 인증	본인 인증: Chapter 4		
기타			
총평/계			

* 학생 자체 평가는 점수에 반영되지 않음.
* 학생 스스로 자신의 보고서를 평가하면서, 체계적으로 프로젝트를 마무리하도록 유도하는 것이 목적임.