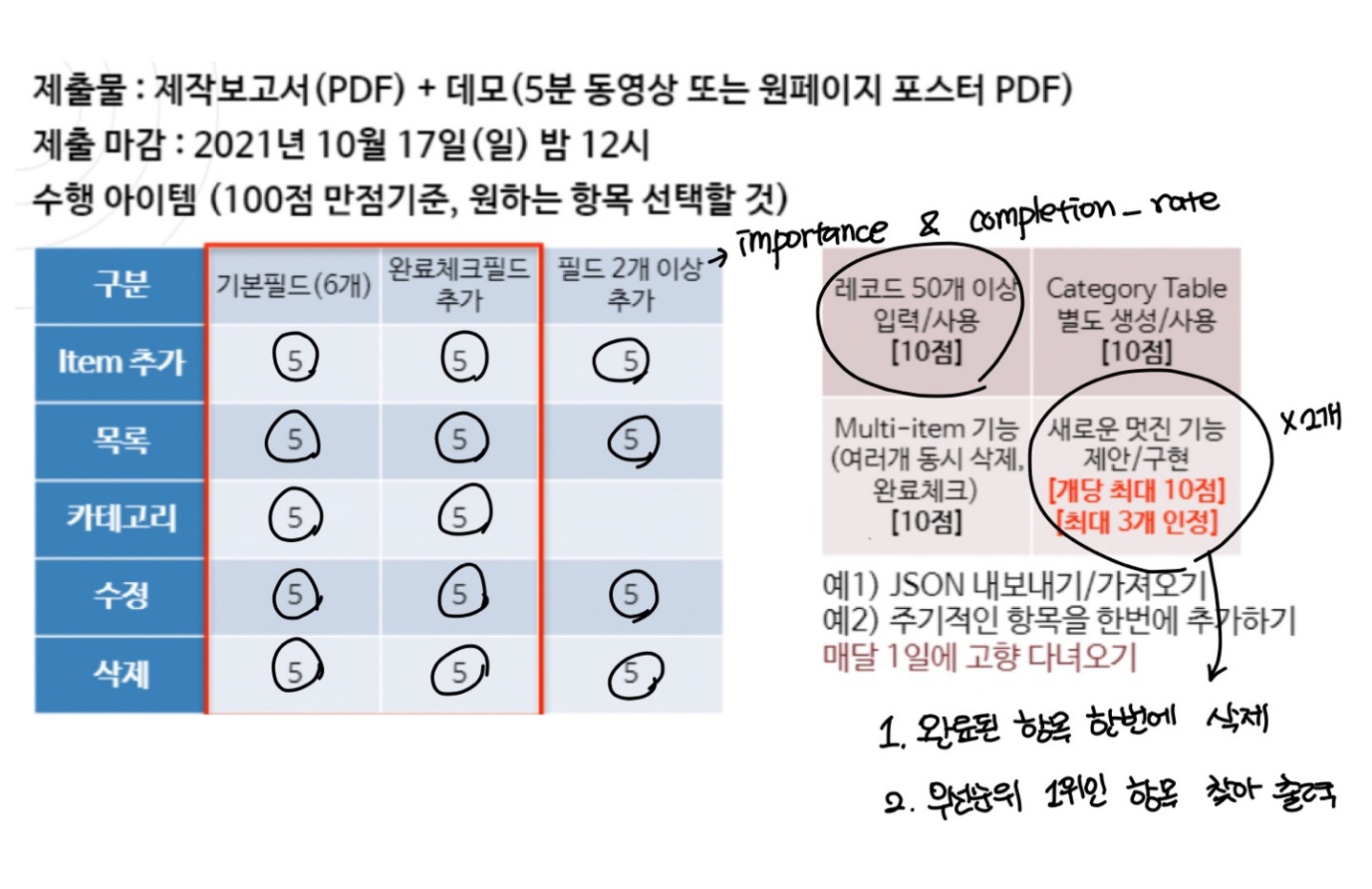
**<< 희망 점수 >>**

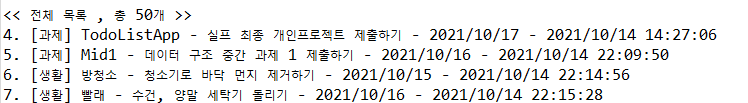
****

**기본 50점 + 필드 2개 추가 20점 + 레코드 50개 이상 사용 10점 + 새로운 기능 2개 20점**

**= 100점 희망**

**<< Git Repository URI :**

**<< 추가 점수 1 : 레코드 50개 이상 입력/사용 – 10점 충족 >>**

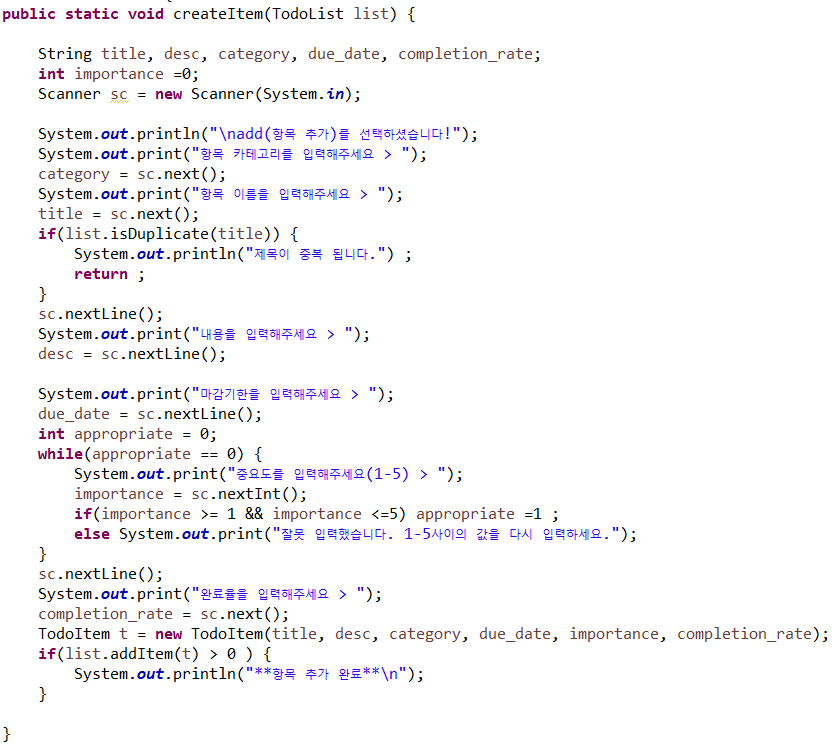


전체 목록 50개인 상태로 프로그램을 시작한다. (캡쳐 이미지는 일부 생략)

**<< Item 추가 : 기본필드 + 완료 체크 필드 + 2개의 추가 필드 – 15점 충족 >>**

**[ add ]**

1. **TodoUtil > CreateItem**

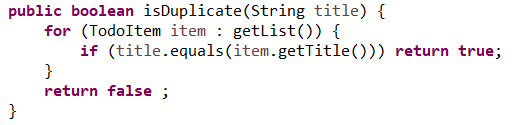


**항목추가 선택 -> 카테고리 입력 -> 이름 입력 -> 내용 입력 -> 마감 기한 입력 -> 중요도 입력 -> 완료율 입력 -> TodoItem 생성 -> addItem 메소드로 항목 추가 -> 항목추가 완료**

위와 같은 과정으로 항목을 추가합니다.

* 1. **중복 확인 기능 : TodoList > isDuplicate**

항목의 이름을 입력한 후, isDuplicate가 true를 반환하면 중복 확인 후 add기능 종료



getList로 ArrayList<TodoItem>을 반환받아서 item객체에 한 항목씩 가져와 item의 title과, 파라미터로 받은 title을 equals메소드로 값을 비교 -> 반복이 끝나기 전에 하나라도 값이 같은 항목이 나온다면 true리턴

* + 1. **모든 리스트 가져오는 기능 : TodoList > getList**



리턴값 : ArrayList<TodoItem> -> 데이터베이스로부터 모든 항목을 받아와서 리스트로 만들어서 반환

SQL실행을 위한 statement 객체 생성 -> sql문 작성 -> sql문을 statement객체 메소드를 통해 실행 -> 실행된 결과를 rs객체에 ResultSet으로 받음 -> rs가 끝날때까지 한줄씩 받아와 각 항목의 9개의 필드값 가져오기 -> 가져온 값으로 TodoItem생성 -> id, current\_date, is\_completed는 set으로 지정 -> 리스트에 아이템 추가 -> 완성된 리스트반환

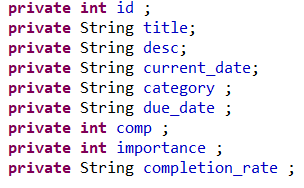
2개 필드 추가로 인한 변동 사항 : importance(중요도)와 completion\_rate가 추가됨에 따라 get할 데이터가 늘어났고, TodoItem을 만들 때에도 두 필드를 추가하여 만든다.

* 1. **중요도 적절한 값 입력할 때까지 묻는 기능**

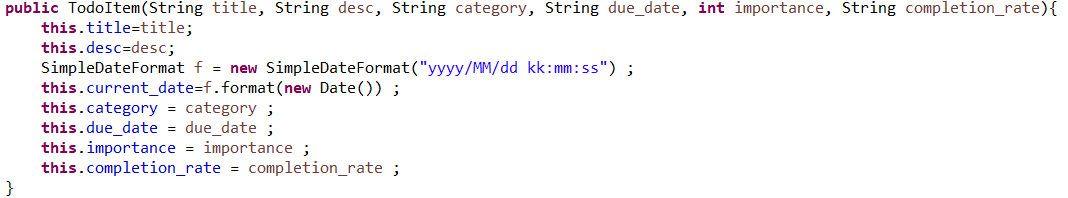
중요도는 int값으로 1-5사이의 값이 입력되어야 한다. 해당 값이 입력되었을 경우에만 appropriate라는 플래그 변수를 1로 바꾸고 while문을 빠져나온다

* 1. **새로운 아이템 생성 : TodoItem**

2개의 필드가 새로 추가 되면서 TodoItem constructor가 변했음을 알 수 있다.

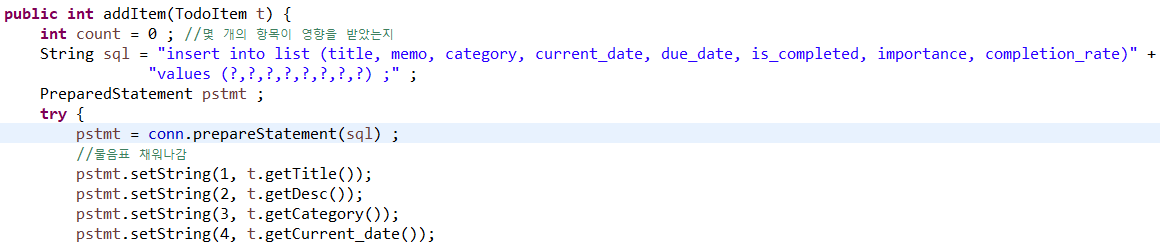


TodoItem의 instance variables는 위와 같이 총 9개가 되었다.



Constructor는 title, desc, category, due\_date, importance, completion\_rate를 받아 instance variable의 값에 저장하고, current\_date는 기존과 동일하게 데이터 포맷을 활용해 현재 시간을 넣는다. Id와 is\_completed 필드에는 자동으로 값이 들어간다. id에는 1씩 증가하는 값이 들어가고, is\_completed에는 0이 들어간다. 0은 완료되지 않았음을 의미하고 1은 완료를 의미한다.

* 1. **리스트에 아이템 추가 : TodoList > addItem**





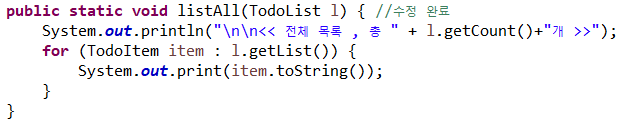
TodoItem객체를 파라미터로 받는다 -> sql문 생성(각 필드의 values는 ?로 비워 둔 상태) -> PreparedStatement 객체 생성(?값을 나중에 넣어서 실행시키기 위함) -> ?의 개수 만큼 setString / setInt 메소드로 해당 물음표의 위치에 TodoItem의 instance variables 값 get해서 값 지정 -> executeUpdate 메소드로 Read 실행 -> 영향받은 데이터의 개수 count에 반환 -> count 리턴

이 메소드에서도 2개의 필드가 추가 됨에 따라 sql문이 달라지고, setString과 setInt가 추가되었음을 알 수 있다.

**<< 목록 : 기본필드 + 완료 체크 필드 + 2개의 추가 필드 – 15점 충족 >>**

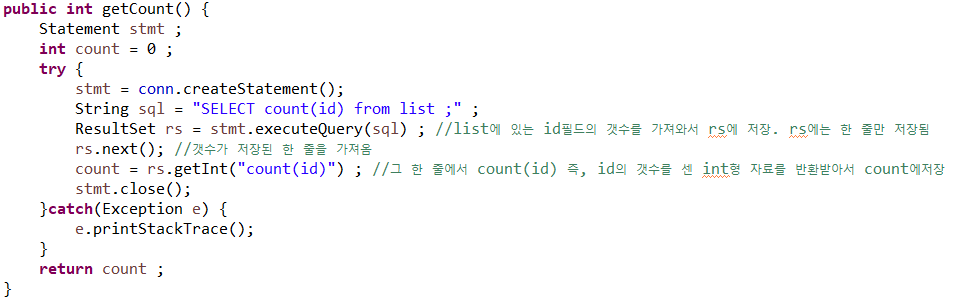
**[ ls ]**

1. **TodoUtil > listAll**



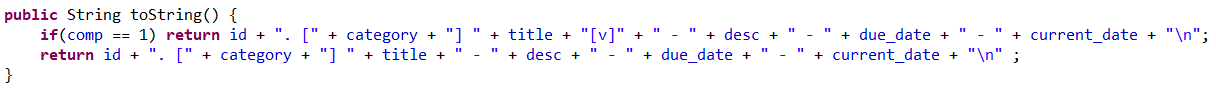
리스트를 파라미터로 받아 리스트에 저장된 항목의 총 개수를 출력 -> 전체 리스트를 가져와 각 항목을 toString 메소드를 통해 해당 형식으로 출력

* 1. **리스트의 항목 개수를 알아내는 기능 : TodoList > getCount()**



sql실행을 위한 statement생성 -> sql문 생성(list로부터 id필드의 개수를 새서 반환) -> 실행 -> id의 개수가 저장된 한 줄이 반환될 것. 반환된 값을 count에 저장 -> 반환

* 1. **TodoItem > toString**

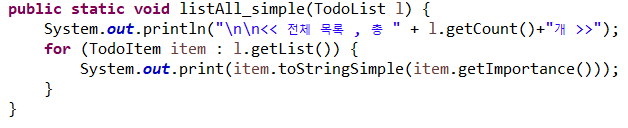


항목의 comp가 1이면 완료 체크 [v] 표시를 추가한 string을 반환한다. 출력 형식을 규정한 것으로 id, category, title, desc, due\_date, current\_date를 출력한다

새로 추가된 2개의 필드 importance와 completion\_rate에 관한 출력은 ls\_simple메뉴에서 다룬다.

**[ ls\_simple ]**

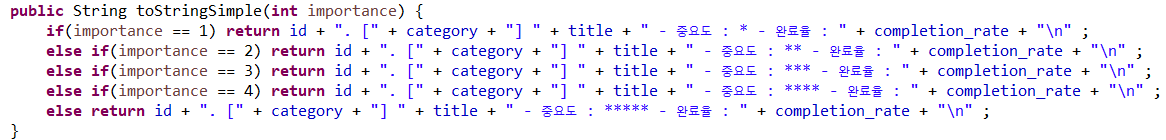
1. **중요도와 완료율만 간단히 보여주는 기능 : TodoUtil > listAll\_simple**



기존의 리스트에서 새로 추가한 필드까지 모두 한 줄에 출력하려면 사용자의 입장에서 너무 많은 정보를 한 줄로 읽게 되어서 불편할 것 같다는 생각이 들었다. 그래서 중요도와 완료율만 따로 간단히 리스트로 볼 수 있는 메뉴를 만들었다. 이 메뉴를 통해 중요한 항목들이 얼마나 완료 되었는지 쉽게 볼 수 있다.

기능은 ls와 거의 비슷하고 다른 점은 toStringSimple을 사용해 항목의 중요도를 넘겨서 ls와는 다른 형식으로 출력한다.

* 1. **TodoItem > toStringSimple**

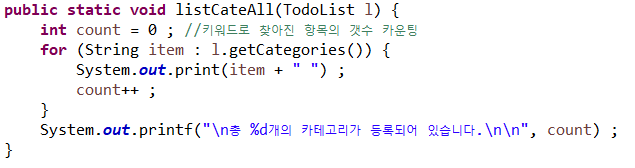


중요도는 1-5까지 있다. 각 경우에 따라 중요도를 \*의 개수로 표현하고, 완료율도 표현한다.

**<< 카테고리 : 기본필드 + 완료 체크 필드 – 10점 충족 >>**

**[ ls\_cat ]**

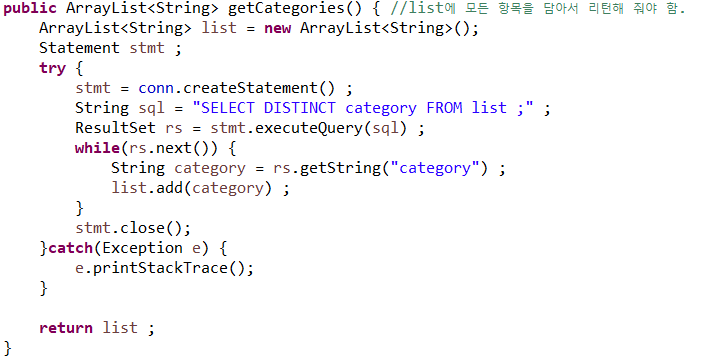
1. **TodoUtil > listCateAll**



TodoList를 파라미터로 받아 리스트로부터 카테고리만 받아서 각 카테고리를 공백과

함께 출력 -> 출력 반복하면서 카테고리의 개수 카운팅 -> 카테고리의 개수 출력

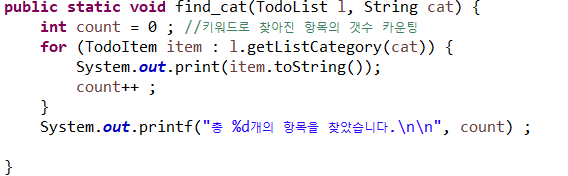
* 1. **카테고리만 리스트로 만들어서 반환하는 기능 : TodoList > getCategories**



ArrayList 생성( 항목 리스트가 아닌, 카테고리만 가져와서 리스트로 만들어서 반환) -> statement 생성 -> select distinct category from list 문을 이용해 카테고리만 중복되지 않게 rs에 ResultSet으로 리턴받음 -> 한줄씩 읽어오면서 category저장 후, 리스트에 추가 -> 완성된 리스트 반환

**[ find\_cat ]**

1. **TodoUtil > find\_cat(TodoList, keyword)**



리스트와, 입력받은 카테고리 string을 파라미터로 받는다 -> getListCategory 메소드로 리스트에서 파라미터로 받은 카테고리에 해당하는 항목만 리스트로 받아서 각 항목을 toString으로 출력한다. -> 출력하면서 카운팅한 항목의 개수를 출력한다

* 1. **카테고리에 해당하는 항목만 리스트로 만들어서 반환하는 기능 : TodoList > getListCategory**

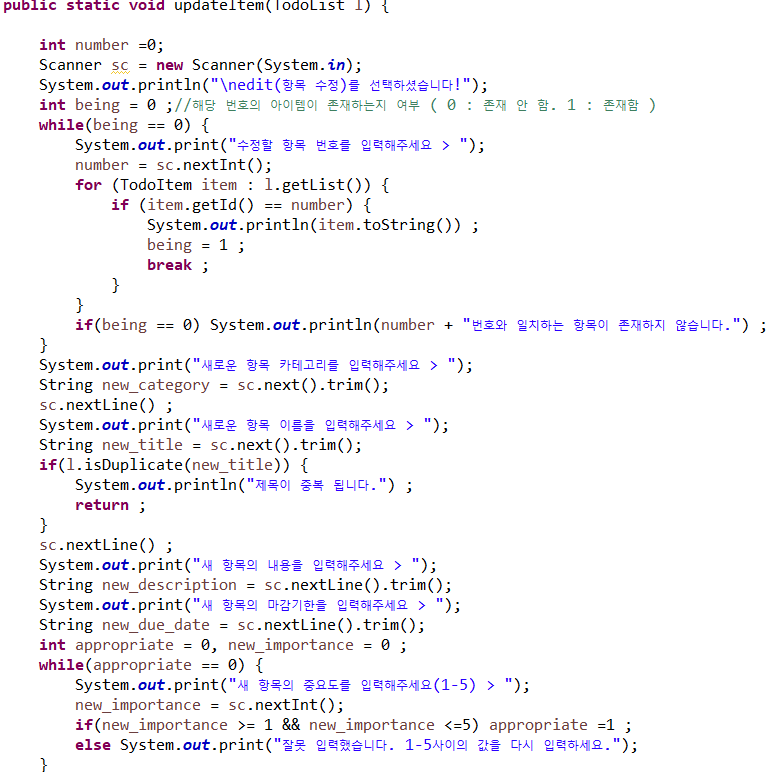


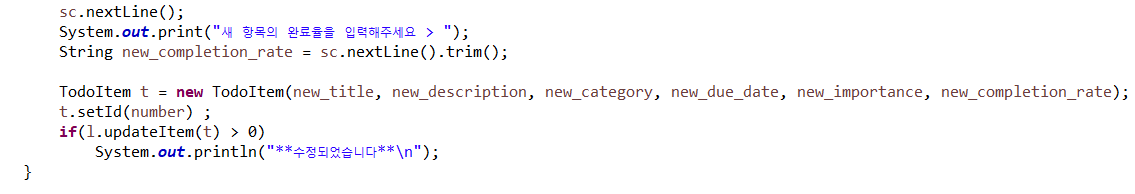
ArrayList 생성(카테고리에 해당하는 항목만 리스트로 반환) -> preparedStatement 생성 -> sql문 생성(카테고리에 해당 키워드가 들어가는지 볼 수 있도록 like 사용) -> 파라미터로 받은 string을 sql문에 set -> 실행 -> ResultSet에서 각 항목의 필드값 가져옴 -> 새로운 아이템 생성해서 리스트에 추가

이 과정에서도 importance와 completion\_rate 필드가 추가 됨에 따라 sql문과 아이템 생성, getInt/getString의 개수에 변화가 생겼음을 알 수 있다.

**<< 수정 : 기본필드 + 완료 체크 필드 + 2개의 추가 필드 – 15점 충족 >>**

1. **TodoUtil > UpdateItem**





Scanner객체 생성 -> 항목수정 선택 출력 -> 수정할 번호 입력 -> 새 카테고리, 항목이름, 내용, 마감 기한, 중요도, 완료율 입력-> 입력받은 데이터로 새로운 아이템 생성 -> updateItem 메소드로 수정 -> 영향받은 항목이 0개보다 크면 수정되었다고 출력

* 1. **수정할 항목 번호 입력 -> 번호 존재 여부 체크 기능**

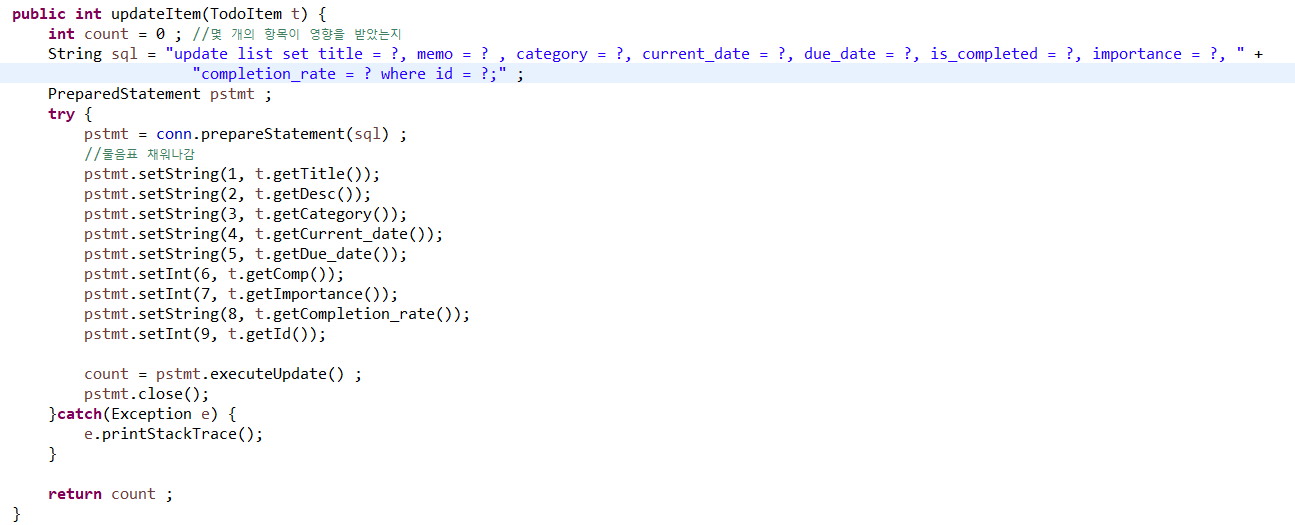
수정할 항목 번호를 입력받고, 입력받은 번호가 리스트에 있는 항목의 id와 같으면 플래그 변수를 1로 바꾸고, 해당 항목을 출력한다. -> 이 경우 while문을 빠져나오게 된다.

만약 리스트로부터 모든 항목을 가져와 입력받은 번호와 id가 같은지 검사를 끝냈음에도 플래그 변수가 0이라면 잘못 입력되었다고 알려준 후, 다시 입력받는다.

* 1. **중요도 입력 적절성 체크 & 제목 중복 체크 기능**

이 기능은 add에서 사용하고 설명했던 기능과 동일하다

* 1. **새로운 아이템으로 데이터를 수정하는 기능 : TodoList > updateItem**



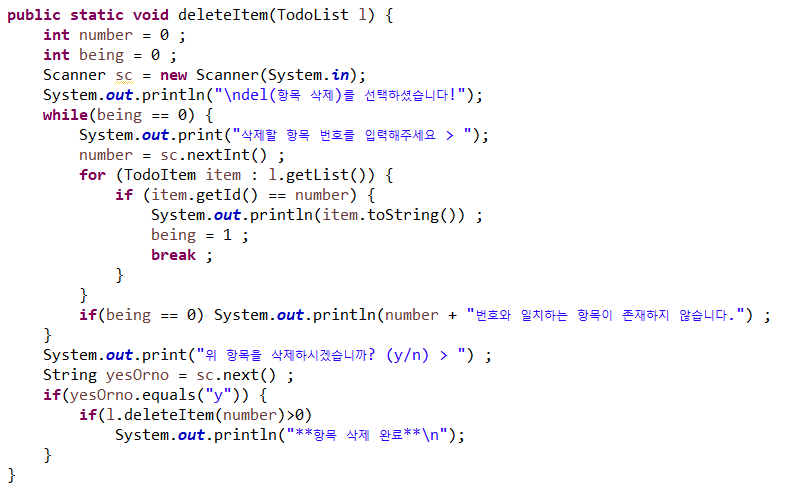
?포함된 sql문 생성(8개 필드 값 set, where로 id 위치 지정) -> preparedStatement 생성 -> ?에 TodoItem에서 값을 가져와서 set -> 실행 -> 실행으로 영향받은 항목의 개수 리턴 -> 정상적으로 수행되었다면 1이 리턴 될 것.

이 과정에서도 새로운 아이템을 추가하고, 수정할 때 새로 추가 된 두 가지 필드가 추가되어 수정되었음을 알 수 있다.

**<< 삭제 : 기본필드 + 완료 체크 필드 + 2개의 추가 필드 – 15점 충족 >>**

**[ del ]**

1. **TodoUtil > deleteItem**



Scanner객체 생성 -> 항목 삭제 선택 출력 -> 삭제할 항목 번호 입력 -> 정말 삭제할 것인지 재확인 -> 항목 삭제

* 1. **삭제할 번호 존재 여부 확인, 재입력 기능**

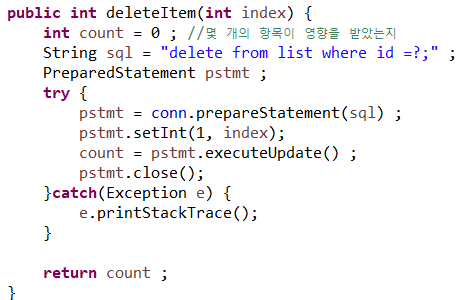
리스트로부터 모든 아이템을 하나씩 가져와서 각 아이템의 id와 입력받은 삭제 항목 번호가 일치하는지 확인 -> 일치하는 항목을 찾으면 출력 후, 플래그 변수를 1로 변경 -> while문을 빠져나옴.

만약 일치하는 항목을 끝까지 찾지 못했다면 일치하는 번호가 존재하지 않는다고 출력 후, 다시 삭제할 항목 번호를 입력받기

* 1. **삭제 여부를 재확인 하는 기능**

삭제여부를 묻고 y가 입력되면 deleteItem메소드 호출 후, 정상적으로 삭제되었다면(0보다 큰 값이 반환되었다면) 항목 삭제 촨료라고 출력

* 1. **TodoList > deleteItem**

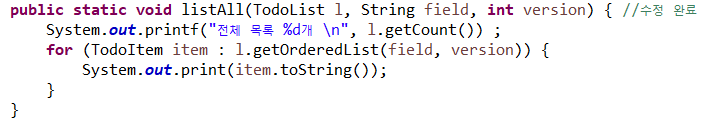


Sql문 생성 -> preparedStatement생성 -> 파라미터로 받은 index로 조건 setting -> 실행 -> 영향받은 항목 개수 count에 저장 후 반환 -> 정상적으로 실행 되었다면 1이 반환 됨.

**<< 기타 기능 >>**

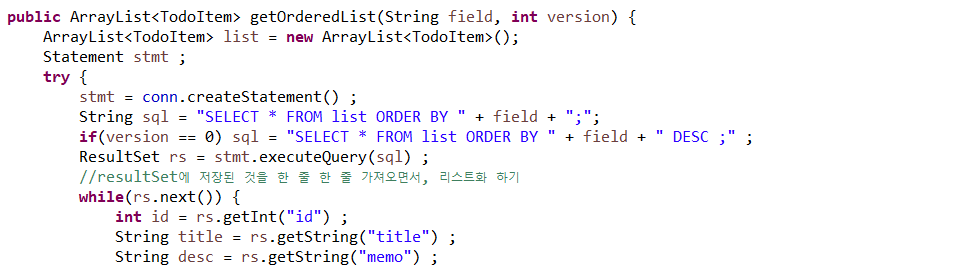
**[ ls\_name\_asc ] , [ ls\_name\_desc ], [ ls\_date ] , [ ls\_date\_desc ]**

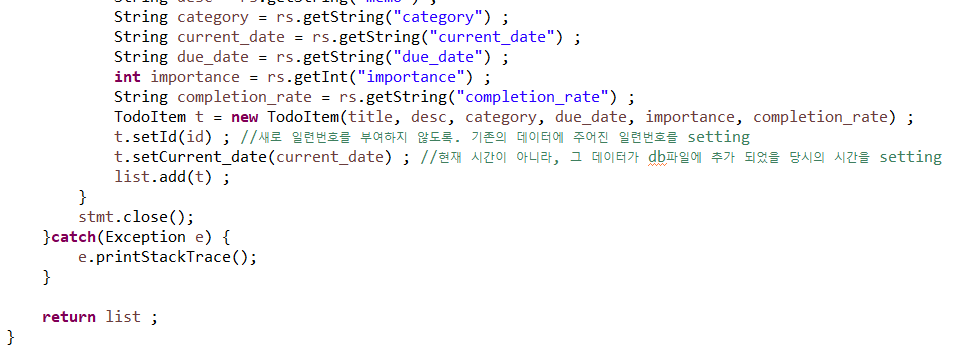
1. **TodoUtil > listAll(TodoList , 기준 필드, 오름/내림 차순 버전)**



전체 목록 개수 출력 -> 해당 필드와 버전을 기준으로 정렬하여 리스트를 반환받아 각 항목을 순차적으로 출력

* 1. **TodoList > getOrderedList**



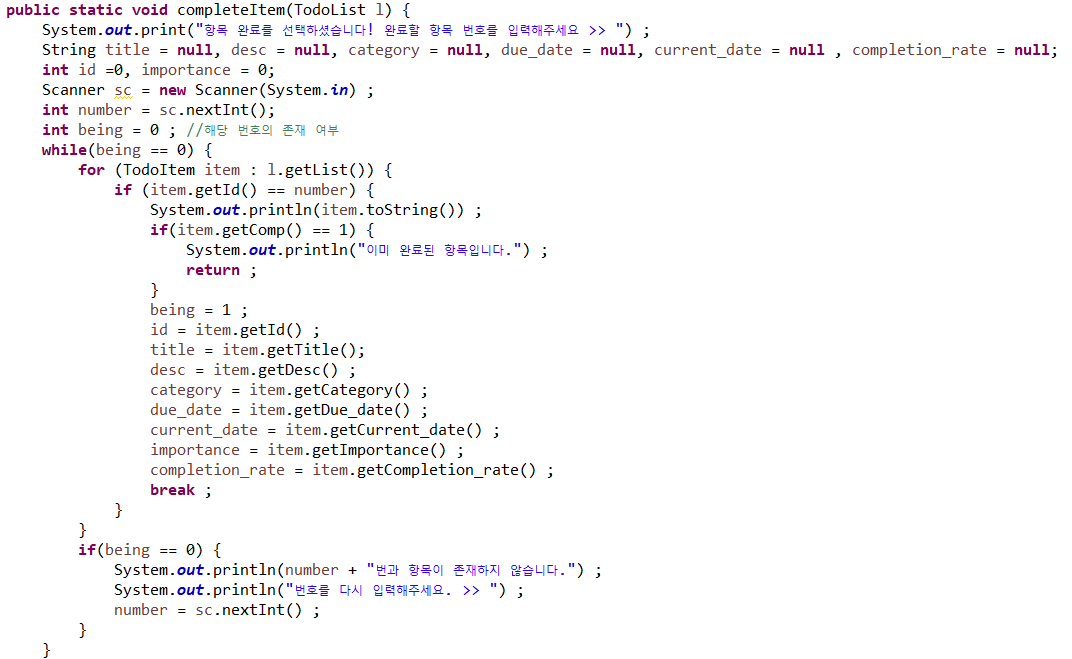


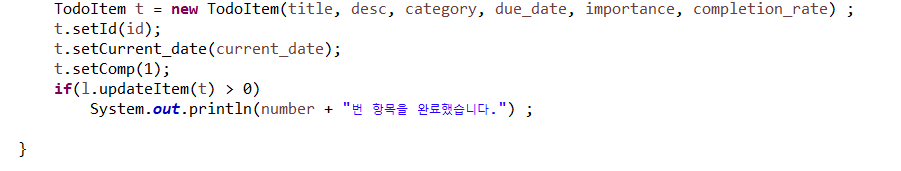
반환할 정렬된 리스트를 저장할 ArrayList 생성 -> 파라미터로 받은 정렬 버전에 따라 sql문 생성(0 : 내림차순 / 1: 오름차순) -> 실행 -> 반환받은 ResultSet으로부터 필드 값을 가져와서 새로운 아이템 생성 -> 아이템을 리스트에 추가하여 완성된 리스트 반환

이 과정에서도 새로 추가 된 두 개의 필드로 인해 ResultSet에서 필드 값을 가져오고, 새로운 아이템을 생성하는 부분의 코드가 수정되었다.

**[ comp ]**

1. **TodoUtil > completeItem**





항목 완료 메뉴 선택 출력 -> Scanner 객체 생성 -> 완료할 항목 번호 입력 -> 해당번호의 존재 여부 확인 -> 존재할 경우, 해당 아이템의 필드값 get으로 가져와서 저장 -> 저장한 값으로 새로운 아이템 생성 -> comp를 1로 세팅하고 updateItem을 호출하여 0보다 큰 값이 반환되면 해당 항목을 완료했다고 출력

이 메뉴에서도 두 가지 필드가 새로 추가 됨에 따라 아이템을 새로 생성하고, 아이템을 생성하기 위한 값을 받는 과정의 코드가 수정되었다.

* 1. **번호 존재 여부 확인 기능**

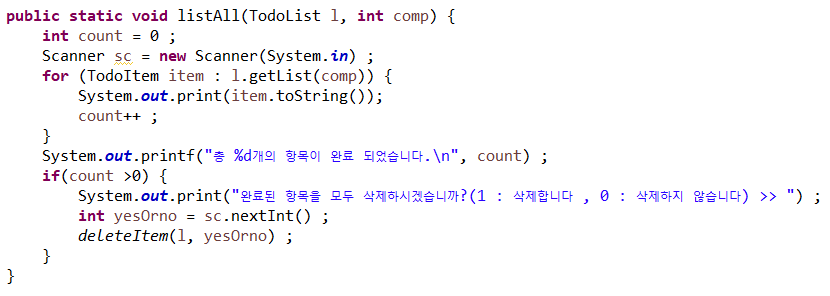
해당 기능은 del 메뉴 기능에서 사용된 기능과 동일하다.

* 1. **이미 완료된 항목인지 확인**

입력받은 번호를 id로 가지는 항목을 발견했을 경우, 해당 항목이 이미 완료된 항목인지 아닌지를 getComp의 값으로 확인한다. 만약 이미 완료된 항목이라면 완료된 항목이라고 출력한 후, comp메뉴를 종료한다

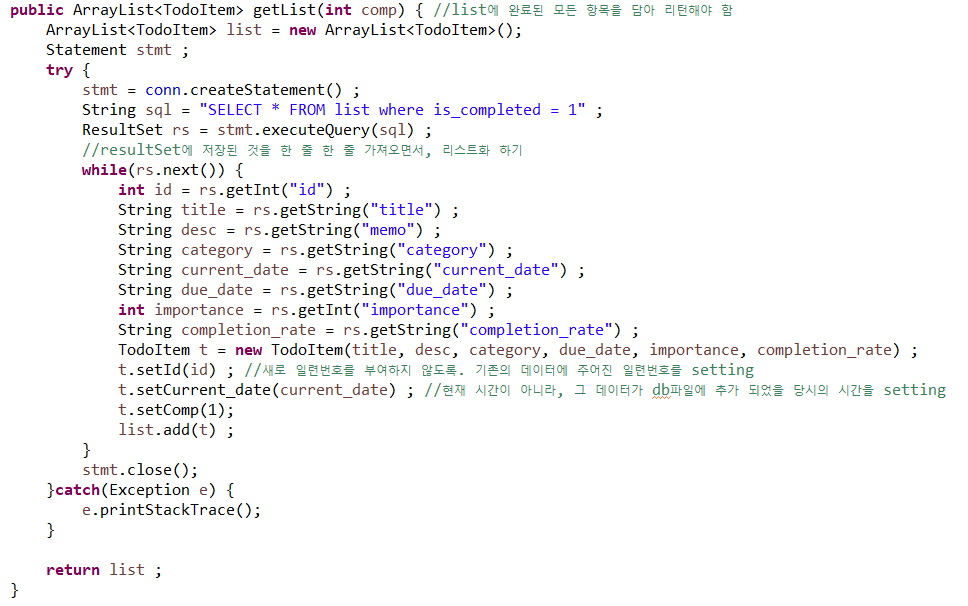
**[ ls\_comp ]**

1. **TodoUtil > listAll**



Scanner 객체 생성 -> 완료된 항목만 리스트로 만들어 반환받아 각 항목을 출력 -> 출력하면서 완료된 항목의 개수를 카운팅한 결과를 출력 -> 완료된 항목을 한 번에 삭제하는 기능 실행

* 1. **TodoList > getList(comp)**



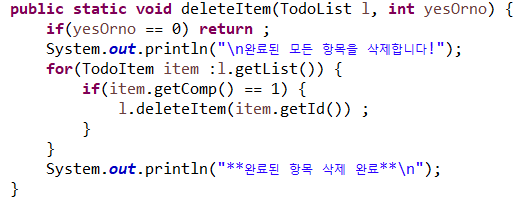
완료된 항목만 담아 리턴할 ArrayList 객체 생성 -> statement생성 -> sql문 생성(is\_completed 필드에 1이 저장되어 있는 항목만 받아옴) -> 실행 -> 받아온 각각의 항목에서 필드 값을 받아와서 저장 -> 저장된 값으로 새로운 아이템 생성 -> 완료된 항목을 리스트화 하는 것이므로 comp는 1로 세팅 -> 리스트에 아이템 추가 -> 완성된 리스트 반환

이 메소드에서도 새로 추가된 필드 두개로 인해, ResultSet으로부터 각 항목의 필드 값을 받아오는 코드와, 새로운 항목을 추가하는 코드가 수정되었다.

* 1. **TodoUtil > deleteItem(TodoList , int yestOrno )**

**<< 새로운 멋진 기능 제안/구현 1 : 완료된 항목만 골라서 한 번에 삭제하기 – 10점 >>**

이 부분은 ls\_comp 메뉴를 실행했을 때, 완료된 항목을 사용자가 한 번에 삭제할 수 있도록 하면 편리할 것이라는 생각이 들어 구현한 새로운 기능이다.



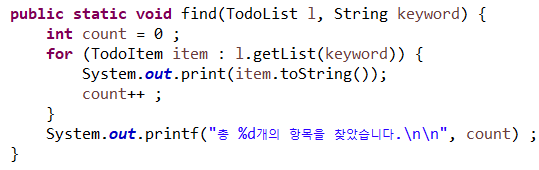
완료된 모든 항목을 한 번에 삭제할 것인지 여부를 파라미터로 받았다. yesOrno에는 1 또는 0의 값이 들어있다. 0은 삭제하지 않겠다는 의미이므로 return해서 ls\_comp메뉴를 종료시킨다. 그렇지 않을 경우에는 모든 항목을 삭제한다고 출력한 후 실행한다. -> 모든 리스트로부터 각 항목을 가져와 각 항목이 완료 되었는지 확인한다. -> 완료된 항목만 삭제한다.

모든 삭제가 끝났으면 완료된 항목 삭제 완료라고 출력한다.

완료된 항목만 찾아서 출력하는 과정에서 l.getList(comp) 메소드를 활용해 조건문 하나를 줄일 수도 있다.

**[ find ]**

1. **TodoUtil > find**



TodoList와 찾고자 하는 키워드를 파라미터로 받는다 -> keyword가 제목 또는 내용에 포함되어 있는 항목만 리스트로 반환 받는다 -> 각항목을 출력한다 -> 각 항목을 출력하면서 카운팅한 개수를 출력한다

* 1. **TodoList > getList(keyword)**



키워드가 포함된 항목만 담아 리턴할 ArrayList 생성 -> preparedStatement 생성 -> sql문 생성(제목 또는 메모에 키워드가 포함되는 것을 조건으로 한다) -> sql문에 keyword를 setting하여 실행 -> ResultSet으로부터 각 항목으로부터 필드 값을 가져온다 -> 가져온 값으로 새로운 아이템 생성 -> 리스트에 추가 -> 완성된 리스트 반환

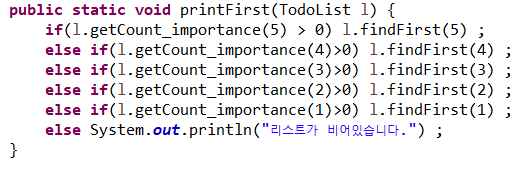
이 과정에서도 두 개의 필드가 추가 됨에 따라 각 항목으로부터 필드 값을 가져 오는 코드와 새로운 아이템을 생성하는 코드가 수정되었다.

**<< 새로운 멋진 기능 제안/구현 2 : 우선순위가 가장 높은 항목 찾아서 보여주기 – 10점 >>**

새로운 필드인 importance – 중요도를 추가했기 때문에 사용자에게 더 좋은 기능을 제공할 수 있게 되었다. 해야할 일이 많아도 그 중에 가장 중요하고 우선순위를 두어야 할 항목들을 먼저 하는 것이 효율적이기 때문에 이러한 과정을 편리하게 하기 위한 기능이다. 이 기능에서는 due\_date와 importance 필드를 사용해, importance가 가장 높은 것 중에, due\_date가 가장 얼마 남지 않은 항목을 1st 메뉴에서 보여준다.

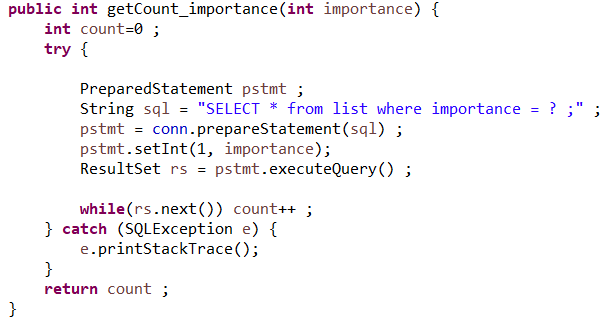
**[ 1st ]**

1. **TodoUtil > printFirst**



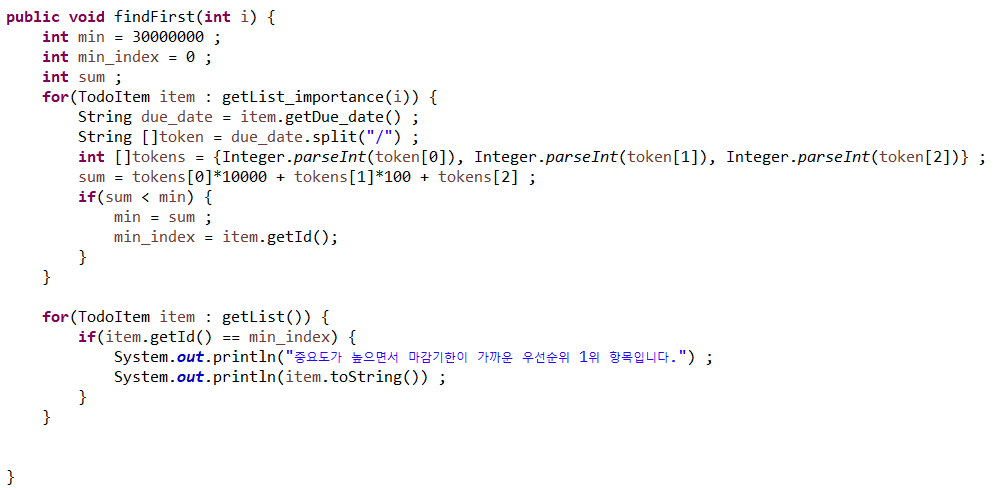
리스트를 파라미터로 받아 조건문 안에서 importance가 각각 5인, 4인, 3인, 2인, 1인 항목들의 개수를 반환 받는다. 중요도가 가장 높은 것부터 검사하면서 만약 중요도가 높은 항목이 하나라도 있다면, 해당 중요도 항목 안에서 findFirst를 호출한다. 조건문을 모두 통과한다면 아직 아무 항목도 들어있지 않은 것이므로 리스트가 비어 있다고 출력한다.

* 1. **TodoList > getCount\_importance**



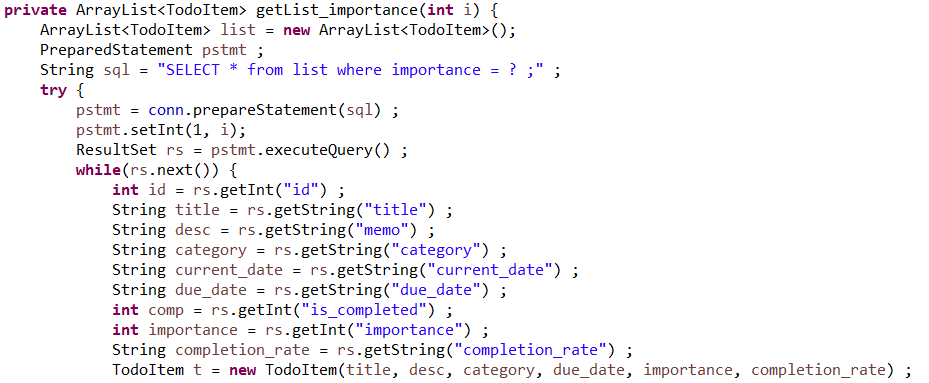
preparedStatement생성 -> sql문 생성(importance를 조건으로 모든 리스트를 불러옴) -> 파라미터로 받은 imortance를 sql문에 setting하여 실행 -> importance의 값에 부합하는 항목들을 반환받음 -> 반환받은 ResultSet으로부터 한줄씩 읽으면서 항목의 개수 세기 -> 결과 반환

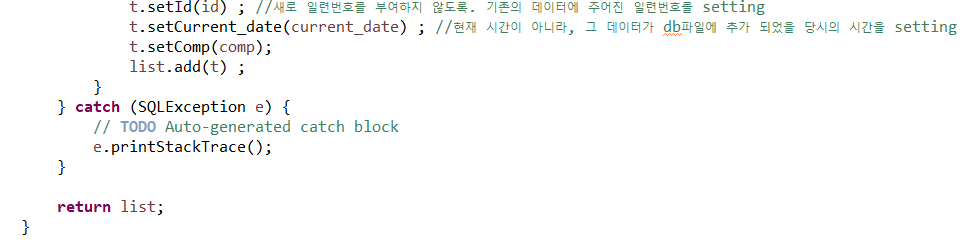
* 1. **TodoList > findFirst**



최소값 30000000으로 초기화 ( 각 항목별로 due\_date를 비교하기 위해 3000년이라는 큰 값으로 초기화 해 둠) -> 가장 due\_date가 가까운 항목의 id를 저장하기 위한 변수 정의 -> getList\_importance 메소드로 파라미터로 받은 중요도 값을 가지는 모든 항목을 가져온다 -> 각 항목의 마감 기한을 읽어옴 -> 마감기한을 split를 이용해 /를 기준으로 token을 나눠서 배열에 저장 -> string으로 저장되어 있는 토큰을 정수형으로 변환하여 in형 배열에 저장 -> 년에는 10000을, 달에는 100을 곱해서 세 토큰을 모두 더하기 -> 더한 값을 최소값과 비교하여 더 작은 경우에 최소값을 바꾸고, 인덱스는 해당 항목의 id를 가져와서 저장 -> 반복이 끝나면 마감기한이 가장 얼마 남지 않은 항목의 id를 알고 있는 상태 -> 다시 반복을 돌려서 해당 id를 가지는 항목을 찾아 출력

**1-2-1. TodoList -> getList\_importance**





중요도를 조건으로 항목을 가져와 리스트로 만들어 리턴할 ArrayList 생성 -> 파라미터로 받은 importance 값을 조건으로 하여 모든 항목을 가져오는 sql문 생성 -> preparedStatement 생성 -> ? 값을 파라미터로 받은 importance로 세팅 -> ResultSet에 받은 모든 항목으로부터 각각의 필드 값을 읽어옴 -> 읽어온 값으로 새로운 아이템을 생성하여 리스트에 추가 -> 완성된 리스트 반환

이 과정에서도 필드가 두 개 추가됨에 따라 ResultSet으로부터 필드 값을 읽어오는 코드와, 새로운 아이템을 생성하는 코드가 수정되었다.

**<< 소감문 >>**

자바를 배우고 수업을 들은 것이 아니어서 처음에는 객체 지향 프로그래밍에 익숙하지 않아 어려움이 많았지만 이론부터 배우지 않은 상태로 직접 해보고 부딪히면서 배워갈 수 있었고 그 결과 객체 지향 프로그래밍에 대한 이해도가 많이 높아졌습니다. 앞으로 컴공 공부를 하면서 또 모르는 것이 나왔을 때 직접 찾아보고 실습해가면서 배워가는 것이 중요할 텐데 그 경험을 해볼 수 있어서 좋았습니다. 그리고 데이터 베이스를 활용하는 것을 배우지 않았다면 데이터를 자바 프로그램 안에서만 만들고 파일에 저장하며 효율적이지 못한 방식만 사용했을 텐데 효율적으로 데이터를 관리할 수 있는 방법을 알게 되어 좋았습니다. 개인 프로젝트 중후반 까지는 열심히 제출해주시는 과제의 기능을 구현하며 따라가다가 마무리를 하면서 사용자가 더 편리하게 이용할 수 있도록 제목 중복체크, 숫자가 적절히 입력되었는지 체크, 정말 삭제할 것인지를 재확인하는 등의 기능을 꼼꼼히 생각해보고 구현했습니다. 이렇게 하면서 사용하기에 편리한 프로그램을 만들기 위해 프로그래머가 고려해야 할 부분이 무엇인지 알 수 있었습니다.