CMPE 321 - Summer '18

Assignment 2

Implementing Storage Manager System

Gamze Gülbahar
2016400333

1. Introduction

In this project I am implementing the storage manager system designed in my previous assignment. There are some changes from the first design. I will explain them in the changes part of my report.

Our system consists of a system catalog and .dta files for each type. All files are made up of pages that hold records. System catalog is a much smaller file than the others, however its records are greater in size. The system is optimized for a 10-MB disc drive.

I created two separate java classes for DDL and DML, in which I defined a method for each operation. The user chooses the operation in the main class and then the method for that operation is called from the related class.

When a type is created, besides inserting a record to the system catalog, a .dta file with the same name as this type is also created in the same directory as the system catalog file. For example, for a type named ‘Teacher’, there will be a Teacher.dta file in our base directory.

Most of the code is based on the pseudocode, especially those in the DML class. I explained the numbers, variables and other complicated points in the code with brief comment lines. Slight deviances from the pseudocode will be explained in the next section with their reasons.

2. Changes From The Initial Design

Changes in assumption and constraints:

- Firstly, I reduced the size of system catalog file from 12 KB to 2 KB after realizing that it is not necessary to create so many different types.
- I was storing the information of which records are in that page in the page header and it was called the record range (2 bytes). I decided it was unnecessary, so removed it from the design while implementing.
- Remaining space and next page pointer information in the page headers were 1 byte. However, as there is no 1 byte data type for numbers in programming languages, I defined them as short integers so they are now 2 bytes. I did the same for next record pointer in the record headers.
- I reduced the maximum size of type name field from 18 characters (18 bytes) to 12 characters (12 bytes) and of the field name from 24 characters (24 bytes) to 9 characters (9 bytes), because these fields are kept in system catalog file and the former sizes seemed too much. Thereby, it is possible to keep much more records in the system catalog.
- After these changes, each record of sys.cat file is 100 bytes and each record of .dta files is 41 bytes. I fixed these sizes for the records and also fixed the sizes of the fields of the sys.cat records in order to read and manipulate the files more easily.
- In the implementation deleted flag comes before next record pointer, just the opposite of former design. This change is not based on a reason, it stems from my confusion during coding.

Apart from these changes, all the assumptions and constraints were applied the same as the former design.

Changes in the pseudo code:

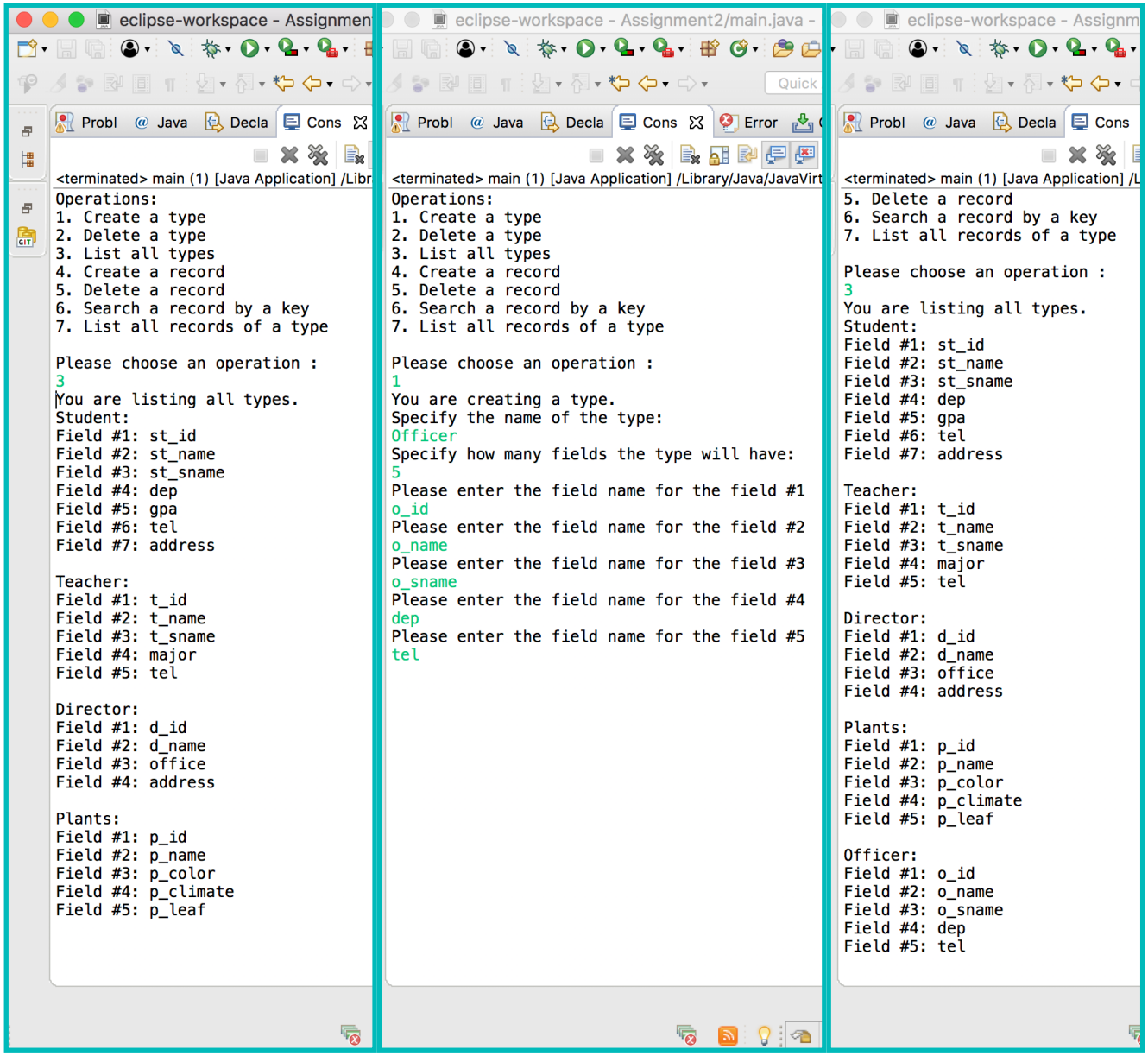
- **createType:** isDeleted variable was unnecessary in the createType method so I didn't use it. I was checking the page remaining space before the page loop, now it is checked after sifting through existing records. I turned the 'If next record exists' condition into a loop so that the algorithm checks until the end of the page if another record exists.
- **deleteType:** I used the 'If next record exists' condition in the initial algorithm I turned it into a loop so that the algorithm checks until the end of the page if another record exists.
- **listingAllTypes:** I wrote the pseudo code so that it only lists type names in the previous design. While implementing, I changed listing only type names to listing type names and field names of the type so that this method is more comprehensive. In addition, I forgot to check if the type is deleted before printing.In the code I included that condition.
- **createRecord:** There is no obvious change in this method.
- **deleteRecord:** 'recordFound' is replaced with 'recordDeleted' variable. Also, algorithm was checking 'if the next record exists' with a condition block, I implemented it as a loop so that it checks until the last record.
- **searchRecord:** Again I made the next record condition a loop for the same reason as the previous method.
- **listAllRecords:** Again, I forgot to check if the record is deleted before printing.In the code I included that condition.
- I defined an additional method in DML class, which return a string array holding the field names of the type that is given as a parameter. I found it useful to print the the field names of a type while creating a record for it or listing all records of it. It also provided me with the field number of the concerned type, which I used in my loop to specify the number of fields that will be printed. In my design, records of the .dta files do not keep the information of field number of the record as a field in it, so I needed this additional method.

3. Sample Usage & Outputs

DDL Operations

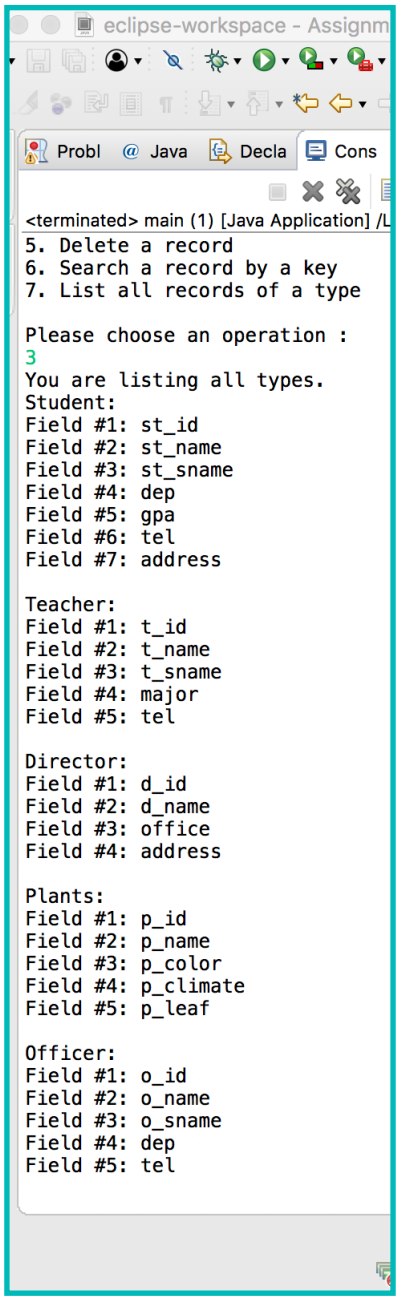
---

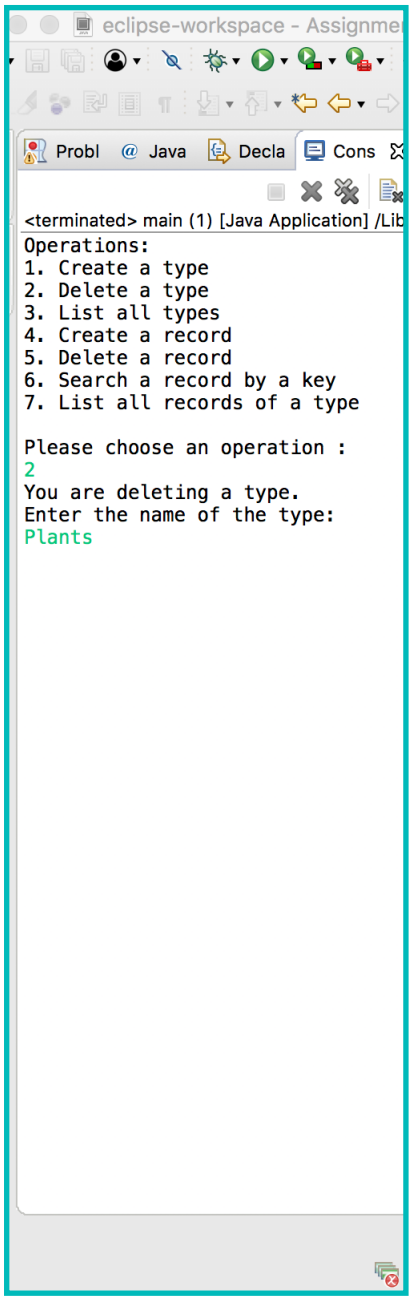
1. Creating a type:

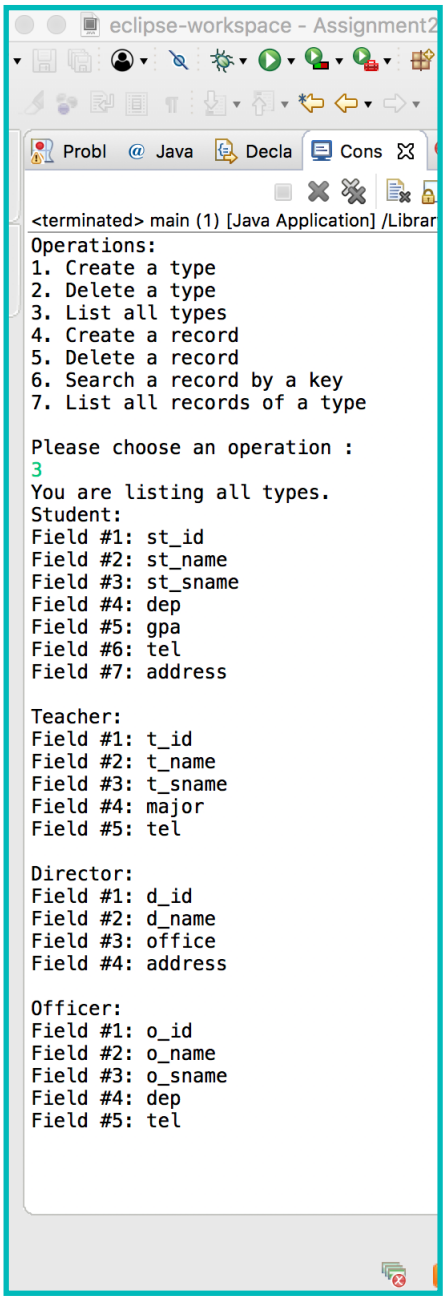


before creating after

2. Deleting a type:

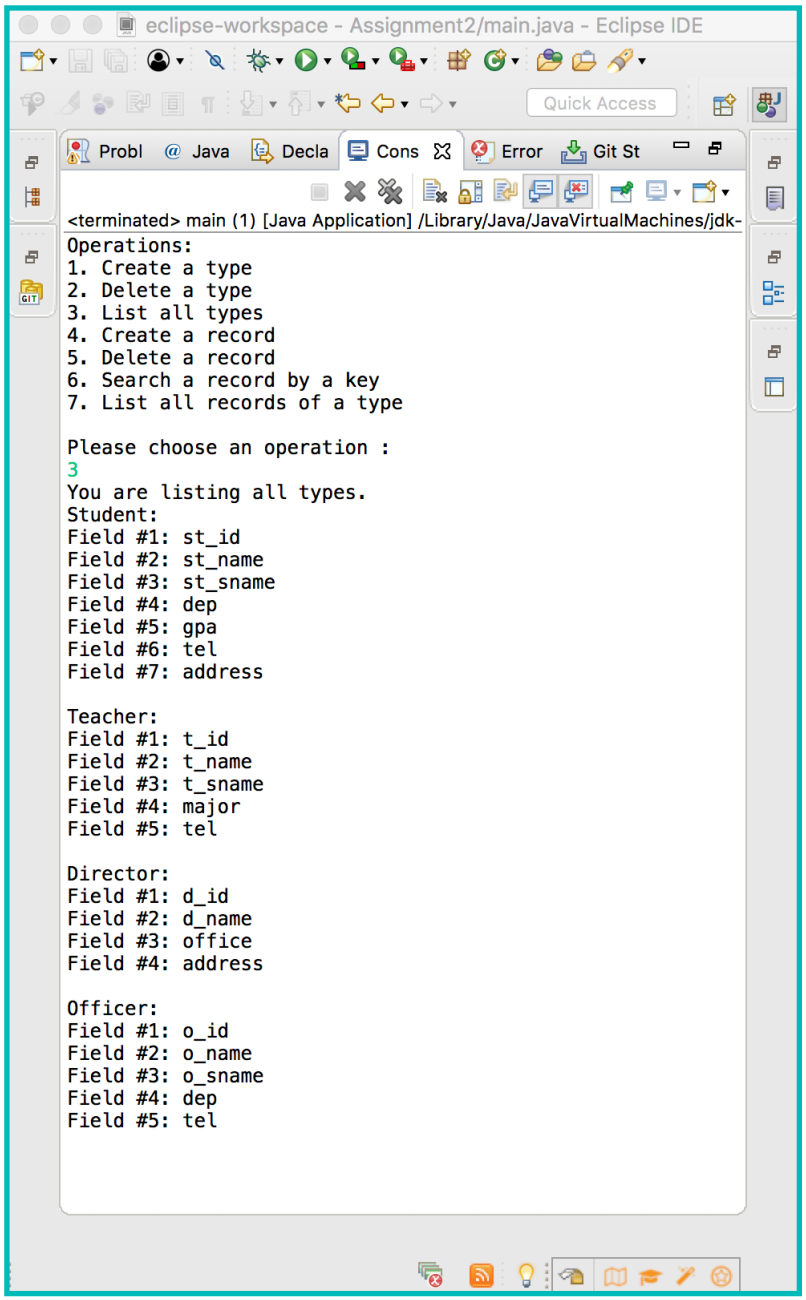






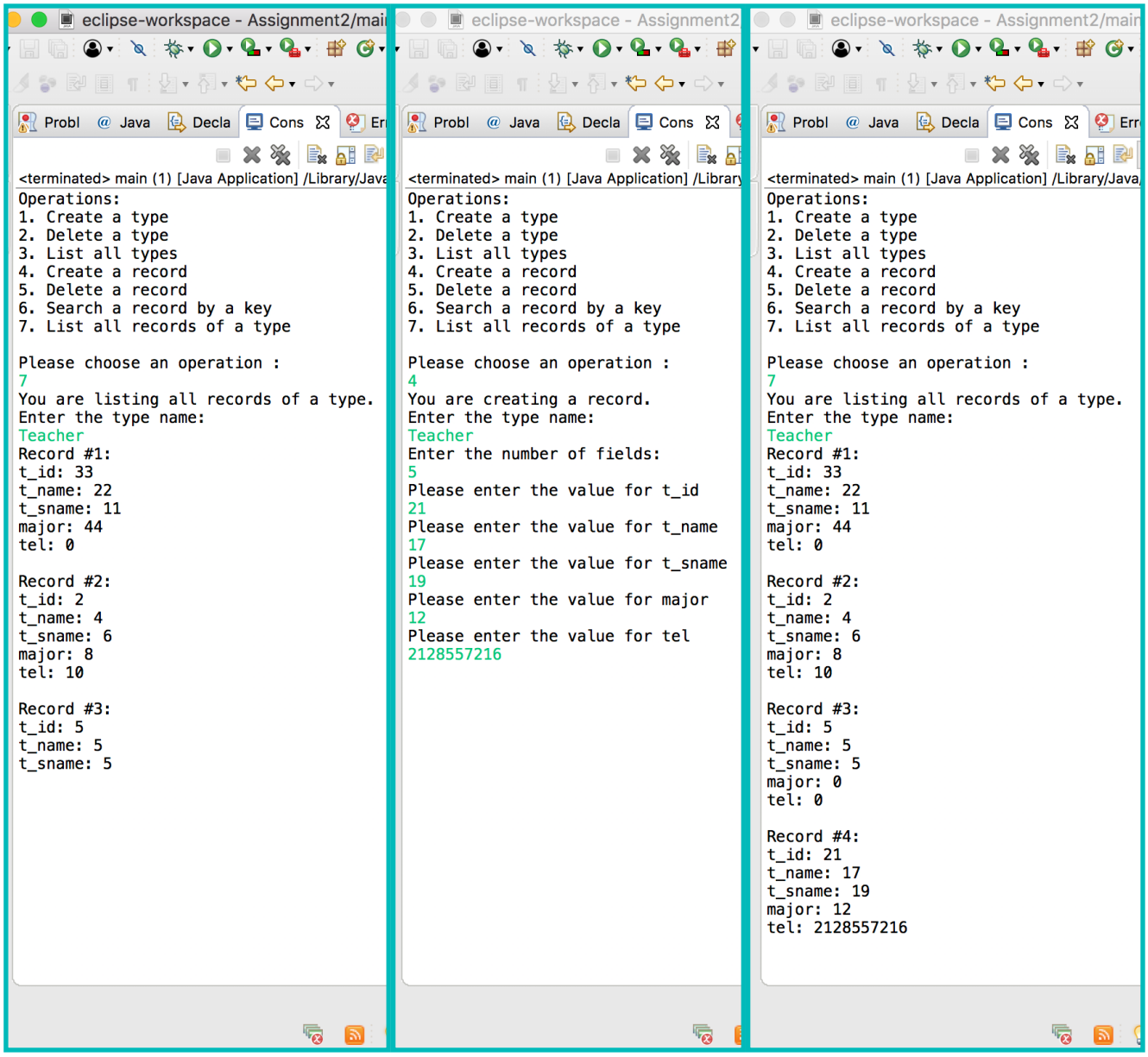
before deleting after

3. Listing all types:



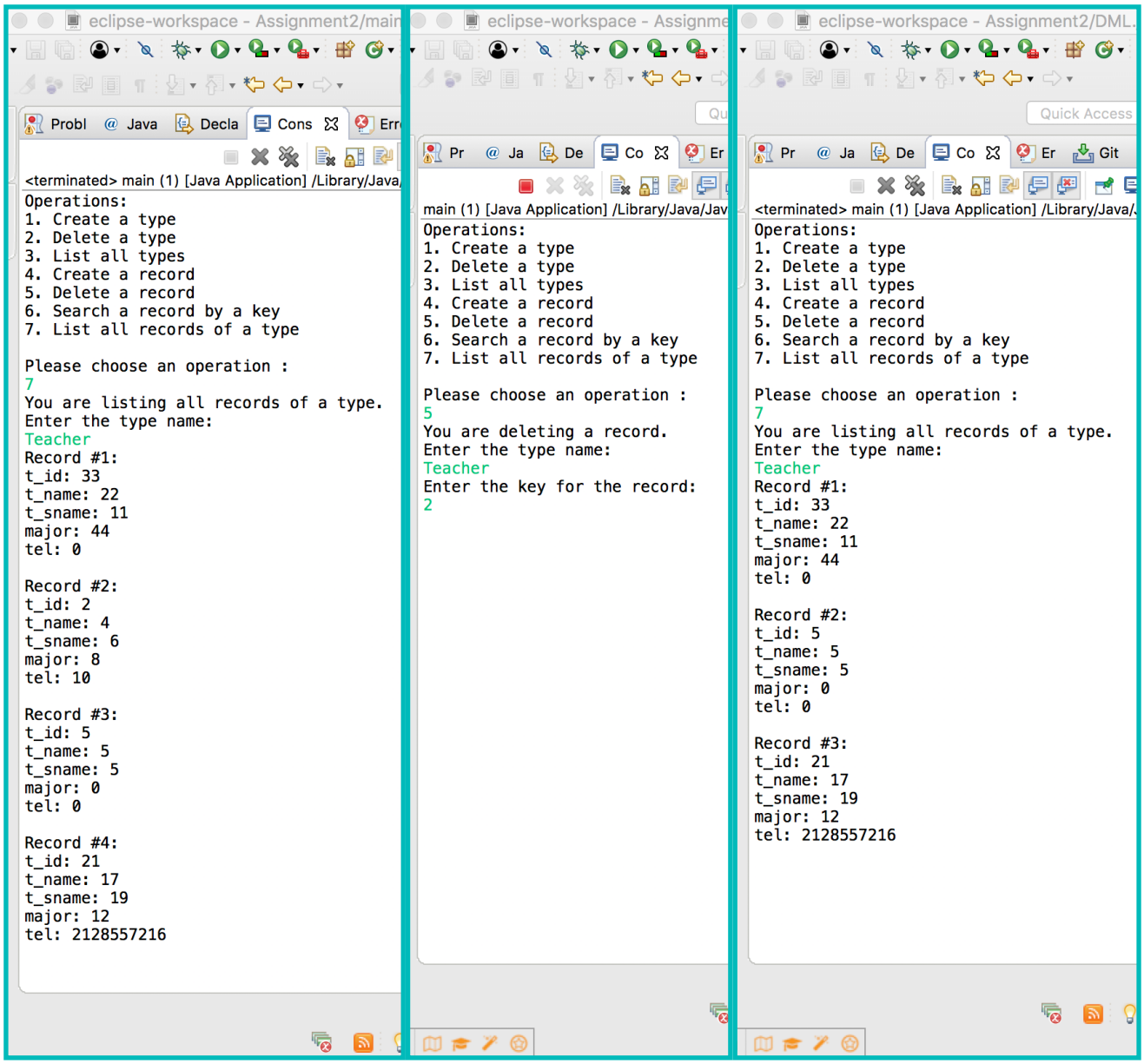
DML Operations

1. Creating a record:



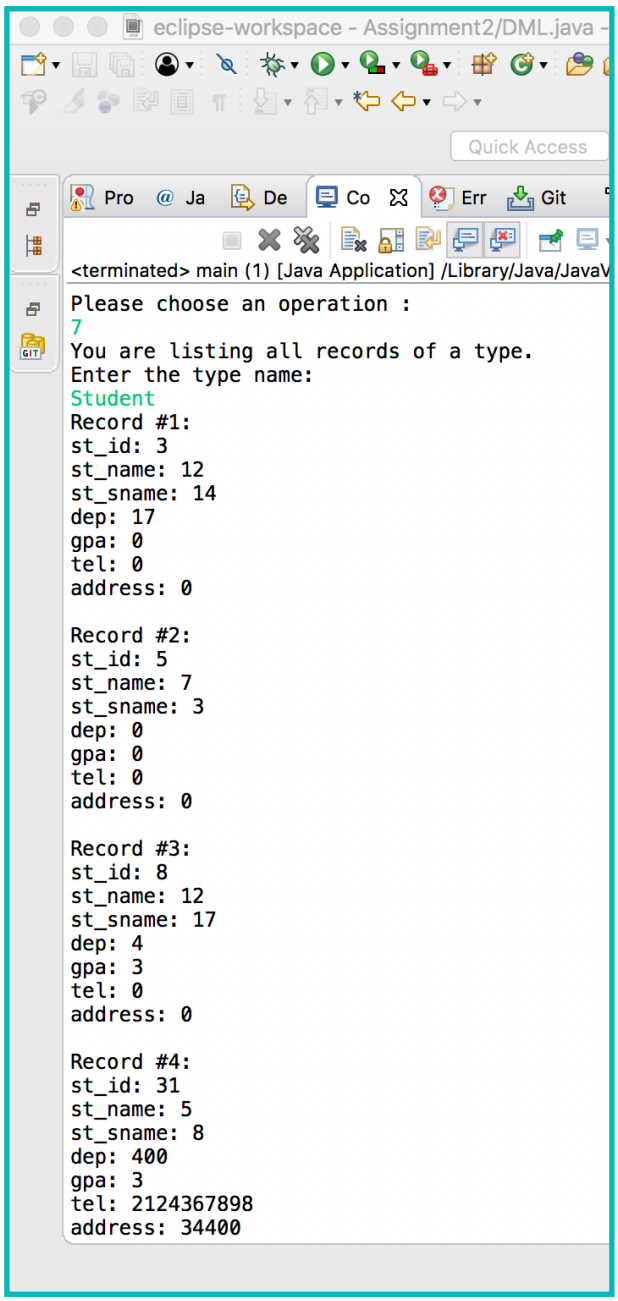
before creating after

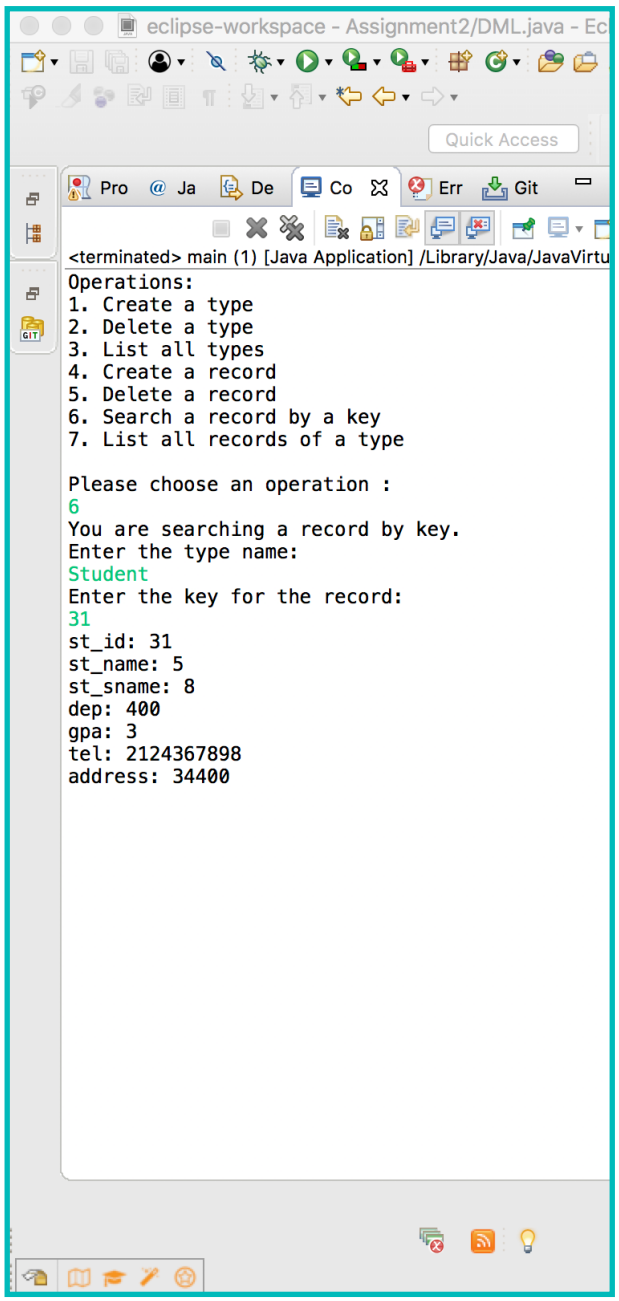
2. Deleting a record by key:



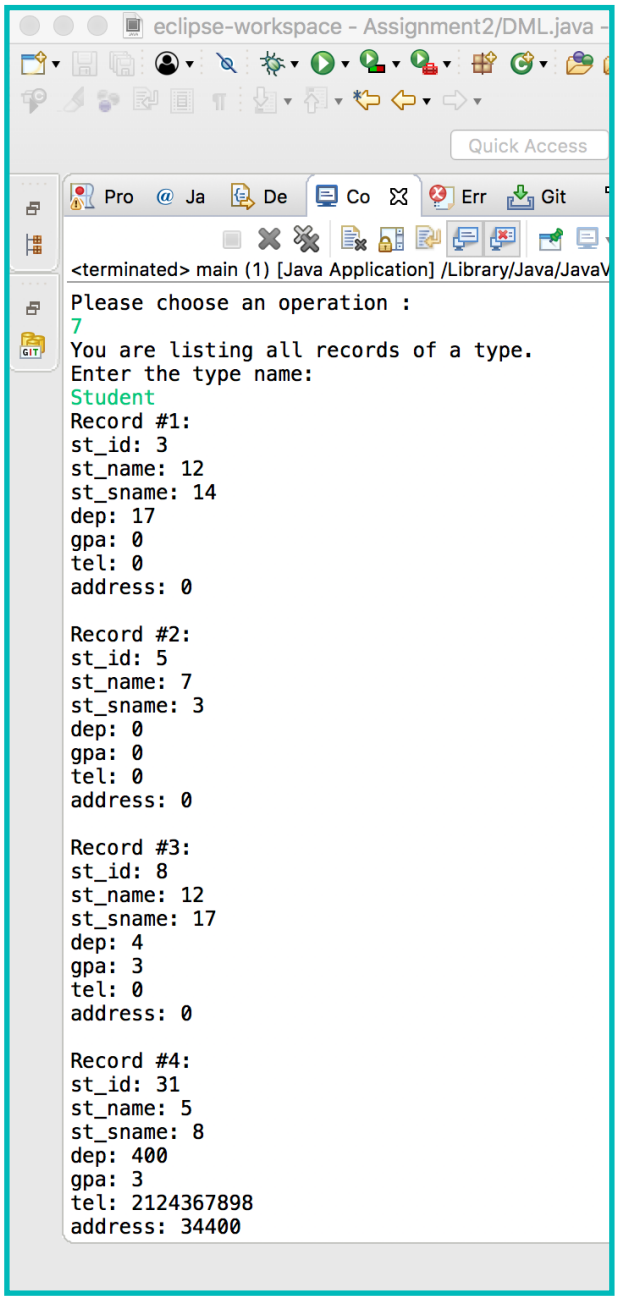
before deleting after

3. Searching a record by key:





4. Listing all records of a type:



4. Conclusions & Assessment

I personally experienced that it is almost impossible to implement the initial design exactly as it is. There will always be infeasable points and error-throwing algorithms. In this second assignment, I learnt to get over those setbacks and find solutions to errors. My first realization was about the system catalog. It was quite big in size and inefficient to be a metadata file. I fixed it with more optimum field smizing and numbering. Another point I realized was that even shortest integer variables are 2 bytes so I cannot assume any number is stored in 1 byte.

I further comprehended what is meant by the description that ‘a page is the amount of data transferred from the hard disk to the buffer of the CPU’. The that helped me get it was getting my program to read all the data in the files page by page. I used an integer variable called ‘cursor’ to move along the bytes of the file and through some operations using this variable and multiples of 2048 bytes (2KB) I constituted the concept of page on my program. So many time during this process I face errors because of misplacing this cursor variable and tried hard to debug them.

By implementing a design which was too abstract and conceptual at the beginning, I gained a great insight into how a real database works and how the data is stored in files at the background. This is very advantageous in terms of being aware of what is going on behind the scenes while creating and manipulating databases in real life situations.