CS348 Project #1

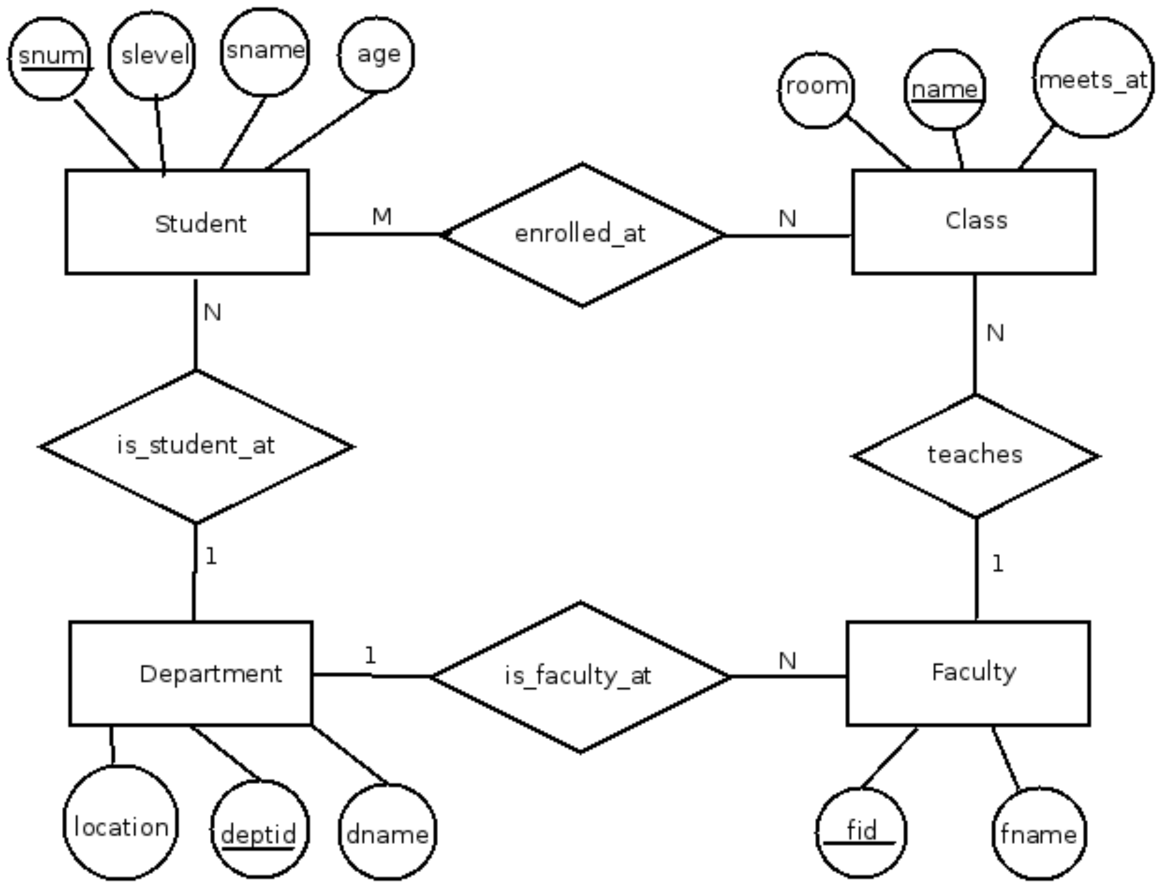
Overview

In this project, you'll create a simple information system to store course enrollment data for a university. This project will let you see the close similarity between the ER model and Object Oriented Programming.

Due Date: Sep 13, 2013 @ 11:59 pm

Requirements

You will write a console-based Java application to serve as a database for university course enrollment information. Your application will essentially store data based on the following ER diagram:



When you start your application, you should present the user with the following prompt:

```
Please enter a command.  Enter 'help' for a list of commands.
>
```

If the user enters ***help***, your application should print the following:

```
add_department DEPTID DNAME LOCATION
add_student SNUM SNAME SLEVEL AGE DEPTID
add_faculty FID FNAME DEPTID
add_class NAME ROOM MEETS_AT FID
enroll SNUM NAME
get_students_in_department DEPTID
get_students_in_class NAME
get_classes_for_student SNUM
get_classes_for_faculty FID
delete_class NAME
save FILENAME
load FILENAME
exit
```

You should implement each of the above commands. A detailed description of each command is provided below.

Description of commands

add_department:
DEPTID – the unique identifier for a department (string)
DNAME – name of the department (string)
LOCATION – address of the department (string)

This command should add a department with the given parameters to the database. Adding duplicate DEPTID should not be allowed. In the case of attempting duplicate record addition, your program should print "Error: Department already exists!".

add_student:
SNUM – unique identifier for the student (string)
SNAME – name of the student (string)
SLEVEL – year of the student in the department (Freshman, Sophomore, Junior or Senior) (string)
AGE – age of the student (integer)
DEPTID – identifier of the department the student is majoring at (string)

This command should add a student with the given parameters to the database. Note that a department identified by DEPTID should already exist in the database to be able to add a student majoring in that department. Otherwise, you should print "Error: Department does not exist!". Addition of duplicate SNUM is not allowed. In the case of attempting duplicate SNUM addition, your program should print "Error: Student already exists!".

add_faculty:

FID – unique identifier for the faculty (string)
FNAME – full name of the faculty (string)
DEPTID – identifier of the department this faculty is affiliated with (string)

This command should add a faculty with the given parameters to the database. Note that a department identified by DEPTID should already exist in the database to be able to add a faculty affiliated with that department. Otherwise, you should print "Error: Department does not exist!". Addition of duplicate FID is not allowed. In the case of attempting duplicate FID addition, your program should print "Error: Faculty already exists!".

add_class:

NAME – unique identifier for the class (string)
ROOM – the classroom for this class (string)
MEETS_AT – day and time this class meets at (string)
FID – identifier of the faculty teaching this class (string)

This command should add a class with the given parameters to the database. Note that a faculty identified by FID should already exist in the database to be able to add a class taught by that faculty. Otherwise, you should print "Error: Faculty does not exist!". Addition of duplicate NAME is not allowed. In the case of attempting duplicate NAME addition, your program should print "Error: Class already exists!".

enroll:

SNUM – identifier of the student to enroll in the class
NAME – identifier of the class the student will enroll in

This command should add an enrollment record with the given parameters to the database. Note that a student identified by SNUM, and a class identified by NAME should already exist in the database. Otherwise, you should print "Error: Missing entry!". Addition of duplicate SNUM-NAME tuples is not allowed. In the case of attempting duplicate records, your program should print "Error: Enrollment already exists!".

get_students_in_department:

This command should print the name, year and age of all students majoring in the department identified by DEPTID, displaying one record on each line

Example:

```
> get_students_in_department 001
Pelin_Angin, Freshman, 18
John_Smith, Sophomore, 19
Joe_Black, Junior, 20
```

get_students_in_class:

This command should print the name, year and age of all students enrolled in the class identified by NAME, displaying one record on each line

Example:

```
> get_students_in_class INFSYS01
John_Smith, Sophomore, 19
Joe_Black, Junior, 20
```

get_classes_for_student:

This command should print the name, room, time and faculty identifier of each class the student identified by SNUM is taking, displaying one record on each line

Example:

```
> get_classes_for_student 00178
CS34801, HAASG066, TR12:00pm, 101
CS30702, LWSN1106, MWF8:30am, 105
```

get_classes_for_faculty:

This command should print the names of all classes taught by the faculty identified by FID, displaying one record on each line

Example:

```
> get_classes_for_faculty 105
CS30702
CS31504
```

delete_class:

This command should delete the class identified by NAME from the database. Note that when a class is deleted, all student enrollment records for that class as well as the faculty teaching records for that class should be deleted from the database too.

save:

This command should save the current contents of the database to a file named FILENAME. You may choose a file format as you see fit.

load:

This command should load the contents of the database from the file named FILENAME. This command should clear out existing data before loading the contents of FILENAME.

exit:

Exit the application. Unsaved data will be lost.

Evaluation

Your project will be evaluated based on how well it meets the specification, i.e. the correctness of the output. We might use automated testing, so make sure your program does not crash in between any commands.

Submitting Your Work

Please create a README file that contains identifying information. For example:

CS348 - Project 1

Author: John Doe
Login: jdoe
Email: jdoe@cs.purdue.edu

Include here anything you might want us to know when grading your project.

All commands should be implemented in the template file provided (SimpleDatabase.java). This template provides a way to read in commands from the console and parse the parameters until the user exits the program, so you don't need to worry about that. It also takes care of incorrect user input such as non-existing command, typo or missing parameters. You just need to provide implementations of the commands in the corresponding methods.

To turn in your project, ssh to *lore.cs.purdue.edu*, create a folder named *project1* in your home directory and copy your source files and your README.txt to that folder. We should be able to compile your program by executing:

*javac *.java*

and run your program by executing:

java SimpleDatabase

After copying your files in the folder project1, execute the following command in your home directory:

turnin -c cs348 -p proj1 project1

To verify the contents of your submission, execute the following command:

turnin -c cs348 -p proj1 –v

***** Please make sure your source code is compliant with Java 5.**

Hints:

- Note that order doesn't matter when printing out the results of a query. Your program should not print anything after executing commands for adding any records to the database as well as saving to/loading content from a file if the command was executed successfully. For the error cases described for each command above, you should print the specific error message listed. For the query type commands (i.e. *get* commands), you should print the matching records, one record per line, with a comma in between the different fields of each record.
- You can make use of Java's serialization capabilities (and file I/O) for saving/loading data (i.e. you will not need access to any real database for this project). For in-memory storage of your data during runtime, you can use a data structure like *ArrayList* or *Vector* in Java. Searching for "Java serialization" online should give you many examples of writing/reading of objects to/from a binary file in Java.
- You may want to create separate classes for each entity (Student, Department, Faculty, Class) and relation (enrolled_at, student_at, faculty_at, teaches) and keep objects for each record of an entity in a list (e.g. ArrayList/Vector) maintained for that entity/relation. When saving to/loading data from a file, you can simply read/write these lists, as they are really objects. When loading data from a file, you should first clear the content of all in-memory data structures (Java provides convenience methods for emptying its built-in data structures, so this should be easy).
- Make sure you don't violate any referential integrity constraints when adding data to/deleting data from your database. These constraints are described in the specification of each command above.
- Be careful about not letting duplicate records in your database. Remember duplicate records are identified using the primary key.