# CIST2362 Software Design Document

CSU Database

Kimberly Jackson

# Introduction

CSU has requested a database program be developed for them to maintain information on their student body. All information on the students will be held in files external to the program. These files will be read from and written to, but only at the beginning of the program and before it is closed. There will also be logins in the files, as this database should only be accessible to administrators at the school.

The program will utilize classes for the students' information. A basic Student object will hold the most basic information about the student, like name, student number, major, etc. This class is to be inherited by two different classes, Undergraduate and Graduate, that add a classification for year then degree type and thesis topic respectively.

It will be menu driven. Firstly, of course, will be a login barrier to ensure that the person altering student information is an administrator. That will open up the menu for actually reading student information, altering student information, and editing course information.

All the information pertaining to logins will be read at the beginning of the program and loaded into vectors or arrays. Once the login information is correct, the program will load in the student information files into vectors or arrays for the administrator to utilize.

The menu, once the admin has logged in, will consist of a number of options.

1. Search for student by first or last name

2. Show entire university

3. Show/Edit information

4. Add courses to student's schedule

5. Remove courses from student's schedule

6. Log out

It is set up this way to avoid students accessing their own information.

UML diagram will display all classes and their attributes/operations.

# Use Cases

*Use Case: Correct login - Show information*

* 1. *Program loads and loads in the file containing the logins*
  2. *User is prompted to enter in their username.*
  3. *User is prompted for their password.*
  4. *Both are correct. Student information is loaded into the program.*
  5. *User is treated to a menu consisting of the above 6 options*
  6. *User chooses option 3. Show/Edit student information*
  7. *User is prompted for Student ID - it is assumed that the student is a currently enrolled student and there for some sort of consult, thereby allowing the admin to search via student ID*
  8. *User enters in the 900 number of the student.*
  9. *A search is performed on the student number of all students currently enrolled until it finds the correct student.*
  10. *Student information is displayed.*
  11. *User is prompted if they would like to alter this information.*
  12. *User enters in N or n for no alterations.*
  13. *Returned to the main menu with 6 options.*

*Use Case: Correct login - List All*

1. *Program loads and loads in the file containing the logins*
2. *User is prompted to enter in their username.*
3. *User is prompted for their password.*
4. *Both are correct. Student information is loaded into the program.*
5. *User is treated to a menu consisting of the above 6 options*
6. *User chooses option 2. Show entire university*
7. *Program sorts array alphabetically by last name*
8. *Program displays every 10 students alphabetically by last name*
9. *Every 10 students, user is prompted if they would like to continue viewing the entire university or they would like to exit*
10. *Display completes and user is returned to main menu*

*Use Case: Correct Login - Edit course information*

1. *Program loads and loads in the file containing the logins*
2. *User is prompted to enter in their username.*
3. *User is prompted for their password.*
4. *Both are correct. Student information is loaded into the program.*
5. *User is treated to a menu consisting of the above 6 options*
6. *User chooses option 4. Add course*
7. *User is prompted for Student ID - it is assumed that the student is a currently enrolled student and there for some sort of consult, thereby allowing the admin to search via student ID*
8. *User enters in the 900 number of the student.*
9. *A search is performed on the student number of all students currently enrolled until it finds the correct student.*
10. *Student information is displayed.*
11. *User is prompted if they would like to alter this information*
12. *User enters Y or y to alter the information and is taken to the course edit function.*
13. *User is shown the student’s current classification, courses, and hours.*
14. *User is prompted about whether they would like to add a course or edit current courses.*
15. *User chooses to edit current courses.*
16. *Current courses are displayed and the user is asked which course to remove/edit.*
17. *User chooses course 3.*
18. *Prompt user if they want to edit or remove this course. User chooses edit.*
19. *User is prompted to input the new course.*
20. *New course replaces the current course.*
21. *Prompt user if they would like to edit more courses.*
22. *User enters N. Student info is saved*
23. *User is returned to the main menu.*

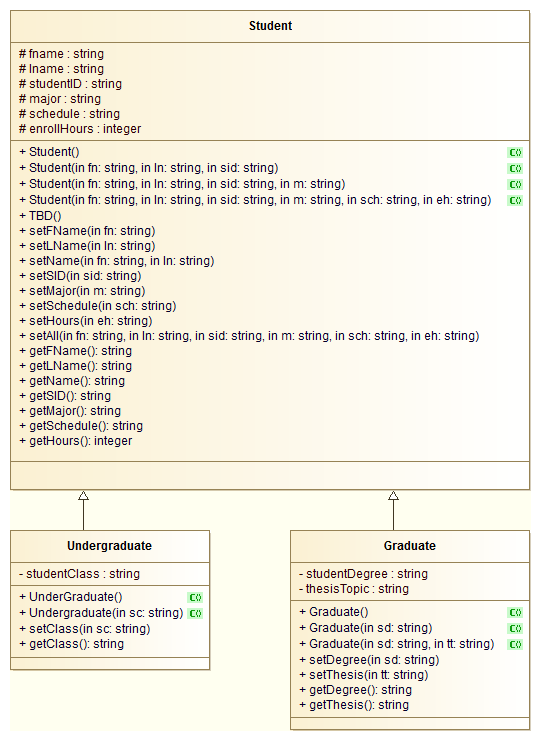
*Use Case: Incorrect Login - Incorrect password*

1. *Program loads and loads in the file containing the logins*
2. *User is prompted to enter in their username.*
3. *User is prompted for their password.*
4. *Username is correct, password is incorrect. Prompt user for password again.*
5. *Username and password are tested again.*
6. *Both are correct. User is taken to the main menu.*

*User Case: Exit*

1. *User has selected 6. Log out*
2. *Program saves all student information to the external files.*
3. *Exit greeting.*
4. *Program saves logins to external file and exits.*

# Design Overview



# System Tasks Description

### Function main

*read from login file until there is nothing left to read*

*username goes in one spot, password in another*

*(some sort of loop)*

*prompt user for username*

*prompt user for password*

*for count = 0 to logins.size()*

*if username == fileuser[count]*

*if password == filepassword[count]*

*welcome admin*

*load up the menu function*

*else*

*password incorrect*

*else*

*username incorrect*

*(end the loop here)*

*display exit greeting*

*write to login file*

*end function*

### function menu

*read from the studentinfo file into a studentinfo array or vector of Student objects*

*probably two different files to separate out the undergrads and grads*

*do*

*display menu*

*prompt user for option*

*switch*

*case 1*

*search function*

*case 2*

*listAllStudents function*

*case 3*

*search function*

*case 4*

*search function > editcourses function*

*case 5*

*search function > editCourses function*

*case 6*

*break*

*default*

*that’s not a valid option*

*while menuoption != 6*

*write to the studentinfo file*

*end menu function*

## function search

*variables for first, last name, and student id*

*(start loop)*

*prompt user for which attribute they would like to use to look a student up*

*input the first name, last name, OR student ID based on their choice*

*perform a search of the student array/vector*

*if student’s name/ID is found*

*display information*

*else*

*student not found. are you sure you put the info in correctly?*

*(end loop)*

## function editcourses

*variables for course information and enrollment hours*

*prompt user for if they want to add or remove/edit student’s current courses*

*if adding*

*accept input, verify input, and save it to that student’s information - make this another function maybe*

*else if removing*

*verify current courses, identify what to remove and how many hours the student will have then, then remove it from the list of courses - to call the add courses function again here?*

*else*

*that’s not an option*

*end courses function*

## function listAllStudents

*accept array/vector as an argument*

*sort array/vector alphabetically by last name*

*display array/vector*

*add a clause in the for loop to set up a paging function - if count % 10 == 0 or something*

# Classes

## Student

**Student constructor - no arguments**

*fname, lname, studentID, major, courses = “ “*

*hours = 0*

**Student - fn, ln, sid as arguments**

*fname = fname*

*lname = ln*

*studentID = sid*

*set the rest per the no argument constructor*

**Student - fn, ln, sid, m as arguments**

*same as constructor 2, but set major = m*

**Student - fn, ln, sid, m, sch, eh as arguments**

*set all attributes at instantiation*

### Mutators

**setFname(fn), setLName(ln), setName(fn, ln)**

*accepts respective values and sets to fname and lname*

**setSID(sid)**

*studentID = sid*

**setMajor(m)**

*major = m*

**setSchedule(sch)**

*schedule = sch*

**setHours(eh)**

*enrollHours = eh*

**setAll(fn, ln, sid, m, sch, eh)**

*set all the things*

### Accessors

**getFname, getLname**

*returns the fname and lname attributes as strings*

**getName**

*string wholename = fname*

*append a space to the end of wholename*

*append lname to wholename*

*return wholename*

**getsid**

*returns studentID*

**getMajor**

*returns major*

**getSchedule**

*returns the schedule*

**getHours**

*returns enrollHours*

## Undergraduate

**Undergraduate default constructor**

*classification = Freshmen*

**Undergrad constructor 2**

*classification = sc*

**setClass(sc)**

*studentClass = sc*

**getClass**

*return studentClass*

## Graduate

**Graduate default constructor - no args**

*degree type and thesis set to nothing*

**Graduate constructor 2**

*degree type = sd*

*thesisTopic = “ “*

**Graduate constructor 3**

*degreetype = sd*

*thesisTopic = tt*

**setDegree(sd)**

*degreetype = sd*

**setTopic(tt)**

*thesisTopic = tt*

**getDegree**

*return degreetype*

**getThesis**

*return thesisTopic*