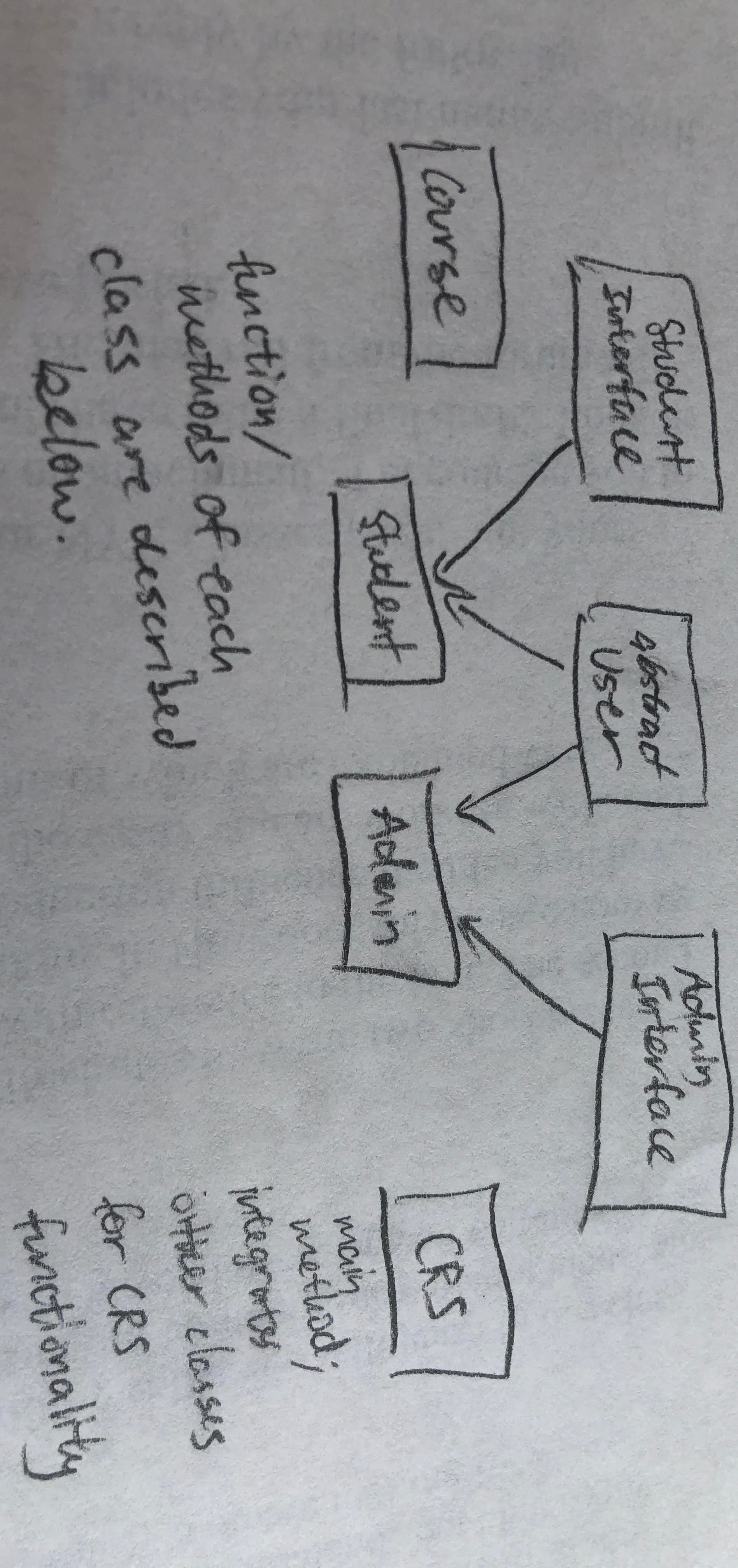
Thomas Huang CS102 Data Structures HW #1

Design Summary of Course Registration System



As shown in my crude diagram above, my Course Registration System consists of a Course class, an Admin class and a Student class that each extend an abstract User class, a StudentInterface and AdminInterface that are implemented by Student and Admin respectively, and a CRS class that includes the main method. Due to the need to frequently use system input and output, most methods revolve around printing statements and reading user input through a scanner within the method body instead of taking method parameters.

**Course**: instance variables stores all the necessary course info (enrollment size, maximum capacity, section, course name, id, instructor, location, and an ArrayList of enrolled Students). Apart from getters and setters, it includes a boolean isFull() method and can also add/remove/sort/list students. Courses are serializable and comparable as I specified.

**User**: includes names and login info, as well as an abstract method for viewing a student’s registered courses and a method for viewing all courses. This class is useful as a superclass because I can declare a User early in my main method and specify its instantiation as an Admin or Student later through my login method.

**Admin**: instance variables store names and login info and implement the methods of AdminInterface, which reflect the project requirements for Admin. Admins are also serializable; my CRS allows the creation and use of more than one Admin, though the default is username Admin and password Admin001.

**Student**: Same general reasoning as Admin class, though Students are Comparable (compareTo method based on student name) so that lists of students will be displayed alphabetically. Also includes an ArrayList of Courses to represent which courses this student is enrolled in.

**CRS**: Contains the main method and various helper methods, such as those for logging in, reading the .csv file of courses, serializing/deserializing relevant objects, and one for prompting the user to press ENTER to continue (helps break up the flow of the program and allows the user to read outputted information before continuing to the next function). First, unless it is the first time the program runs (in which case, there are no students, there is a default admin, and the courses are read in through the .csv file) files containing information for courses, students, and admins are deserialized and the lists are updated. After welcoming the user to the program, they are prompted to login, based on existing login credentials and accounting for a variety of inputs; the login loop continues until a valid user type and login has been determined (Admin/Student), and the rest of the program consists of several while loops that keep the user looped on a certain level of functionality (as determined by the permissions of the user type) until the user decides to return to a more general level of functionality or exit the program altogether. For example, if I login as Admin, I enter one level of functionality: to manage courses, to view reports, to logout, or to exit the program. If I choose to logout, I return to the login loop until a new valid login is made. If I choose to manage courses, I move to a while loop that allows me to execute any of the Admin course management commands (by calling the corresponding methods in the Admin class, which may each contain their own while loops for making sure input and modifications are valid) until I either decide to exit the program or return to the previous level of functionality (course management, viewing reports, logging out, exiting). If the user ever selects to exit the program (not forcibly terminating it), all loops are broken, but the ArrayLists of students, admins, and courses are appropriately serialized to their respective .ser files before the program terminates. When the CRS main method is run again, those serialized objects are read back in, so that the data modified in the prior session carry over to the current session, and so on.

**KEY CONCEPTS**

Method overloading: I used several examples of method overloading. For example, in my Admin class, I have a public displayCourseInfo() method that is called in the main method of CRS and selects a course based on user info, then calls the private overloaded version displayCourseInfo(Course c) that takes the course as parameter and then displays its information.

Method overriding (give at least two examples): Both Student and Course classes override the Comparable interface’s compareTo() method, based on last name and class size, respectively. My Admin class also overrides its superclass User’s viewCourses() method, because the Admin’s version need to display additional information as well.

Abstract Class: My User class is abstract, so that it can’t be instantiated but can be generally declared and specifically instantiated as a Student or Admin as determined by the login process. It also contains methods common to Student and Admin that must be implemented by them.

Inheritance: Both Student and Admin inherit from their abstract superclass User.

Polymorphism: User declaration and its polymorphism, its ability to vary in specific instantiation through login has been described above.

Encapsulation: Each class has its private instance variables, and some of its public methods invoke private methods, all to help organize my program, but also to keep the complexity of each part of the program within its class. My CRS class can access all the necessary functions of each other class without directly accessing its internal information or dealing with the nitty-gritty.

The concept of ADT (Abstract Data Types): the User class is abstract, and a couple of its methods do not specify implementation, so they must be implemented in subclasses.