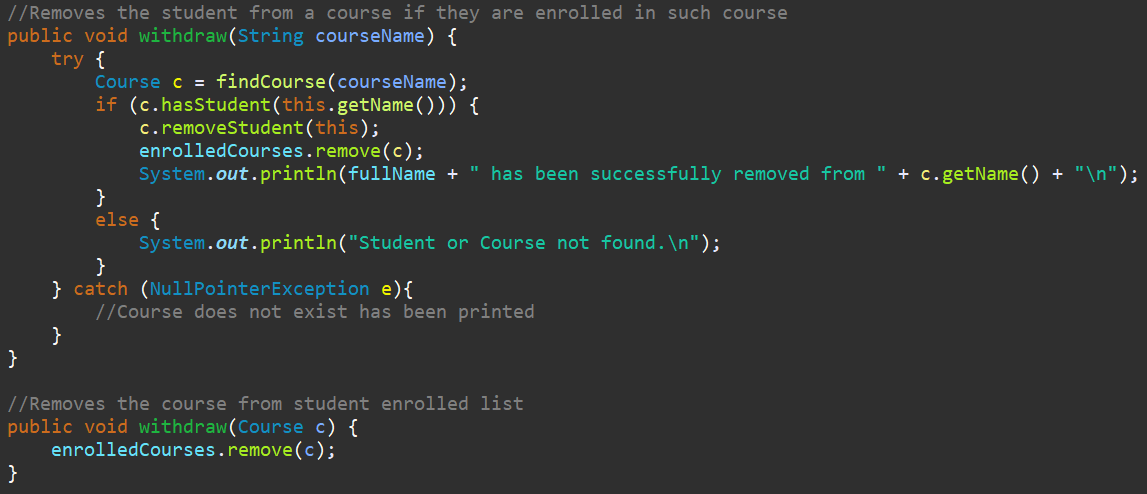
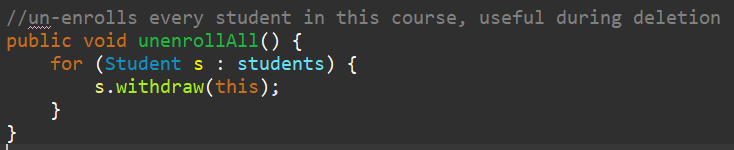
Method Overloading

The most common example of method overloading in my program is object to object methods and user input to object methods.

Example:

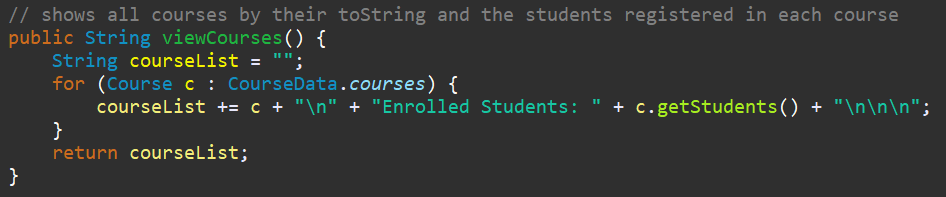
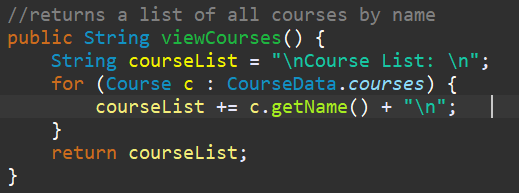




The first withdraw() function is called directly from main after given string input, however this conversion is unnecessary if called from another function which already finds or has access to the course object it needs. UnenrollAll() is called by the Admin upon deletion of a course, which obviously already has a reference to the course it needs.

Method Overriding

Example:



The viewCourses() method changes depending on whether it is called from User or Admin. The user can only see the names of each course. Since the admin has higher rights so to speak it gets to see all the information of the specified course in addition to each of the students registered in that course. No other method overriding was necessary for this project.

Abstract Class

The class user is never actually needed because you must always log-in as a Student or an Admin, both of which are child objects. Because User doesn’t exist in this context (never needs to be instantiated) it is therefore abstract and simply acts as a go between for its child classes.

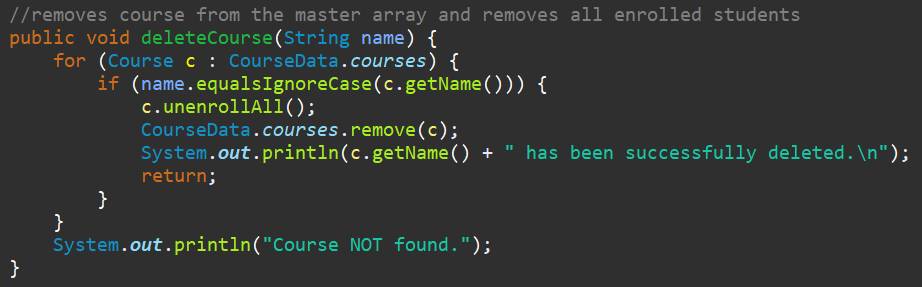
Inheritance

As stated previously, both Student and Admin extend User, which is useful because Students and Admins are users and share common data that is inherited. Both classes need to be able to close the program, view courses, and contain first, last, full name, username, and password.

Polymorphism

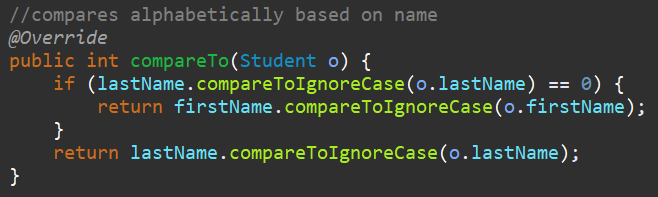
This concept is taken advantage of through the inheritance and method overriding previously mentioned.

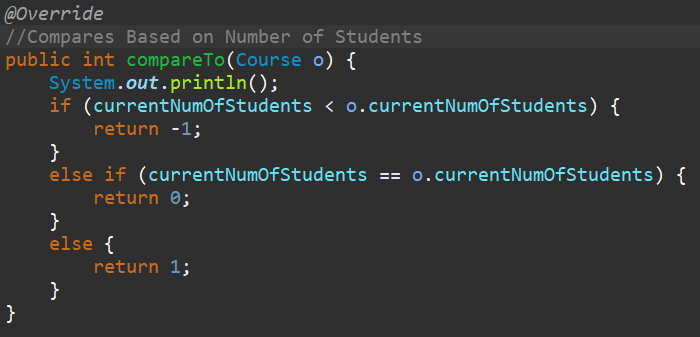
Encapsulation



Encapsulation is utilized by all classes with the exception of the CourseData class which contains only static types as it’s data needs to be accessed by all classes in the program. One example of encapsulation is the Admin needs to know virtually everything about each Course and be able to edit it all, however, this is done exclusively through the methods provided by the Course class because it provides an added layer of security by keeping things simple and keeping Courses independant of the Admin. It also makes things easier to keep track of ensures data is always edited properly. The example above uses CourseData directly, but interacts with Course objects view their methods.

Abstract Data Types





The Interface comparable was implemented by both student and course because it defines a method of comparison which can be used by Collection.sort but it does not define the actual implementation. This is extremely useful because Collection.sort knows how to use comparable without me explicitly defining yet and allowing me to define it how I like for each use case.