Allison Majors

* Method overloading
  + In my CourseDirectory class, I have 2 methods called deleteCourse(). The first one is deleteCourse(String name, String Id, int sectnum), and the second one is deleteCourse(String name, int sectnum). This method is overloaded because it exists twice, but has different parameters.
  + I also used method overloading with the constructors in the Student class. We have a constructor called Student() as well as another being Student(String firstName, String lastName) and another being Student(String firstName, String lastName, String username, String password). These constructors all have the same name, but they have different parameters so that the compiler knows which method to call based on the parameters we give it.
* Method overriding (at least two examples)
  + In the class User, there is a method called viewCourses(). The classes Student and Admin both inherit this method, but they override it in their own classes and provide their own implementation. The viewCourses() method in Admin calls the directory’s viewAllCourses method, but the viewCourses() method in Student calls the directory’s viewCoursesForStudent() method.
  + In the class User, there is a method called printMyInfo() that prints the user’s name. The classes Student and Admin both inherit this method, however they both override it in their own class. Specifically, Admin has the same method printMyInfo(), but it prints “I am an admin”. The Student class also has printMyInfo(), but it prints “I am a student”.
* Abstract Class
  + The class User is an abstract class. This means that it can’t be instantiated. Student and Admin both inherit from User and can make use of the User class through their own constructors. Student and Admin automatically inherit User’s variables and methods.
* Inheritance
  + Both Student and Admin inherit from a class called User. The user class contains protected variables which both a student and an admin inherit. The student and admin also can use any of the methods included in the user class. This is all thanks to the super() keyword and to Admin and Student using their constructors to access the User class.
* Polymorphism
  + Every admin is a user, every student is a user, but not every user is a student and not every user is an admin.
  + The term “polymorphism” means many forms and exists within the context of inheritance. In my program, I have a class called User and then a class Admin that extends from User and a class Student that extends from User. Both an Admin and a Student have their own implementation of the method printMyInfo(), but if we wanted to, we could use the keyword super to call the User’s printMyInfo(). In this case, we are executing the same action in various ways. I can use declared type and actual type to access different methods.
* Encapsulation
  + Encapsulation hides the complexity from the user. In my program, the class Course has all private variables and these variables are accessed in other classes through getters and setters. A course interacts with other classes through its constructors and its public methods. The code is more simple and is easier to view.
* The concept of ADT (Abstract Data Types)
  + An abstract data type hides the complexity from the user. In this program, I use a lot of Arraylists, like coursesList, which are abstract data types. I use methods like .add() and .remove() without having to worry about the implementation. I also use a lot of variables that are of type String. The String type is immutable.