Method Overloading:

I used method overloading on my inherited view_courses method that I put into my User class. First, I had to override it because I declared it as an abstract method in the User class. Then, I changed the method signature of method inside my Admin and Student classes to allow for different functionality. Admin was to allow for viewing of courses for a specific student, Student was to allow for the student to view their registered courses.

Code Example:

```
public void view_courses(ArrayList<Course> courses, String first_name, String last_name) # Overrode the view_courses method
```

Method Overriding:

I used method overriding on my abstract method, view_courses that I created in my User class (as shown before, I also performed overloading on it). I also used method overriding on my register() method, which I also declared as an abstract method in my User_class. Overriding the method allowed me to give it functionality specific to the user (admin or student).

Code Examples:

```
public void view_courses(ArrayList<Course> courses, boolean Full) { // Overrides the abstract method 'view_courses' in the User class to either display ALL COURSES to the admin or ALL COURSES that are full (Overriding)
public void register(ArrayList<Course> courses) { // Overrides the register method in the User class for the Student; student needs class name, section, and first and last name (Overriding)
```

Abstract Class:

I declared my User class as an abstract class. This allowed me to specify abstract methods in order to make them more flexible for implementing differing functionality (depending on the type of User). The view_course and register methods were abstract in the abstract_class.

Code Example:

```
public abstract class User implements Serializable { // Abstract User class that
implements serializable; our admin and student class will extend this abstract user
class
public abstract void register(ArrayList<Course> courses); // abstract method to
register a student (either into the system or into a course); implementation will
differ depending on whether admin or student is calling the method

    public abstract void view_courses(ArrayList<Course> courses, boolean full); //
displays information about selective courses to the user based on certain criteria
```

Inheritance and Polymorphism:

I used inheritance to inherit the public instance fields and methods from the User class into my Admin and Student class. This allowed me to utilize the methods with the same method signature in multiple ways, or saved time so that I didn't have to redeclare my methods separate in the Admin or Student class.

Code Example:

```
public class Admin extends User implements AdminInterface
public class Student extends User implements StudentInterface { // Student class that
implements the Student and serializable interface and extends the User Class
```

Encapsulation:

I used encapsulation by keeping certain instance variables private from the user. For example, I made certain information, such as the individual username and password for each student private. This ensures that there are no data leaks, and is a more realistic way of creating a course registration system. I used getter and setter methods instead to create a more robust approach of changing attributes of classes, hiding information from the user (even the admin itself).

Code Example:

```java
public String get_course_name() { // Returns the course name
        return name;
    }

    public String get_course_id() { // Returns the course id
        return id;
    }

    public int get_max_student() { // Returns the max number of students
        return max_num;
    }

    public int get_current_student() { // Returns the current number of students
        return current_num;
    }

    public ArrayList<String> get_Student_ArrayList() {
        return student_list;
    }
```

 # A sample of getter methods for the Course class. This makes sure that the admin or student cannot directly access or modify the instance fields, which would break down the system.

```java
public String get_course_info(String user_Type) { // Returns information for the
course; useful for various tasks of the admin and student
        String information = "";
        if (user_Type.equals("Admin")) { // The information that the admin sees
differs from the information the student sees
        information = "Course Name: " + name + ", ";
        information += "Course ID: " + id + ", ";
        information += "Maximum Number of Students: " + max_num + ", ";
        information += "Current Number of Students: " + current_num + ", ";
        information += "List of Students: " + get_student_names() + ", ";
        information += "Instructor: " + instructor + ", ";
        information += "Section Number: " + section_number + ", ";
        information += "Location: " + location;
        }
        else { // The information that the student sees differs from the information
that the admin sees
        information = "Course Name: " + name + ", ";
        information += "Course ID: " + id + ", ";
        information += "Instructor: " + instructor + ", ";
        information += "Section Number: " + section_number + ", ";
        information += "Location: " + location;
        }
        return information;
    }
```

 # Another example is this method, which hides the implementation of retrieving the information for a requested course from the admin or student user.

The Concept of ADT (Abstract Data Types):

I used interfaces, which are considered abstract data types, to specify the method signatures of the functions to be implemented for the admin and student. This allowed me to define a set of operations that the Admin and Student classes had to provide, without specifically specifying how they are performed. This is the general purpose of ADTs.

I also used ArrayLists, a concrete version of the List ADT, to store Students and Courses. This allowed for easier modifications and serialization.

Code Example:

```
public interface AdminInterface { // Define the interface for the admin here.
    // Each method signature below corresponds to a possible action the admin can
take.
public interface StudentInterface { // Student Interface for the Student class. All
methods in the interface are implemented in the Student class.
ArrayList<Course> courses = null; // Creates the ArrayList to store courses with their
respective information
        ArrayList<Student> student_Body = new ArrayList<>(); // The student body
ArrayList will store all registered students, regardless of the courses they are
taking
```