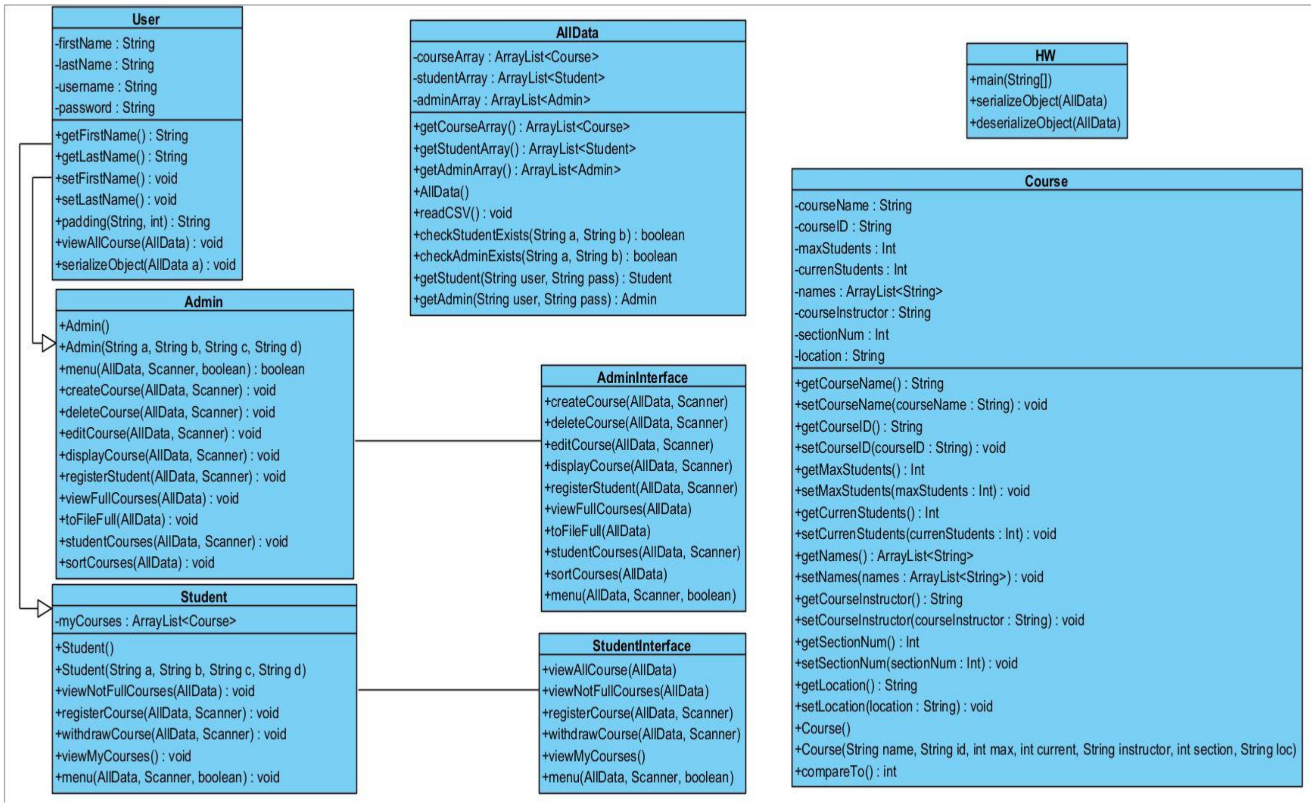
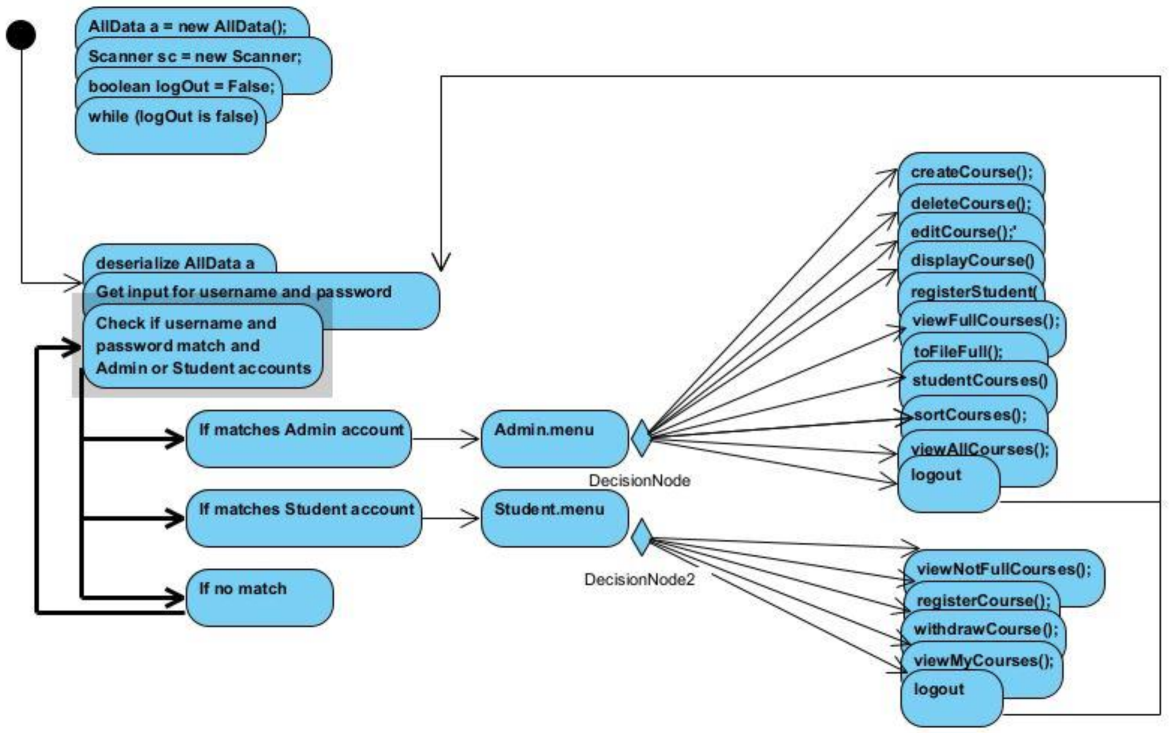
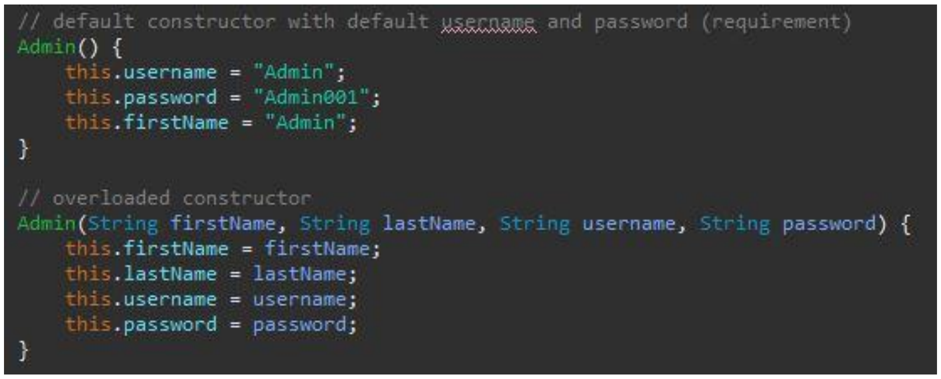

User
-firstName : String
-lastName : String
-username : String
-password : String
+getFirstName() : String
+getLastName() : String
+setFirstName() : void
+setLastName() : void
+padding(String, int) : String
+viewAllCourse(AllData) : void
+serializeObject(AllData a) : void
AllData
-courseArray : ArrayList<Course>
-studentArray : ArrayList<Student>
-adminArray : ArrayList<Admin>
+getCourseArray() : ArrayList<Course>
+getStudentArray() : ArrayList<Student>
+getAdminArray() : ArrayList<Admin>
+AllData()
+readCSV() : void
+checkStudentExists(String a, String b) : boolean
+checkAdminExists(String a, String b) : boolean
+getStudent(String user, String pass) : Student
+getAdmin(String user, String pass) : Admin
HW
+main(String[])
+serializeObject(AllData)
+deserializeObject(AllData)
Course
-courseName : String
-courseID : String
-maxStudents : Int
-currenStudents : Int
-names : ArrayList<String>
-courseInstructor : String
-sectionNum : Int
-location : String
+getCourseName() : String
+setCourseName(courseName : String) : void
+getCourseID() : String
+setCourseID(courseID : String) : void
+getMaxStudents() : Int
+setMaxStudents(maxStudents : Int) : void
+getCurrenStudents() : Int
+setCurrenStudents(currenStudents : Int) : void
+getNames() : ArrayList<String>
+setNames(names : ArrayList<String>) : void
+getCourseInstructor() : String
+setCourseInstructor(courseInstructor : String) : void
+getSectionNum() : Int
+setSectionNum(sectionNum : Int) : void
+getLocation() : String
+setLocation(location : String) : void
+Course()
+Course(String name, String id, int max, int current, String instructor, int section, String loc)
+compareTo() : int
Admin
+Admin()
+Admin(String a, String b, String c, String d)
+menu(AllData, Scanner, boolean) : boolean
+createCourse(AllData, Scanner) : void
+deleteCourse(AllData, Scanner) : void
+editCourse(AllData, Scanner) : void
+displayCourse(AllData, Scanner) : void
+registerStudent(AllData, Scanner) : void
+viewFullCourses(AllData) : void
+toFileFull(AllData) : void
+studentCourses(AllData, Scanner) : void
+sortCourses(AllData) : void
AdminInterface
+createCourse(AllData, Scanner)
+deleteCourse(AllData, Scanner)
+editCourse(AllData, Scanner)
+displayCourse(AllData, Scanner)
+registerStudent(AllData, Scanner)
+viewFullCourses(AllData)
+toFileFull(AllData)
+studentCourses(AllData, Scanner)
+sortCourses(AllData)
+menu(AllData, Scanner, boolean)
Student
-myCourses : ArrayList<Course>
+Student()
+Student(String a, String b, String c, String d)
+viewNotFullCourses(AllData) : void
+registerCourse(AllData, Scanner) : void
+withdrawCourse(AllData, Scanner) : void
+viewMyCourses() : void
+menu(AllData, Scanner, boolean) : void
StudentInterface
+viewAllCourse(AllData)
+viewNotFullCourses(AllData)
+registerCourse(AllData, Scanner)
+withdrawCourse(AllData, Scanner)
+viewMyCourses()
+menu(AllData, Scanner, boolean)


AllData a = new AllData();
Scanner sc = new Scanner;
boolean logOut = False;
while (logOut is false)
deserialize AllData a
Get input for username and password
Check if username and password match and Admin or Student accounts
If matches Admin account
If matches Student account
If no match
Admin.menu
Student.menu
DecisionNode
DecisionNode2
createCourse();
deleteCourse();
editCourse();'
displayCourse()
registerStudent(
viewFullCourses();
toFileFull();
studentCourses()
sortCourses();
viewAllCourses();
logout
viewNotFullCourses();
registerCourse();
withdrawCourse();
viewMyCourses();
logout

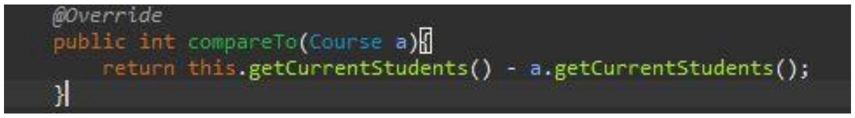
In a word document, you will need to briefly explain how you used these concepts into your program. You will need to provide examples from you code. **Please be as brief as possible.**

- Method Overloading: This concept was used in the constructors of Student class and Admin class. Specifically, this allowed a default Admin to be created upon the first run of the program with the default username and password. Later on in the admin menu, there is an option to create a new Admin account that would utilize the overloaded constructor of Admin class.



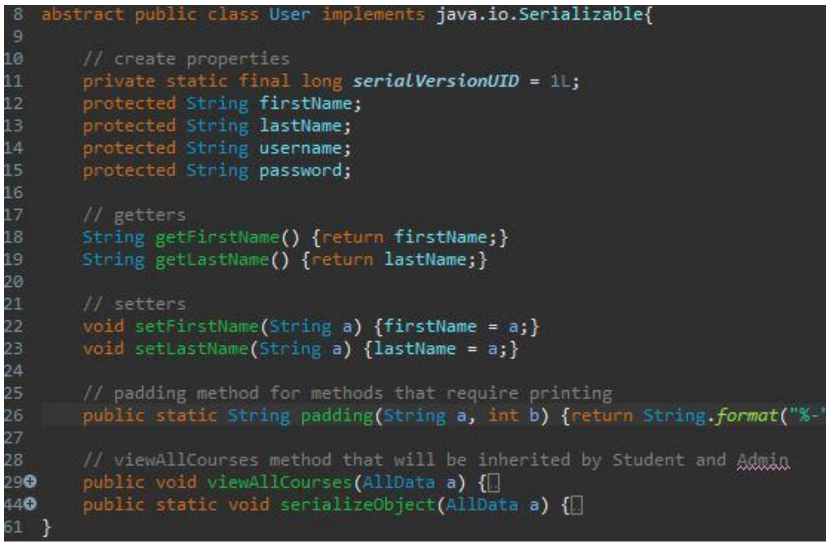
```
// default constructor with default username and password (requirement)
Admin() {
    this.username = "Admin";
    this.password = "Admin001";
    this.firstName = "Admin";
}

// overloaded constructor
Admin(String firstName, String lastName, String username, String password) {
    this.firstName = firstName;
    this.lastName = lastName;
    this.username = username;
    this.password = password;
}
```

- Method Overriding (two examples): In order to sort the courseArray of my program, I overrode the compareTo() method of the Course class in order to care a specific property of an object. This allowed the program to sort the courses in ascending order according to the number of current students enrolled.



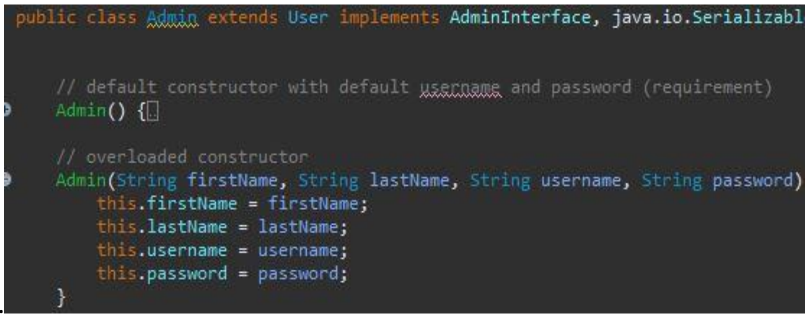
```
    @Override
    public int compareTo(Course a){
        return this.getCurrentStudents() - a.getCurrentStudents();
    }
```

- Abstract Class: The User class is an abstract class in my progam. It was designed to allow common functionality and some shared properties of its subclasses (Admin and Student).



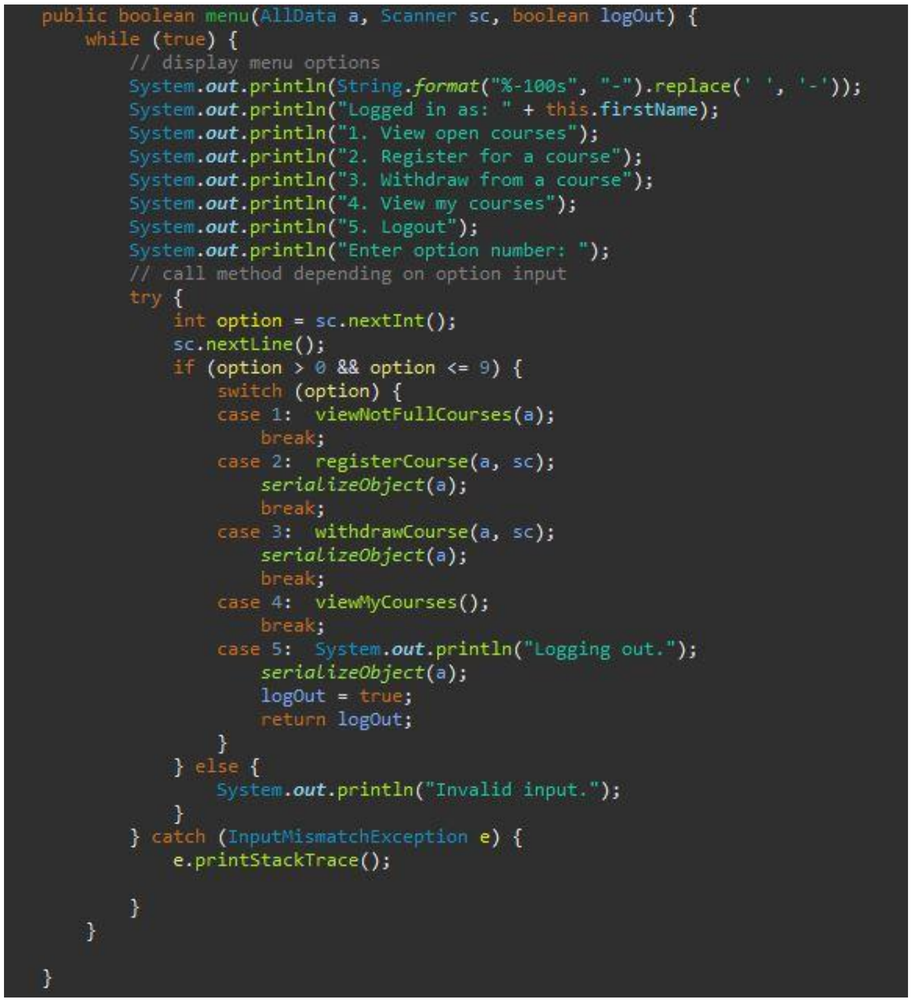
```
abstract public class User implements java.io.Serializable{

    // create properties
    private static final long serialVersionUID = 1L;
    protected String firstName;
    protected String lastName;
    protected String username;
    protected String password;

    // getters
    String getFirstName() {return firstName;}
    String getLastName() {return lastName;}

    // setters
    void setFirstName(String a) {firstName = a;}
    void setLastName(String a) {lastName = a;}

    // padding method for methods that require printing
    public static String padding(String a, int b) {return String.format("%-

    // viewAllCourses method that will be inherited by Student and Admin
    public void viewAllCourses(AllData a) {
    public static void serializeObject(AllData a) {
}
```

- Inheritance: This main concept of OOP was used with abstract class User and its subclasses. Admin and Student extends User, therefore inheriting all of User's methods and attributes. Here, Admin inherited the fields of User.



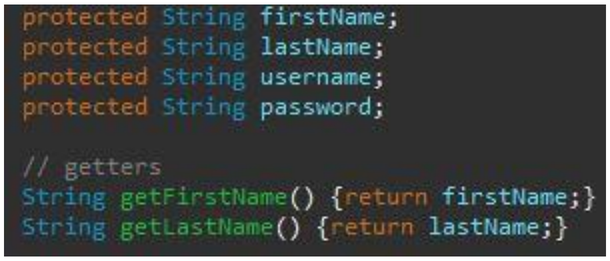
```
public class Admin extends User implements AdminInterface, java.io.Serializabl

    // default constructor with default username and password (requirement)
    Admin() {

    // overloaded constructor
    Admin(String firstName, String lastName, String username, String password)
        this.firstName = firstName;
        this.lastName = lastName;
        this.username = username;
        this.password = password;
    }
```

- Polymorphism: This concept was used to conduct inherited methods from User in different ways according to the child class type. For example, the User class contains a menu() method that only has the logout option. The Admin class also inherits the menu() method however implements it differently.

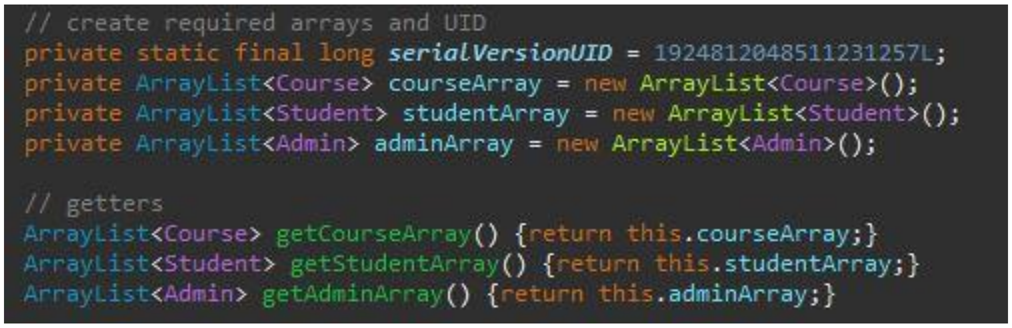


```
    public boolean menu(AllData a, Scanner sc, boolean logOut) {
        while (true) {
            // display menu options
            System.out.println(String.format("%-100s", "-").replace(' ', '-'));
            System.out.println("Logged in as: " + this.firstName);
            System.out.println("1. View open courses");
            System.out.println("2. Register for a course");
            System.out.println("3. Withdraw from a course");
            System.out.println("4. View my courses");
            System.out.println("5. Logout");
            System.out.println("Enter option number: ");
            // call method depending on option input
            try {
                int option = sc.nextInt();
                sc.nextLine();
                if (option > 0 && option <= 9) {
                    switch (option) {
                    case 1:  viewNotFullCourses(a);
                        break;
                    case 2:  registerCourse(a, sc);
                        serializeObject(a);
                        break;
                    case 3:  withdrawCourse(a, sc);
                        serializeObject(a);
                        break;
                    case 4:  viewMyCourses();
                        break;
                    case 5:  System.out.println("Logging out.");
                        serializeObject(a);
                        logOut = true;
                        return logOut;
                    }
                } else {
                    System.out.println("Invalid input.");
                }
            } catch (InputMismatchException e) {
                e.printStackTrace();

            }
        }

    }
```

- Encapsulation: Was used when dealing with variables/pointers that held important or sensitive information. In my program, encapsulation was used when appropriate to hide data fields that can only be accessed through getters.


```
protected String firstName;
protected String lastName;
protected String username;
protected String password;

// getters
String getFirstName() {return firstName;}
String getLastName() {return lastName;}
```

- Concept of ADT (Abstract Data Type): In the program, StudentInterface and AdminInterface are used to semi-defines an ADT. In addition, there are multiple ArrayLists of type Student, Admin and Course in the program.


```
// create required arrays and UID
private static final long serialVersionUID = 1924812048511231257L;
private ArrayList<Course> courseArray = new ArrayList<Course>();
private ArrayList<Student> studentArray = new ArrayList<Student>();
private ArrayList<Admin> adminArray = new ArrayList<Admin>();

// getters
ArrayList<Course> getCourseArray() {return this.courseArray;}
ArrayList<Student> getStudentArray() {return this.studentArray;}
ArrayList<Admin> getAdminArray() {return this.adminArray;}
```