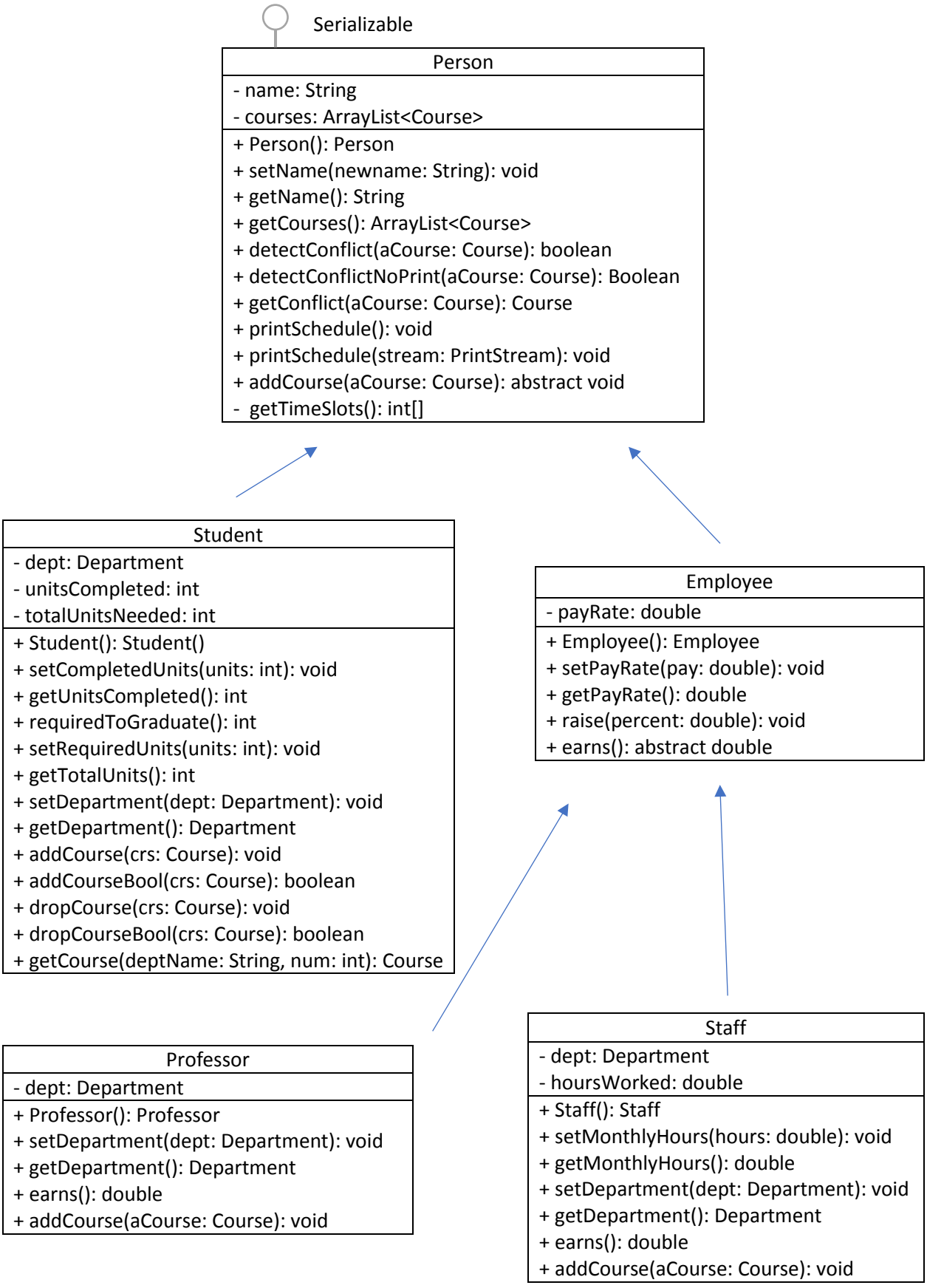
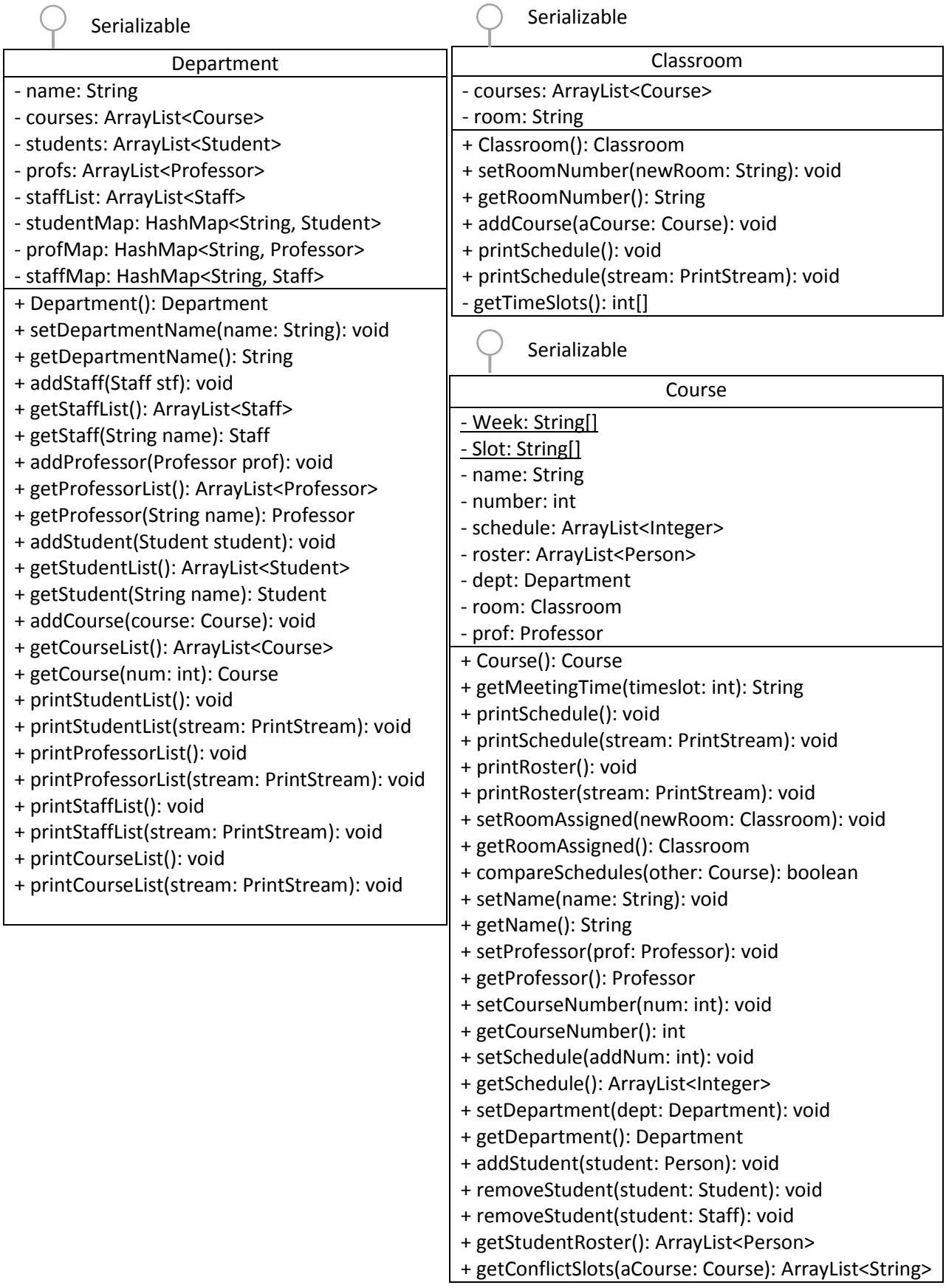
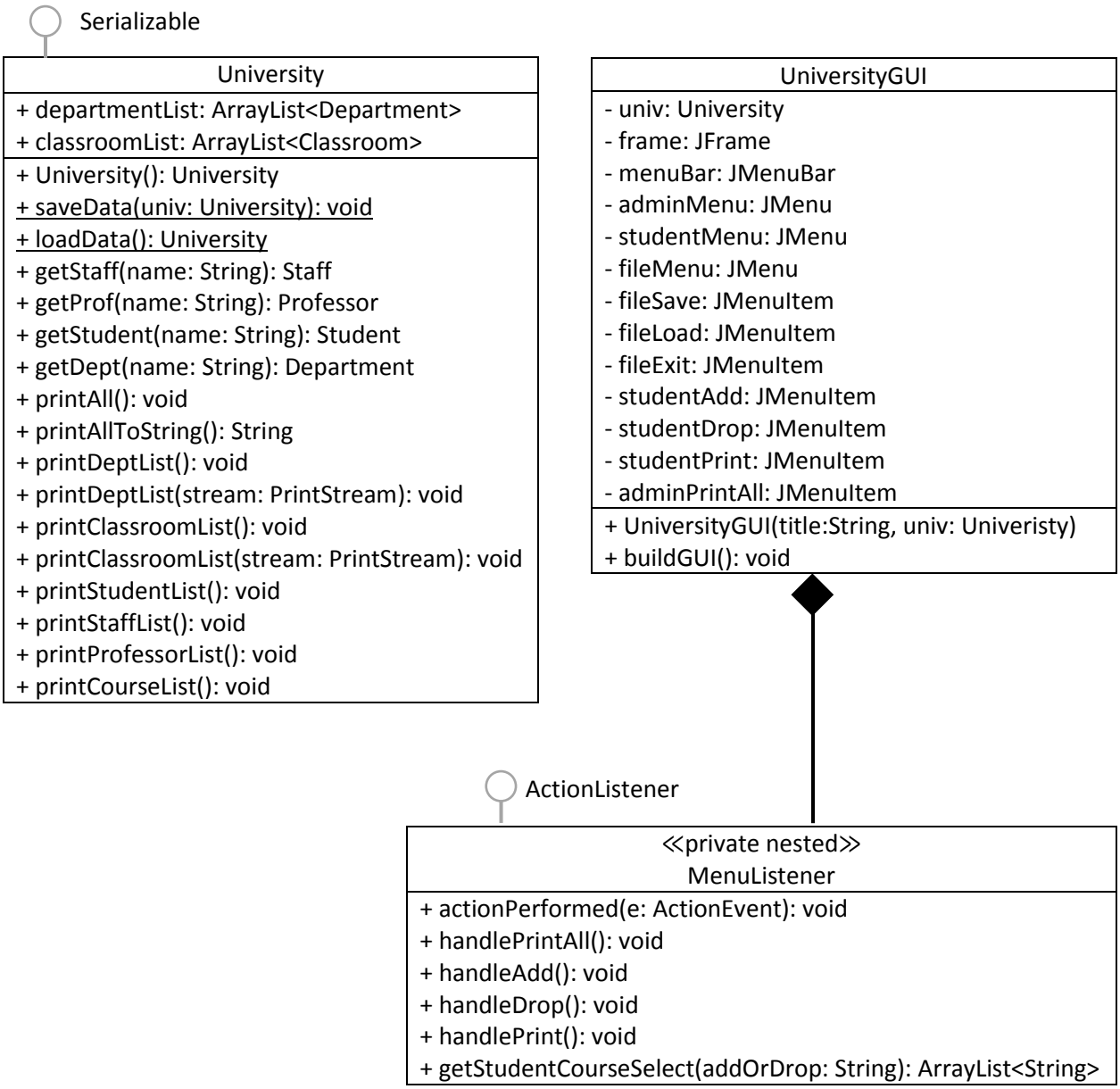

Serializable
Person
- name: String
- courses: ArrayList<Course>
+ Person(): Person
+ setName(newname: String): void
+ getName(): String
+ getCourses(): ArrayList<Course>
+ detectConflict(aCourse: Course): boolean
+ detectConflictNoPrint(aCourse: Course): Boolean
+ getConflict(aCourse: Course): Course
+ printSchedule(): void
+ printSchedule(stream: PrintStream): void
+ addCourse(aCourse: Course): abstract void
- getTimeSlots(): int[]
Student
- dept: Department
- unitsCompleted: int
- totalUnitsNeeded: int
+ Student(): Student()
+ setCompletedUnits(units: int): void
+ getUnitsCompleted(): int
+ requiredToGraduate(): int
+ setRequiredUnits(units: int): void
+ getTotalUnits(): int
+ setDepartment(dept: Department): void
+ getDepartment(): Department
+ addCourse(crs: Course): void
+ addCourseBool(crs: Course): boolean
+ dropCourse(crs: Course): void
+ dropCourseBool(crs: Course): boolean
+ getCourse(deptName: String, num: int): Course
Employee
- payRate: double
+ Employee(): Employee
+ setPayRate(pay: double): void
+ getPayRate(): double
+ raise(percent: double): void
+ earns(): abstract double
Professor
- dept: Department
+ Professor(): Professor
+ setDepartment(dept: Department): void
+ getDepartment(): Department
+ earns(): double
+ addCourse(aCourse: Course): void
Staff
- dept: Department
- hoursWorked: double
+ Staff(): Staff
+ setMonthlyHours(hours: double): void
+ getMonthlyHours(): double
+ setDepartment(dept: Department): void
+ getDepartment(): Department
+ earns(): double
+ addCourse(aCourse: Course): void


Serializable
Department
- name: String
- courses: ArrayList<Course>
- students: ArrayList<Student>
- profs: ArrayList<Professor>
- staffList: ArrayList<Staff>
- studentMap: HashMap<String, Student>
- profMap: HashMap<String, Professor>
- staffMap: HashMap<String, Staff>
+ Department(): Department
+ setDepartmentName(name: String): void
+ getDepartmentName(): String
+ addStaff(Staff stf): void
+ getStaffList(): ArrayList<Staff>
+ getStaff(String name): Staff
+ addProfessor(Professor prof): void
+ getProfessorList(): ArrayList<Professor>
+ getProfessor(String name): Professor
+ addStudent(Student student): void
+ getStudentList(): ArrayList<Student>
+ getStudent(String name): Student
+ addCourse(course: Course): void
+ getCourseList(): ArrayList<Course>
+ getCourse(num: int): Course
+ printStudentList(): void
+ printStudentList(stream: PrintStream): void
+ printProfessorList(): void
+ printProfessorList(stream: PrintStream): void
+ printStaffList(): void
+ printStaffList(stream: PrintStream): void
+ printCourseList(): void
+ printCourseList(stream: PrintStream): void
Serializable
Classroom
- courses: ArrayList<Course>
- room: String
+ Classroom(): Classroom
+ setRoomNumber(newRoom: String): void
+ getRoomNumber(): String
+ addCourse(aCourse: Course): void
+ printSchedule(): void
+ printSchedule(stream: PrintStream): void
- getTimeSlots(): int[]
Serializable
Course
- Week: String[]
- Slot: String[]
- name: String
- number: int
- schedule: ArrayList<Integer>
- roster: ArrayList<Person>
- dept: Department
- room: Classroom
- prof: Professor
+ Course(): Course
+ getMeetingTime(timeslot: int): String
+ printSchedule(): void
+ printSchedule(stream: PrintStream): void
+ printRoster(): void
+ printRoster(stream: PrintStream): void
+ setRoomAssigned(newRoom: Classroom): void
+ getRoomAssigned(): Classroom
+ compareSchedules(other: Course): boolean
+ setName(name: String): void
+ getName(): String
+ setProfessor(prof: Professor): void
+ getProfessor(): Professor
+ setCourseNumber(num: int): void
+ getCourseNumber(): int
+ setSchedule(addNum: int): void
+ getSchedule(): ArrayList<Integer>
+ setDepartment(dept: Department): void
+ getDepartment(): Department
+ addStudent(student: Person): void
+ removeStudent(student: Student): void
+ removeStudent(student: Staff): void
+ getStudentRoster(): ArrayList<Person>
+ getConflictSlots(aCourse: Course): ArrayList<String>


Serializable
University
+ departmentList: ArrayList<Department>
+ classroomList: ArrayList<Classroom>
+ University(): University
+ saveData(univ: University): void
+ loadData(): University
+ getStaff(name: String): Staff
+ getProf(name: String): Professor
+ getStudent(name: String): Student
+ getDept(name: String): Department
+ printAll(): void
+ printAllToString(): String
+ printDeptList(): void
+ printDeptList(stream: PrintStream): void
+ printClassroomList(): void
+ printClassroomList(stream: PrintStream): void
+ printStudentList(): void
+ printStaffList(): void
+ printProfessorList(): void
+ printCourseList(): void
UniversityGUI
- univ: University
- frame: JFrame
- menuBar: JMenuBar
- adminMenu: JMenu
- studentMenu: JMenu
- fileMenu: JMenu
- fileSave: JMenuItem
- fileLoad: JMenuItem
- fileExit: JMenuItem
- studentAdd: JMenuItem
- studentDrop: JMenuItem
- studentPrint: JMenuItem
- adminPrintAll: JMenuItem
+ UniversityGUI(title:String, univ: Univeristy)
+ buildGUI(): void
ActionListener
≪private nested≫
MenuListener
+ actionPerformed(e: ActionEvent): void
+ handlePrintAll(): void
+ handleAdd(): void
+ handleDrop(): void
+ handlePrint(): void
+ getStudentCourseSelect(addOrDrop: String): ArrayList<String>

ReadMe

The following steps were taken to implement this program:

1. Task One
    a. Change classes as necessary to implement the Serializable interface to allow for saving and loading of Universities.
    b. Create and implement the saveData and loadData methods in University, using existing lab 3 code as a reference.
    c. Create the printAll() method in University, using the various print methods in other classes to reduce redundancy in code.
2. Task Two
    a. Create the UniversityGUI class, with University and various swing objects as private attributes. The constructor initializes all of the attributes, and the buildGUI function creates the GUI using these attributes.
    b. Create a private nested MenuListener class inside of UniversityGUI that implements ActionListener to control menu actions. The ActionPerformed method handles saving, loading, exiting the GUI, and printing all data. Student adds, drops, and prints are handled by separate specialized methods within this class.
    c. Create printAllToString method in University, which prints to a local PrintStream object combined with a ByteArrayOutputStream instead of System.out. It then creates a String out of the ByteArrayOutputStream to capture the data printed by the University, which is identical to the data printed by the printAll() method, minus the pay information. This required creating overloads of several print methods inside of other classes to print to the PrintStream instead of System.out.
    d. Create the handlers for adding and dropping Courses to/from Students' schedules.
        i. The add and drop methods have near identical implementation in the student/course selection stage, so a method was created to capture that stage of the process, and is called by both handlers. This method returns the student's name and the course information in a String ArrayList.
        ii. To use this information to fetch students and courses, several getters were created in the University and Department classes. In addition, to improve search times, HashMaps from String names to Students, Staff, Professors, and Courses were added to the Department class, making retrieval of a Person or Course an O(n) operation, where n is the number of Departments in the University.
        iii. The handlers use the new getters to retrieve the Student by name and the course by Department name and Course number. If any of these do not exist, the getters return null, which is error checked by the handlers to create error messages as necessary. Finally, once all information is retrieved correctly, the Course is added to or dropped from the Student's schedule, using a modified version of the existing addCourse and dropCourse methods that do not print errors to System.out.
        iv. Adding a course has the additional check to see if the new course creates any conflicts. This required a modified version of the detectConflicts method to be created in the Person class that did not print error messages to System.out, and

a new method that returned the conflicting Course in the Person's schedule in order to display information about which Course created the conflict.

e. Create the handler to print Student schedules.
    i. The handler only required an overloaded printSchedule method that prints to a PrintStream to be created in the Person class. The handler first gets the user to select a Student, then checks to see if the student exists, and finally uses the newly created method in Person to print the student's schedule to a message dialogue.