CSE 403, Winter 2007
Final Exam
Due: Thursday, March 15, 2007, 12:00pm (noon)

Student Name: _____

Signature: _____

Roughly how much time did you spend on this exam? _____ *hours*

- This take-home exam is open-book and open-notes.
- You must work alone on this exam and should not use a computer.
  You are strictly forbidden from discussing your answers with another student.
- Write legibly and case-sensitively. Do not abbreviate Java code. You don't need to write `import`s. You do not need to comment your code, except if the question says to do so.
- Several questions require written answers. Please explain yourself as clearly and concisely as you can. Long answers are not encouraged; try to find the fewest words necessary to fully explain your answer.
- This exam will not be accepted after the listed due date above. Late exams will receive zero points.

Good luck!

Score summary: (for grader use only)

Problem	Description	Max	Earned
1	Lifecycle Models	15	
2	Use Cases	20	
3	UI Prototyping	15	
4	Object-Oriented Design	15	
5	UML	20	
6	Unit Testing	15	
TOTAL		**100**	

Background:

For the rest of the exam, assume that you are working as the project manager for the TheirUW project, and that you've been given the following specifications from your university "customers."

Dear Project Team,

We would like your team to build our new online student course registration system, *TheirUW*. We aren't programmers, but we know what we'd like the system to do. Here are your specifications:

Students can log into the system using their net ID and password, and can then view their courses and schedules, enroll in courses, drop courses, see their grades, and pay student fees. The student gets 3 attempts to log in; if the student guesses their password incorrectly 3 times, their account should be locked, and they'll have to see an advisor in person to fix it. A student can only be logged into the system from one place at a time. If a student tries to log in from a second place at the same time, the second login fails. The login is a critical feature; if it doesn't work, the whole system can't be used. We've been told that security is really important for logins, so that nobody can access or change another student's personal information.

Once the student logs in to the system, we want them to see a UI where courses can be added, dropped, and so on. Adding courses is the main reason students will log in to TheirUW. Without this feature the advisors would be swamped with students trying to add courses manually.

UW policy says that students can only carry up to 19 credits per quarter, and can only be enrolled in up to 5 courses at a time. We want it so that the student can view the course enrollment and see how many other students are taking the course. You can store the course data in one of those "Sequel" databases on the university's server. Students cannot add courses with full enrollment unless they have a special "add code" from the instructor. (Some instructors give add codes themselves, and others ask their department advisors to provide them.) Before letting the student into the class, the system should probably check whether they meet the prerequisites to take it. We can provide your team with a complete list of all courses and their prerequisites.

Can you also make the system maintain waiting lists for classes and allow students to put themselves on the waiting list for a full class, so that they will be automatically added if a slot becomes available? That would be a great feature to have. A slot in a course can become available if the enrollment is increased or if a student drops the course. We want to allow students to drop courses using *TheirUW* within the first 10 days; after that, they must see an advisor and receive special permission to drop the course. Not as many students need to drop courses as add them, and students can also drop courses through their academic advisor. Currently the advisors are handling all drop requests manually, and they have told us the load is pretty manageable.

We haven't got the schedule database working yet, but we have a team of freshmen from the I-School working on it. We assume it will be ready about halfway through the quarter. So you should talk to them, because you'll depend on their data for *TheirUW* to view student schedules. For paying student fees, you'll probably have to come up with a secure way to read and validate credit card numbers, or take money out of Husky accounts. That's easy to do with computers, right? Can you please figure that out for us? Currently the students use their telephone to handle fee transactions, but we thought it would be nice to have that in this new online system.

We have read that users don't like web apps that run slowly, so we're hoping you can create a system that will respond to all user actions within 4 seconds. The database has just been upgraded, so we assume that any database query should return in under 2 seconds. Also, we received complaints that the last system, "MyUW," was hard to use. So we'd really like your team to come up with a well-designed user interface that looks nice.

Your team will work on this project throughout the 10-week winter quarter. We'd like to show a rough version of the system to the provosts by halfway through the quarter, and we really need it to be working without any bugs by the end of the quarter so that we can officially unveil it for the students to add their spring courses.

Sincerely,
Marty Stepp
President, University of Washington

1. Software Lifecycle Models

Your customers in the university have asked you to give them a rough schedule that explains what tasks you'll do during what time periods. In addition to their written instructions on the first page, they have sent an additional email that says they'd like you to emulate what is done out in the "real world" at places such as Microsoft and Google.

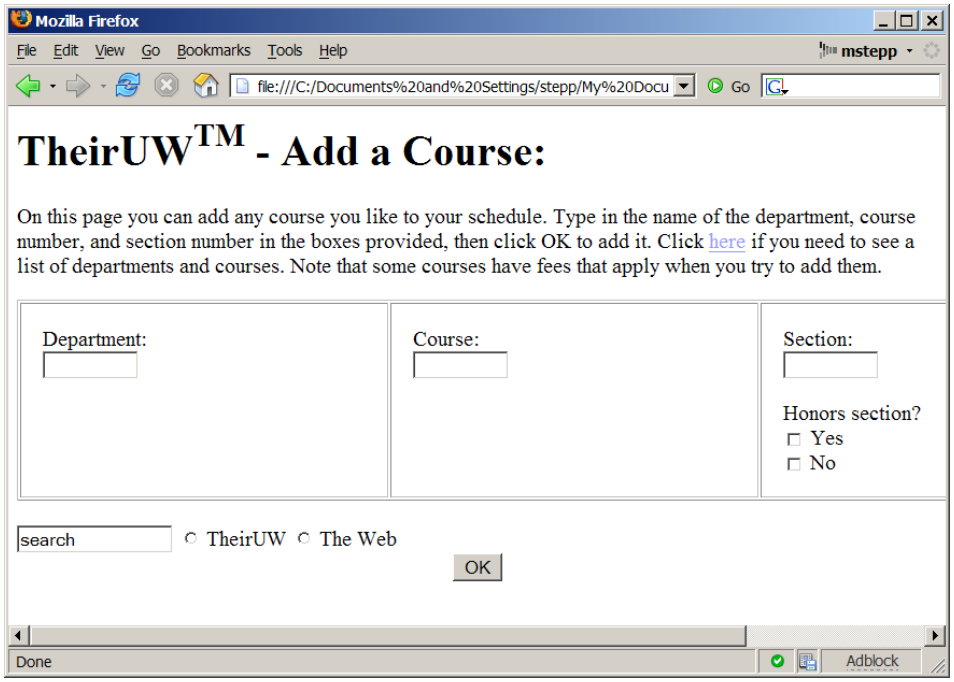
What sort of development lifecycle will you use? What features will you attempt to implement during the ten-week project phase, and in what prioritized order? Briefly explain your decisions.

2. Use Cases

Write a formal use case for the *Add a Course* scenario for the *TheirUW* system to match the preceding specifications. Include all the applicable elements of formal use cases that we have discussed in class, as were included in the use cases of your SRS.

3. UI Prototyping

One of your teammates made a mockup of what the user interface for adding a course might look like:



Describe three specific problems with this UI, and draw your own sketch of a UI that solves these problems.

4. Object-Oriented Design

One of your project partners has been working on a design for the TheirUW system. He has emailed you the following questions:

a) Should a `Student` object store a list of the student's courses, or should a `Course` object store a list of the course's students? Or both, or neither?

b) Right now our `RegistrationSystem` class holds all courses and students, contains all logic for checking prerequisites, and has the methods for adding and dropping a course. It also has the database connection logic right now. Is this okay?

c) Since there are different levels of students, we made an inheritance hierarchy. The superclass is called `Student`, and the subclasses are called `Freshman`, `Sophomore`, `Junior`, and `Senior`. We were going to add another subclass called `GradStudent`; is this the right design?

d) Right now the `Student` class has a `display` method that prints out the HTML for the student to be displayed on the web site. We also have a similar `display` method in the `Course` class. Is this okay, and should we add a `display` method to other classes?

e) Right now, the `Student` object talks to the `Registrar` object when it wants to send messages to a `Course` object. The `Registrar` object intercepts the messages and passes them along to the relevant `Course`. The `Course` also sends messages to the `Registrar` object when it wants to access a particular `Student`. What changes, if any, would you suggest to this design?

How would you answer these questions? Support your decisions by citing relevant design concepts and heuristics presented in the course.

5. UML

Draw a UML class diagram for the "model" portion of the *TheirUW* system. Draw all classes related to the representation of the data and behavior of students adding/dropping courses. You don't need to show any behavior related to the user interface or connecting to databases.

Your diagram should show all relevant classes, methods, fields, access modifiers, and relationships between classes/objects with appropriate adornments in proper UML syntax. Follow appropriate object-oriented design guidelines as discussed in class, including using design patterns as appropriate. Make sure that your design accurately represents the major elements in the spec. You may add UML comments to your diagram if necessary for clarification.

6. Unit Testing

Write a JUnit test class for part of your *TheirUW* system. Your test class should cover the following functionality:

- No student can hold more than 19 units or more than 5 courses.
- A student cannot add a course without satisfying all of its prerequisites.

You should take into account the different types of unit testing discussed in class as well as the various constraints listed in the *TheirUW* project specification.