**Software Engineering ESOF 322 HW2 (50 pts)**

- Hand in a print out of your work during class on 10/02.
- You must use software to draw UML. Hand-drawn diagrams will receive 0.
- Use a partner. Clearly label every page with all names.
- Staple the pages together.

**Question 1 (25 pts)**

As the software architect of your organization you have been asked to come up with a design for a software tool.

1) Use the Unified Modeling Language (UML) **class diagram** to represent your design (15 pts).

2) draw a (UML) **sequence diagram** to show how to arrange a meeting (i.e. get a room, invite people, and specify a time) (10 pts).

The software tool is a daily planner. A typical daily planner allows you to arrange meetings, schedule appointments, invite people, reserve rooms, record your own tasks, plan vacations, etc. The following is a list of minimal functional requirements that was gathered from the eventual customers.

**Requirements**

- The software tool should allow a user to:
  - arrange a meeting or an appointment
  - re-schedule a meeting or an appointment of existing invitees
  - add or delete invitees to a meeting or appointment
  - respond to an invitation
  - propose a change to an invitation
  - record a task
  - record a vacation

**Hint:**

As you develop your design, think of the concepts and classes first, then try to identify which concepts should be related to each other. Once you have established the relationships between the concepts think about whether the relationships are associations, generalizations, aggregation, composition, etc. Then think about multiplicities between the relationships and finally use annotations, label your relationships, add roles, etc. to make your design more clear.

There is no one answer to a design, but in general a good architect can clearly capture the essential aspects of a design without unnecessary information.

**Question 2 (25 pts)**

The **Strategy** pattern is a design pattern used to encapsulate different behaviors and/or algorithms. The idea is to allow you to swap those strategies at will during the execution of a program. The architecture of the program remains the same. According to Gamma et. al, the “Strategy pattern is intended to define a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithm vary independently from clients that use it.”
In this exercise, you are to design, then implement a program in Java (or your OO language of choice) that uses the Strategy pattern to solve the following problem.

**Problem**

You as a client are required to meet with your TAX accountant. Depending on your level of income, investments, business investments, etc., your accountant will select a default taxing strategy provided by the TAX firm. The default taxing strategy basically adds your total income (salary and investments) and calculates your default tax rate of 28%. There are no deductions.

Use the Strategy pattern to create a Taxing Strategy that implements this default strategy as well as 3 alternate strategies:

**Alternative 1:** If your total income is > $118,000, then the tax rate goes up to 30%

**Alternative 2:** Your total income will be adjusted by subtracting from it all possible deductions (passed as an additional parameter, or via an attribute, etc.—your design and choice). The Tax rate is calculated after the deductions are subtracted from the income.

**Alternative 3:** If your income comes from your own business then this strategy uses your business income taxed at a rate of 30% minus business expenses (also a parameter or attribute).

You will test your code by creating a client program where you simulate the interaction between a client and his/her tax accountant. The accountant will request your information (interactively) then execute the default strategy. Then dynamically, the accountant will switch strategies given your information to one of the other 3 alternatives. The accountant will report the output of both, the default strategy and the alternative strategy.

You are to hand in a UML class diagram of your design, a printout of your code that clearly shows the interaction between the accountant and his/her client and the output from executing both strategies (switched dynamically!). Include print statements on every method to help instrument your code. Your code should be well documented (but not excessively). I will request that selected assignments give me a private demo of their running code.