Engineering 9874
Software Design & Specification
Final Exam

Dr. D. K. Peters

April 13, 2005

Instructions: Answer all questions. Write your answers to all questions except question 8 in the space provided on this paper. Write your answer to question 8 in the answer books provided.

This is a closed book test, no textbooks, notes, calculators or other aides are permitted. A formula sheet is provided.

Write your name and student number in the space provided on this sheet. Write your student number only in the space provided on other sheets of this paper. Please ensure that your answer book is clearly identified as indicated on the front cover and that you follow the rules listed there.

Q2	/ 10
Q3	/ 6
Q4	/ 7
Q5	/ 7
Q6	/ 10
Q7	/ 5
Q8	/ 45
Total	/ 90

1. [0 points, but an answer is **required**] For the project component of this course, please enter in the table below the percentage of the work done by each student on the team, including yourself. The total must be 100%. This information will be kept strictly confidential.

Name	% of work
Total:	100

2. [10 points] In the following table enter "T" (true) or "F" (false), as appropriate, for each statement.

Statement	**T/F**
In incremental development, increments are defined by the subset of the system components that will be implemented in each increment.	*F*
In a traditional software development process, the output of each stage is a product that can be verified against the output of the previous stage.	*T*
"Validation" is another name for software testing.	*F*
In incremental development, the design documents are updated only after the development is finished.	*F*
In incremental development it is normal for the team to be working on several increments at the same time.	*F*
The output of each increment in an incremental development process should be a tested and working system.	*T*
When using an incremental development process you must work hard to ensure that you have a complete detailed design of the system early because it will be difficult to modify the design at later stages.	*F*
If class A has a method m1, and class B extends class A overriding m1, the principle of substitutability states that the post-conditions for B::m1 can be stronger than those for A::m1.	*T*
It is possible to show that a system is correct by testing it.	*F*
A pattern is a collection of classes and interfaces that can be used as part of an application implementation.	*F*

3. [6 points] Give brief definitions of each of the following terms as they were used in this course:

   a) Interface
   The set of assumptions that designer/implementers of other components can make about a particular component. In classes this is represented by the set of public methods and attributes of the class together with their signature (syntactic interface) and behaviour (semantic interface).

   b) Verification
   Any activity that is undertaken to determine if a system meets its objectives or not. This may include testing, static analysis, symbolic analysis and model checking.

   c) Class
   A specification for a set of objects. It consists of a name, a set of attributes and a set of operations. A class represents a concept in the problem or solution domain.

4. [7 points]

   a) [4 points] The model-view structure was used repeatedly as an example in this course since it illustrates many useful design patterns. Name two design patterns that typically are used in a model-view system structure and briefly explain how they are used in m-v.

   Observer – *Views register as observers on with the model. They are notified whenever the model changes so that they can be updated.*

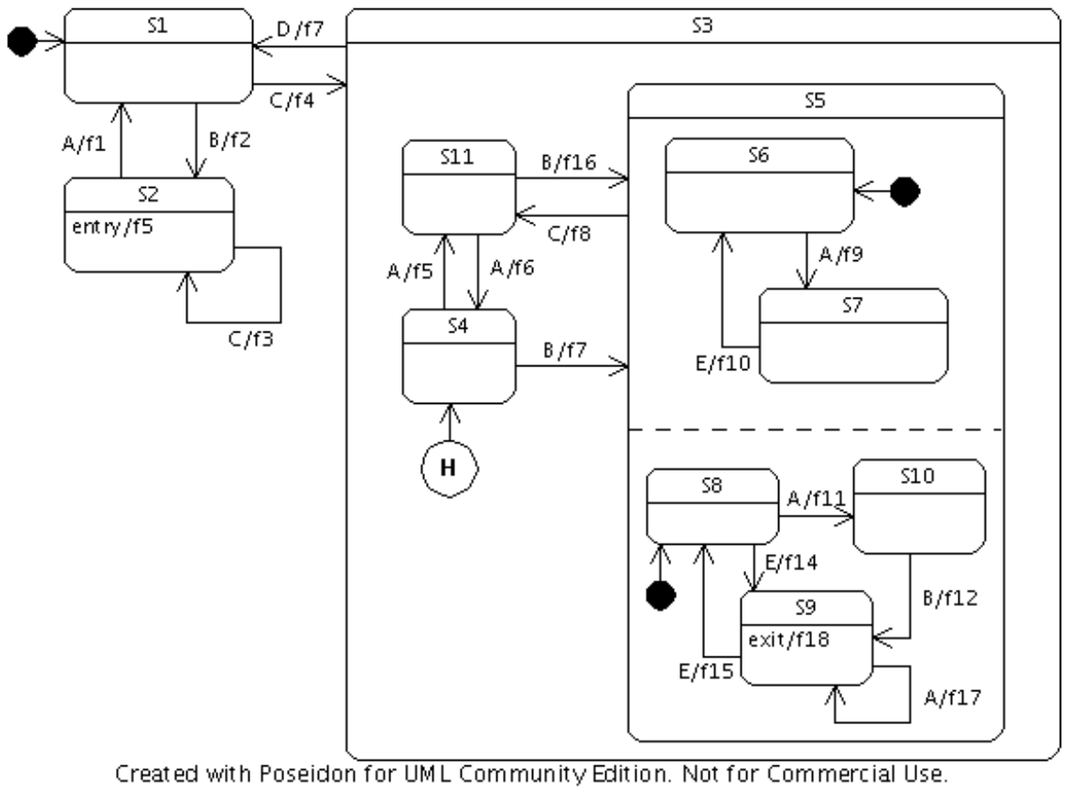
   Façade – *A model uses a Façade class to isolate the view from internal class structure in the model.*

   Composite – *A UI window is a composite of components (panel, button etc.).*

   Command – *UI actions are passed as commands to methods that implement the appropriate behaviour (controller).*

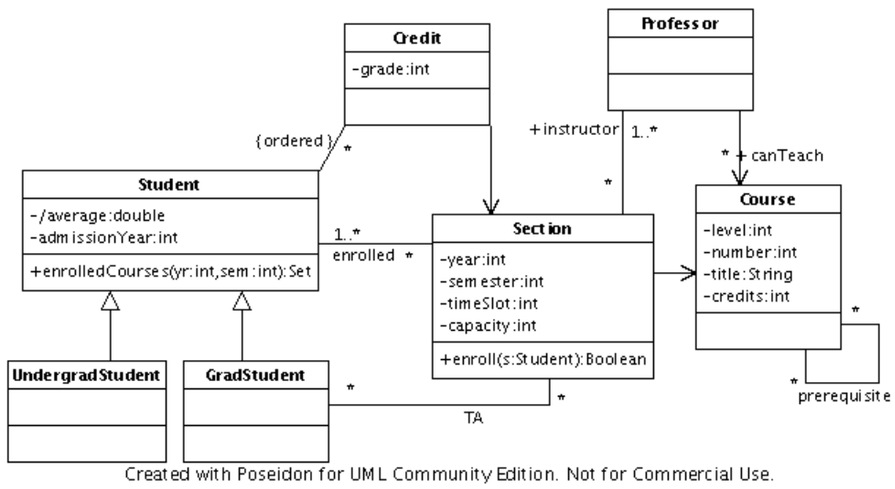
   b) [3 points] Briefly explain why design patterns are seen to benefit the software development community.
   DP are a means for experienced designers to communicate their repertoire of general solutions to problems that occur repeatedly. They can highlight pitfalls and good solution structures. DP are reusable solutions.

5. [7 points]



For the statechart above, give the reaction(s) and the new active state(s) that result from the each trigger in the given sequence.

Trigger	**Reaction(s)**	**New state(s)**
B	*f2, f5*	*S2*
C	*f3, f5*	*S2*
A	*f1*	*S1*
C	*f4*	*S3,S4*
B	*f7*	*S3, S5, S6, S8*
A	*f9, f11*	*S3, S5, S7, S10*
B	*f12,*	*S3, S5, S7, S9*
A	*f18, f17*	*S3, S5, S7, S9*
E	*f10, f18, f15*	*S3, S5, S6, S8*
D	*f7*	*S1*
C	*f4*	*S3, S5, S6, S8*
C	*f8*	*S3, S11*
A	*f6*	*S3, S4*
B	*f7*	*S3, S5, S6, S8*



6. [10 points]
Express the following constraints using OCL, with respect to the above model.

a) [2 points] A Course should have no prerequisites that are higher level courses.

```
context: Course
inv: prerequisite->forAll(c | c.level <= self.level)
```

b) [2 points] All instructors for a Section must be able to teach (canTeach) the Course.

```
context: Section
inv: instructor->forAll(p | p.canTeach->includes(self.Course))
```

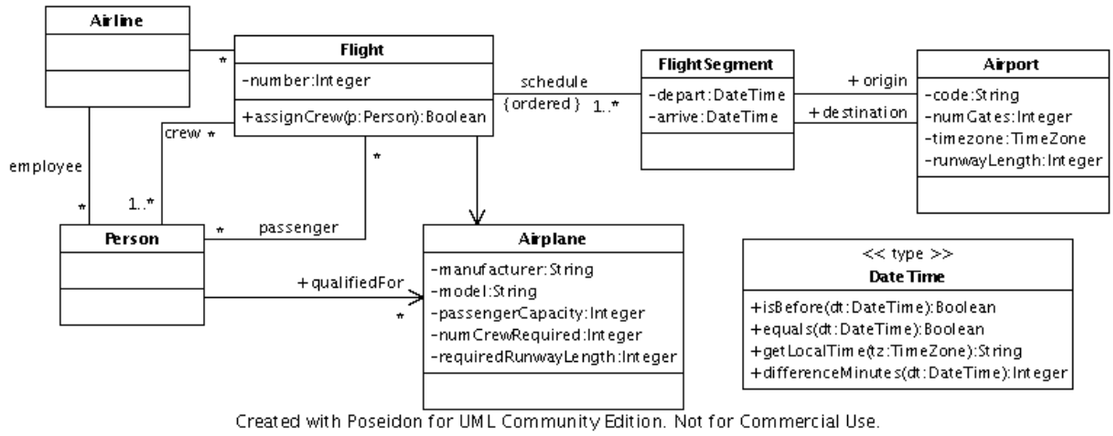
c) [2 points] A GradStudent cannot be a TA for a Course in which s/he is enrolled.

```
context: GradStudent
inv: self.TA.Course->excludesAll(self.enrolled.Course)
```

d) [4 points] The requirements for the method `Section::enroll(s:Student) : Boolean`, as follows: The method will assume that the Student has Credits for the required prerequisites and is not already enrolled in the Section. It will add the given Student to those enrolled in the Section only if there is room (i.e., the number of Students enrolled in the section is less than the capacity). The returned value is true iff the Student is enrolled in the Section when the operation completes.

```
context: Section::enroll(s:Student) : Boolean
pre : s.Credit.Section.Course->includesAll(self.Course.prerequisite)
pre : self.enrolled->excludes(s)
post: result = self.enrolled->includes(s)
post: self.enrolled@pre->size() < self.capacity implies
      self.enrolled->includes(s)
post: self.enrolled->includesAll(self.enrolled@pre)
```

7. [5 points]



Express the following constraints using OCL, with respect to the above model.

a) [2 points] All crew on a Flight must be qualifiedFor the Airplane.

```
context: Flight
inv: crew->forAll(c : Person | c.qualifiedFor->includes(self.Airplane))
```

b) [3 points] All Airports that a Flight is scheduled to land at have sufficient runwayLength for the Airplane.

```
context: Flight
inv: let rl : Integer = self.Airplane.requiredRunwayLength
     in schedule->forAll(s : FlightSegment | s.destination.runwayLength >= rl)
```

8. [45 points] (***Note:*** *Answer this question in the answer books provided.*) In this question you are to design a software system to allow two players using different computers on the internet to play the game of "dots", as described below. In your answer you may include any diagrams or text that you feel helps to clearly specify your design, but at a minimum you should include the following. (Points give the approximate weight that will be applied to each.)

   a) [20 points] UML class diagram(s) for the system, including public operations and public attributes for all classes and all class associations.

   b) [10 points] UML sequence diagram(s) illustrating the interactions involved in a player's turn.

   c) [15 points] A CRC description of each class, containing:

      1. the class name,
      2. a brief description of its role in the system, and
      3. a list of its responsibilities and the other classes that it collaborates with in order to accomplish each responsibility.

   Note that points will be awarded based on how well your design reflects the principles taught in this course.

   Game Rules for "Dots"

   This is a two-player game played on a rectangular grid of dots. The size of the grid is chosen by the player who initiates the game. Players take turns selecting a pair of horizontally or vertically adjacent dots to be connected with a line segment, as illustrated below. The object of the game is to be the player who connects the dots to form the fourth segment to close a square, which results in a point for that player, the enclosed square is coloured that player's colour and s/he is awarded another turn.

   The game ends when there are no pairs of unconnected dots available on the board. The winner is the player with the most points.