Name: Solutions Student #: ____________________

Engineering 9874
Software Design & Specification
Final Exam

Dr. D. K. Peters

April 17, 2006

Instructions: Answer all questions. Write your answers to all questions except question 7 in the space provided on this paper. Write your answer to question 7 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.

If you find a question to be ambiguous or feel that you must make assumptions in order to complete your answer, please clearly state those assumptions on your paper.

Q1	/ 10
Q2	/ 6
Q3	/ 9
Q4	/ 15
Q5	/ 9
Q6	/ 6
Q7	/ 45
Total	/ 100

Student #: ____________________

1. [10 points] In the following table enter "T" (true) or "F" (false), as appropriate, for each statement.

Statement	**T/F**
A "traditional" software development process is better suited to projects with large teams and stable, well defined, requirements.	*T*
In incremental development, the easy use cases should be in the early increments.	*F*
Stepwise refinement is another term for incremental development.	*F*
The invariant for a derived class can be weaker than for the base class.	*F*
It is the duty of the calling code to ensure that the pre-conditions for the methods being called are true.	*T*
Dynamic aspects of a system design can be represented in class diagrams.	*F*
If done well, UML Diagrams are sufficient for a designer to give to a programmer as a complete specification of the system behaviour.	*F*
An Application Framework may include a complete working application.	*T*
Verification will always result in a yes or no answer.	*F*
Exhaustive testing is not possible for realistic systems.	*T*

2. [6 points] Give brief definitions of each of the following terms as they were used in this course:

   a) Object
   An (run-time) instance of a class. Contains attributes and methods that represent some concept in the system.

   b) Design Pattern
   A named problem-solution pair that can be applied in new contexts.

   c) Black-box Testing
   Method of selecting test cases based on the externally observable behaviour of the system under test.

3. [9 points]

   a) [2 points] What are the essential components of a design pattern description? Why are they essential?

      Name *Allows us to communicate concisely with other designers.*

      Problem *Describes the kind of situations where this pattern is applicable.*

      Solution *Describes how the problem can be solved.*

      Consequences *Discussion of pros and cons of this pattern. Highlights some potential pitfalls.*

   b) [2 points] The LayoutManager in Java AWT is a good example of what pattern? Explain.
      Strategy (also accept Delegation).
      The strategy for organizing components in a container is determined by the choice of LayoutManager. Each container has a LM, and new LM can be added without old containers needing to know about it.

   c) [5 points] The observer pattern was used frequently in this course. Briefly discuss some of the positive and negative consequences, implementation issues and variations of this pattern.

      Observer Consequences

      + *ConcreteSubject may be reused without reusing observers.*

      + *Observer classes may be added or removed without modifying ConcreteSubject or other observer classes.*

      + *Observers may belong to higher level in a layered system.*

      + *Supports broadcast to many observers*

      - *Cost of update is hidden from subject.*

      - *No indication of how subject has changed — may lead to costly unneeded updates.*

      - *Subject must be consistent when it calls notify.*

      - *Too many notifications — every change causes notify.*

      Implementation Issues and Variations

      Dangling references *Deleting either a subject or observer may leave dangling references — need to ensure that referring objects are informed of delete (de-register self before delete).*

      Update triggering *Whose responsibility is it to call update?*
      - *state-setting methods in subject — may lead to too many updates.*
      - *clients — error prone.*

      Observing more than one subject *— update needs a parameter to identify itself to the observer.*

      Update protocols *How is the information about he change communicated to the observer?*
      - *Push model — subject sends observers detailed information about the change. Subject needs to know more about observers.*
      - *Pull model — subject sends minimal information, observer goes and gets it. May be less efficient.*

      Specifying changes of interest explicitly *Observers register as being interested in specific kinds of changes.*

      Change manager *Encapsulates particularly complex update semantics (example of* Mediator *pattern).*

4. [15 points] In the space below, draw a statechart diagram to describe the behaviour of a simple microwave oven controller. The events that the controller must react to are as follows:

Name	**Description**	**Effect**
Add1Sec	User pressed the button labeled "+1sec" on the oven keypad.	Adds 1 second to the cook time.
Add10Sec	User pressed the button labeled "+10sec" on the oven keypad.	Adds 10 seconds to the cook time.
StartStop	User pressed the "Start/Stop" button on the oven keypad.	Starts cooking if there is a cook time set and the oven is not cooking. If the oven is cooking it pauses the cooking.
Reset	User pressed the "Reset" button on the oven keypad.	Clears the cook time and stops any cooking.
Power	User pressed the "Power Level" button on the oven keypad.	Only effective when a cook time is set and the oven is not cooking. Causes the cook power level to be decreased by 10%.
Tick	Occurs once per second.	
DoorOpen	The oven door is opened.	The oven should not cook with the door open.
DoorClosed	The door is closed.	

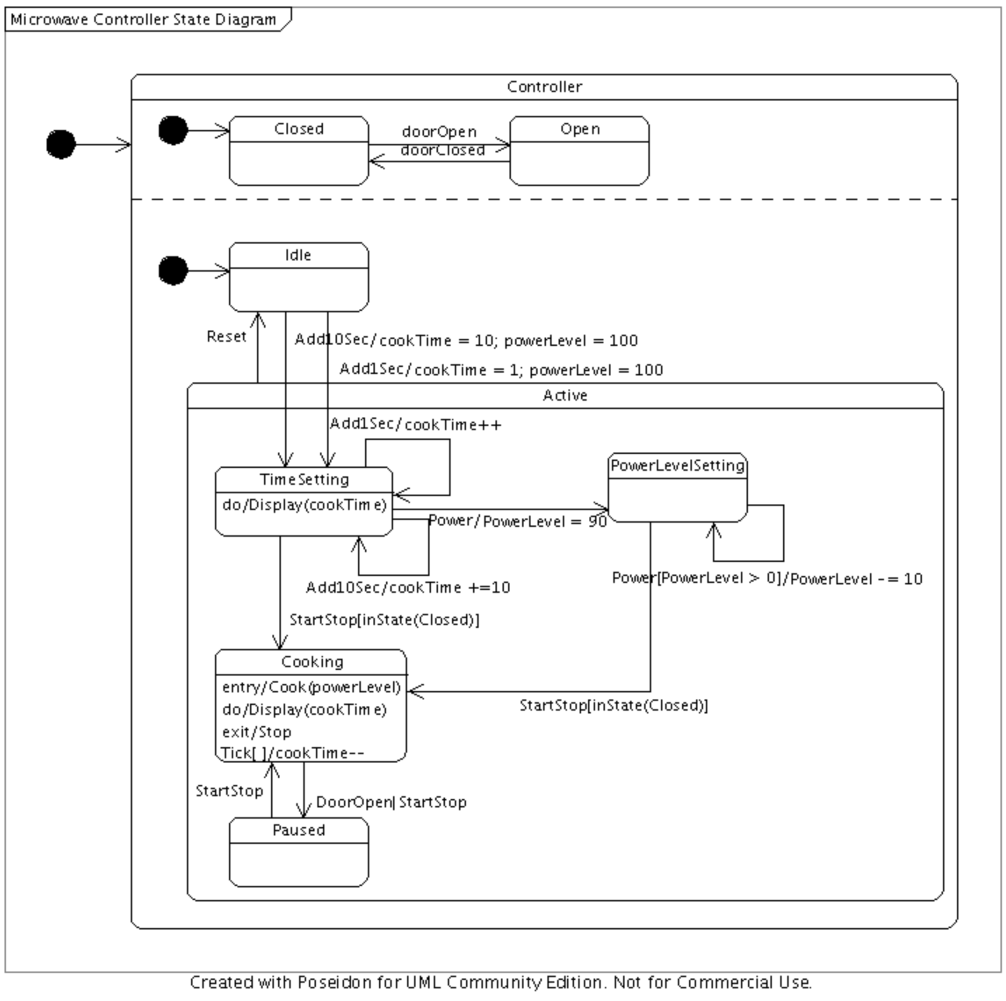
The following sequence of events would start the oven cooking for 13 seconds at 80% power, assuming that the door is closed: Add10Sec, Add1Sec, Add1Sec, Add1Sec, Power, Power, StartStop.

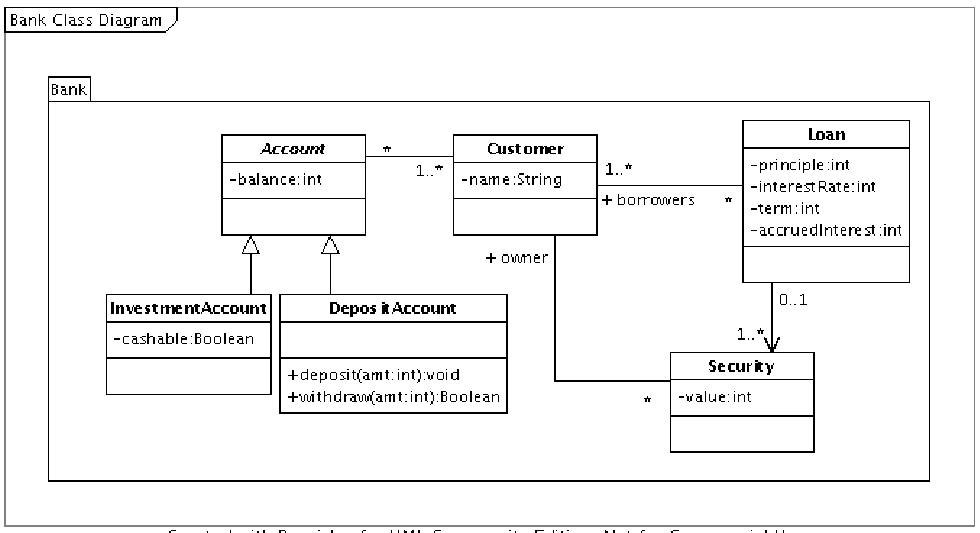
The following outputs are to be controlled:

Cook(*pl*) Causes the oven to cook at power level *pl*.

Stop Causes the oven to stop cooking.

Display(*val*) Displays the string *val* on the display.


Microwave Controller State Diagram
Controller
Closed
doorOpen
doorClosed
Open
Idle
Reset
Add10Sec/cookTime = 10; powerLevel = 100
Add1Sec/cookTime = 1; powerLevel = 100
Active
Add1Sec/cookTime++
TimeSetting
do/Display(cookTime)
PowerLevelSetting
Power/PowerLevel = 90
Add10Sec/cookTime +=10
Power[PowerLevel > 0]/PowerLevel -= 10
StartStop[inState(Closed)]
Cooking
entry/Cook(powerLevel)
do/Display(cookTime)
exit/Stop
Tick[ ]/cookTime--
StartStop[inState(Closed)]
StartStop
DoorOpen| StartStop
Paused
Created with Poseidon for UML Community Edition. Not for Commercial Use.



Created with Poseidon for UML Community Edition. Not for Commercial Use.

5. [9 points]
Express the following constraints using OCL, with respect to the above model.

a) [2 points] The requirements for the deposit method of a DepositAccount. The account balance must be increased by the deposit amount (amt), which must be positive.

```
context DepositAccount::deposit(amt: Integer)
pre: amt >= 0
post: balance = balance@pre + amt
```

b) [3 points] The requirements for the withdraw method of a DepositAccount. If there are sufficient funds in the account then the balance must be decreased by the withdraw amount (amt), which must be positive, and the returned value is true. If there are insufficient funds then there is no change to the account balance and false is returned.

```
context DepositAccount::withdraw(amt: Integer) : Boolean
pre: amt >= 0
post: if (balance@pre >= amt) then
result = true and balance = balance@pre - amt
else result = false and balance = balance@pre
endif
```

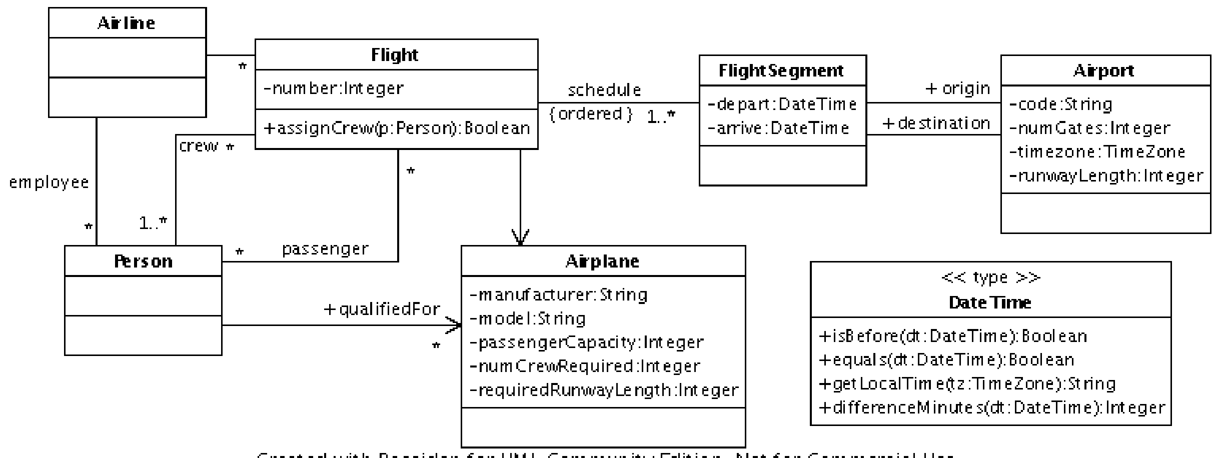
c) [2 points] All the Securities for a Loan are owned by borrowers of the Loan.

```
context Loan
inv: Security->forAll(s : Security | borrowers->includes(s.owner))
```

d) [2 points] The total value of the Securities associated with a Loan is at least as much as the Loan principle.

```
context: Loan
inv: principle <= Security.value->sum()
```

6. [6 points]



Created with Poseidon for UML Community Edition. Not for Commercial Use.

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

a) [2 points] The origin of a FlightSegment in a schedule must be the same Airport as the destination of the previous FlightSegment in that schedule.

```
context: Flight
inv: Sequence{ 1..schedule->size()-1 }->forAll(i |
schedule->at(i+1).origin = schedule->at(i).destination)
```

b) [2 points] A flight's stopover time at each airport (the difference between arrival on a FlightSegment and departure on the following FlightSegment) must be at least 30 minutes. Assume that DateTime is based on a fixed time zone (e.g., GMT).

```
context: Flight
inv: Sequence{ 1..schedule->size()-1 }->forAll(i |
       schedule->at(i+1).depart.differenceMinutes(schedule->at(i).arrive) >= 30)
```

c) [2 points] All Airports that a Flight is scheduled to land at have sufficient runwayLength for the Airplane.

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

7. [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 "Othello", 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 "Othello"

   The playing space consists of a 8 by 8 board of possible positions, which is presented to the players in a graphical display. At the start of the game the board is laid out as illustrated in Figure 1(a). Two players, randomly designated "White" and "Black", play the game by alternately taking turns selecting (by mouse click) an unmarked position in the game space on which to place a marker (a white disc for "White" or a black disk for "Black"). A marker can only be placed such that it becomes an endpoint of at least one straight line (horizontal, vertical or diagonal) terminated by another marker of the same colour and containing at least one marker of the opposite colour (i.e., a sandwich) as illustrated in Figure 1(b). For every such line formed by placing the marker, all of the opposite coloured markers in the sandwich are changed to the same colour as the newly laid marker (see Figure 1(c)). If a player cannot place a marker then she must skip a turn (pass). If a player can place a marker then she must do so. Figure 2 shows other sample moves. The game ends when neither player can place a marker. The winner is the player with the most markers of their colour on the board.

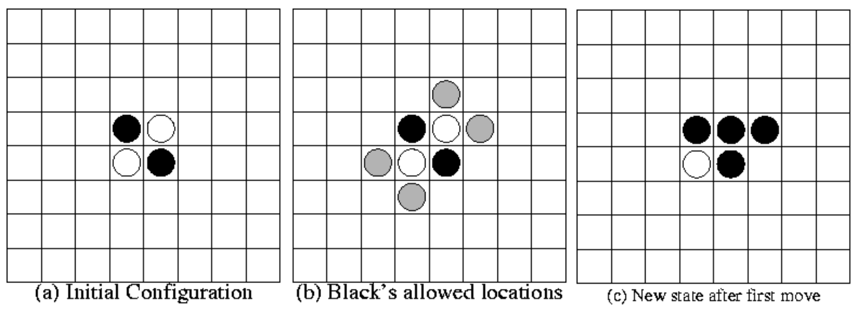

(a) Initial Configuration
(b) Black's allowed locations
(c) New state after first move

Figure 1: Game Layout

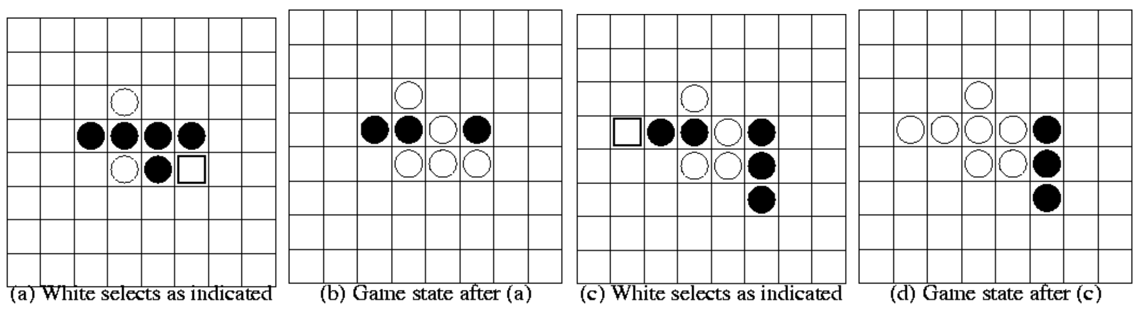

(a) White selects as indicated
(b) Game state after (a)
(c) White selects as indicated
(d) Game state after (c)

Figure 2: Sample Moves