COMPUTER ENGINEERING DEPARTMENT

CMPE 202

Software Systems Engineering

Fall 2011

MIDTERM PROJECT

(80 points Total)

You may use your notes, books and Internet resources. But, you must do your own work. Copying someone else's source code or UML diagrams will result in an automatic "Zero" on the midterm for all parties involved. Please refer the on the University's Policy on the next page for guidelines on how to avoid plagiarism.

YOUR **LAST** NAME	
YOUR **FIRST** NAME	
LAST **4-DIGITS** OF YOUR STUDENT ID	

Honesty Policy

Cheating

San José State University defines cheating as the act of obtaining or attempting to obtain credit for academic work through the use of any dishonest, deceptive, or fraudulent means. Cheating includes:

- Copying, in part or in whole, from another's test or other evaluation instrument including homework assignments, worksheets, lab reports, essays, summaries, quizzes, etc.;
- Submitting work previously graded in another course without prior approval by the course instructor or by departmental policy;
- Submitting work simultaneously presented in two courses without prior approval by both course instructors or by the department policies of both departments;
- Using or consulting sources, tools or materials prohibited by the instructor prior to, or during an examination;
- Altering or interfering with the grading process;
- Sitting for an examination by a surrogate, or as a surrogate;
- Any other act committed by a student in the course of their academic work that defrauds or misrepresents, including aiding others in any of the actions defined above.

Plagiarism

San José State University defines plagiarism as the act of representing the work of another as one's own without giving appropriate credit, regardless of how that work was obtained, and submitting it to fulfill academic requirements. Plagiarism includes:

- Knowingly or unknowingly incorporating the ideas, words, sentences, paragraphs, or parts of, or the specific substance of another's work, without giving appropriate credit, and representing the product as one's own work;
- Representing another's artistic/scholarly works such as musical compositions, computer programs, photographs, paintings, drawing, sculptures, or similar works as one's own.

Honesty Pledge

- I have read and understand the university definition of cheating and plagiarism.
- I will not copy ("*cut and paste*") *any* material from *any* source without proper attribution.
- I will not discuss *any* assignment that is part of the course grade with *anybody* without prior approval from the instructor.
- I understand that there is no make-up opportunity for assignments where breaches of academic integrity have occurred.
- I understand that breaches of academic integrity will result in a failing grade in the course, as well as possible referral for additional administrative sanctions.
- I will ask the instructor for guidance if I have any questions about how the above is to be understood.

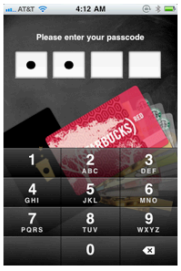
Pay on the Go (5 points)

Given the "Pay on the Go" Use Case as discussed in class, implement your "final" Astah UML design in Java using BlueJ. Make sure your Astah UML file includes the following diagrams:

1. **Detailed Sequence Diagram for "Pay on the go"**

2. **UML Class Diagram.**
   Should include classes, interfaces and markers on classes for "boundary", "control" and "entity" stereotypes. Note: your solution will contain only "one" class diagram that may evolve as you work on the midterm requirements. Submit only the "final" version of your design.

3. **State Diagram for Pin State Machine**
   Note: pin authentication, as discussed in class, allows for 4-digit pins with keypad coordinates mapping to pin digit as shown below.



Pin Screen

The Key Pad should convert touch coordinates into "keys". Use the following coordinates-to-key mappings.

(1,1) = **1**	(2,1) = **2**	(3,1) = **3**
(1,2) = **4**	(2,2) = **5**	(3,2) = **6**
(1,3) = **7**	(2,3) = **8**	(3,3) = **9**
(1,4) = _	(2,4) = **0**	(3,4) = **X**

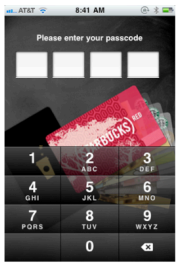
Coordinates: (x,y) = Key

NOTE

Implement your solution with a default PIN of "1234".

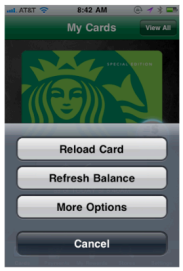
PART 2 – Interface Requirements (5 points)

Your design and solution should include classes for all of the following Screens. Additionally, you should also implement an "App Controller" class which should contain a "display()" method to displays the current Screen Name as well as well as call the "display()" method of the current Screen each time the Screen changes.



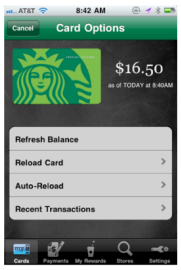
Pin Screen

My Cards - Main

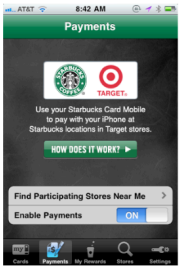


My Cards - Options

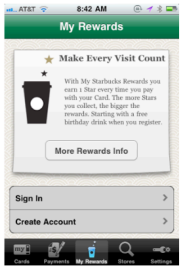
My Cards - Pay



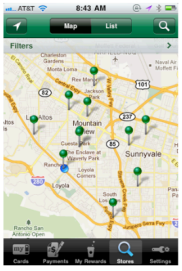
My Cards
More Options



Payment Setup



Rewards Setup

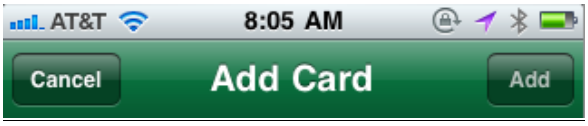


Find Starbucks

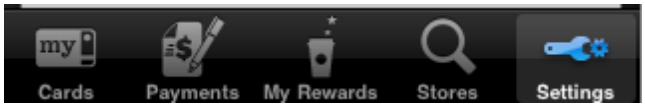
All Screens (above) should implement a common interface which includes "display()" and "touch(x,y)" methods. The "touch(x,y)" method accepts the coordinates below. Screens can contain components (such as the "keypad") which have their own relative coordinates. Touch events can be mapped from the Screen coordinates into the widget's relative coordinates. For example, given the following layout of the PinScreen, a "touch(1,5)" would map into a "keypress(1,1)" on the KeyPad.

(1,1)	(2,1)	(3,1)
(1,2)	(2,2)	(3,2)
(1,3)	(2,3)	(3,3)
(1,4)	(2,4)	(3,4)
(1,5)	(2,5)	(3,5)
(1,6)	(2,6)	(3,6)
(1,7)	(2,7)	(3,7)
(1,8)	(2,8)	(3,8)

App Controller should implement "Top Left Command" and "Top Right Command" which would be mapped to certain operations in a particular screen (if any at all). For example, the Add Card Screen maps "Top Left" to "Cancel" and "Top Right" to "Add".



App Controller also must implement a Menu Bar (typically position at the bottom of an iPhone Screen). This should be specified in your design as an Interface that App Controller implements and should contain call signatures for menu1(), menu2(), menu3(), menu4() and menu5() to represent each of the five menu options an application may have. For example, menu1() maps to "Cards", menu2() maps to "Payments", etc…



To summarized, there should be two interfaces with the following suggested names and call signatures:

AppScreen Interface:

touch(x, y)
display( )
topLeftCmd( )
topRightCmd( )

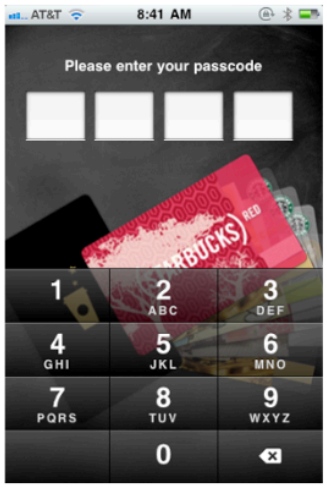
MenuBar Interface:

menu1( )
menu2( )
menu3( )
menu4( )
menu5( )

All Screens must implement the **"AppScreen" Interface.**
AppController must implement the **"AppScreen" and "MenuBar" Interfaces**.

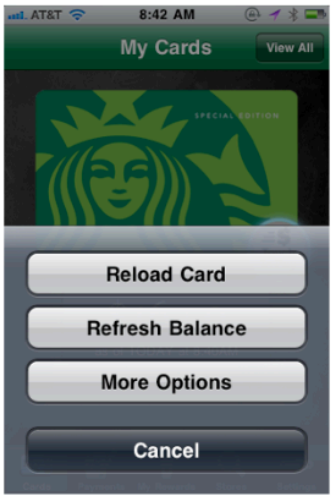
NOTE: The MenuBar should not be active until after the user has successfully authenticated with a valid Pin from the Pin Screen.

PART 3 – Screen Flow Requirements (10 points)

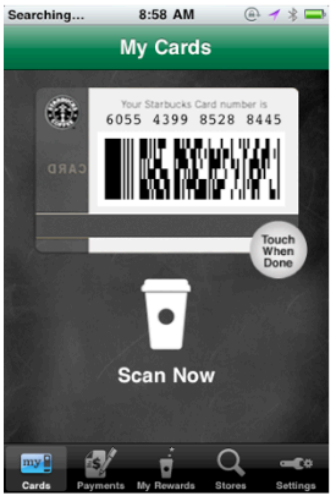


Pin Screen

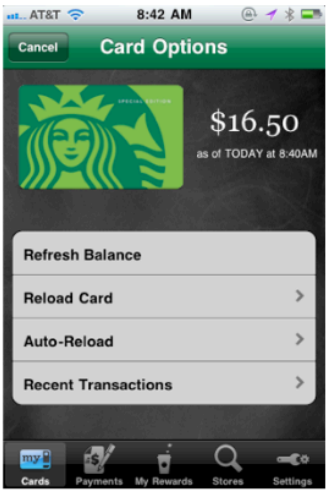
My Cards - Main



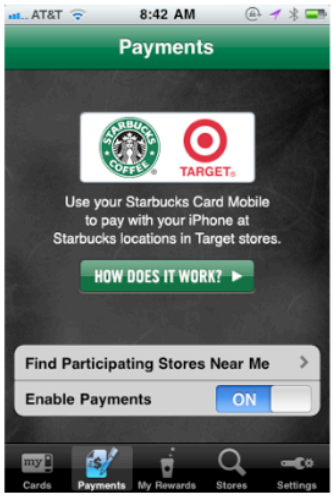
My Cards - Options



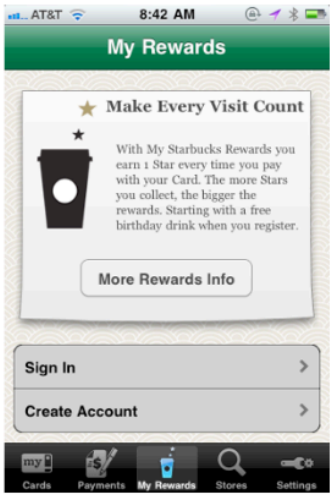
My Cards - Pay



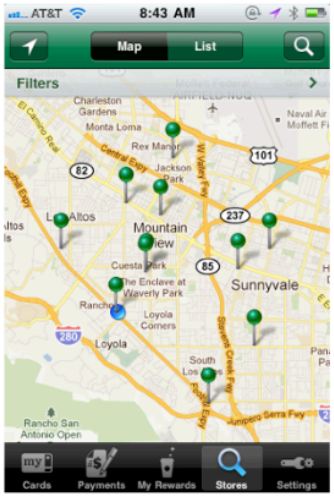
My Cards More Options



Payment Setup



Rewards Setup

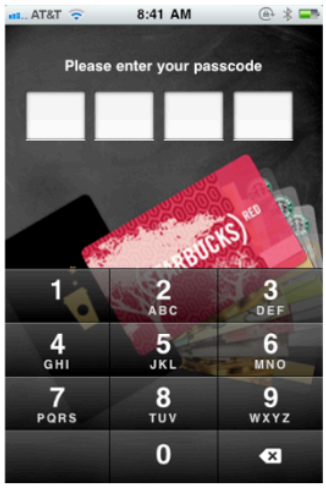


Find Starbucks

Implement the following Screen Flows and Touch Behaviors:

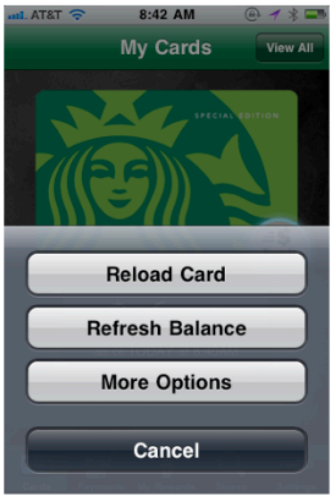
1. After a "Successful" Pin Validation, the "My Cards – Main" Screen appears
2. A **touch(3,3)** on the "My Cards – Main" screen switches to the "My Cards – Pay" Screen
3. A **touch(2,4)** on the "My Cards – Main" screen switches to the "My Cards – Options" Screen
4. A **touch(1,7), touch(2,7) or touch(3,7)** on the "My Cards – Options" Screen switches to the "My Cards – More Options" screen
5. A **touch(3,3)** on the "My Cards – Pay" screen switches to "My Cards – Main" Screen.

PART 4 – Screen Display Requirements (5 points)

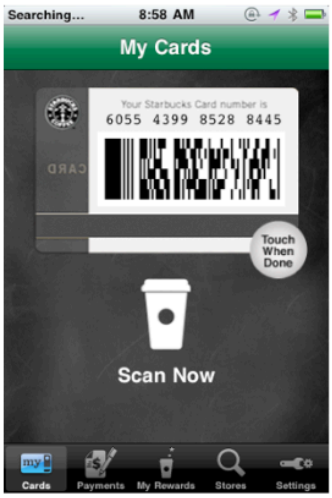


Pin Screen

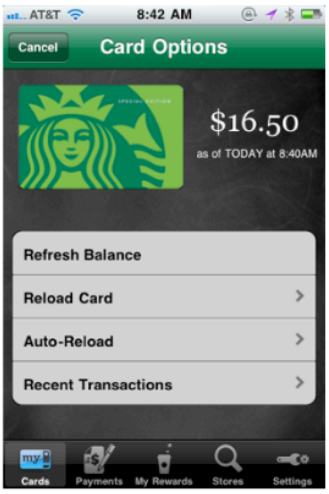
My Cards - Main



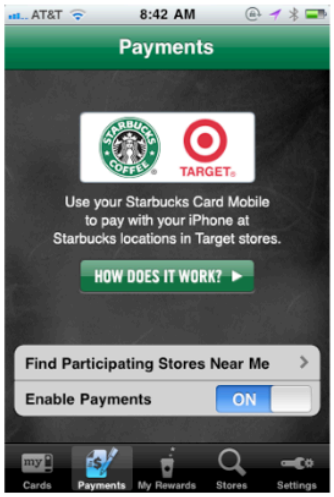
My Cards - Options



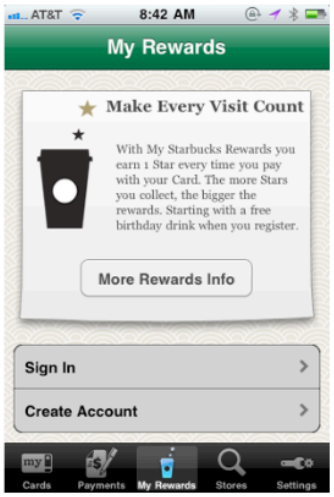
My Cards - Pay



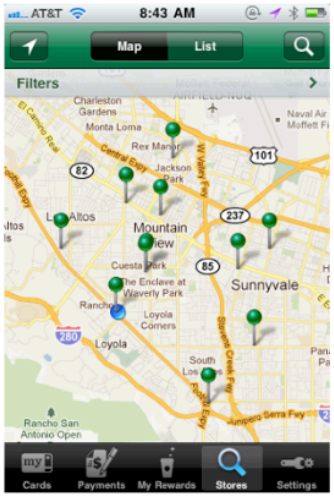
My Cards More Options



Payment Setup



Rewards Setup



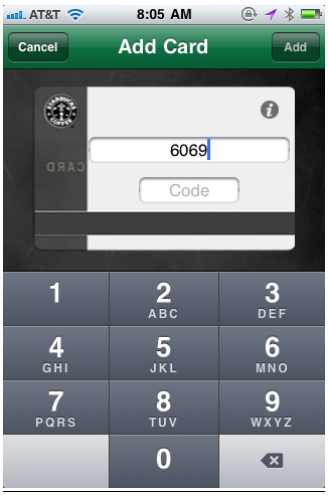
Find Starbucks

Implement the following Display Output for each screen:

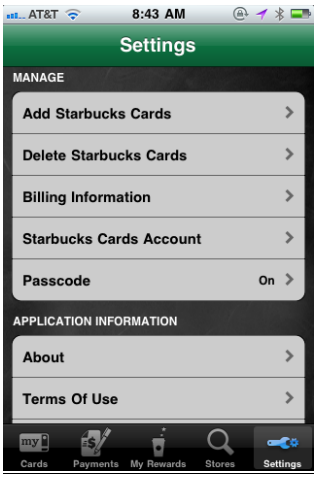
1. Calling "display()" on the "**My Cards – Main**" Screen should **show the balance of the current active card**.
2. Calling "display()" on the "**My Cards – Pay**" Screen should print-out the words "**Scan Now**"
3. Calling "display()" on the "**My Cards – Options**" Screen should print-out the words "**Reload, Refresh Balance, or More Options**".
4. Calling "display()" on the "**My Cards – More Options**" Screen should print-out the words "**Refresh, Reload or View Recent Transactions**"
5. Calling "display()" on the "**Payment Setup**" Screen should print-out the words "**Enable Payments?**"
6. Calling "display()" on the "**Rewards Setup**" Screen should print-out the words "**Make Every Visit Count**".
7. Calling "display()" on the "**Find Starbucks**" Screen should print-out "**Google Map of Local Starbucks**"

PART 5 – Additional Screen Requirements (15 points)

Add the following "Add Card Screen" which reuses the KeyPad widget. In the "Add Card Screen", 16 digits can be entered for the Card ID and 8 digits can be entered for the Card Code. A touch() with coordinates of (1,3), (2,3) and (3,3) should set the Card ID entry as the current focus. Likewise, touch (2,4) will switch the focus to the Card Code. With the focus of the cursor, new digits "touched" should be appended to the current "Card ID" or "Card Code" numbers. The "Top Right Cmd" on the Add Card Screen maps to "Submit the Add New Card Request" with the digits enterred and the "Top Left Cmd" maps to "Cancel Card Entry" and return to the previous screen.



Add Card Screen

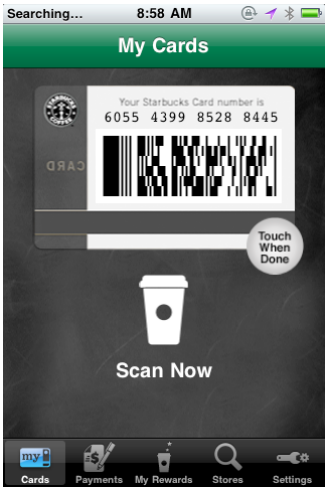


Settings Screen

To add a card, a user would touch the "Settings" menu and then the "Add Starbucks Cards" option. Design a **Detailed Sequence Diagram** for this workflow, which should result in creating a new Card object making it the new "Default" card for payment.

Additionally, the "display()" method for Add Card Screen should print "Enter a new Card"; likewise, the "display()" method for Settings Screen should print "Manage Card, Billing, Passcode. Show About & Terms."

NOTE: Add all new cards with a default initial balance of $16.50.



For Payments, "hard code" and **charge of $1.50 for each transaction** made with the current card.

The following touch(x,y) coordinates should trigger the payment.

touch(1, 2), touch (2,2), touch (3,2)
touch(1, 3), touch (2,3), ~~touch (3,3)~~
touch(1, 4), touch (2,4), touch (3,4)

PART 6 – Patterns & Unit Testing (40 points)

Implement the following Patterns in your solution:

1. In addition to the "touch(x,y)" and "display()" methods, the ***App Controller*** should also implement the following methods for the menu ribbon: menu1(), menu2(), menu3(), menu4() and menu5(). Using the ***Command Pattern***, wire these menus to switch the current screen to: My Cards - Main, Payments, My Rewards, Find Starbucks and Settings, respectively. For example, calling menu1() on the App Controller should result in the display of the "My Cards – Main" screen.

2. Implement the **State Pattern** for your Pin Entry process. The State Machine should track the number of pins entered and automatically try to validate the pin upon the fourth touch. Additionally, implement the **Observer Pattern** for the Passcode Display assigning the Observer responsibility to the Passcode Display widget, which echos the number of digits entered so far upon being notified by the change by a Subject object.

3. Implement the **Singleton Pattern** for the App Controller.

4. Implement a **Chain of Responsibility** for Widgets composed within the ***Pin Screen*** and ***Add Card Screen***. The objects within the chain should determine if the "touch(x,y)" event is interesting to it and may have to ***Adapt (i.e. Adapter Pattern)*** the event to the target widget for processing. For example, the Pin Screen may have two objects in the event chain, one what "wraps" Passcode Display widget and another that wraps the Key Pad widget. The Passcode Display receiver ignores all touch events since it is an output only widget. The Key Pad receiver will convert touch(x,y) events into keypad events if it determines that the touch is over the KeyPad region.

Create Unit Tests for:

1. Your ***Pin Entry State Machine***. Your unit tests should cover all state transitions and must validate the correct outcome of each transition from state-to-state.

2. Your Menu Selections for ***App Controller***. Your unit tests should cover all "Commands" and should verify that the correct Screen is set as the current screen.

SUBMISSION:

1. One Zip File with your name and course section number.

   For example: ***lastname-firstname-cmpe202-01.zip*** *or* ***lastname-firstname-cmpe202-01.zip***

   If your submission doesn't have this naming convention, an automatic deduction of 10 points will be accessed.

2. The Zip file should contain: BlueJ Java Source Code and Astah UML Project Files. UML diagrams should include Sequence Diagrams, State Diagram(s) and the "***Final***" UML Class Diagram which reflects your implementation for Parts 1 – 6 of this midterm project.

3. The Zip file should be uploaded to D2L by the due date. Late Submission will be access at -10 points per day late.