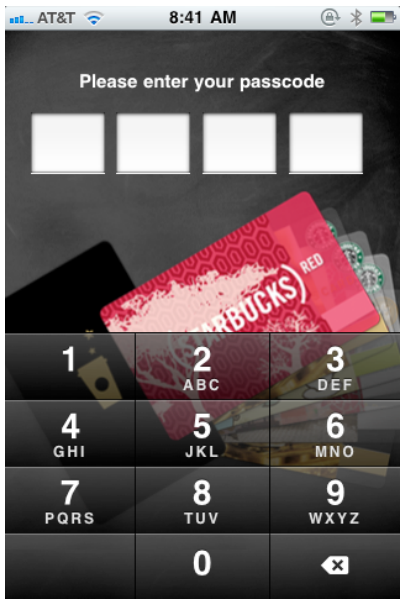
Starbucks Mobile App

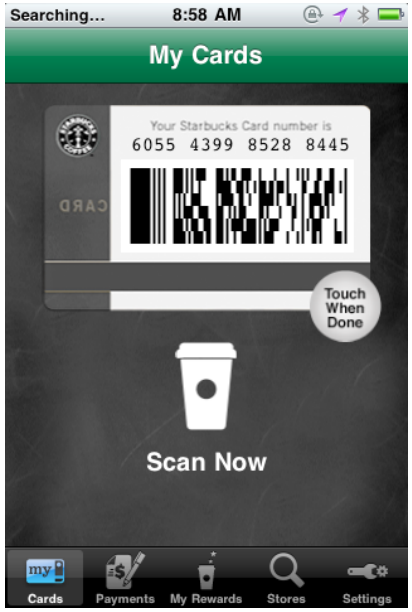
Project Source Code Package
(*implementation with design patterns*)

Project Description

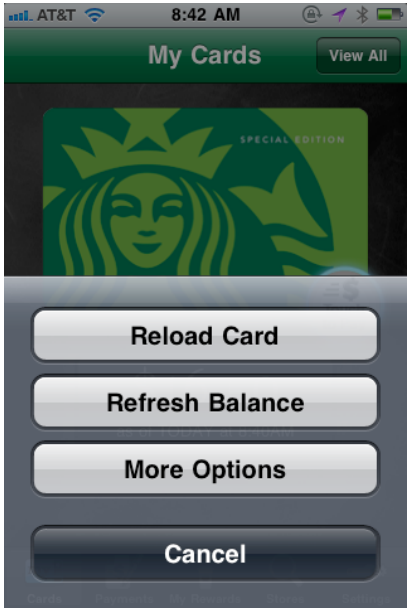


Pin Screen

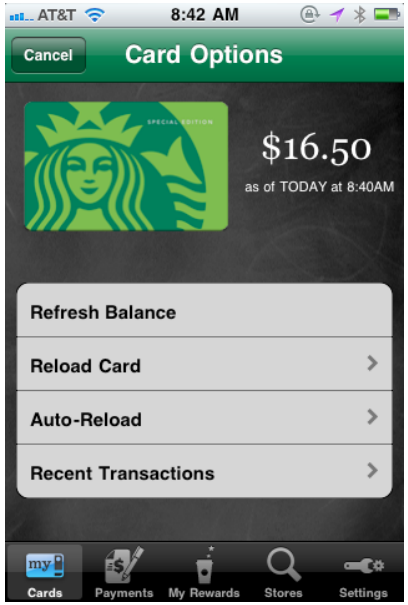
My Cards - Main



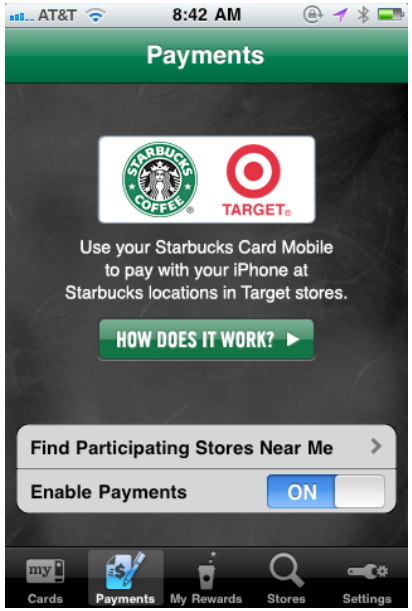
My Cards - Pay



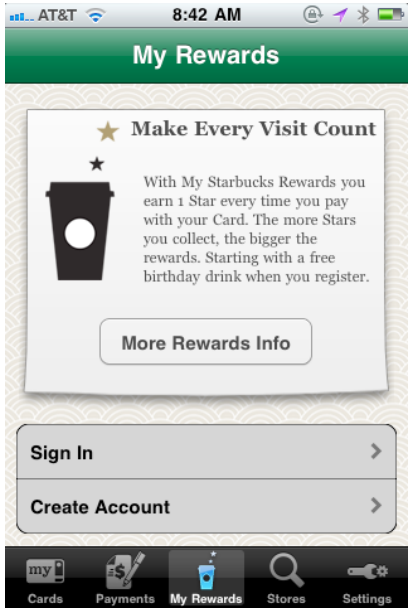
My Cards - Options



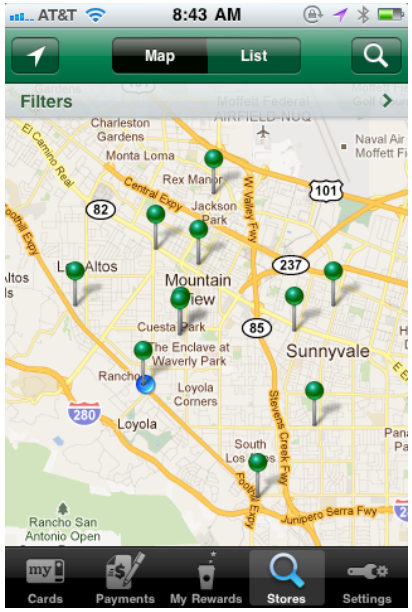
My Cards
More Options



Payment Setup

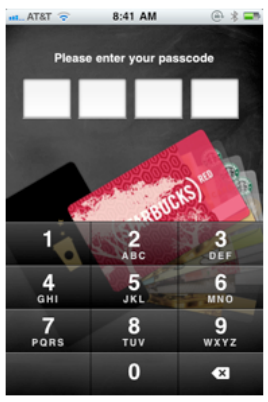


Rewards Setup



Find Starbucks

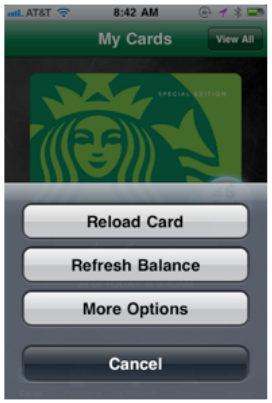
Solution should implement an “App Controller” class which should contain a “display()” method to display the current ***Screen Name*** as well as well as call the “***display()***” method of the current Screen each time the Screen changes



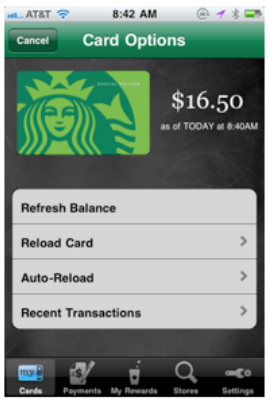
Pin Screen

My Cards - Main

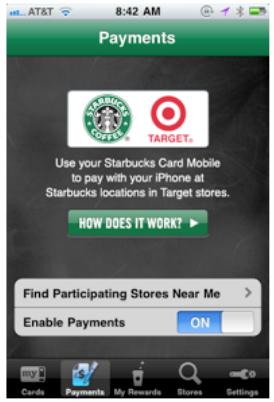
My Cards - Pay



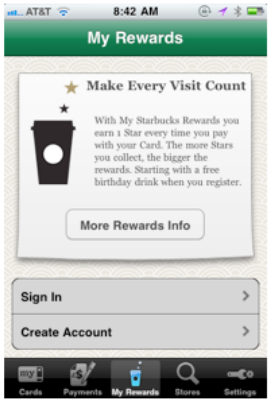
My Cards - Options



My Cards
More Options



Payment Setup



Rewards Setup

Find Starbucks

All Screens should implement a common interface which includes "***display()***" and "***touch(x,y)***" methods.

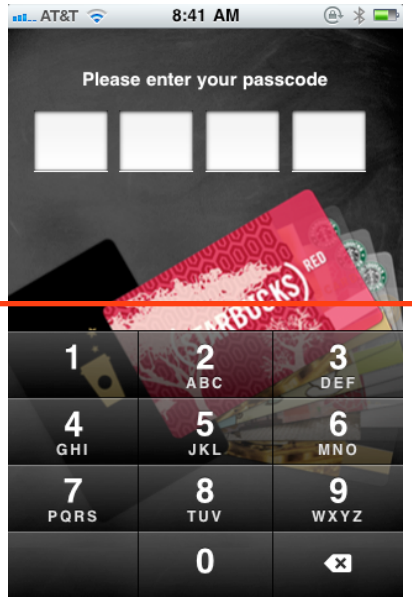
The "***touch(x,y)***" method accepts the coordinates below. Screens can contain components (such as the "keypad") which have their own relative coordinates.

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)

PinScreen Coordinates (x,y)

Pin Screen



(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)

(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**

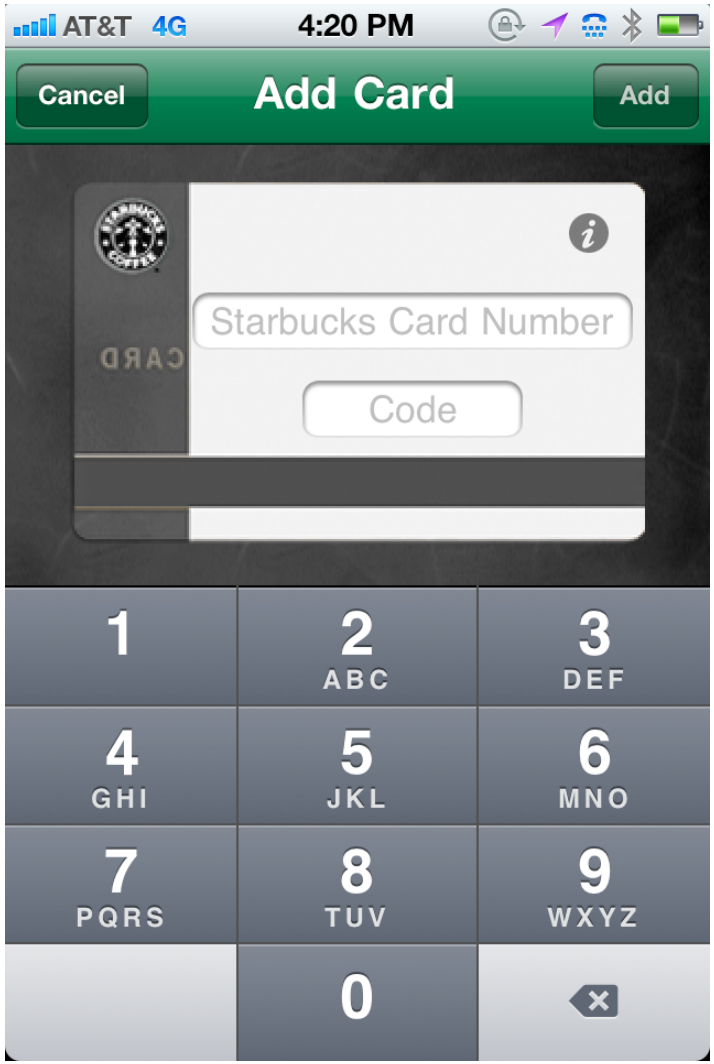
Coordinates: (x,y) = Key

KeyPad
inside PinScreen

Sample Touch Event Mapping:

== >Touch (1,5)
==> KeyPress(1,1)
==> Key Number 1

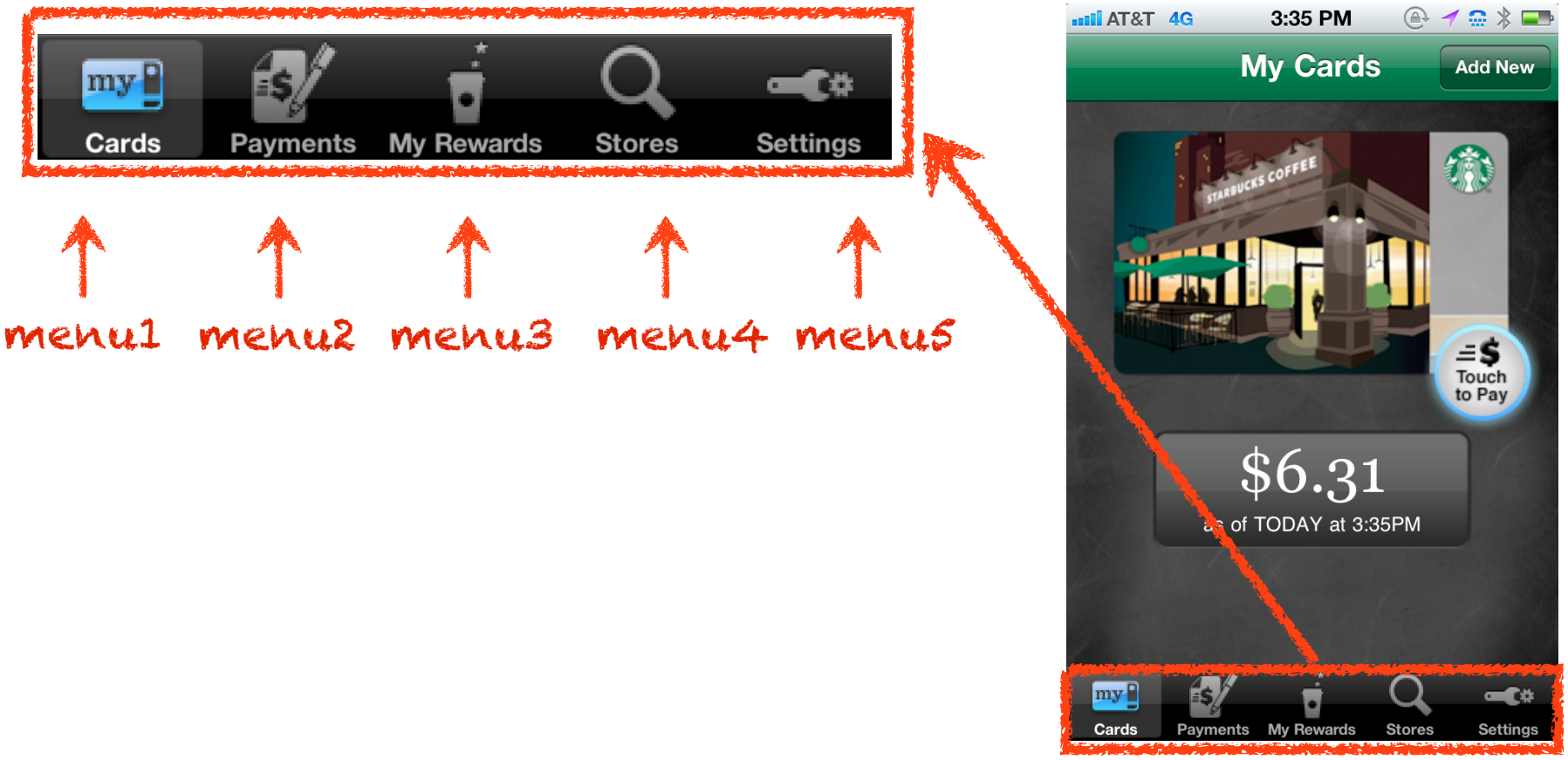
App Controller should implement "***Top Left Command***" and "***Top Right Command***" which would be mapped to certain operations in a particular screen (if any at all). For example, the ***Add Card Screen maps "Top Left" to "Cancel"*** and ***"Top Right" to "Add"***.



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…



AppScreen and MenuBar Interfaces

AppScreen Interface:

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

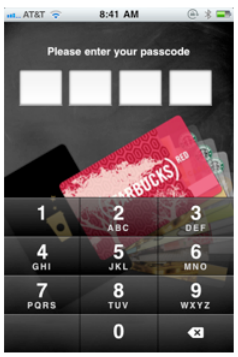
MenuBar Interface:

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.

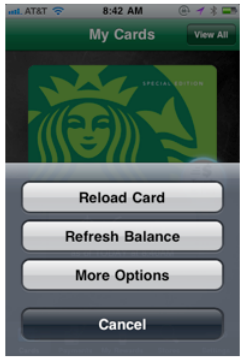
Screen Flows



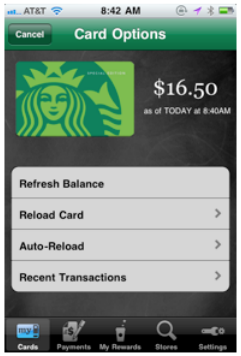
Pin Screen

My Cards - Main

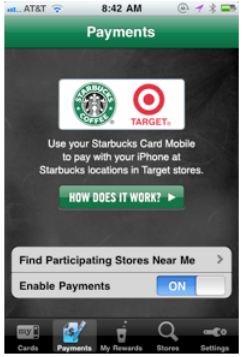
My Cards - Pay



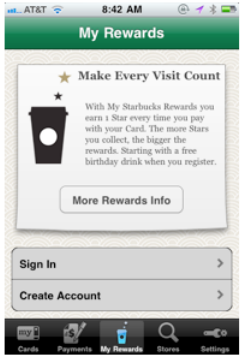
My Cards - Options



My Cards
More Options



Payment Setup



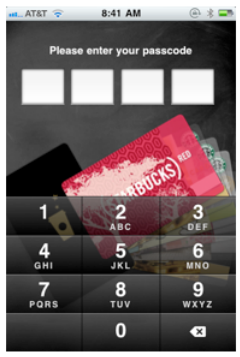
Rewards Setup

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.

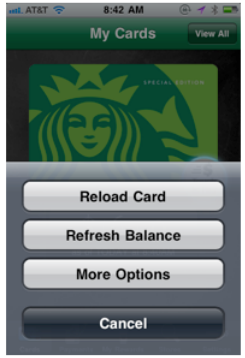
Display Requirements



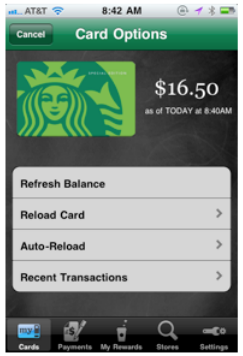
Pin Screen

My Cards - Main

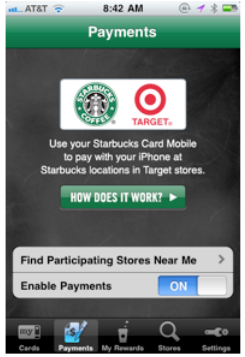
My Cards - Pay



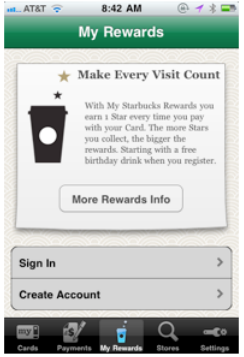
My Cards - Options



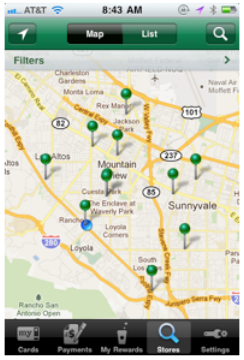
My Cards
More Options



Payment Setup



Rewards Setup



Find Starbucks

Implement the following Display Output for each screen:

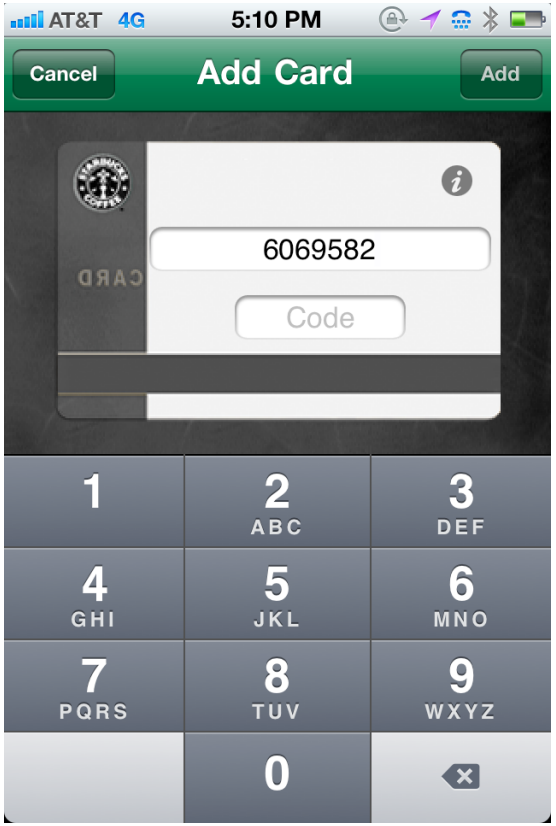
1. Calling "display()" on the "**My Cards – Main**" Screen should **show the balance of the current active card**.
2. Calling "display()" on the "**My Cards – Pay**" Screen should print-out the words "**Scan Now**"
3. Calling "display()" on the "**My Cards – Options**"
   Screen should print-out the 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**"

Add Card Screen

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) **or** (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.

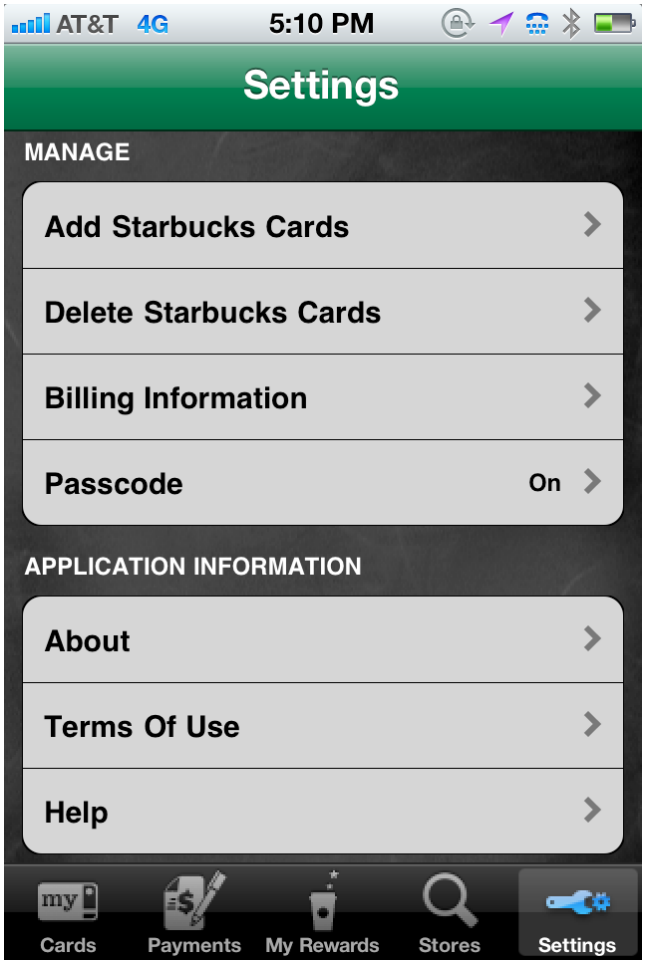
The "***Top Right Cmd***" on the Add Card Screen maps to "***Submit the Add New Card Request***" with the digits entered and the "***Top Left Cmd***" maps to "***Cancel Card Entry***" and return to the previous screen.



Settings Screen

To add a card, a user would touch the "***Settings***" menu and then the "***Add Starbucks Cards***" option. This option has the following touch coordinates: (1,2), (2,2) and (3,2). 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.

Making Payments

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), or touch (3,2)

Unit Testing

Create Unit Tests for:

1. ***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. 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.

Patterns

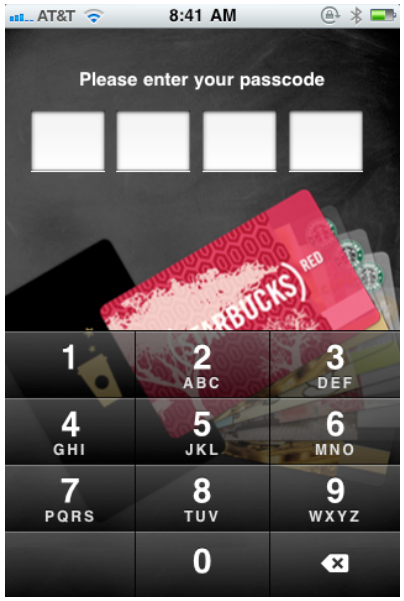
1. In addition to the "touch(x,y)" and "display()" methods, the ***App Controller*** should also implement the following methods for the menu ribbon: menu1(), menu2(), menu3(), menu4() and menu5(). Using the ***Command Pattern***, wire these menus to switch the current screen to: My Cards - Main, Payments, 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.

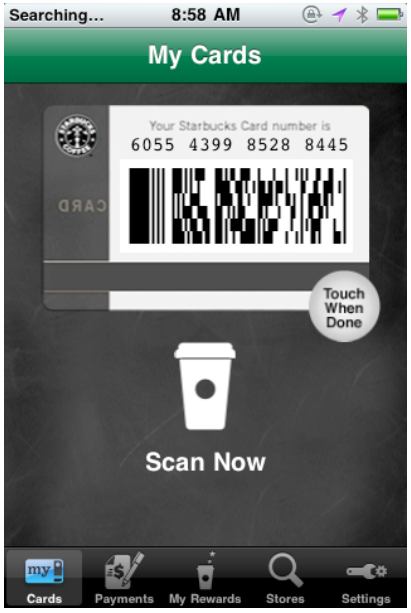
4. Implement a **Chain of Responsibility** for Widgets composed within the ***Pin Screen*** and ***Add Card Screen***. The objects within the chain should determine if the "touch(x,y)" event is interesting to it and may have to ***Adapt (i.e. Adapter Pattern)*** the event to the target widget for processing. 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.

“Pay on the Go” Use Case



Pin Screen

My Cards - Main



My Cards - Pay

1. Pin Screen
2. Main Screen
3. Pay Screen

Use Case Name: Pay on the Go

Brief Description:

Customer wishes to use the Starbucks App to pay for a drink.

Actors: Customer

Basic Flow:

1. Customer Starts-Up the Apps and the App challenges the Customer to enter a Pin.
2. Customer enters the Pin which the App validates successfully and then shows the Customer the last card used, the current balance on the card and the option to pay with the card.
3. The Customer selects the option to pay and the App displays a bar-code for the POS scanner.
4. The Customer presents the bar-code to the scanner and completes the payment transaction.
5. The Customer then checks the remaining balance and closes the App.

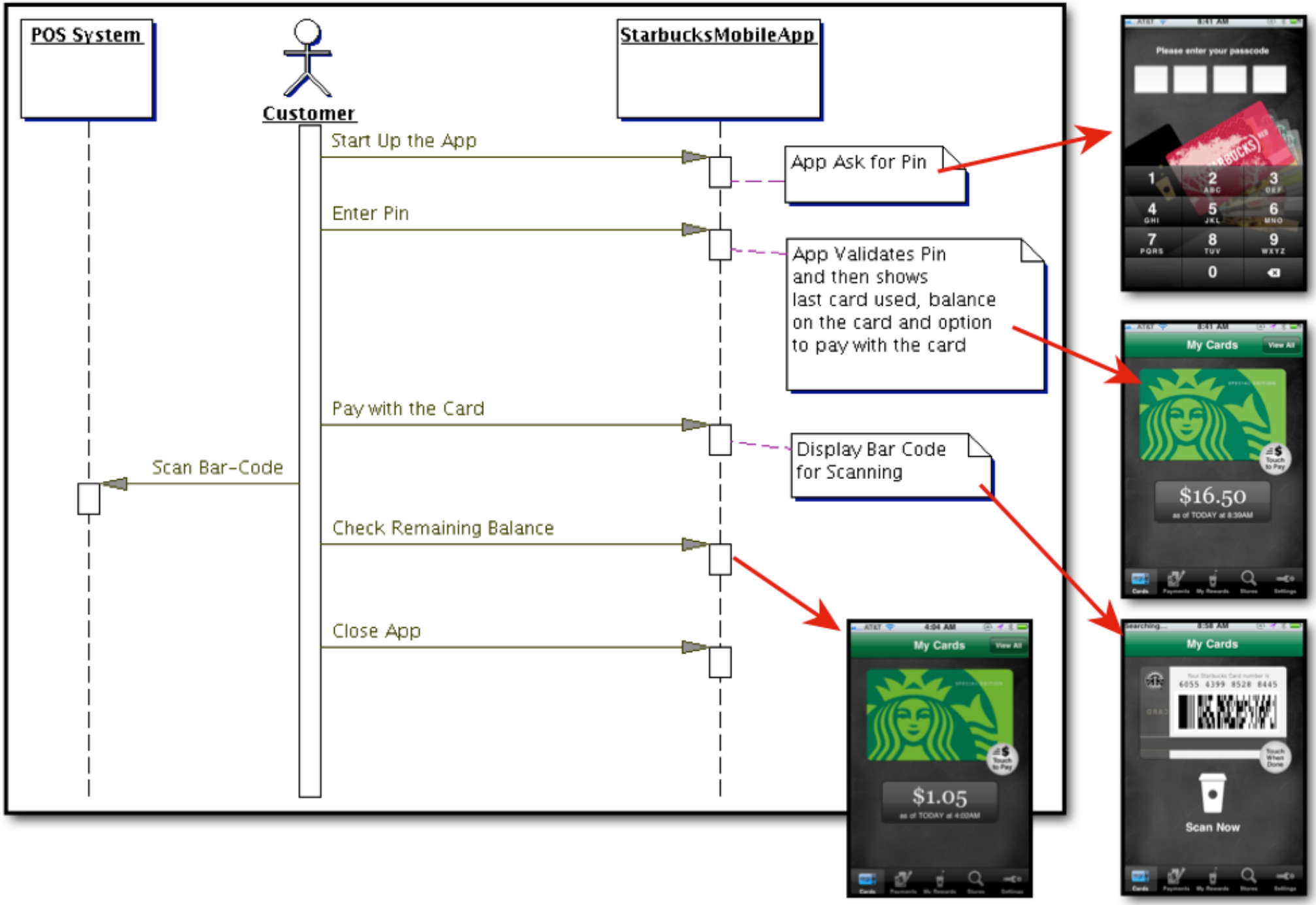
Alternate Flow:

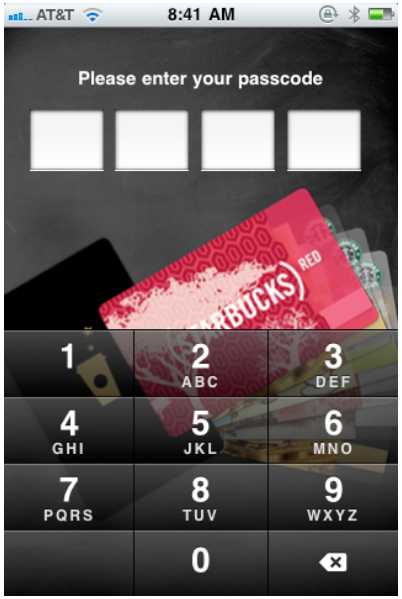
2a. The Customer realizes that there is not enough credit on the card and decides to chose another card or reload the current card.

Preconditions: Customer as a Card setup with the App and a Pin configured.

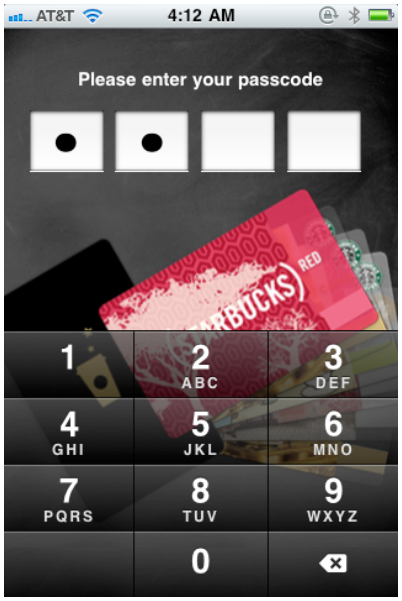
Success Guarantee: An electronic receipt for the transaction is created and made available to view on the App.

Minimal Guarantee: If the transaction failed, no credits on the card will be deducted and the balance on the Card will remain the same.


POS System
Customer
StarbucksMobileApp
Start Up the App
App Ask for Pin
Enter Pin
App Validates Pin
and then shows
last card used, balance
on the card and option
to pay with the card
Pay with the Card
Display Bar Code
for Scanning
Scan Bar-Code
Check Remaining Balance
Close App
Please enter your passcode
My Cards
$16.50
$1.05
Scan Now



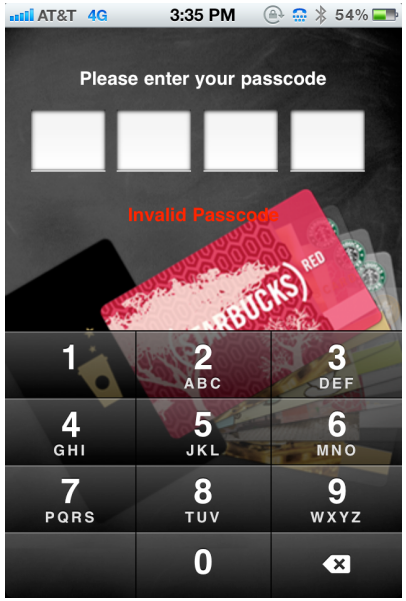
Pin Screen



For the "Pin Screen", design a composition with:

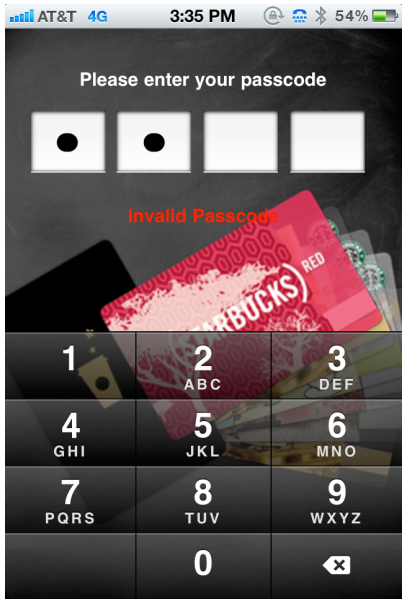
Key Pad
Passcode Display

Passcode Display should echo how many pin values were entered. If the "backspace" key was pressed, one pin value is removed.



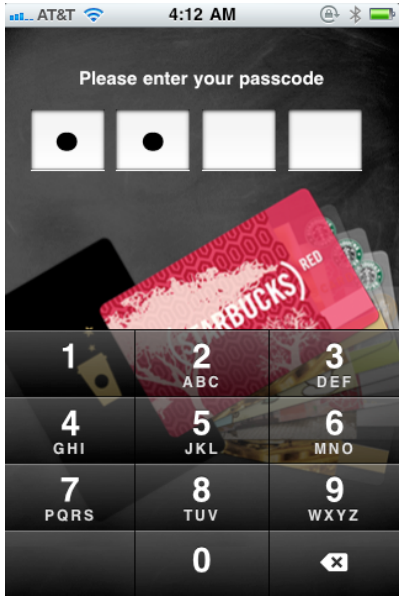
Pin Screen
(with error message)

If an invalid pin was entered, the pin authentication starts again.



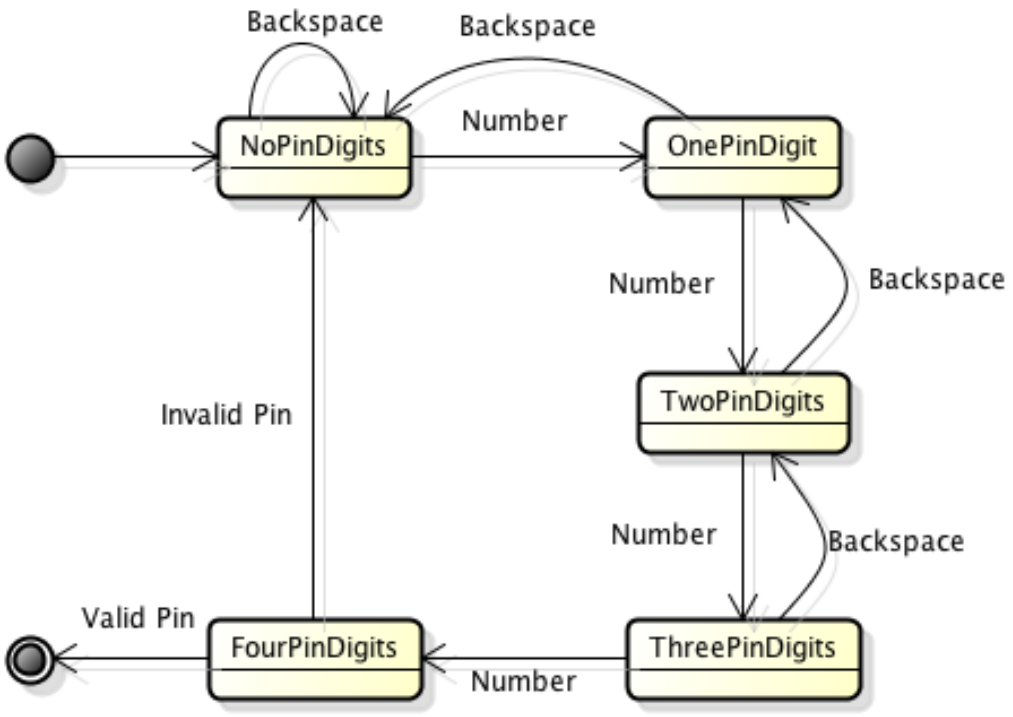
Pin Screen

User has to enter another pin (i.e. try again)

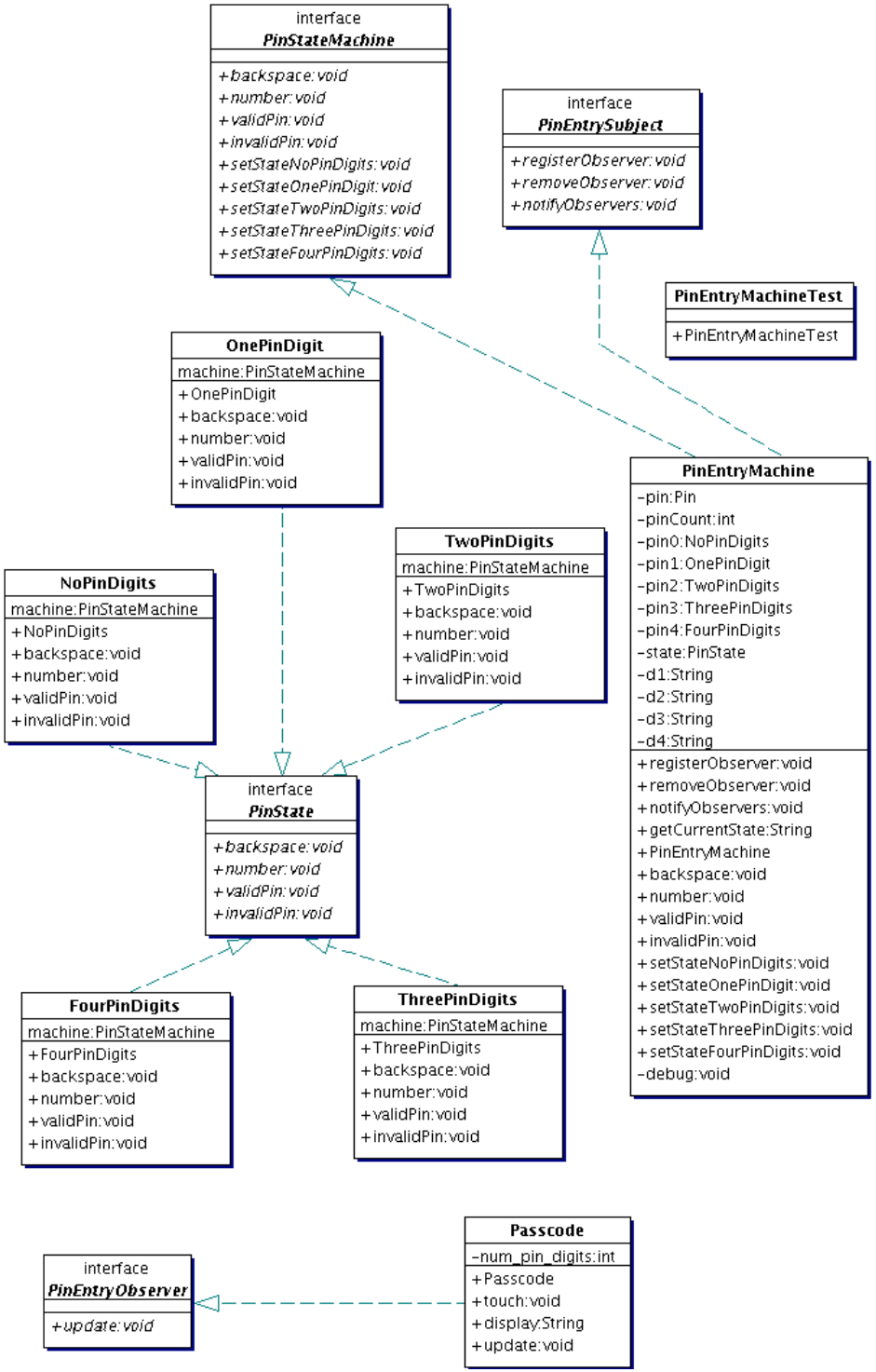


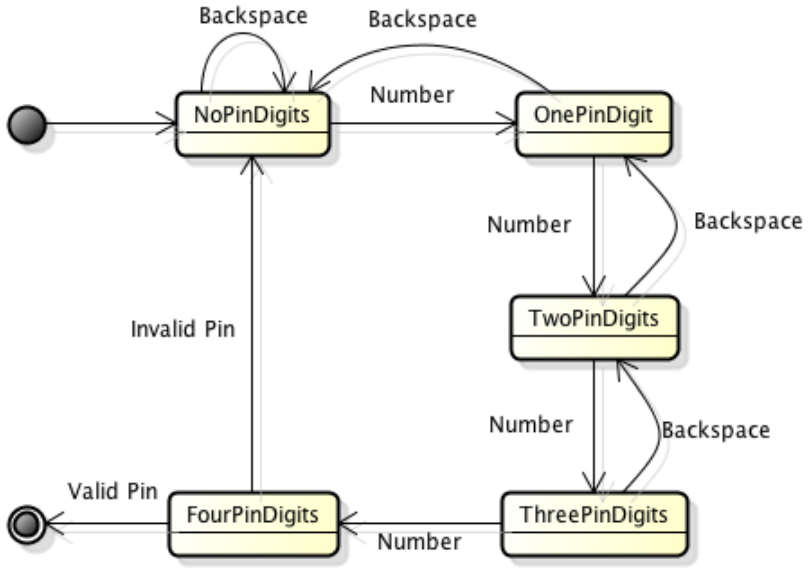
Pin Screen

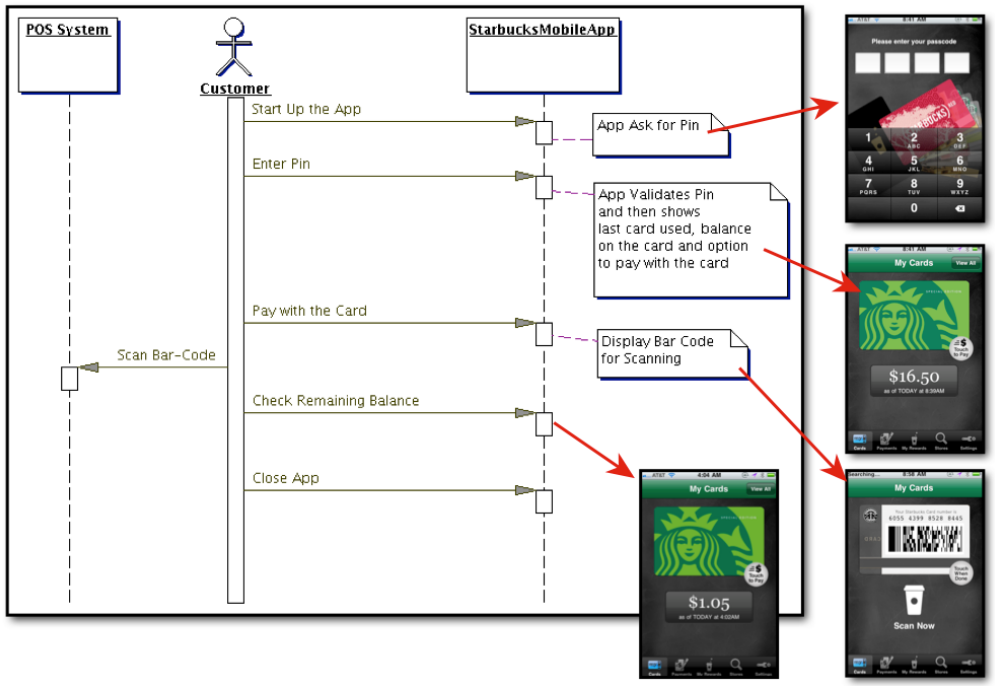
Model the Pin Screen Behavior as a "State Machine".



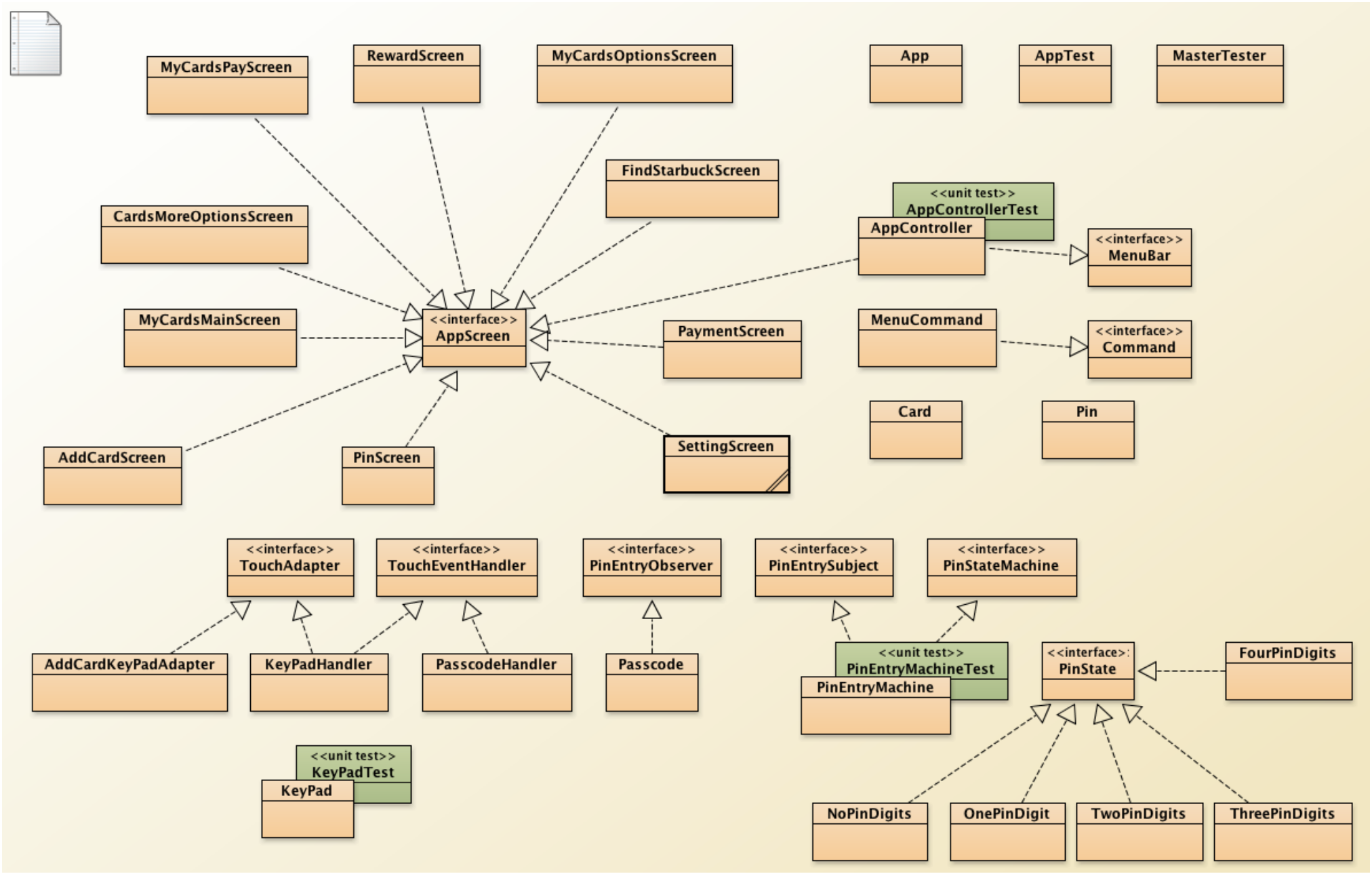
Use a default PIN of "1234" for testing.

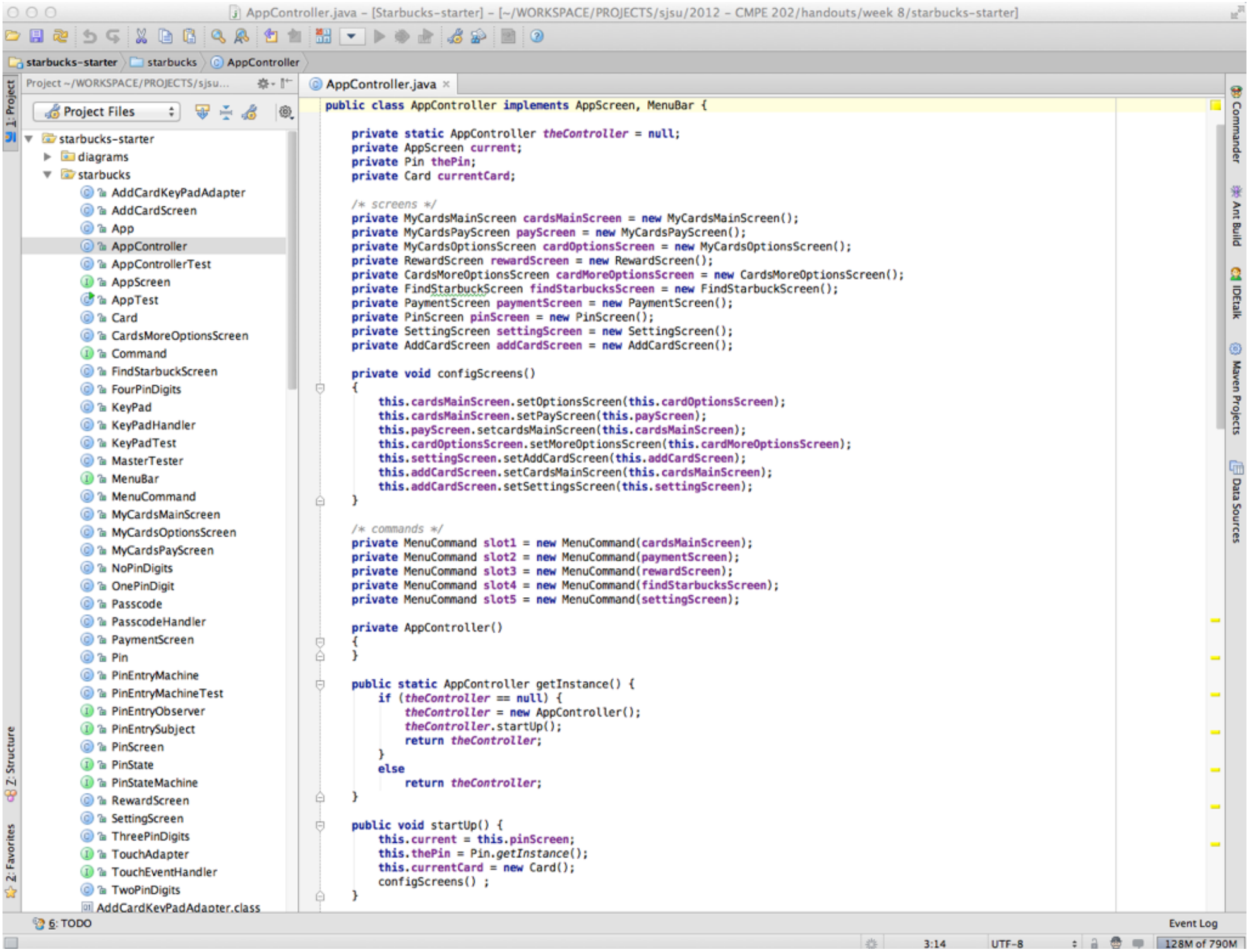

interface
PinStateMachine
+backspace:void
+number:void
+validPin:void
+invalidPin:void
+setStateNoPinDigits:void
+setStateOnePinDigit:void
+setStateTwoPinDigits:void
+setStateThreePinDigits:void
+setStateFourPinDigits:void
interface
PinEntrySubject
+registerObserver:void
+removeObserver:void
+notifyObservers:void
PinEntryMachineTest
+PinEntryMachineTest
OnePinDigit
machine:PinStateMachine
+OnePinDigit
+backspace:void
+number:void
+validPin:void
+invalidPin:void
PinEntryMachine
-pin:Pin
-pinCount:int
-pin0:NoPinDigits
-pin1:OnePinDigit
-pin2:TwoPinDigits
-pin3:ThreePinDigits
-pin4:FourPinDigits
-state:PinState
-d1:String
-d2:String
-d3:String
-d4:String
+registerObserver:void
+removeObserver:void
+notifyObservers:void
+getCurrentState:String
+PinEntryMachine
+backspace:void
+number:void
+validPin:void
+invalidPin:void
+setStateNoPinDigits:void
+setStateOnePinDigit:void
+setStateTwoPinDigits:void
+setStateThreePinDigits:void
+setStateFourPinDigits:void
-debug:void
TwoPinDigits
machine:PinStateMachine
+TwoPinDigits
+backspace:void
+number:void
+validPin:void
+invalidPin:void
NoPinDigits
machine:PinStateMachine
+NoPinDigits
+backspace:void
+number:void
+validPin:void
+invalidPin:void
interface
PinState
+backspace:void
+number:void
+validPin:void
+invalidPin:void
FourPinDigits
machine:PinStateMachine
+FourPinDigits
+backspace:void
+number:void
+validPin:void
+invalidPin:void
ThreePinDigits
machine:PinStateMachine
+ThreePinDigits
+backspace:void
+number:void
+validPin:void
+invalidPin:void
interface
PinEntryObserver
+update:void
Passcode
-num_pin_digits:int
+Passcode
+touch:void
+display:String
+update:void


Backspace
Backspace
Number
NoPinDigits
OnePinDigit
Number
Backspace
Invalid Pin
TwoPinDigits
Number
Backspace
Valid Pin
FourPinDigits
ThreePinDigits
Number

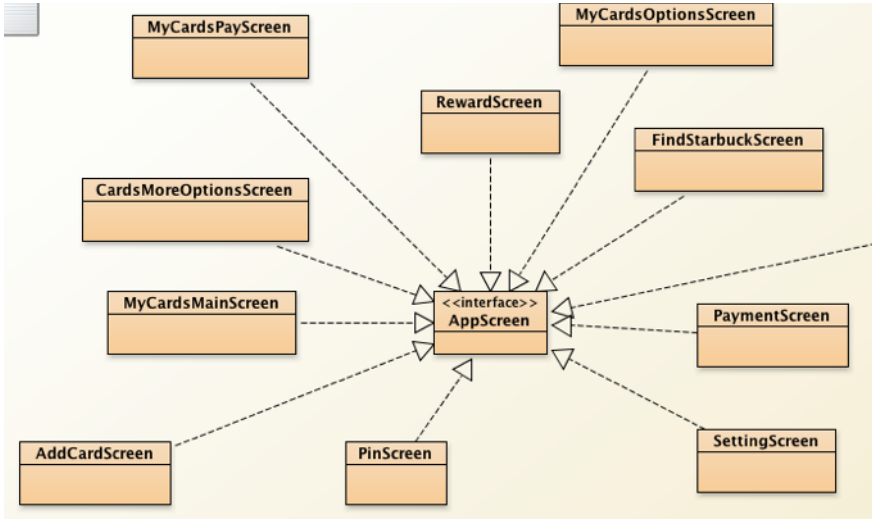

POS System
Customer
StarbucksMobileApp
Start Up the App
App Ask for Pin
Enter Pin
App Validates Pin
and then shows
last card used, balance
on the card and option
to pay with the card
Pay with the Card
Display Bar Code
for Scanning
Scan Bar-Code
Check Remaining Balance
Close App
$16.50
$1.05
Scan Now

Source Code Example


MyCardsPayScreen
RewardScreen
MyCardsOptionsScreen
App
AppTest
MasterTester
FindStarbuckScreen
<<unit test>>
AppControllerTest
CardsMoreOptionsScreen
AppController
<<interface>>
MenuBar
MyCardsMainScreen
<<interface>>
AppScreen
PaymentScreen
MenuCommand
<<interface>>
Command
Card
Pin
AddCardScreen
PinScreen
SettingScreen
<<interface>>
TouchAdapter
<<interface>>
TouchEventHandler
<<interface>>
PinEntryObserver
<<interface>>
PinEntrySubject
<<interface>>
PinStateMachine
AddCardKeyPadAdapter
KeyPadHandler
PasscodeHandler
Passcode
<<unit test>>
PinEntryMachineTest
PinEntryMachine
<<interface>>
PinState
FourPinDigits
<<unit test>>
KeyPadTest
KeyPad
NoPinDigits
OnePinDigit
TwoPinDigits
ThreePinDigits


AppController.java – [Starbucks-starter] – [~/WORKSPACE/PROJECTS/sjsu/2012 – CMPE 202/handouts/week 8/starbucks-starter]
starbucks-starter
starbucks
AppController
Project ~/WORKSPACE/PROJECTS/sjsu...
Project Files
starbucks-starter
diagrams
starbucks
AddCardKeyPadAdapter
AddCardScreen
App
AppController
AppControllerTest
AppScreen
AppTest
Card
CardsMoreOptionsScreen
Command
FindStarbuckScreen
FourPinDigits
KeyPad
KeyPadHandler
KeyPadTest
MasterTester
MenuBar
MenuCommand
MyCardsMainScreen
MyCardsOptionsScreen
MyCardsPayScreen
NoPinDigits
OnePinDigit
Passcode
PasscodeHandler
PaymentScreen
Pin
PinEntryMachine
PinEntryMachineTest
PinEntryObserver
PinEntrySubject
PinScreen
PinState
PinStateMachine
RewardScreen
SettingScreen
ThreePinDigits
TouchAdapter
TouchEventHandler
TwoPinDigits
AddCardKeyPadAdapter.class
AppController.java
public class AppController implements AppScreen, MenuBar {
private static AppController theController = null;
private AppScreen current;
private Pin thePin;
private Card currentCard;
/* screens */
private MyCardsMainScreen cardsMainScreen = new MyCardsMainScreen();
private MyCardsPayScreen payScreen = new MyCardsPayScreen();
private MyCardsOptionsScreen cardOptionsScreen = new MyCardsOptionsScreen();
private RewardScreen rewardScreen = new RewardScreen();
private CardsMoreOptionsScreen cardMoreOptionsScreen = new CardsMoreOptionsScreen();
private FindStarbuckScreen findStarbucksScreen = new FindStarbuckScreen();
private PaymentScreen paymentScreen = new PaymentScreen();
private PinScreen pinScreen = new PinScreen();
private SettingScreen settingScreen = new SettingScreen();
private AddCardScreen addCardScreen = new AddCardScreen();
private void configScreens()
{
this.cardsMainScreen.setOptionsScreen(this.cardOptionsScreen);
this.cardsMainScreen.setPayScreen(this.payScreen);
this.payScreen.setcardsMainScreen(this.cardsMainScreen);
this.cardOptionsScreen.setMoreOptionsScreen(this.cardMoreOptionsScreen);
this.settingScreen.setAddCardScreen(this.addCardScreen);
this.addCardScreen.setCardsMainScreen(this.cardsMainScreen);
this.addCardScreen.setSettingsScreen(this.settingScreen);
}
/* commands */
private MenuCommand slot1 = new MenuCommand(cardsMainScreen);
private MenuCommand slot2 = new MenuCommand(paymentScreen);
private MenuCommand slot3 = new MenuCommand(rewardScreen);
private MenuCommand slot4 = new MenuCommand(findStarbucksScreen);
private MenuCommand slot5 = new MenuCommand(settingScreen);
private AppController()
{
}
public static AppController getInstance() {
if (theController == null) {
theController = new AppController();
theController.startUp();
return theController;
}
else
return theController;
}
public void startUp() {
this.current = this.pinScreen;
this.thePin = Pin.getInstance();
this.currentCard = new Card();
configScreens() ;
}
Commander
Ant Build
IDEtalk
Maven Projects
Data Sources
1: Project
2: Favorites
Z: Structure
6: TODO
Event Log
3:14
UTF-8
128M of 790M

AppScreen Interface

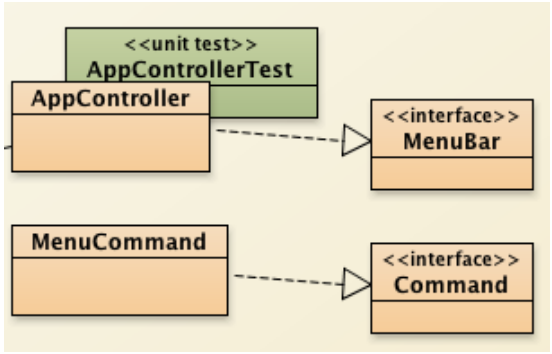



```
public interface AppScreen {

    public void touch(int x, int y);
    public String display() ;
    public void topLeftCmd() ;
    public void topRightCmd() ;
}
```

```
public class RewardScreen implements AppScreen {

    public void touch(int x, int y) {

    }

    public String display()
    {
        String message = "Make Every Visit Count"
        System.out.println( message ) ;
        return message ;
    }

    public void topLeftCmd() {
    }

    public void topRightCmd() {
    }

}
```

```
public class PaymentScreen implements AppScreen {

    public void touch(int x, int y) {

    }

    public String display()
    {
        String message = "Enable Payments?" ;
        System.out.println( message ) ;
        return message ;
    }

    public void topLeftCmd() {
    }

    public void topRightCmd() {
    }

}
```

AppController & Menu Commands



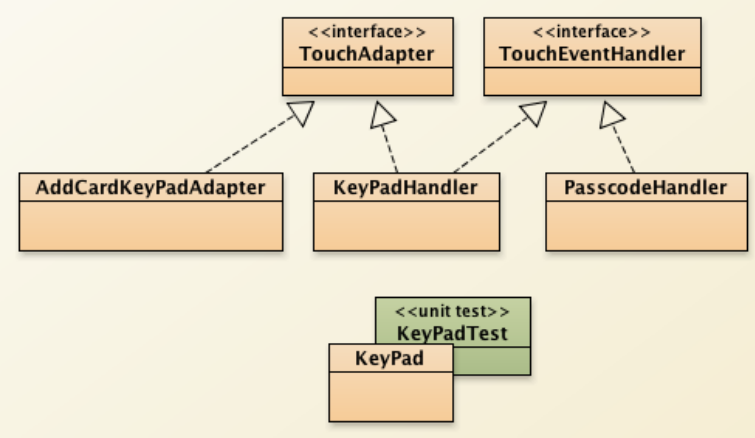

```
public interface MenuBar
{

    public void menu1() ;
    public void menu2() ;
    public void menu3() ;
    public void menu4() ;
    public void menu5() ;

}
```

```
public interface Command
{

    public void execute() ;
}
```

```
public class AppController implements AppScreen, MenuBar {

    private static AppController theController = null;
    private AppScreen current;
    private Pin thePin;
    private Card currentCard;

    /* screens */
    private MyCardsMainScreen cardsMainScreen = new MyCardsMainScreen();
    private MyCardsPayScreen payScreen = new MyCardsPayScreen();
    private MyCardsOptionsScreen cardOptionsScreen = new MyCardsOptionsScreen();
    private RewardScreen rewardScreen = new RewardScreen();
    private CardsMoreOptionsScreen cardMoreOptionsScreen = new CardsMoreOptionsScreen();
    private FindStarbuckScreen findStarbucksScreen = new FindStarbuckScreen();
    private PaymentScreen paymentScreen = new PaymentScreen();
    private PinScreen pinScreen = new PinScreen();
    private SettingScreen settingScreen = new SettingScreen();
    private AddCardScreen addCardScreen = new AddCardScreen();

    private void configScreens()
    {
        this.cardsMainScreen.setOptionsScreen(this.cardOptionsScreen);
        this.cardsMainScreen.setPayScreen(this.payScreen);
        this.payScreen.setcardsMainScreen(this.cardsMainScreen);
        this.cardOptionsScreen.setMoreOptionsScreen(this.cardMoreOptionsScreen);
        this.settingScreen.setAddCardScreen(this.addCardScreen);
        this.addCardScreen.setCardsMainScreen(this.cardsMainScreen);
        this.addCardScreen.setSettingsScreen(this.settingScreen);
    }

    /* commands */
    private MenuCommand slot1 = new MenuCommand(cardsMainScreen);
    private MenuCommand slot2 = new MenuCommand(paymentScreen);
    private MenuCommand slot3 = new MenuCommand(rewardScreen);
    private MenuCommand slot4 = new MenuCommand(findStarbucksScreen);
    private MenuCommand slot5 = new MenuCommand(settingScreen);
```

```
public class MenuCommand implements Command
{
    private AppScreen screen ;

    public MenuCommand( AppScreen scr )
    {
        this.screen = scr ;
    }

    public void execute() {

        AppController app = AppController.getInstance() ;
        app.setScreen( screen ) ;

    }
}
```

Touch Interface & Adapters

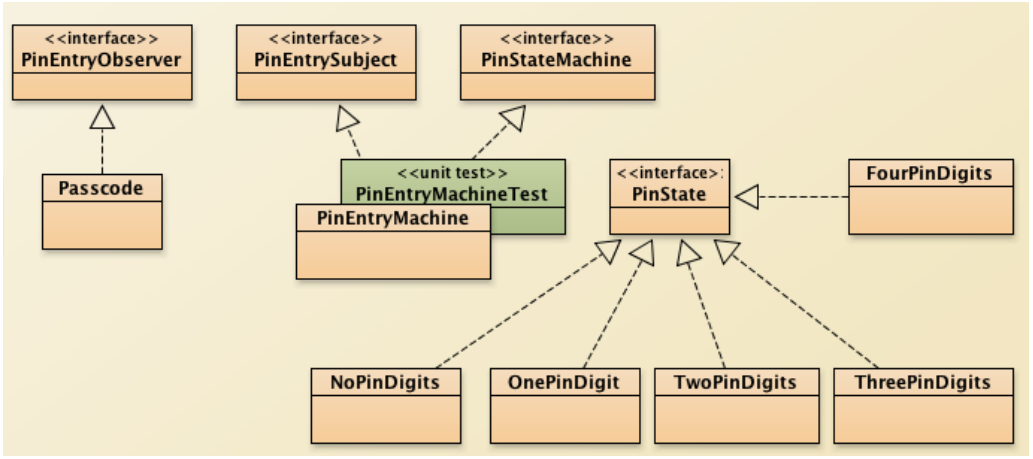



```
/*

Note: Alternative Implementation using an Abstract Class

public abstract class TouchEventHandler
{
    private TouchEventHandler next ;
    abstract void touch(int x, int y) ;
}

*/

public interface TouchEventHandler
{
    void touch(int x, int y) ;
    void setNext( TouchEventHandler next ) ;
}
```

```
public interface TouchAdapter
{
    void touch(int x, int y) ;
}
```

```
public class KeyPadHandler implements TouchEventHandler,TouchAdapter
{
    private TouchEventHandler nextHandler ;
    private KeyPad keypad ;
    private PinStateMachine pinStateMachine ;

    public KeyPadHandler( PinStateMachine pm )
    {
        this.keypad = new KeyPad() ;
        this.nextHandler = null ;
        this.pinStateMachine = pm ;
    }

    public void setNext( TouchEventHandler next )
    {
        this.nextHandler = next ;
    }

    public void touch(int x, int y)
    {
        if ( y > 4 )
        {
            String key ;
            int kx, ky ;
            kx = x ; ky = y-4 ;
            key = this.keypad.keyPress(kx, ky) ;
            System.out.println( "Key => " + key ) ;
            if ( key.equals(" ") )
                /* nothing */ ;
            else if ( key.equals("X") )
                pinStateMachine.backspace() ;
            else
                pinStateMachine.number( key ) ;
        }
        else if ( this.nextHandler != null )
            this.nextHandler.touch( x, y );
    }

}
```

Pin Observer & State Machine




```java
public interface PinEntrySubject
{
    void registerObserver( PinEntryObserver obj ) ;
    void removeObserver( PinEntryObserver obj ) ;
    void notifyObservers( ) ;

}
```

```java
public interface PinEntryObserver
{
    void update(int count) ;
}
```

```java
public class PinEntryMachine implements PinStateMachine, PinEntrySubject
{
    // Observers
    private ArrayList<PinEntryObserver> observers ;

    public void registerObserver( PinEntryObserver obj )
    {
        observers.add( obj ) ;
    }

    public void removeObserver( PinEntryObserver obj )
    {
        int i = observers.indexOf(obj) ;
        if ( i >= 0 )
            observers.remove(i) ;
    }

    public void notifyObservers( )
    {
        for (int i=0; i<observers.size(); i++)
        {
            PinEntryObserver observer = observers.get(i) ;
            observer.update( this.pinCount ) ;
        }
    }
```

```java
public class Passcode implements PinEntryObserver {

    private int num_pin_digits ;

    public Passcode()
    {
        this.num_pin_digits = 0 ;
    }

    public void touch(int x, int y) {
    }

    public String display()
    {
        String str ;
        switch( num_pin_digits )
        {
            case 0: str = "_ _ _ _" ; break ;
            case 1: str = "* _ _ _" ; break ;
            case 2: str = "* * _ _" ; break ;
            case 3: str = "* * * _" ; break ;
            case 4: str = "* * * *" ; break ;
            default: str = "_ _ _ _" ;
        }
        System.out.println( str ) ;
        return str ;
    }

    public void update(int count)
    {
        this.num_pin_digits = count ;
        display() ;
    }
}
```