University of Texas at Austin

Mobile Application Development using OOP Techniques

CS 370 - Fall 2010

Author:
Amaan Penangwala

Supervisor:
Glen Downing

December 5, 2010

Contents

1 Introduction

The explosive growth of smartphones over the course of the last few years has given rise to a new platform for users to consume the content available on the Internet. Although the Internet available on these mobile devices is virtually the same as the one they access from their desktop machines at home, the users experience is quite different. Mobile phones grasp the undivided attention of its users, which means that users are less tolerant to app crashes. This means that user experience, which was previously a secondary goal for consumer applications, is now the center of attention. Developers can no longer create apps that simply solve a user's problem, they must make their application intuitive to use by using a combination of creativity and a deep understanding of the platform he is developing for. Fortunately, there are many existing frameworks available that enable the developer to focus on the core of the problem and use the abstractions provided by these frameworks to interact with the user. In this research project, I plan to analyze for an audience of first time developers of mobile applications the minimum knowledge necessary to create a mobile application and the set of challenges that must be overcome by all developers before harnessing the true potential of todays smartphones.

In order to demonstrate the principles behind building a simple application, we narrowed our focus to Apple's mobile iOS, and build a Loan Calculator that would empower its user with the knowledge necessary to make a purchase on credit. More specifically this application allows a user to input the requirements of a Loan and see in detail what portion of their money is spent on paying interest vs. paying off the principal. The nave approach to solve this problem would simply guide the user to a one of the many loan calculators on the web. But accessing web applications on mobile phones is almost always an inferior experience than to use an application that runs natively from the phone. The calculation behind this application is fairly straightforward for anyone with some knowledge about compound interest. The challenge here is to create an intuitive interface that allows the user to discover pertinent information regarding their financial transactions. The key components of the application are the calculator engine that computes the missing information for new transactions and the persistence of these transactions for later retrieval. The application does not take into account any tax considerations or other flavors of a loan such as a variable interest rate. Additionally, the application data cannot be extracted or imported by other applications.

2 Related Work

Other alternatives to gain familiarity with the development of mobile applications include obtaining one of the many books available in the market on the subject or to read the official developer documentation of the platform available online. Additionally, there are many other resources online that are freely available that focus on the educating new developers on the subject. Our approach was to use a combination of those resources in addition to learning from the experience of building the loan calculator.

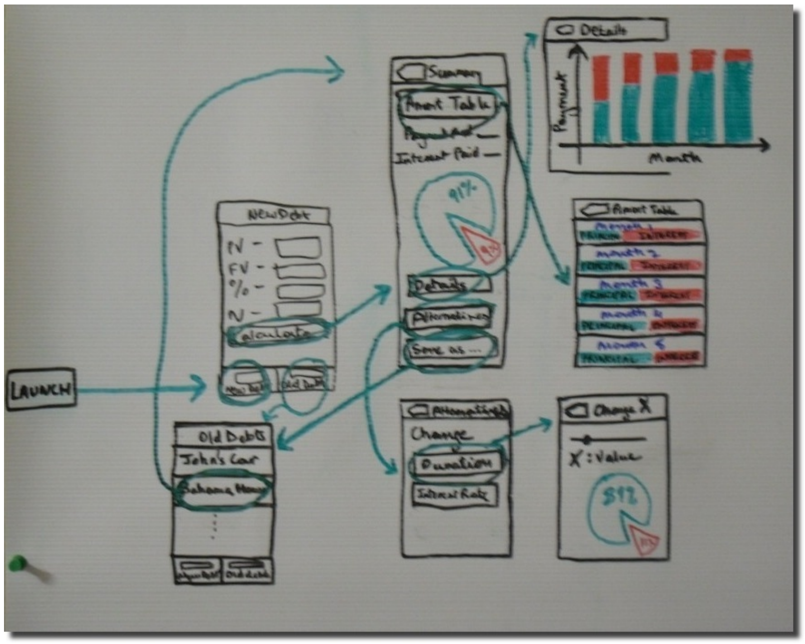

Figure 1: UML Diagrams of Models used in the application.

3 Design

The loan calculator has two main functions i.e., to produce an Amortization Schedule for a loan given its Present Value, Future Value, Yearly Rate of Interest and the Duration and allow the user to save a particular combination of these for later retrieval. An example of a web application that provides the same functionality but without the ability to save transactions can be found at http://amort.appspot.com. The design patterns used in architecting the application are the Model-View-Controller (MVC) Pattern to separate the systems responsibilities among manageable subsystems. Apples framework also makes extensive use of the Delegation Pattern in order to notify different objects when the Event handler gets new information.

The flowchart in Figure 1. loosely depicts the different Views encountered by a user New Debt View (displayed upon launch) - Gathers information about a loan instance Summary View - Displays the payment amount and total interest paid. Allows the user to save this loan for later retrieval Details View - Shows each payment broken down as money towards principal and money towards interest Amortization Table View - Shows the Amortization Schedule for each payment. Each cell is colored according the proportion of the payment towards principal and towards interest. Alternatives View - Specifies which variable can change Duration or Interest Rate. Change X View - Allows the user to change the specified variable and instantaneous view its effect on the total payment breakdown Old Debt View - Lets the user review previously saved loans and load it up in Summary View.

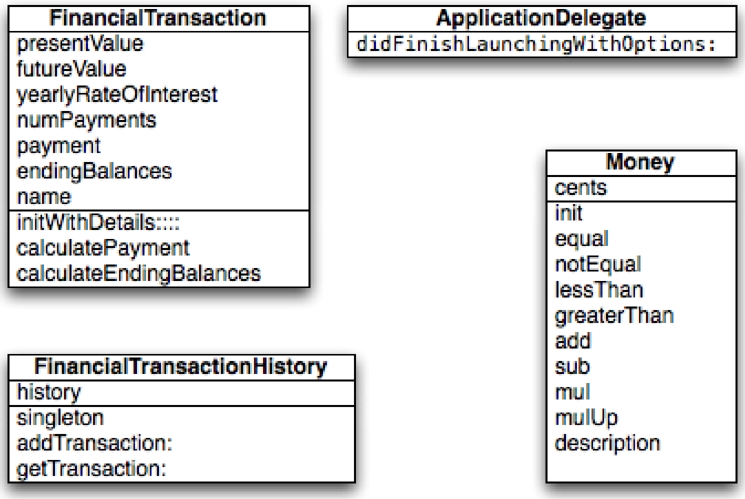

Figure 2: UML Class Diagram of the Models of the application

The Models used in the application are ApplicationDelegate, Money, FinancialTransaction and FinancialTransactionHistory.

ApplicationDelegate - the class that is responsible for responding to event notifications sent by the operating system such as: the user has launched the application, the user will quit the application, warnings about memory leaks, etc. This object is the glue that holds all other objects together. It is created by default when starting a project in Xcode. Money - is an enhancement within the application to represent money. Money allows the calculations pertaining to money within the app to be more accurate. The use of floating point to represent money would cause subtle inaccuracies due to how floating-point arithmetic works. This class was added later after noticing inconsistencies between the expected results and actual results. FinancialTransaction - used to create instances of a loan and responsible for computing the missing details that are not provided by the user. FinancialTransactionHistory - singleton in the scope of the application. A collection of FinancialTransaction that a user has created or modified within a session.

Apart from the basic functionality described above, the users are also presented with the ability to modify parameters of a loan and simultaneously view its effect on the graphs. No other special features were added.

4 Implementation

The first implementation of the calculator, which lacked the Money object, failed to produce accurate balances at the end of the month because of inconsistency in floating point arithmetic, which propagated a noticeable difference in the final payment. The alternative was to use integer arithmetic and represent the dollar values as cents. The current implementation of Money does exactly that in an encapsulated form. The rest of the code use both the frameworks provided by Apple to handle the presentation of the Views and capturing user input to update the data internally. The Foundations Framework provides an interface to the underlying OS and contains all the data structures used within the app. The Cocoa Touch Framework is contains classes dealing with creating the Graphical User Interface (GUI) and drawing it on the screen. The GUI can either be created programmatically in Xcode or visually in Interface Builder. The views described earlier were initially created in Interface Builder, but later customized from within Xcode.

Some of other tools used to manage this project were Mercurial and Fogbugz/Kiln integrated solution. Mercurial is a distributed version control system that allows programmers to commit incremental changes throughout the code in a local repository and then push those changes to a central repository if those changes need to be shared with other developers. Kiln is a web application that lets users maintain a central repository and allows users to review each others code from within a web browser. Fogbugz integrates with Kiln and allows the users to maintain a bug tracker and offers other tools that help in planning for a large project.

5 Evaluation

Testing in the earlier stages of building the application was done entirely using the iPhone Simulator by creating comprehensive test cases that can be loaded from within the application. Since the tests were not automated the results were confirmed by logging the output to the console of the Virtual iPhone. This entire testing process was highly inefficient and can certainly be avoided by using Unit Testing frameworks that essentially provide the same assurance to the developer. The application was also tested using Instruments, which is a profiler included in Apples Development Kit. It played a critical role in identifying the source memory leaks that were causing the application to crash inexplicably.

6 Conclusion

The current version of the application meets the basic requirements listed earlier. After having gone through multiple rounds of refactoring, the readability of the code is certainly at an all time high. The calculations for all payments happen on the main thread without penalizing the GUIs responsiveness. There are some pending issues pertaining to drawing the pie charts and graphs as shown in the Summary and Detail Views. Since the application is modularized, it could be extended to include some the features necessary to make this application useable in real life settings. There are two features currently on the roadmap for this application; one is to allow users to extract loan data from the application and the other is the ability to add payment reminders for existing loans that users are in the process of paying off.

Throughout this project I learned the value of designing for code reuse, a principal which was previously only emphasized in my formal education but one whose benefits I had not seen in action. I had never been involved in a programming project in which a huge portion of my code depended on previously written code i.e., my own evolving version and the underlying frameworks that did quite a bit of heavy lifting in dealing with the user's interactions. Writing large programs in an environment where memory must managed meticulously also showed me the importance of dynamic program analysis, which offers a comprehensive view of the stack's allocation and other runtime aspects of the code. The overall experience was quite fulfilling because many of the principles that I had only seen in a classroom setting with made up examples were actually being applied throughout a framework that is used by a large number of people around the world.