Project: MP2
Course: ITM-411
Author: Brian T. Bailey

Project Description:

The object of this project was to create an application that read in the sample data file, bank-data.csv, and perform some basic operations and analysis on the data. The data file needs to be parsed and stored in a BankRecord object model we defined. The objects need to be sorted by region and stored in lists of BankRecords which in turn need to be stored in a Map object using the region as the Key.

A PersistentObject class needs to be defined that will hold the Map object and a timestamp. This PersistentObject then needs to get serialized to a file. After 5 seconds, that file needs to be deserialized back into the application. Once that is complete some basic data analytics need to be performed.

These analytics include the time difference between serialization and deserialization, average income per region, max and min age per region, number of females with a mortgage and savings account, number of males with a car and children. All output also needs to be displayed to the console and written to a text file.

Installation, Compile and Run-Time Requirements:

This project was written in Java using JDK version 1.6. The IDE used to write and compile the application was NetBeans 7.1 on the Macintosh platform. The computer used was a 2.7 GHz Dual Core Intel core i7 13" MacBook Pro with 4GB of RAM running OS X Lion 10.7.3.

The application folder can be copied to a folder on the system and opened in NetBeans. The project can then be build and run using the play button in netbeans. The application can also be run from the standard terminal. To do this the data directory needs to be copied to the dist directory if it is not already there. The application jar file can then be run with the command "java -jar mp2.jar" from within the dist directory. There is also a test directory located in the main project directory. In the test directory there are two files. One is called mp2.bat and is a windows batch file for running the application on Microsoft Windows. The other is called mp2.sh and is a UNIX shell script for running the application on Mac, Linux or Unix.

Insights and Expected Results:

This project did not give me any real difficulties. The data file in this application was much better formatted than in MP1. In MP1 the data file had problems, this data file was a properly formatted csv file.

Professor Kimont provided the readRecordData() method located in the BankRecordUtils class. This code reads the data file and parses it into Record objects and returns a List. He provided this code for everyone to use as a solid starting point. I used his code as is except for one minor change. In his code he is creating Record objects and a List<Record>. Since the project required us to create our Record class as an abstract class and have a subclass called BankRecord which we would instantiate, I needed

to change his code to create BankRecord objects and a List<BankRecord>. His code would not work as is because you can not instantiate a abstract class.

In working on this project, I changed some design patterns from previous project. In MP1 I broke out my code into static methods in the main driver class and called those methods from the main method. This did have the effect of reducing code in the main method to mostly method calls but created a lot of other methods in that class. In MP2 I created two new utility classes, BankRecordUtils and OutputUtils. I used these utility classes to define static methods I wanted to use for calculations and data output. This left me with a main driver class that was mostly method calls to other classes in the main method. This also allowed me to have a clean main driver class that didn't have other methods in it other than main.

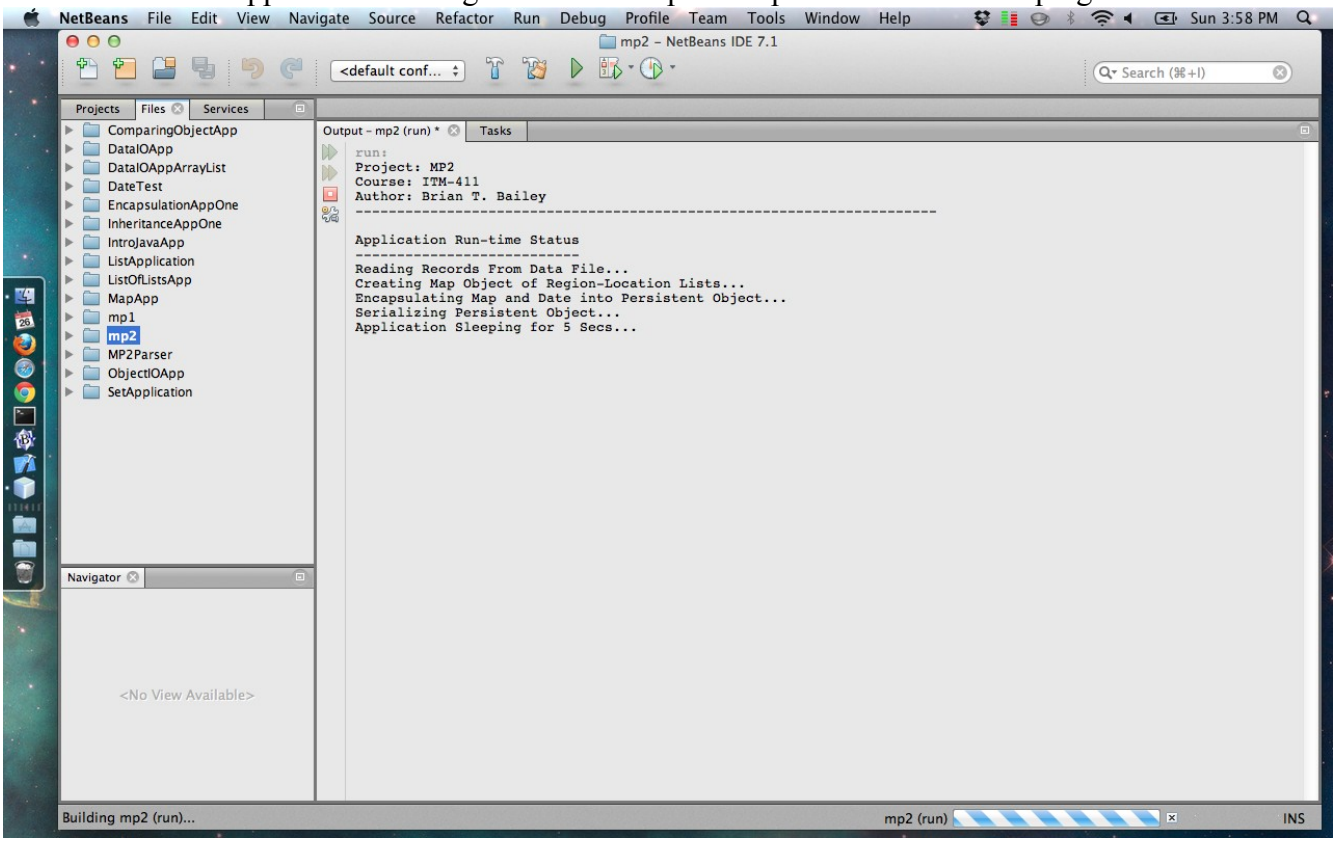
Serializing the PersistentObject which encapsulated the Map object and timestamp happened without any issue. After serializing the data, I programed the application to sleep for 5 seconds. When the program returned from sleeping, I deserialized the PersistentObject. I was able to pull the Map object out of the deserialized object and compare the timestamps. The time difference was a little over the 5 seconds which was expected because of the 5 seconds of sleep plus a little bit of processing time.

For the data analysis we were supposed to use Comparator classes. I created 4 classes to represent each of the data analysis questions. The min and max age by location analysis was the only one that I used the compare() method of the Comparator class. I needed to sort the lists by age and coded the AgeComparator's compare() method to do that. The other three analysis questions didn't need any sorting to produce their results. They were more calculation based. I did go ahead and make Comparator classes for each and implemented a compare() method that would sort the records but didn't call that method in the analysis. I also implemented some static methods in each of the Comparator classes to calculate the required data for that particular question. I then called those methods from the Comparator classes to get my results.

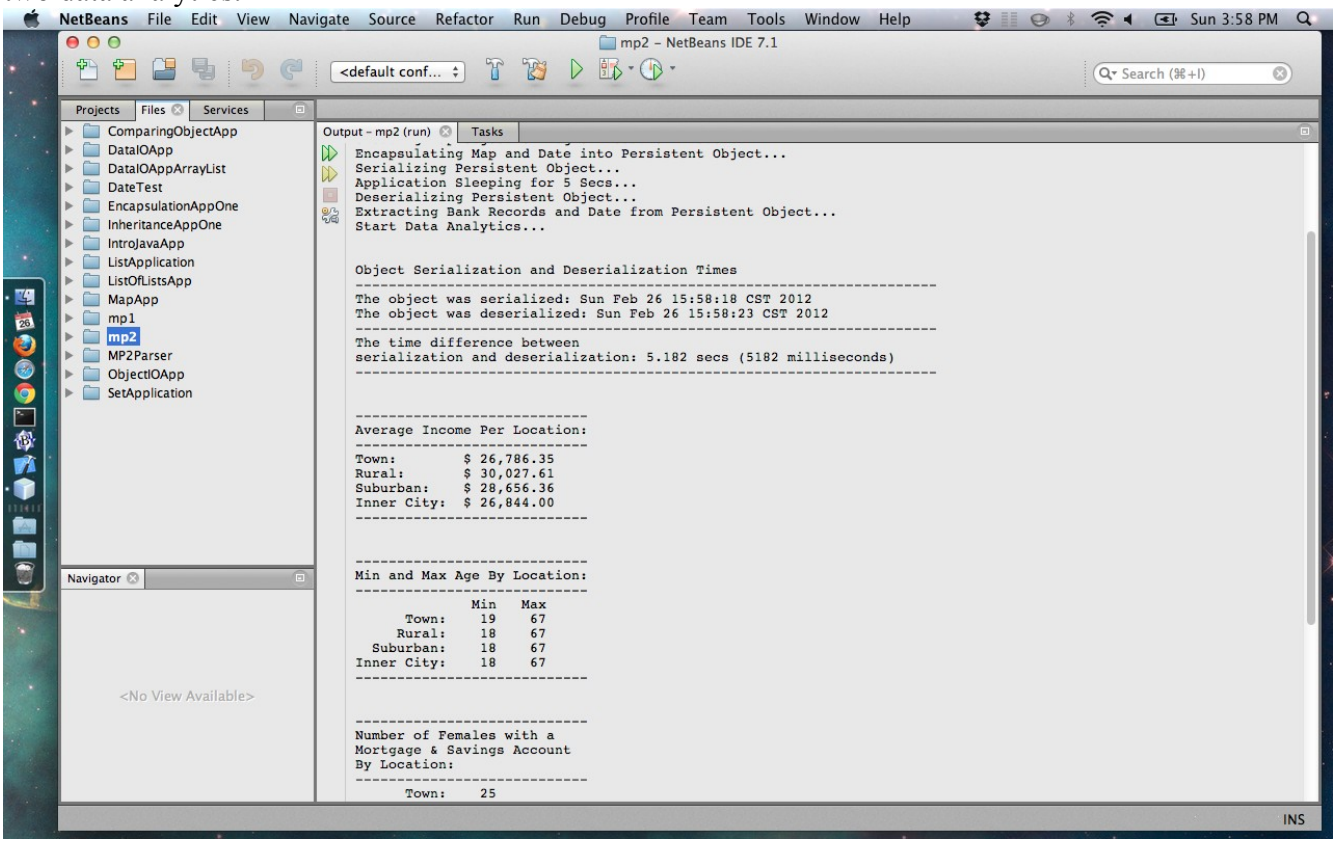
Screenshots Demonstrating Application:

Screenshots of the application running appear on the following pages.

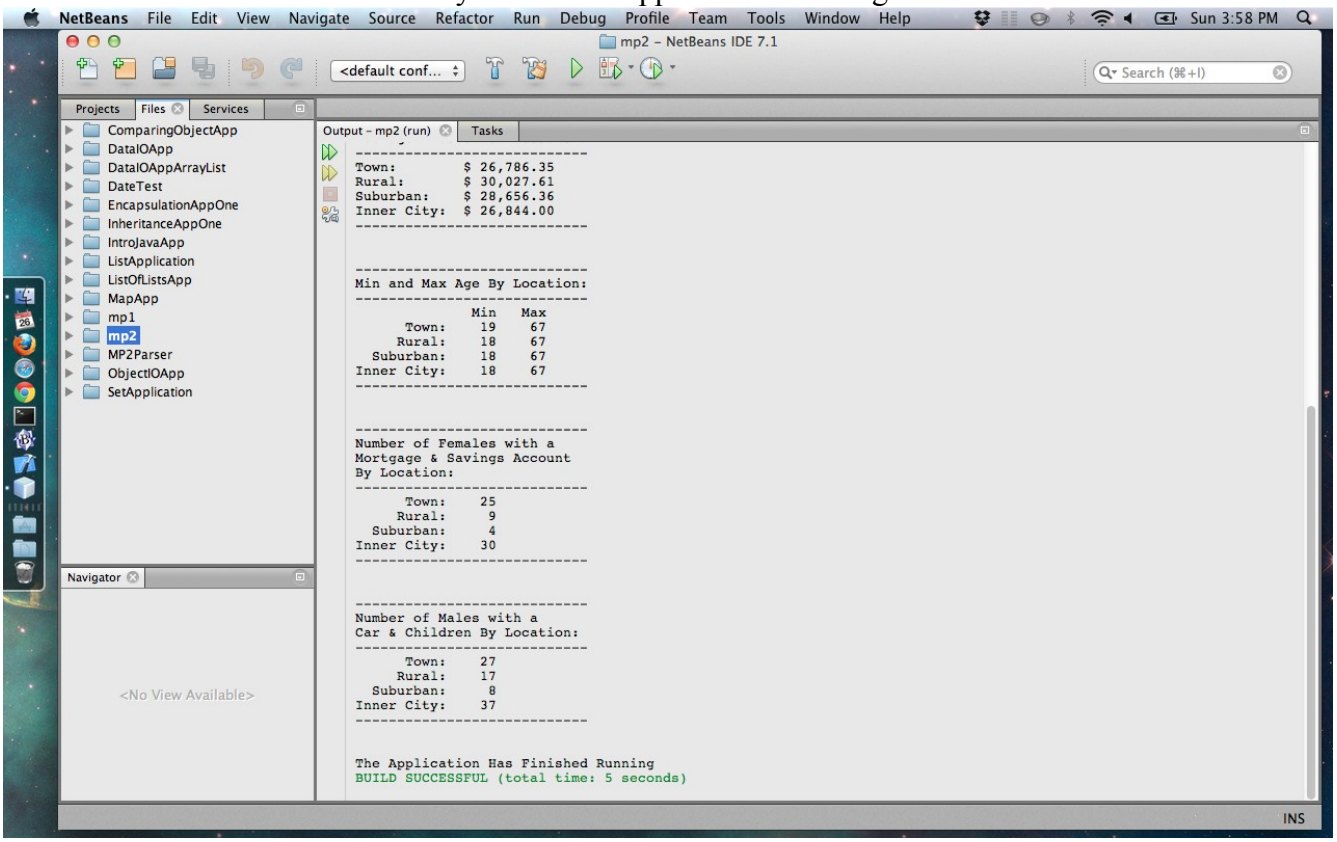
Screenshot of the application starting in NetBeans up to the point where it is sleeping for 5 secs.



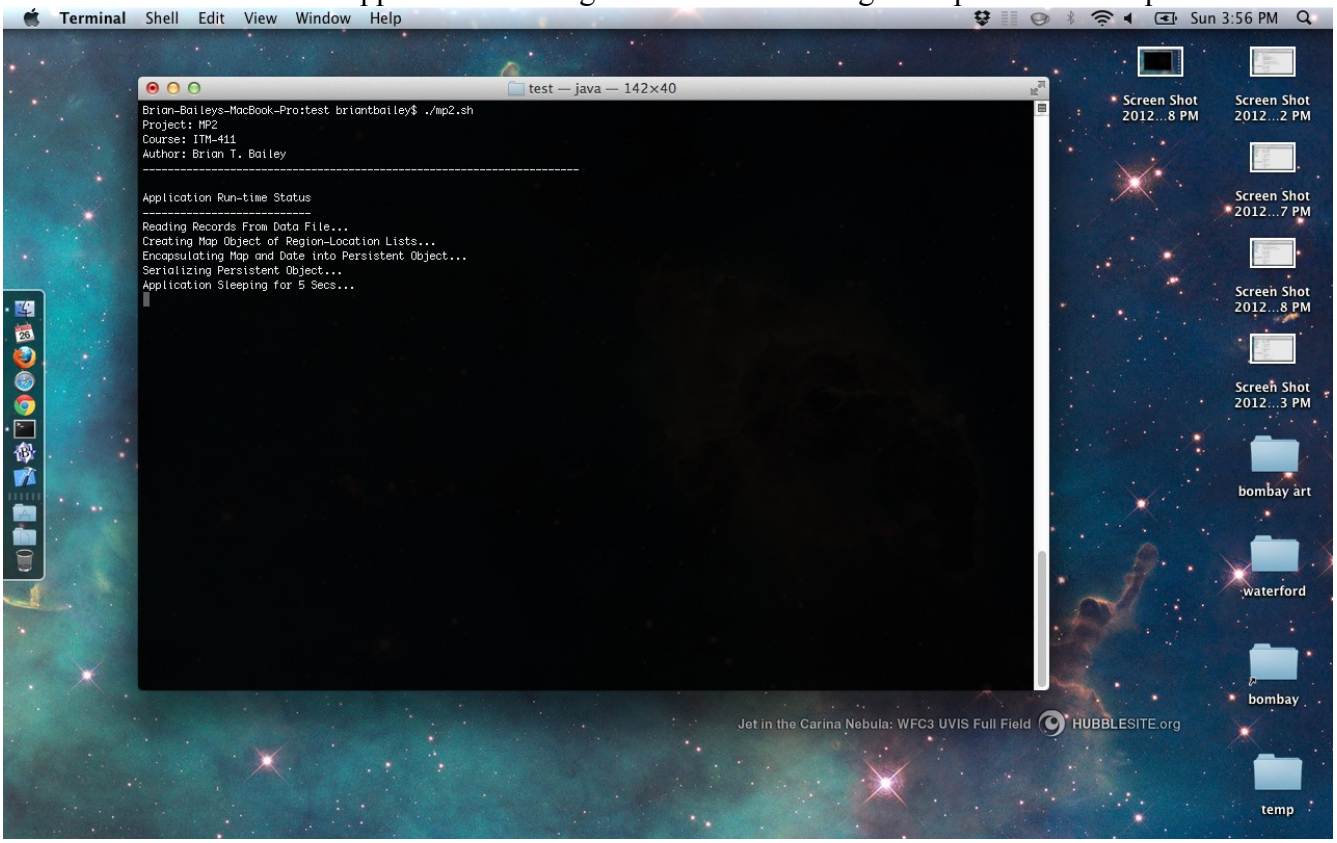
Screenshot of the application continuing from sleep and displaying the time differences and the first two data analytics.



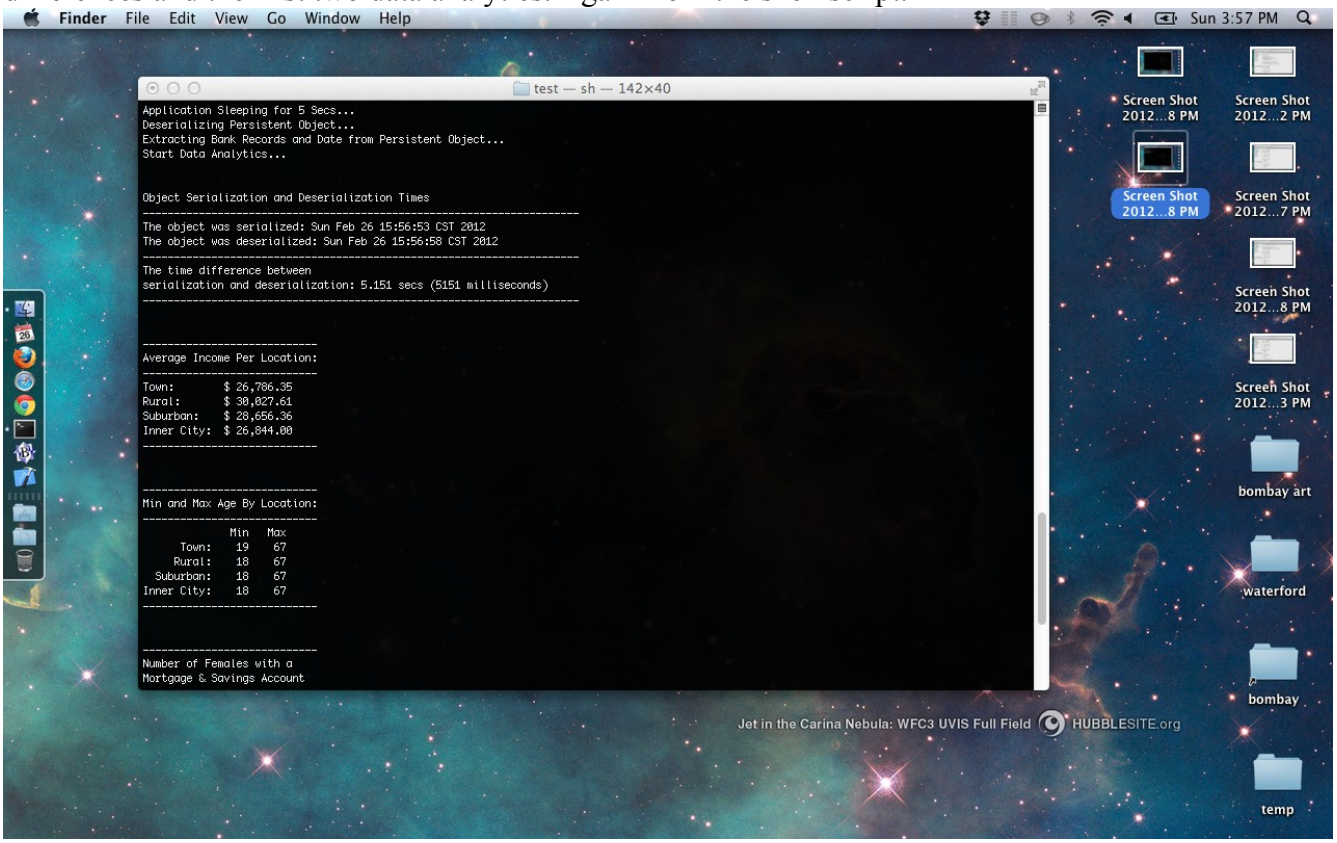
Screenshot of the last two data analytics and the application finishing.



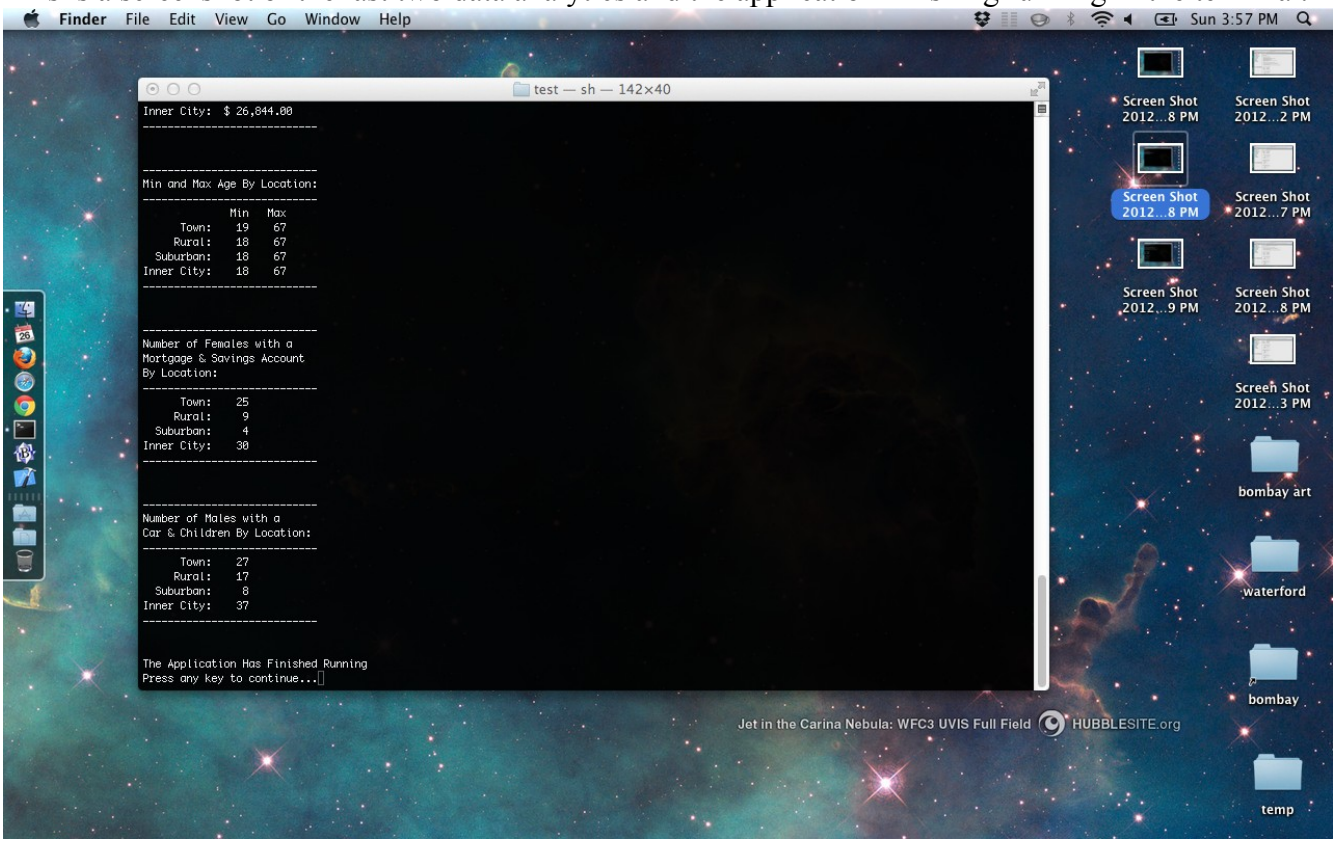
This screenshot shows the application running in the Terminal using the mp2.sh shell script.



The following screenshot shows the application continuing from sleep and displaying the time differences and the first two data analytics. Again from the shell script.



This is a screenshot of the last two data analytics and the application finishing running in the terminal.



Screenshot of the application running in Windows XP using the bat file.