**uBillity Program Test Plan**

**Test Plan**

The test plan will outline the strategy that will be used to test the database functionalities, the resource that will be used, the test environment in which the testing will be performed and the limitations of the testing.

**Assumptions:**

**Database Class Perspective**

* The data input type and controls are handled by the software and not the database.
* Expect method calls input to have matching data types with the actual method.
* Assume a data file, may or may not, exists and it must be called “Mock\_Data.txt”
* Data stored in the data file matches the structure of the database where the delimiter for columns is represented by “tab” spaces and “new line” represents rows

**Customer Class Perspective**

* Contains getter/setter methods for every field
* Initializes each customer object
* Receives a list<string> of varying length to create or edit customer
* Class is completely independent and relies on no separate method calls
* Contains information to generate each bill based on rate, client type, and meter reading

**UI Classes Perspective**

* Provides all the functionality to navigate between every screen
* Provides the form in order to request a new customer
* Provides the form and functionality needed to search for customers, edit customers, and delete customers
* Supports functionality to request billing information and to lay all bill information into a designated format for the user to print
* Full printing functionality, depends on current screen position.

**uBillity Main Class Perspective**

* Initializes database class and creates and array of all customer objects (if any are present)
* Initializes the primary UI class
* Contains methods to add customer, which relays information to both the Customer class and Database class.
* Contains methods to update customer, which sends a request to the database to delete a specific instance of the customer, and resends the new information in order to update that customer.
* Contains methods to return both the array of customers, and able to find specific customers based on name or social security number, as discussed in the design specifications

**Database Class Test Procedures**

This database class is meant assume the tasks of server/client communication.

**List of Test Cases:**

1. test\_InitializeDB()
2. test\_getData()
   1. test\_dataAccess()
3. test\_addCustomer()
4. test\_removeCustomer()
5. test\_customerObj\_toString()

**Features to be Tested:**

1. DataBase initialization
2. Gets data from data text file and stores into a dictionary
   1. Checks if the data stored in file are accessible
3. Adds new customer data to the text file and verify
4. Remove customer data from text file
5. Checks if string is output is valid

**Testing Approach:**

The common testing procedure is to create a new customer object with random data input and verify if the customer data is properly stored.

**Testing Schedule of Tasks:**

* Test getData() method
  + Check if file is created
  + Check if data output matches with the content of the file
  + Check if data output is complete
* Test initialize\_DB() method
  + Assuming a data file does not exist:
    - Check if data file is created successfully and is empty
  + Assuming data file exists and contains some data
    - Check if data output matches the data contained in the file
* Test custObj\_toString(Customer cust) method
  + Check if string output matches with customer object data
  + Check if customer object is translated into a structured string where the object data is separated by a delimiter “tab space”
* Test addCustomer(Customer cust) method
  + Check if new customer object data matches the data store in the storage file
* Test removeCustomer(Customer cust) method
  + Check if Customer object data is still contained in the file

**Customer Class Test Procedures**

**List of Test Cases:**

1. setFirstName(string name)
2. setLastName(string name)
3. setSocialSecurity(string num)
4. setContractType(string type)
5. setContractLength(string length)
6. setBillStreet(string street)
7. setBillCity(string city)
8. setBillState(string state)
9. setBillZip(string zip)
10. setBill(string street, string city, string state, string zip)
11. setServiceStreet(string street)
12. setServiceCity(string city)
13. setServiceState(string state)
14. setServiceZip(string zip)
15. setService(string street, string city, string state, string zip)
16. setPhoneNumber(string num)
17. setEmai(string mail)
18. setBillType(string type)
19. setAutoBilling(string auto)
20. setBillRate(string rate)
21. setCurrentMeterReading(string meter)
22. setPastMeterReading(double num)
23. setMoneyOwed(double num)
24. setAccountNumber()
25. setCardName(string name)
26. setCardNumber(string num)
27. setCardExpiration(string exp)
28. setCardSecurity(string code)
29. setCardZip(string zip)
30. setCard(string name, string num, string exp, string code, string zip)
31. getFirstName()
32. getLastName()
33. getFullName()
34. getSocialSecurity()
35. getContractType()
36. getContractLength()
37. getBillStreet()
38. getBillCity()
39. getBillState()
40. getBillZip()
41. getBillInfo()
42. getServiceStreet()
43. getServiceCity()
44. getServiceState()
45. getServiceZip()
46. getServiceInfo()
47. getPhoneNumber()
48. getEmail()
49. getBillType()
50. getAutoBilling()
51. getBillRate()
52. getCurrentMeterReading()
53. getPastMeterReading()
54. getMoneyOwed()
55. getAccountNumber()
56. getCardName()
57. getCardNumber()
58. getCardExpiration()
59. getCardSecurity()
60. getCardZip()
61. getCardInfo()
62. updateMeterReading(double num)
63. getBillAmount(double meterReading)
64. makePayment()

**Features to be Tested:**

1. Setting customer information
2. Getting customer information
3. Calculating bill amount
4. Make payment

**Testing Approach:** (a couple lines description)

The testing procedure was to create a couple customers with random data and look at each field of those customers. The set methods were then called to change customer information and the get methods would be called again to make sure that customers information was updated.

**Testing Schedule of Tasks:** (testing logic)

* setFirstName(string name)

o change firstName to name

* setLastName(string name)

o change lastName to name

* setSocialSecurity(string num)

o change socialSecurity to num

* setContractType(string type)

o change contractType to type

* setContractLength(string length)

o change contractLength to length

* setBillStreet(string street)

o change bill.billStreet to street

* setBillCity(string city)

o change bill.billCity to city

* setBillState(string state)

o change bill.billState to state

* setBillZip(string zip)

o change bill.billZip to zip

* setBill(string street, string city, string state, string zip)

o change all billing information to appropriate parameters

* setServiceStreet(string street)

o change service.serviceStreet to street

* setServiceCity(string city)

o change service.serviceCity to city

* setServiceState(string state)

o change service.serviceState to state

* setServiceZip(string zip)

o change service.serviceZip to zip

* setService(string street, string city, string state, string zip)

o change all service information to appropriate parameters

* setPhoneNumber(string num)

o change phoneNumber to num

* setEmail(string mail)

o change email to mail

* setBillType(string type)

o change billType to type

* setAutoBilling(string auto)

o set autoBilling to true or false

* setBillRate(string rate)

o change billRate to rate

* setCurrentMeterReading(string meter)

o change currentMeterReading to meter

* setPastMeterReading(double num)

o change pastMeterReading to num

* setMoneyOwed(double num)

o add num to moneyOwed

* setAccountNumber()

o generate a random 9 digit account number for a customer

* setCardName(string name)

o change card.cardName to name

* setCardNumber(string num)

o change card.cardNumber to num

* setCardExpiration(string exp)

o change card.cardExpiration to exp

* setCardSecurity(string code)

o change card.cardSecurity to code

* setCardZip(string zip)

o change card.cardZip to zip

* setCard(string name, string num, string exp, string code, string zip)

o change all credit card information to appropriate parameters

* getFirstName()

o return first name of customer

* getLastName()

o return last name of customer

* getFullName()

o return first and last name of customer

* getSocialSecurity()

o return social security number of a customer

* getContractType()

o return contract type of customer

* getContractLength()

o return contract length of a customer

* getBillStreet()

o return billing street of a customer

* getBillCity()

o return billing city of a customer

* getBillState()

o return billing state of a customer

* getBillZip()

o return billing zip of a customer

* getBillInfo()

o return full billing information of a customer

* getServiceStreet()

o return service street of a customer

* getServiceCity()

o return service city of a customer

* getServiceState()

o return service state of a customer

* getServiceZip()

o return service zip of a customer

* getServiceInfo()

o return full service information of a customer

* getPhoneNumber()

o return phone number of a customer

* getEmail()

o return email address of a customer

* getBillType()

o return billing type of a customer

* getAutoBilling()

o return whether or not a customer has auto billing

* getBillRate()

o return billing rate of a customer

* getCurrentMeterReading()

o return current meter reading of a customer

* getPastMeterReading()

o return past meter reading of a customer

* getMoneyOwed()

o return money owed by a customer

* getAccountNumber()

o return account number of a customer

* getCardName()

o return name on credit card of a customer

* getCardNumber()

o return credit card number of a customer

* getCardExpiration()

o return expiration date of a customers credit card

* getCardSecurity()

o return card security code of a customer

* getCardZip()

o return zip code of customers credit card

* getCardInfo()

o return full credit card information of a customer

* updateMeterReading(double num)

o if num is greater than currentMeterReading, set past meter reading to current meter reading and set current meter reading to num

* getBillAmount(double meterReading)

o calculate customers bill amount based on monthly usage calculated by subtracting currentMeterReading from meterReading. If monthly usage is greater than 0, calculate customers bill amount. If customers contract type is commercial, bill amount is billRate \* monthly usage. If customers contract type is residential, their bill will be calculated with a discounted rate after each 100 KwH usage.

* makePayment(double num)

o If num is greater than moneyOwed, set moneyOwed to 0, else update moneyOwed to the correct amount a customer owes.

**uBility Main Class Test Procedures**

**List of Test Cases:**

1. addCustomer(List<string> newCustList)

2. deleteCustomer(string socialSecurityNum)

3. updateCustomer(Customer customer)

4.updateCustomer(List<string>updateCustList)  
5.getCustomers()  
6.findCustByName(string firstName, string lastName)  
7. findCustBySSN(string socialSecurityNum)

**Features to be Tested:**

1. adding customer to the dictionary and database

2. deleting a customer from the dictionary and database

3. updating customers in the dictionary and database

4. returning the dictionary of customers

5. searching for customer by name, or by social security number

**Testing  Approach:**

Test all of the above manipulations of the dictionary of customers and the database file, ensure that the correct manipulations have been made to both the dictionary and the database by the end of the method call. I tested this by creating test customers and performing the above methods, then checked the contents of both the dictionary and the database to make sure both were in agreement.

**Testing Schedule of Tasks:**

1. addCustomer(List<string> newCustList)

* Create a test customer list
* Call this method on the list
* Check the dictionary to make sure the correct customer object has been added
* Check the database file to make sure the correct customer object has been added

2. deleteCustomer(string socialSecurityNum)

* pass the string of a social security number that belongs to a customer in the database
* Make sure that the customer is deleted from the dictionary and the database
* If the customer does not exist, nothing happens

3. updateCustomer(Customer customer)

* Create a test customer
* Pass the customer to this method
* Ensure that the customer has been updated in both the database and the dictionary

4. updateCustomer(List<string> updateCustList)

* Create a test customer list
* Pass the customer list to this method
* Ensure that the customer has been updated in both the database and the dictionary

5. getCustomers()

* Ensure that the dictionary returned is the same as the one maintained in the main class

6. findCustByName(string firstName, string lastName)

* pass strings for both the first and last name of a customer
* If the customer exists, make sure that it is returned
* If the customer does not exist, make sure that a null pointer is returned

7. findCustBySSN(string socialSecurityNum)

* pass the string of a social security number for a customer
* If the customer exists, make sure that it is returned
* If the customer does not exist, make sure that a null pointer is returned

**UI Classes Test Procedures**

**List of Test Cases:**

1. uBillity\_UI()

2. addCustomerToolStripMenuItem\_Click()

3. homeToolStripMenuItem\_Click()

4. addCustomerButton\_Click()

5. searchCustomerToolStripMenuItem\_Click()

6. initSearchComboBox()

7. verifyScreenValidity()

8. clearCustomerResults()

9. clearSearchResults()

10. searchCustomerButton\_Click()

11. generateAllBillsToolStripMenuItem\_Click()

12. customerInfo()

13. saveButton\_Click()

14. deleteButton\_Click()

15. billButton\_Click()

16. BillControl()

17. captureScreen()

18. printDocument1\_PrintPage()

19. printPreviewToolStripMenuItem\_Click()

20. selectPrinterToolStripMenuItem\_Click()

21. printToolStripMenuItem\_Click()

22. printDocument2\_PrintPage()

23. printAllPreviewToolStripMenuItem\_Click()

24. printAllToolStripMenuItem1\_Click()

25. Bill()

26. getPanel()

27. ProgressClass()

28. init()

29. update()

**Features to be Tested:**

1. Verify primary screen loads properly

2. Verify screen layout of all screens is correct

3. Verify tab order for adding and editing customers

4. Verify information is sent in a proper order from the addCustomerScreen

5. Able to retrieve information for Bills and customer searches

6. Verify information is sent from searchCustomerScreen for searching, editing, deleting, and printing bills

7. Verify that generating all bills works properly

**Testing  Approach:** Ideally, the UI should only layout information and make specific calls to each other class. The only time information is created within the UI is for adding a new customer, searching for a customer, or editing customer information. At all other times, the UI will only need to display information. To verify correctness, all buttons and screens must be tested to see if they function correctly

**Testing Schedule of Tasks:**

1. uBillity\_UI()

* Make sure initial homescreen works
* Click each tool strip button to verify screen transitions work

2. addCustomerToolStripMenuItem\_Click()

* Verify add customer screen appears
* Verify the add customer tool strip icon isn’t visible

3. homeToolStripMenuItem\_Click()

* Verify home screen appears
* Verify the home tool strip icon isn’t visible

4. addCustomerButton\_Click()

* Enter customer information and test tab order
* Set autobilling to automatic and don’t enter credit card information to verify error checking
* Set autobilling to manual and enter credit card information to verify it is deleted
* Enter any amount of information and press on either home screen or search customer screen tool strip item to verify warnings appear for information being present
* Enter complete customer information and verify the information is sent to the appropriate classes

5. searchCustomerToolStripMenuItem\_Click()

* Verify the search customer screen appears
* Verify the search customer tool strip item isn’t visible
* Test tab order
* Verify all boxes except combo box need to be filled in order to proceed
* Verify combo box contains info if valid customers are present

6. initSearchComboBox()

* Verify that during each new search the customer box is cleared
* Verify that all updates are noticed by the combo box

7. verifyScreenValidity()

* Test if warnings appear during screen transitions if information is present within the add customer screen

8. clearCustomerResults()

* Verify all screen information in the add customer screen clears

9. clearSearchResults()

* Verify all screen information in the search customer screen clears

10. searchCustomerButton\_Click()

* Verify appropriate boxes are filled properly
* Verify customer information is sent to the appropriate classes in order to search
* Verify ability to receive appropriate customer information and can display customer screen

11. generateAllBillsToolStripMenuItem\_Click()

* Verify creation of progress bar class
* Verify new bills can be generated using the array of customers

12. customerInfo()

* Verify all screen elements are correct
* Verify all customer attributes are correct
* Check tab order

13. saveButton\_Click()

* Verify iteration of all text boxes and combo boxes to send to the update method
* Make sure error checking is provided by testing autobilling box as both auto and manual, with and without information provided in the credit card information boxes.
* Verify list<string> is being sent to the appropriate location to update customers

14. deleteButton\_Click()

* Check if customer deletion was successfull

15. billButton\_Click()

* Verify appropriate information is being gathered for bill generation
* Test bill accuracy for customer

16. BillControl()

* Make sure initial tab order is blank
* Verify printer tab is correct
* Check all printer functions work properly

17. captureScreen()

* Verify screen capture represents printing area

18. printDocument1\_PrintPage()

* Verify correct printing document

19. printPreviewToolStripMenuItem\_Click()

* Verify correct preview is being displayed

20. selectPrinterToolStripMenuItem\_Click()

* Verify printer dialog is correct

21. printToolStripMenuItem\_Click()

* Verify correct page is printed

22. printDocument2\_PrintPage()

* Verify all tabs produce a printed page

23. printAllPreviewToolStripMenuItem\_Click()

* Verify all tabs printing preview is correct

24. printAllToolStripMenuItem1\_Click()

* Verify all tabs print

25. Bill()

* Verify all information is laid out correctly
* Verify customer’s money owed is updated

26. getPanel()

* Verify correct panel is returned

27. ProgressClass()

* Verify progress bar is created

28. init()

* Takes integer and produces a max size for progress bar,.

29. update()

* Updates by certain amount, verify that amount is correct