George Mason University
INFS 519 – Data Structures
Fall 2018
Programming Assignment 1 – Linked Lists and Dynamic Arrays

Your program will create a class to implement the Table abstract data type and then use this class to implement a simple address book.

Table is an ADT which stores pairs of values: a *key* and a *value*. The key must be unique and is used to identify the pair. There are three operations:

- `value lookup(key)` searches the table for a pair with the given key and returns the corresponding value.
- `boolean insert(key, value)` stores the key/value pair in the table.
- `boolean delete(key)` removes the key/value pair with given key from the table.

Public methods for your `Table` class are described below.

The address book will store names (used as keys) and addresses that are associated with the names. The program will present the user a menu with a choice of operations: *add* a contact (name and address), *look up* a contact (displaying the associated address), *update* the address for a contact, *delete* a contact, *display all contacts*, and *quit*. Furthermore, the program will allow a user to *send* a text message to a contact and to *display all messages* to a contact.

If the user chooses *add* the program prompts for a name. If the name already exists in the address book a message is displayed to that effect and no change is made. Otherwise the program asks for an associated address and the new entry is made. If the user chooses *look up* he or she is asked for a name. The program will look up the entry for that name and display the associated address. If the user chooses *update* he or she is asked for a name. The name is looked up and the address displayed. The user is then prompted for a new address which is accepted and stored in place of the old address. For *delete* the user is asked for a name and the entry for that name is removed from the address book. In any of these cases (other than insert) if the name given by the user is not found an appropriate message is displayed. If the user selects *display all* all of the names and addresses in the address book are displayed (the order is not important).

If the user chooses *send* the program prompts for a name. The program will look-up the entry for that contact and will accept a brief message to be texted to the contact. If the name does not exist in the address book, a new contact entry is created and a text message is accepted to be sent to that contact.

The program continues displaying the menu and taking commands until the user chooses *quit*. If the user enters a selection which is not in the menu, the menu will be displayed again and the user is prompted again for a command. The names, addresses, and messages can be any strings -- your program need not check that they make any sense.

A sample run of the program may look like:

```
Add a contact (a)
Send a text message (m)
Look up a contact (l)
Update address (u)
Delete a contact (dc)
Delete a text message (dm)
Display all contacts (ac)
Display all messages to a contact (am)
Quit (q)
->  a

Name:  Genghis Khan
Address:  Khentii Aimag, Mongolia

Add a contact (a)
Send a text message (m)
Look up a contact (l)
Update address (u)
Delete a contact (dc)
Delete a text message (dm)
Display all contacts (ac)
Display all messages to a contact (am)
Quit (q)
->  a

Name:  Rasputin
Address:  St. Petersburg, Russia

Add a contact (a)
Send a text message (m)
Look up a contact (l)
Update address (u)
Delete a contact (dc)
Delete a text message (dm)
Display all contacts (ac)
Display all messages to a contact (am)
Quit (q)
->  a

Name:  Helen Mirren
Address:  London, England

Add a contact (a)
Send a text message (m)
Look up a contact (l)
Update address (u)
Delete a contact (dc)
Delete a text message (dm)
Display all contacts (ac)
Display all messages to a contact (am)
Quit (q)
->  l

Name:  Rasputin
Address is St. Petersburg, Russia

Add a contact (a)
Send a text message (m)
Look up a contact (l)
Update address (u)
Delete a contact (dc)
Delete a text message (dm)
Display all contacts (ac)
Display all messages to a contact (am)
Quit (q)
->  u

Name:  Rutherford B. Hayes
Rutherford B. Hayes is not in the book

Add a contact (a)
Send a text message (m)
Look up a contact (l)
Update address (u)
Delete a contact (dc)
Delete a text message (dm)
Display all entries (ac)
Display all messages to a contact (am)

Quit (q)
->  u

Name:  Helen Mirren
Old address is London, England
New address:  Hollywood, California

Add a contact (a)
Send a text message (m)
Look up a contact (l)
Update address (u)
Delete a contact (dc)
Delete a text message (dm)
Display all contacts (ac)
Display all messages to a contact (am)
Quit (q)
->  d

Name to delete:  Rasputin
Add a contact (a)
Send a text message (m)
Look up a contact (l)
```

```
Update address (u)
Delete a contact (dc)
Delete a text message (dm)
Display all contacts (ac)
Display all messages to a contact (am)
Quit (q)
->  ac

Name:  Helen Mirren
Address:  Hollywood, California

Name:  Genghis Khan
Address:  Khentii Aimag, Mongolia

Add a contact (c)
Send a text message (m)
Look up a contact (l)
Update address (u)
Delete a contact (dc)
Delete a text message (dm)
Display all contacts (ac)
Display all messages to a contact (am)
Quit (q)
->  q
```

Internals

You will write (among other classes) a class `Table` which will store entries which are (key/value) pairs of Strings. You will need to write a `Node` class which will have (as private fields) a `key` String (used to identify the entry), a `value` String (used to store some data associated with the key), a dynamic array `msgs` (with an initial size of two) for storing text messages to a contact, and a `next` pointer used for linking. `Table` will have public methods:

- `public boolean insert(String key, String value):` Inserts a new entry to the table. If an entry already exists with the given key value make no insertion but return `false`.
- `public String lookUp(String key):` Looks up the entry with the given key and returns the associated value. If no entry is found `null` is returned.
- `public boolean deleteContact(String key):` Deletes the entry with the given key. If no entry is found returns `false`.
- `public boolean update(String key, String newValue):` Replaces the old value associated with with the given key with the newValue string.
- `public boolean markToStart():` Sets the *mark* (see below) to the first item in the table. Returns `false` if the table is empty.
- `public boolean advanceMark() :` Moves the mark to the next item in the table. If there is no next item (e.g. at the end of the list) returns `false`.
- `public String keyAtMark():` Returns the key stored in the item at the current mark.

- `public String valueAtMark()` : Returns the value stored in the item at the current mark.
- `public int displayAll()` : Displays Name/Address for each table entry. Returns total entry count.
- `public boolean deleteContact(String key)`: Deletes the entry with the given key. If no entry is found returns `false`.
- `public boolean SendMessage(String key, String msg)`: adds a message into the dynamic array associate with this entry node. If no entry is found or no messages are found in the dynamic array returns `false` and displays appropriate message.
- `public int deleteMessages(String key)` : Displays all text messages sent to a contact. Returns total message count.

`Table` will keep a *linked list* of `Nodes`. Each contact entry (from the methods listed above) will be stored in an instance of `Node` stored in this linked list. The list is maintained in no particular order, but no two `Nodes` in the list can have the same key field. `Table` will also have a (private) field, `mark`, which is a reference to `Node`. This field is modified or used by the `markToStart`, `advanceMark`, `keyAtMark`, and `valueAtMark` methods. Together these methods can be used to traverse the list and read the contents of all the entries in the table.

You may add any more methods you need to the `Table` class but they must be private. The only public methods are those listed above. *All* instance fields in all classes in your program will be private.

You will notice that this `Table` class is not specifically an address book but just stores and operates on pairs of strings. You will implement your address book as an instance of `Table` and will implement the address book operations, listed at the beginning of this page, using public methods of `Table`.

DynamicArray<T> (DynamicArray.java):
This class represents a **generic** collection of objects in an array whose capacity can grow and shrink as needed. It supports the usual operations of adding and removing objects including *add*() and *remove*(). The capacity of a DynamicArray must change as items are added or removed and the following rules must be followed in your implementation:

- The default initial capacity is fixed to be 2.
- When you need to add items and there is not enough space, grow the array to double its capacity.
- When you delete an item and the size falls below 1/3 of the capacity, shrink the array to half its capacity.

To turn in

You will turn in a (well-commented) source listing via Blackboard. A rubric will be posted for the next class.