Term Project

Group 9

Michael Salemi – 0751614

Ryan Mattsson – 0748663

Shiyaani Jegatheeswaranathan - 0572907

Term-Project Report

In this project, we demonstrated the utilization of 3 design methods. The methods we chose to implement were the facade method, chain of responsibility, and the singleton method. The facade method was implemented through the use of an interface inherited by our Contact class. This interface was responsible for handling how other classes would interact with our contact class without gaining direct access to its private member variables. The chain of responsibility method was implemented by using handler functions that would ensure that Contact objects would only be created through a specific method in the PhoneBook class that would use safeguards to ensure non-null values. Finally the Singleton method was implemented to ensure that only one PhoneBook object would be created.

Our program implemented the bubble sort method in order to sort the contact list when new contacts were added. This ensured that the contact list was always in alphabetical order, regardless of what order the contacts were added in. Due to the limited size of our contact list (10 contacts), this meant that bubble sort was an adequate sorting algorithm for speed, especially given that our contacts were stored in a linked list.

The 3 data structures we implemented were an **array** of type Contact, a L**inked List** of type Contact, and a **queue** of type contact. Our array was used to store the initial contacts that would be added to the contact list at program start. We chose to use an array because there would only be 1 object type utilised, and no objects would be added or removed in this array. Also, because arrays have the quickest index time. Our Linked List was used for our contact list in the phone book. We chose to use a linked list in order to make removal and addition of Contacts extremely fast due to the nature of linked lists relation between its nodes. Finally, we implemented a queue in order to store the contacts that were not able to fit within the contact list. This would prove useful when we were to remove a contact from the contact list and add the contacts that would be first dequeued from our contact waitlist.

Below is an example of our PhoneBoook program output. We can see how the contacts are added to the linked list until the set amount is reached (10). Then, the remaining contacts are enqueued to our contact waitlist until further notice. Afterwards, we display the count of our queue. Finally we display the PhoneBook information such as the owner's namer and the amount of contacts in the phonebook (not in the waitlist) followed by all of the contacts name and numbers that were sorted in alphabetical order by name using our implemented sorting algorithm.

```
John Davis was added to contact list.
Lisa Kudrow was added to contact list.
Wayne Simmonds was added to contact list.
Happy Gilmore was added to contact list.
Tiger Williams was added to contact list.
Tiger Woods was added to contact list.
Courtney Cox was added to contact list.
Wayne Gretzky was added to contact list.
Jerome Seinfeld was added to contact list.
Sean Combs was added to contact list.
Contact list is full. Denzel Washington was added to waitlist.
Contact list is full. Jo Beck was added to waitlist.
Contact list is full. Eryn Washington was added to waitlist.
Contact list is full. Muhammad Ali was added to waitlist.
Contact list is full. Mike Tyson was added to waitlist.
Contact list is full. Wayne Brody was added to waitlist.
Contact list is full. Michael Jordan was added to waitlist.
Contact list is full. Michael Schumacher was added to waitlist.

8 contacts in wait-list.

Mike's phonebook: 10 contacts
1: Courtney Cox 1012355602
2: Happy Gilmore 6632544587
3: Jerome Seinfeld 7985501234
4: John Davis 7946258746
5: Lisa Kudrow 8746215487
6: Sean Combs 1002356986
7: Tiger Williams 8988874521
8: Tiger Woods 9758543225
9: Wayne Gretzky 2011143986
10: Wayne Simmonds 9230038706
```

This program compiled and executed without any errors or warnings.

Source code will be submitted as 4 separate files.

In the following UML class diagram, the 3 different classes are displayed. The first class is the ***ICallable*** interface. This *interface* acts as a way for the **Phonebook** and **Contact** objects to interacts without the **Phonebook** having direct access to the **Contact**'s member variables. This *interface* is realized by the Contact class and that relationship is displayed by the dotted line and arrow connecting the Contact class and the ***ICallable*** interface. Secondly, we have the relationship between the **Contact** class and the **Phonebook** class. Since the **Phonebook** class uses the **Contact** class in order to populate its LinkedList and Queue, we display the aggregation with a solid line and hollow diamond. This is because the **Contact** objects are not deleted when the **Phonebook** object is.