Credit Name: CSE 3110 Iterative Algorithms 1

Assignment: Friends

How has your program changed from planning to coding to now? Please Explain

In this application there are 5 classes; one designates an interface to compare either first or last names, another is a class that holds the defines the relevant information for each friend on the app, then there is a class for the database which allows you to add, remove and find friends on the app saving their information to a file. Afterwards there is a class which has a selection sort method to organize names alphabetically. The last class is the tester class which is an application for adding and receiving your friends' information.

```
public class comparableNames {
    public interface ComparableNames {
        int compareToFirstName(Object obj);
        int compareToLastName(Object obj);
    }
}
```

Our interface is called comparableNames and has 2 comparable methods: one to compare first names and another to compare last names. It takes in 1 object as a parameter which will be a friend object, we will define below.

```
package Mastery.Friends;

import java.io.*;

public class friend implements ComparableNames, Serializable {
    // Friend attributes
    String firstName,lastName,phoneNo,email;
```

Then I made the friend class to store and call all the relevant information for every friend. This class implements 2 interfaces ComparableNames and Serializable. The Serializable class is to be able to save objects to files by converting data structures not recognized by file into binary data(serialization). It can then be called from the file and reconstructed to java code(deserialization).

Then I defined instance variables for this class which are the attributes of the friend, including first and last name, phone number and email address.

```
// Primary constructor
public friend(String fname, String lname, String phone, String emailAdd) {
    firstName = fname;
    lastName = lname;
    phoneNo = phone;
    email = emailAdd;
}
```

Then i define and initialize my main constructor with all the attributes declared before and initialize it to a string parameter. Note that the phone number, although a number, is recorded as a string because no calculations need to be done with it.

```
// Overloaded constructor for name-only initialization
public friend(String fname, String lname) {
    firstName = fname;
    lastName = lname;
}
```

Then we overload this constructor to initialize it with only a name. This will be used for finding and deleting friends.

```
// Returns full name as "FirstName LastName"
public String firstAndLast() {
    return firstName + " " + lastName;
}

// Getters
public String getLast() {
    return lastName;
}

public String getFirst() {
    return firstName;
}
```

Then we have 3 methods for getting their full name and their last and first name.

```
public String displayByLast() {
    return lastName + ", " + firstName + "\n" + phoneNo + "\n" + email;
}

// Displays friend information with first name first
public String displayByFirst() {
    return firstName + ", " + lastName + "\n" + phoneNo + "\n" + email;
}
```

We then have 2 methods which return a friend's information and they differ in that one displays last name first and the other displays first name first.

```
// Comparison methods implementing ComparableNames interface
public int compareToFirstName(Object f) {
    friend testObj = (friend) f;
    return (firstName.compareToIgnoreCase(testObj.getFirst()));
}

public int compareToLastName(Object f) {
    friend testObj = (friend)f;
    return (lastName.compareToIgnoreCase(testObj.getLast()));
}
```

Now I define the interface methods defined in the comparableNames interface. It takes in an object which we type cast as a friend(a different friend) and then we compare the firstName of the friend defined in this class with the first name of the other friend indicated by testObj. We ignore the case and only compare alphabetically. The same method is defined for the last name.

```
// Equality check based on first and last names (case-insensitive)
public boolean equals(Object f) {
    friend testObj = (friend)f;

    return firstName.compareToIgnoreCase(testObj.getFirst()) == 0 &&
           lastName.compareToIgnoreCase(testObj.getLast()) == 0;
}
```

Then we make an equals method based on if they have the same full name. We compare the using the same compareToIgnoreCase as the previous 2 methods but here we return true if both compareTo statements are equal to 0 otherwise we return false.

```
// String representation of friend object
public String toString() {
    return "Name: " + firstAndLast() + "\nEmail: " + email + "\nPhone No: " + phoneNo;
}
```

Lastly, for the friend class we define a toString method which displays the friends info.

The next class is the selectionSort.

```
public static void sortFirst(ArrayList items) {
    for (int index = 0; index < items.size(); index++) {
        for (int subIndex = index; subIndex < items.size(); subIndex++) {
            ComparableNames item1 = (ComparableNames) items.get(subIndex);
            ComparableNames item2 = (ComparableNames) items.get(index);

            if (item1.compareToFirstName(item2) < 0) {
                ComparableNames temp = item2;
                items.set(index, item1);
                items.set(subIndex, temp);
            }
        }
    }
}
```

The first selectionSort is to sort first names. We start with a list of friend objects, which is why we need to use the ComparableNames interface. Then we have 2 nested for-loops. We then use the ComparableNames interface in the second for loop to define 2 elements. In one iteration of the outer for loop item 1 iterates through every other element that is not item2 while item2 stays constant. While item 1 iterates through all options it is also lexicographically compared to to item 2. If item1 is alphabetically before item2 , -1 is returned and it satisfies the if-statement. Then the positions of item1 and item 2 are swapped. Then this repeats for every other index in the list.

```java
// Sorts by last name
public static void sortLast(ArrayList items) {
    for (int index = 0; index < items.size(); index++) {
        for (int subIndex = index; subIndex < items.size(); subIndex++) {
            ComparableNames item1 = (ComparableNames) items.get(subIndex);
            ComparableNames item2 = (ComparableNames) items.get(index);

            if (item1.compareToLastName(item2) < 0) {
                ComparableNames temp = item2;
                items.set(index, item1);
                items.set(subIndex, temp);
            }
        }
    }
}
```

The exact method is redefined, this time to compare last names by a selection sort.

Then we set up the database for all friends to be saved to a file.

```java
package Mastery.Friends;

import java.util.ArrayList;

public class friendsDB {
    private ArrayList<friend> list;  // Stores friend objects
    private int numFriends;          // Tracks number of friends
```

We declare an array list with friend objects as its entries. We also declare an instance integer variable for the number of friends.

```java
public friendsDB() {
    File friendsFile = new File("C:\\Users\\Aryan Kapoor\\git\\cs30p32025-ak0122\\Chapter12\\src\\Mastery\\Friends\\friendsTextFile");
    list = new ArrayList<>();
    friend aFriend;

    /* Create new file if none exists */
    if (!(friendsFile.exists())) {
        try {
            friendsFile.createNewFile();
            System.out.println("There are no friend records in your database.");
        } catch (IOException e) {
            System.out.println("File could not be created.");
            System.err.println("IOException: " + e.getMessage());
        }
        numFriends = 0;
```

We create and link a new file where our data will be stored and we initialized our friends list. We declared a friend object called aFriend.

We then create a file if none exists catching the relevant exceptions. If this is the case we set the number of friends to 0.

```
} else {
    /* Load existing friends from file */
    try {
        if(friendsFile.length() > 0) {
            FileInputStream in = new FileInputStream(friendsFile);
            ObjectInputStream readFriends = new ObjectInputStream(in);
            numFriends = (int)readFriends.readInt();

            if (numFriends == 0) {
                System.out.println("There are no existing friend records in your database.");
            } else {
                for (int i = 0; i < numFriends; i++) {
                    aFriend = (friend) readFriends.readObject();
                    list.add(aFriend);
                }
            }
            readFriends.close();
            in.close();
        }
    } catch (FileNotFoundException e) {
        System.out.println("File could not be found.");
        System.err.println("FileNotFoundException: " + e.getMessage());
    } catch (IOException e) {
        System.out.println("Problem with input/output.");
        System.err.println("IOException: " + e.getMessage());
    } catch (ClassNotFoundException e) {
        System.out.println("Class could not be used to cast object.");
        System.err.println("ClassNotFoundException: " + e.getMessage());
```

otherwise we load existing friends by using a file input stream around our file. But since the program reads objects we initalize a object input stream that wraps around the file input stream. Then we read the friend objects saved and the integer value for friends. These values were written to the file initially in a different method update() which runs when the program is exited. We display if there are no saved friends otherwise we add each friend to the list that is reinitialized every time the program is run so they remain in the list.

We close our reader objects and write statements to catch the relevant exceptions.

Now we will define a method to add friends.

```java
// Adds a new friend to the database
public void addFriend() {
    String fname, lname, email, phoneNo;
    Scanner input = new Scanner(System.in);

    System.out.println("Enter your friend's first name: ");
    fname = input.next();

    System.out.println("Enter your friend's last name: ");
    lname = input.next();

    System.out.println("Enter your friend's email: ");
    email = input.next();

    System.out.println("Enter your friend's phone number: ");
    phoneNo = input.next();

    friend newFriend = new friend(fname, lname, phoneNo, email);
    list.add(newFriend);
    numFriends++;
}
```

first we define variables for all the inputs we get from the user including full name, email and phone number. Once we prompt and receive these inputs, we create another friend object calling it new Friend and after we initalize our newFriend we add them to our list and increment our number of friends by 1.

```java
// Removes a friend from the database
public void deleteFriend(String fName, String lName) {
    int deleteInd;
    friend removeFriend = new friend(fName, lName);
    deleteInd = list.indexOf(removeFriend);

    if(deleteInd > -1) {
        list.remove(deleteInd);
        System.out.println(removeFriend.firstAndLast().toUpperCase() + " was removed from your friend's list.");
        numFriends--;
    } else {
        System.out.println("This friend does not exist");
    }
}
```

Similarly, we have a method for removing friends. Here we initalize our friend by only their name which is why we needed to have a second overloaded constructor. we then find the index of this friend in the list using the indexOf method. If our index was not -1 then we proceed and remove the person at the index and print a statement outlining the same and also reducing the friend count by 1. Otherwise if the index was -1 then we show that the friend does not exist in the list.

```java
public void displayFriends(int order) {
    friend retrievedFriend;

    if (order == 0) {
        selectionSorts.sortLast(list);
        for (int friendData = 0; friendData < list.size(); friendData++) {
            retrievedFriend = list.get(friendData);
            System.out.println(retrievedFriend.displayByLast().toUpperCase() + "\n");
        }
    } else {
        selectionSorts.sortFirst(list);
        for (int friendData = 0; friendData < list.size(); friendData++) {
            retrievedFriend = list.get(friendData);
            System.out.println(retrievedFriend.displayByFirst().toUpperCase() + "\n");
        }
    }
}
```

To display our friends, we have 2 options : display by first name or display by last name. If they select 0,we sort the list by last name and we iterate through our friend array list and print their info showing last name first. Otherwise we sort by first and print out each friend's info with the first name showing first.

```java
// Finds a friend in the database
public void findFriend(String fName, String lName) {
    int finderInd;
    friend foundFriend = new friend(fName, lName);
    finderInd = list.indexOf(foundFriend);

    if(finderInd > -1) {
        System.out.println(foundFriend.firstAndLast().toUpperCase() + " was found in your friend's list.");
        System.out.println(list.get(finderInd).toString());
    } else {
        System.out.println("The friend was not found on the list.");
    }
}
```

The penultimate method of the database class is to find a friend. It takes in their first and last name. And we initialize a friend object to indicate the information of the friend we are finding. We search for their index in the list. If they were not found we print a not found statement otherwise we print that they were found and then show their information.

```java
/**
 * Saves friend records to file
 * pre: none
 * post: friend records have been written to file
 */
public void update() {
    File updateFile = new File("C:\\Users\\Aryan Kapoor\\git\\cs30p32025-ak0122\\Chapter12\\src\\Mastery\\Friends\\friendsTextFile");

    try {
        FileOutputStream out = new FileOutputStream(updateFile);
        ObjectOutputStream writeFriends = new ObjectOutputStream(out);
        writeFriends.writeInt(numFriends);

        for (int i = 0; i < list.size(); i++) {
            writeFriends.writeObject(list.get(i));
        }
        writeFriends.close();
        out.close();
    } catch (FileNotFoundException e) {
        System.out.println("File could not be found.");
        System.err.println("FileNotFoundException: " + e.getMessage());
    } catch (IOException e) {
        System.out.println("Problem with input/output.");
        System.err.println("IOException: " + e.getMessage());
    }
```

The last method in the friendsDB class is the update method which writes all the changes to the file when we quit the file. Again we link to the same file as before. Now we create an object output stream to write information to a file rather than read it. We then write the number of friends as an int. And then for every friend in the list we write their information to a list. After, we close the output stream objects and catch exceptions as necessary.

The final class in this program is the tester application friends.

```java
import java.io.*;

public class friends {
    public static void main(String[] args) {
        friendsDB friendsList = new friendsDB();
        Scanner input = new Scanner(System.in);
        String firstName, lastName;
        int choice = 0, order = 0;
```

We first define the main method and declare variables: a friendsDB object which has the friends list on it and is initialized. We have 2 string variables for first and last name. Lastly there are 2 int variables for choice and order which dictate if they want to add, remove or find friends(choice) or the order in which they want to display names(order). These 2 were initialized because we use a do-while loop to allow the user to do as many actions as they want before quitting.

```java
do {
    System.out.println("========================================== FRIENDS LIST ==========================================");
    System.out.println("Select one of the following options");
    System.out.println("(1) Display Friends\n(2) Add Friend\n(3) Remove Friend\n(4) Find Friend\n(0) Quit Application");

    choice = input.nextInt();

    if (choice == 1) {
        System.out.println("\nLast Name (0)\nFirst Name (1)");
        order = input.nextInt();
        friendsList.displayFriends(order);
    } else if (choice == 2) {
        friendsList.addFriend();
    } else if (choice == 3) {
        System.out.println("Enter your friend's first name: ");
        firstName = input.next();
        System.out.println("Enter your friend's last name: ");
        lastName = input.next();
        friendsList.deleteFriend(firstName, lastName);
    } else if (choice == 4) {
        System.out.println("Enter your friend's first name: ");
        firstName = input.next();
        System.out.println("Enter your friend's last name: ");
        lastName = input.next();
        friendsList.findFriend(firstName, lastName);
    } else if(choice == 0) {
        System.out.println("You have successfully quit the application");
    }
} while (choice != 0);
```

Finally a do while we show the user multiple options and ask them to choose an option.

We then have a series of if else statements:
-if they select 1 we display friends and so we ask them how they want to display their friends. Depending on what they select we call the displayFriends method with the relevant order.
-if they select 2 we call the addFriend method from the database object.
-if they select 3 we prompt for full name and then attempt to delete a friend from the list with their given name.
-if they select 4 we prompt for full name and then attempt to find a friend from the list with their given name.
-if they select 0 we quit the program and break the do-while loop.