Computer Science 115
Assignment 1
Due Date: January 23, 2011

Objectives

- Exposure to a simple specification
- Exposure to using command line arguments
- Practice with objects
- Practice with file input and output

Introduction

In this assignment you will implement a simple contact manager. Your contact manager will store names and phone numbers.

Your contact manager will use a text file to store the contacts and will support four operations: add, remove, list and look-up.

Contact Storage

In order for our program to "remember" contacts, the contact information needs to be stored somewhere persistent. In this assignment we will use a text file for storage.

The text file will be named: `contacts.txt`

The file contains a single contact on each line. The first thing on the line is the name, followed by a single space, followed by a 10 digit phone number.

For example, if two contacts are added, the file would contain:

```
bob 6045551212
sue 6045551213
```

Note: Our program is not capable of storing full names that contain spaces. You cannot add the contact "john smith".

Input Validation

In order to simplify the assignment, we will make the assumption that all input provided to the program is valid. **The result of running the program with invalid input is undefined.**

Later in the course we will use exceptions to gracefully handle unexpected input.

Step 0 : Get the sample code

Download the three sample files from the course web page. Compile all the files by typing:

```
javac ContactManager.java
```

Run the resulting program by typing:

```
java ContactManager
```

You should see:

```
Usage:
  ContactManager add {name} {phone number}
  ContactManager list
  ContactManager remove {name}
  ContactManager lookup {name}
```

This output is telling you how to use the program. For example, if you want to add a new contact named jane with phone number 6045551212, you would type:

```
java ContactManager add jane 6045551212
```

Try all 4 possible operations – note that the program runs, it just doesn't do what we want (yet!).

Step 1 : Understand the requirements

There are three main components of this assignment:

Provide the implementation for a simple class: `Contact.java`

An instance of the Contact class stores the name and phone number associated with the contact. You need to complete the constructor and provide the implementation for the 3 methods.

Have a look at `Point.java` that was discussed in class this week – the ideas are very similar.

Create an array of Contact objects in ContactList.java

At this point in the course, the only way we can store a list of contacts is to use an array. Since no ordering is required, you can just add elements to the end of the array.

An easy way to remove elements from the array is to find the index of the contact to remove and overwrite that contact with the last contact in the list. You need a special case to handle the situation where there is only one element in the list.

Searching and printing the contents of an array should be review for most people – if it is not, please ask your instructor, lab instructor or a consultant for help.

Read and write contact information to and from a text file

When you create an instance of `ContactList`, populate the `ContactList` with the contacts in `contacts.txt`

When the `save` method of `ContactList` is invoked, erase the existing `contacts.txt` and write the current list to a newly created `contacts.txt`

You've been provided with the java code `FileExample.java` that illustrates how to read from and write to text files.

Step 2 : Read the code

Read the comments (yes, really!) in `Contact.java` and `ContactList.java`.

The comments define the expected behaviour of the classes used in this assignment. We will see later in the course that Java supports a special style of commenting that makes specification of behaviour even more precise.

Read the comments and the code in `FileExample.java`

Don't worry if you don't understand all the comments the first time you read them.

Step 3 : Understand Java String basics

A reminder that you cannot compare String objects using the == operator. The correct way to compare String objects is to use the `.equals` method:

```
String s1 = new String(“jason”);
String s2 = new String(“was here”);
if (s1.equals(s2))
{
      System.out.println(“yes”);
}
```

The easiest way to perform file input in Java is to read the file one line at a time. This means we need some way to split the line into the name and the phone number. Two string methods will be useful for this: `indexOf` and `substring`

```
String s3 = new String(“joeblow”);
s3.indexOf('b');  // this returns 3
s3.substring(1,3); // this returns the string “oeb”
```

For more information about the operations possible on Strings, see:

http://java.sun.com/j2se/1.5.0/docs/api/java/lang/String.html

Step 4: Implement and Test

I don't recommend you try to implement the entire assignment at once – choose small parts of the assignment to implement and then test to make sure that part works. The easiest place to start is to complete the implementation of `Contact.java`.

Your instructor will be giving out more “hints” during lecture this week.

Submission

Submit your files `Contact.java` and `ContactList.java` via connex.

Grading

If you submit something that does not compile you will receive 0 for the assignment.

Your assignment will be marked by a program, not a person. Do NOT change any of the method names provided in `Contact.java` and `ContactList.java`. You are free to add additional methods as you see fit.

You submitted something that compiles	1 mark
You implemented `Contact.java` correctly	3 marks
You implemented `ContactList.java` correctly	
add, list, remove, lookup implemented correctly	5 marks
file input and output implemented correctly	5 marks