

# CS 101: Introduction to Computer Science

## Project Six - Address Book

Student Name    Mohammad El-Abid

Student ID

Professor        Yasser Elleithy

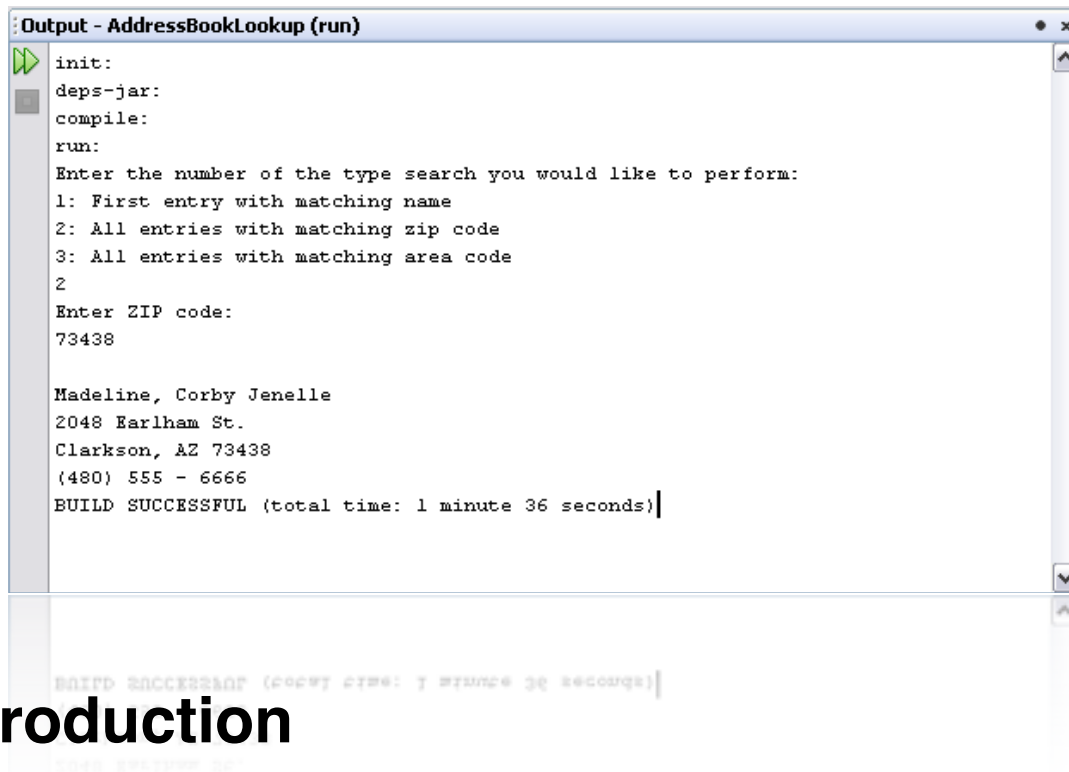
Date              11/11/11

# Table of Contents

<a href="#"><u>Abstract</u></a>	page 3
<a href="#"><u>Introduction</u></a>	page 3
<a href="#"><u>Screenshots</u></a>	page 4
<a href="#"><u>Code</u></a>	page 6
<a href="#"><u>Conclusion</u></a>	page 16
<a href="#"><u>Works Used</u></a>	page 16

# Abstract

This application loads contacts from a text file. These contacts are stored as Entry objects inside of an AddressBook object. The AddressBook is responsible for searching the contacts and parsing the contacts via AddressBook(Scanner file). We we're giving the follow screenshot of an expected output.



```
init:
deps-jar:
compile:
run:
Enter the number of the type search you would like to perform:
1: First entry with matching name
2: All entries with matching zip code
3: All entries with matching area code
2
Enter ZIP code:
73438

Madeline, Corby Jenelle
2048 Earlham St.
Clarkson, AZ 73438
(480) 555 - 6666
BUILD SUCCESSFUL (total time: 1 minute 36 seconds)
```

## Introduction

This application was a bit more complex than usual and required some planning. First I created the driver class and had it construct an AddressBook using the default constructor. Then created the AddressBook class and declared an array of entries. Then I began parsing the file and printing out the data. I then began stubbing out the Entry class. I added the toString method and the fields required, but made them public.

I then went back to the AddressBook class and had it create the Entry objects instead of printing out the parsed data. I then added the search methods by looping through the entries and “pushing” it to an array. Searching by a name was much more simple, if the firstName is not blank, check if it matches. It does this for all the “names” (first, middle, last), allowing for findByName(“Madeline”, “”, “”) to match “Madeline, Corby Jenelle”.

I then went back and made some improvements on the Entry class, such as formatting the phone number, getters, and setters. Since I was done early, I also came back and added a selection sort to the AddressBook class, based on the Entry’s last name.

# Screenshots

Application output, before sort:

```
[exec:exec]
Enter the number for the type of search you would like to perform:
0: All entries with matching name
1: First entry with matching name
2: All entries with matching zip code
3: All entries with matching area code
Input: 2
Enter zip code: 11048

Slyva, Santana Marie
128 Binoway St.
Leafton, MA 11048
(617)-222-1111

William, Herrold Michael
31415 Pieman Lane
Leafton, MA 11048
(617)-888-3333

Laurens, Gruman Franke
1024 Ibiem Road
Leafton, MA 11048
(781)-777-7777

-----
BUILD SUCCESS
-----
```

Application output, after sort:

```
[exec:exec]
Enter the number for the type of search you would like to perform:
0: All entries with matching name
1: First entry with matching name
2: All entries with matching zip code
3: All entries with matching area code
Input: 2
Enter zip code: 11048

Laurens, Gruman Franke
1024 Ibiem Road
Leafton, MA 11048
(781)-777-7777

Slyva, Santana Marie
128 Binoway St.
Leafton, MA 11048
(617)-222-1111

William, Herrold Michael
31415 Pieman Lane
Leafton, MA 11048
(617)-888-3333

-----
BUILD SUCCESS
-----
```

# Code

## File: Driver.java

```
package edu.bridgeport.addressbook;

import java.io.FileNotFoundException;
import java.util.logging.Level;
import java.util.logging.Logger;
import java.util.Scanner;
import java.io.FileReader;

public class Driver
{
    private static Scanner in = new Scanner(System.in);

    public static void main( String[] args )
    {
        try {
            Scanner file = new Scanner(new
            FileReader("addressbook.txt"));
            AddressBook book = new AddressBook(file);
            file.close();

            listOptions();

            int opt = getOption();

            switch(opt){
                case 0:
                    getAllByName(book);
                    break;
                case 1:
                    getFirstByName(book);
                    break;
                case 2:
                    getAllByZipCode(book);
                    break;
                case 3:
                    getAllByAreaCode(book);
                    break;
            }

            } catch (FileNotFoundException ex) {
                System.err.println("Unable to load contacts file!");

                Logger.getLogger(Driver.class.getName()).log(Level.SEVERE, null,
                ex);
            }
        }
    }
}
```

```

    }
}

private static void listOptions(){
    println("Enter the number for the type of search you
would like to perform:");
    println("0: All entries with matching name");
    println("1: First entry with matching name");
    println("2: All entries with matching zip code");
    println("3: All entries with matching area code");
}

private static int getOption(){
    String temp = "";
    while(!temp.matches("[0-3]$")){
        if(!temp.isEmpty()){ println("Please enter a valid
option (0-3)."); }
        System.out.print("Input: ");
        temp = in.nextLine();
    }
    return Integer.parseInt(temp);
}

private static String getString(String name){
    System.out.print("Enter " + name + ": ");
    return in.nextLine();
}

private static void println(String st){
    System.out.println(st);
}

private static void outputEntries(Entry[] entries){
    System.out.println();

    if(entries.length > 0){
        for(int i = 0; entries.length > i; i++){
            System.out.println(entries[i]);
            System.out.println();
        }
    } else {
        println("Unable to find any entries.");
    }
}

private static void getAllByZipCode(AddressBook book){
    String zipCode = getString("zip code");
    Entry[] contacts = book.findAllByZip(zipCode);
    outputEntries(contacts);
}

```

```

    }

    private static void getAllByAreaCode(AddressBook book){
        String areaCode = getString("area code");
        Entry[] contacts = book.findAllByAreaCode(areaCode);
        outputEntries(contacts);
    }

    private static void getAllByName(AddressBook book) {
        String firstName = getString("first name (blank to ignore)");
        String lastName = getString("last name (blank to ignore)");
        String middleName = getString("middle name (blank to ignore)");

        Entry[] contacts = book.findAllByName(firstName,
        lastName, middleName);
        outputEntries(contacts);
    }

    private static void getFirstByName(AddressBook book) {
        String firstName = getString("first name (blank to ignore)");
        String lastName = getString("last name (blank to ignore)");
        String middleName = getString("middle name (blank to ignore)");
        try {
            Entry contact = book.findByName(firstName, lastName,
            middleName);
            System.out.println(contact);
        } catch (Exception ex) {
            System.out.println("Unable to find contact!");
        }

        Logger.getLogger(Driver.class.getName()).log(Level.SEVERE, null,
        ex);
    }
}

```

## File: AddressBook.java

```

package edu.bridgeport.addressbook;

import java.util.Scanner;

public class AddressBook {
    private Entry[] entries;

```



```

    public AddressBook(Scanner file){
        loadData(file);
    }

    public Entry[] getContacts(){
        return entries;
    }

    public Entry findByName(String firstName, String lastName,
String middleName) throws Exception{
        for(int i = 0; entries.length > i; i++){
            Entry entry = entries[i];
            if(
                (firstName.isEmpty() ||
entry.getFirstName().equals(firstName)) && // if first name is
specified, see if match
                (lastName.isEmpty() ||
entry.getLastName().equals(lastName)) &&
                (middleName.isEmpty() ||
entry.getMiddleName().equals(middleName))
            ){
                return entry;
            }
        }

        throw new Exception("Unable to find a contact by supplied
name");
    }

    public Entry[] findAllByName(String firstName, String
lastName, String middleName){
        Entry[] results = new Entry[0];

        for(int i = 0; entries.length > i; i++){
            Entry entry = entries[i];
            if(
                (firstName.isEmpty() ||
entry.getFirstName().equals(firstName)) && // if first name is
specified, see if match
                (lastName.isEmpty() ||
entry.getLastName().equals(lastName)) &&
                (middleName.isEmpty() ||
entry.getMiddleName().equals(middleName))
            ){
                results = pushEntryToArray(results, entry);
            }
        }
    }

```

```

        return results;
    }

    public Entry[] findAllByAreaCode(String areaCode) {
        Entry[] results = new Entry[0];

        for(int i = 0; entries.length > i; i++){
            Entry entry = entries[i];
            if(entry.getPhoneNumber().startsWith(areaCode)){
                results = pushEntryToArray(results, entry);
            }
        }

        return results;
    }

    public Entry[] findAllByZip(String searchZipCode){
        Entry[] results = new Entry[0];

        for(int i = 0; entries.length > i; i++){
            Entry entry = entries[i];
            if(entry.getZipCode().equals(searchZipCode)){
                results = pushEntryToArray(results, entry);
            }
        }

        return results;
    }

    private static Entry[] pushEntryToArray(Entry[] array, Entry
add) {
        Entry[] temp = new Entry[array.length + 1];
        System.arraycopy(array, 0, temp, 0, array.length);
        temp[array.length] = add;
        return temp;
    }

    private void loadData(Scanner file){
        int numOfLines = file.nextInt();
        entries = new Entry[numOfLines];

        for(int i = 0; numOfLines > i; i++){
            entries[i] = new Entry();

            entries[i].setFirstName(file.next());
            entries[i].setLastName(file.next());
            entries[i].setMiddleName(file.next());
        }
    }

```

```

        file.nextLine(); // move pointer to next line

        entries[i].setStreetAddress(file.nextLine().trim());
        entries[i].setCity(file.next());
        entries[i].setState(file.next());
        entries[i].setZipCode(file.next());

        file.nextLine(); // move pointer to next line

        entries[i].setPhoneNumber(file.nextLine().trim());
    }
}

/**
 * Selection sort the entries
 */
public void sortEntries(){
    Entry[] sortedList = new Entry[entries.length];
    Entry[] entryList = new Entry[entries.length];
    int currentIndex = 0;

    // copy the array so we can mangle the data
    System.arraycopy(entries, 0, entryList, 0,
entries.length);
    // Because we can't set primitives to null, we will just
    set -1 as our "null"
    int matchedIndex = -1;
    // Event flag
    boolean allDone = false;

    while(!allDone){
        // loop over the entries
        for(int i = 0; entryList.length > i; i++){
            // ignore null entires, ones that are already
matched
            if(entryList[i] == null){
                continue;
            } else {
                // if there is nothing to compare to, set
current matched index to current iteration
                if(matchedIndex == -1){
                    matchedIndex = i;
                } else {
                    // if there is something to compare, use
String#compareTo to see which last name comes
                    // lexograpically first.

                    if(entryList[matchedIndex].getFirstName().compareTo(entryList[i].
getFirstName()) > 0){

```

```

        matchedIndex = i;
    }
}
}

    if(matchedIndex == -1){
        // if nothing was matched, it was all null
        values, thus we're done
        allDone = true;
    } else {
        // push the lowest value of entryList
        sortedList[currentIndex++] =
entryList[matchedIndex];
        // remove that entry from entryList
        entryList[matchedIndex] = null;
        // reset matchedIndex to our "null" value
        matchedIndex = -1;
    }
}
entries = sortedList;
}
}

```

## File: Entry.java

```

package edu.bridgeport.addressbook;

public class Entry {
    private String firstName, middleName, lastName;
    private String streetAddress, city, state, zipCode;
    private String phoneNumber;

    // constructors
    public Entry(){

        public Entry(String firstName, String lastName, String
middleName, String streetAddress, String city, String state,
String zipCode, String phoneNumber){
            setFirstName(firstName);
            setLastName(lastName);
            setMiddleName(middleName);
            setStreetAddress(streetAddress);
            setCity(city);
            setState(state);
            setZipCode(zipCode);
            setPhoneNumber(phoneNumber);
        }
    }
}

```

```

// formatting

public String formatPhoneNumber() {
    // Strip any non-numeric characters from the string
    String stripped = phoneNumber.replaceAll("[^0-9]", "");

    java.text.MessageFormat phoneMsgFmt = new
java.text.MessageFormat("({0})-{1}-{2}");
    String[] phoneNumArr={stripped.substring(0, 3),
        stripped.substring(3,6),
        stripped.substring(6)};

    return(phoneMsgFmt.format(phoneNumArr));
}

@Override
public String toString(){
    String build = new String();
    build += String.format("%s, %s %s\n", firstName,
lastName, middleName);
    build += String.format("%s\n", streetAddress);
    build += String.format("%s, %s %s\n", city, state,
zipCode);
    build += formatPhoneNumber();
    return build;
}

public String toFile(){
    String build = new String();
    build += String.format("%s %s %s\n", firstName, lastName,
middleName);
    build += String.format("%s\n", streetAddress);
    build += String.format("%s %s %s\n", city, state,
zipCode);
    build += String.format("%s", phoneNumber);
    return build;
}

// Getters

public String getFirstName(){
    return firstName;
}

public String getMiddleName(){
    return middleName;
}

public String getLastName(){

```

```

        return lastName;
    }

    public String getStreetAddress(){
        return streetAddress;
    }

    public String getCity(){
        return city;
    }

    public String getState(){
        return state;
    }

    public String getZipCode(){
        return zipCode;
    }

    public String getPhoneNumber(){
        return phoneNumber;
    }

    // setters

    public boolean setFirstName(String newFirstName){
        firstName = newFirstName;
        return true;
    }

    public boolean setMiddleName(String newMiddleName){
        middleName = newMiddleName;
        return true;
    }

    public boolean setLastName(String newLastName){
        lastName = newLastName;
        return true;
    }

    public boolean setStreetAddress(String newStreetAddress){
        streetAddress = newStreetAddress;
        return true;
    }

    public boolean setCity(String newCity){
        city = newCity;
        return true;
    }
}

```

```

    public boolean setState(String newState){
        state = newState;
        return true;
    }

    public boolean setZipCode(String newZipCode){
        zipCode = newZipCode;
        return true;
    }

    public boolean setPhoneNumber(String newPhoneNumber){
        // ensure format of ddd ddd dddd
        if(newPhoneNumber.matches("[0-9]{3}\\s[0-9]{3}\\s[0-9]{4}$")){
            phoneNumber = newPhoneNumber;
            return true;
        } else {
            return false;
        }
    }
}

```

## addressbook.txt

```

5
Slyva Santana Marie
128 Binoway St.
Leafton MA 11048
617 222 1111
Evan Parsival Sunny
32 East Pleasant St.
Arbuelo AZ 73421
480 444 9999
William Herrold Michael
31415 Pieman Lane
Leafton MA 11048
617 888 3333
Madeline Corby Jenelle
2048 Earlham St.
Clarkson AZ 73438
480 555 6666
Laurens Gruman Franke
1024 Ibiem Road
Leafton MA 11048
781 777 7777

```

# Conclusion

This application was a bit challenging due to the pushing of results and formatting phone numbers. I used the `MessageFormat` class and `String#substring` to format the phone number in the (ddd)-ddd-dddd format. It also required a bit of critical thinking about how to store the zip code and phone number. A number would be more efficient than a string, however less capable. I also implemented a selection sort algorithm, while it does then make the search results appear in alphabetical order, but the reason I did the sort algorithm was to challenge myself.

# Works Used

[Stack Overflow - Java Phone Number Format API](#)

[Stack Overflow - How Do I Format a Phone Number as a String in Java](#)

[WikiMedia Commons - Selection Sort Algorithm Animation](#)