Object-Oriented Design Document for a Personal Address and Phone Book

Part 2: Detailed Design

Prepared for C-S 341: Software Engineering

By group #: 3

Group Members:

Kasi Periyasamy
Tom Gendreau
Dave Riley

November 03, 2003

1. About this document

This document describes a detailed object-oriented design for a personal address and phone book. The architectural design for the phone book is given in [4] and the requirements are given in [1,2,3]. The object-oriented design is described using a collection of class definitions where each class definition includes structural and behavioral properties. This document describes only those classes corresponding to the functional behavior and do not describe the classes corresponding to the graphical user interface. By keeping the two sets of classes separated, a designer has the freedom to change the GUI without affecting the functional behavior.

2. Design Decisions

The following decisions are made during the design process:

1. Phone numbers may be entered/retrieved in only one of the following formats; any phone number not entered in one of these formats will be flagged as an error:
   7 digits – for local phone numbers
   10 digits – for long distance phone numbers
   15 digits – for international phone numbers; if an international number is less than 15 digits long, zeroes will be padded at the front of the phone number while entering the number; however, these zeroes will not show up when the number is retrieved or displayed.
2. Both last name and first name are represented as strings; the number of characters in first name and in last name are limited to 80 characters.
3. An address will be stored and retrieved as one string (maximum 256 characters) per line.
4. Date will be chosen from the system and will be entered, stored and retrieved in the MM/DD/YYYY format.
5. Each appointment will be entered and retrieved as one string (maximum 256 characters).

3. Format of a Class Definition

Each class definition is given in the following format:

Class name: must be unique within the entire design.

Attributes or instance variables:
each variable is given in following format:
<visibility> <type> <name>
where <visibility> is “public” or “private”.

Unless otherwise indicated, there will be a getX() method and a setX() method for each private attribute X.

Methods

Each method will be classified as "public" or "private".
Each method is given in the following structural format:

Name of the method – must be unique within the class
Synopsis of the method – calling syntax for the method
Purpose of the method – a short description of the functionality implemented by this method
Visibility – public or private
Input parameters – a set of parameters in <type><name> format
Output parameter – in <type><name> format
Local variables – a set of variables used in describing the pseudocode (given next) for this method
Pseudocode – an algorithmic (structured) description of the method
Exceptions – a set of exceptions that might arise in executing this method and their corresponding corrective actions
Remarks – additional information about this method and hints for the programmers; typically, it may include the design decisions taken and choices of implementation that the programmer may consider

4. Class definitions

Classname: PhoneNumber

Attributes:

private	Integer	number	
private	Boolean	type	/* "true" indicates phone number "false" indicates fax number */

/ There are no setX() methods for the two private variables listed above. However, there are implicit getX() methods for the variables. **/**

Methods

Name:	CreatePhoneNumber
Synopsis:	ph ← CreatePhoneNumber (num, numType, phoneOrFax)
Purpose:	To create a new phone number.
Visibility:	public

Input parameters:

Integer	num
Integer	numType

```
                        Boolean         phoneOrFax
Output parameters:      PhoneNumber         ph
Local variables:        None
Pseudocode:
        /* verify the format of 'num' */
        if (numType = 1)
                if (numberOfDigits(num) ≠ 7)          /* local */
                        throw PhoneNumberFormatException;
        if (numType = 2)
                if (numberOfDigits(num) ≠ 10)         /* long distance */
                        throw PhoneNumberFormatException;
        if (numType = 3)
                if (numberOfDigits(num) < 11 OR
                    numberOfDigits(num) >  15)       /* international */
                        throw PhoneNumberFormatException;
        if (numType < 1 OR numType > 3)
                throw PhoneNumberFormatException;
        ph.number ← num;
        ph.type ← phoneOrFax;
Exceptions:
        PhoneNumberFormatException – display the error message and ask the user
        to enter the phone number again.
Remarks:        None
                --------*--------------*--------------*------------

Name:           ChangePhoneNumber
Synopsis:       ChangePhoneNumber (newNumber)
Purpose:        To change the number portion of the phone number
Visibility:     public
Input parameters:       Integer         newNumber
Output parameters:      None
Local variables:        None
Pseudocode:
        /* new number must belong to the same category as the old number */
        if (numberOfDigits (number) = 7 AND numberOfDigits (newNumber) ≠ 7)
                throw PhoneNumberFormatException;
        else if (numberOfDigits(number) = 10 AND
                numberOfDigits (newNumber) ≠ 10)
                        throw PhoneNumberFormatException;
        else if (11 <= numberOfDigits (number) <= 15 AND
                NOT (11 <= numberOfDigits (number) <= 15)
                        throw PhoneNumberFormatException;
        /* number of digits in new number may be different from that in the old
           number, but  both must be in the same range. */
        number ← newNumber;
```

Exceptions:
PhoneNumberFormatException – display the error message and ask the user to enter the phone number again.
Remarks: None

--------*--------------*--------------*------------

Name: numberOfDigits
Synopsis: count ← numberOfDigits (num)
Purpose: To return the number of digits in an integer
Visibility: private
Input parameters: Integer num
Output parameters: Integer count
Local variables: None
Pseudocode:

```
count ← 0;
if (num ≠ 0) count ← 1 + numberOfDigits (num / 10);
```

Exception: None
Remarks: None

-------------*--------------*--------------*-----------

Class name: PhoneEntry
Attributes:

private	String	lastname, firstname	
private	Array [1..L] of String	address	/* L lines of address */
private	Array [1..N] of PhoneNumber	numbers	/* N phone numbers */

/* 'L' and 'N' must be chosen by the implementer */
/ There are no setX() methods for any of the three private variables above. But there are implicit getX() methods for all the three variables **/**

Methods:

Name: CreatePhoneEntry
Synopsis: phEntry ← CreatePhoneEntry (last, first, adr, nums)
Purpose: To create a new phone entry.
Visibility: public

Input parameters:	String	last, first
	Array [1..K] of String	adr
	Array [1..M] of PhoneNumber	nums
Output parameters:	PhoneEntry	phEntry
Local variables:	Integer	i

Pseudocode:

```
/* verify the length restrictions for names and address */
if (length(last) > 80) throw NameLengthException;
if (length(first) > 80) throw NameLengthException;
if (K > L) throw AddressLineLimitException;
if (M > N) throw PhoneNumberLimitException;
```

```
for i = 1 to K
        if (length(adr[i]) > 256) throw AddressLineLengthException;
phEntry.lastname ← last;
phEntry.firstname ← first;
for i = 1 to K    phEntry.address[i] ← adr[i];
for i = 1 to M   phEntry.numbers[i] ← nums[i];
```

Exceptions:

NameLengthException – ask the user to re-enter the name again.
AddressLineLengthException – ask the user to re-enter the address again
AddressLineLimitException – warn the user that the lines after the limit will not be included in the phone book
PhoneNumberLimitException – warn the user that the extra phone numbers will not be included in the phone book

Remarks: This method assumes that the individual phone numbers in the array 'nums' have been created already and hence their format has been validated.

---------*--------------*---------------*------------

Name: IsPhoneNumberPresent
Synopsis: answer ← IsPhoneNumberPresent (phNumber)
Purpose: To check whether or not a given phone number exists in the this phone entry.
Visibility: public
Input parameters: PhoneNumber phNumber
Output parameters: Boolean answer
Local variables: Integer i
Pseudocode:

```
answer ← false;
i ← 1;
while (NOT (answer) AND  (i <= length (numbers)) {
        if (numbers[i] = phNumber) answer ← true;
        i ← i + 1;
}
```

Exceptions: None
Remarks: This method assumes that the array 'numbers' is not sorted.

---------*--------------*---------------*------------

Name: AddPhoneNumber
Synopsis: AddPhoneNumber (phNumber)
Purpose: To add a phone number to the phone entry.
Visibility: public
Input parameters: PhoneNumber phNumber
Output parameters: None
Local variables: Integer i
Pseudocode:

```
/* Ensure that the phone number does not exist in this entry */
```

```
        if (isPhoneNumberPresent (phNumber))
            throw PhoneNumberExistsException;
    /* the 'if' condition above checks  for equality of number and type;
       if the numbers are same but the types are different, this condition
       will consider them as two different numbers */
    if (length (numbers) = N) throw PhoneNumberLimitException;
    numbers ← numbers + phNumber;   /* '+' here indicates adding an entry
                                       into an array. */
```

Exception:
    PhoneNumberExistsException – display the appropriate error message.
    PhoneNumberLimitException – warn the user that the limit for maximum number of phone numbers has exceeded and so the new number will not be included in the phone book.

Remarks: The pseudocode uses '+' to add an item to an array. The implementer can choose to add at the end of the array, or at the beginning of the array or sort the array and insert the new number at appropriate place.

-----------*----------------*--------------*---------------

Name:	DeletePhoneNumber	
Synopsis:	DeletePhoneNumber (phNumber)	
Purpose:	To delete a phone number from this entry	
Visibility:	public	
Input parameters:	PhoneNumber	phNumber;
Output parameters:	None	
Local variables:	Integer	i

Pseudocode:

```
    if (NOT(isPhoneNumberPresent(phNumber)))
        throw  PhoneNumberNotExistException;
    numbers ← numbers – phNumber;  /* '-' here indicates removing an
                                      element from the array; may require to find
                                      the index of the element to be removed */
```

Exceptions:
    PhoneNumberNotExistException – display the appropriate error message.

Remarks:
    The pseudocode uses the '-' sign to remove an element from the array. Similar to '+', the implementer is responsible for writing the code to find the index of the element to be removed and update the array.

---------*--------------*---------------*------------

Class name: PhoneDiary

Attributes:

private	Array [1..N] of PhoneEntry	entries;

/* Implementer must choose 'N' */
/ There are no implicit setX() or getX() methods for this attribute. **/**

Methods:

Name: InitializePhoneDiary
Synopsis: InitializePhoneDiary()
Purpose: To initialize the entries of the phone diary with null.
Visibility: public
Input parameters: None
Output parameters: None
Local variables: Integer i
Pseudocode:

```
for i = 1 to N  entries[i] ← null;
```

Exceptions: None
Remarks: "null" must be defined by the implementer.

---------*--------------*---------------*------------

Name: AddEntry
Synopsis: AddEntry (lname, fname, adr, phones)
Purpose: To add an entry in the phone diary.
Visibility: public
Input parameters: String lname, fname
Array [1..K] of String adr
Array [1..M] of PhoneNumber phones
Output parameters: None
Local variables: PhoneEntry phEntry;
Pseudocode:

```
phEntry ← phEntry.createPhoneNumber (lname, sname, adr,
        phones);
entries ← entries + phEntry;  /* '+' denotes adding into an array */
```

Exceptions: None
Remarks: The '+' sign in the pseudocode indicates insertion of a member into an array; the implementer may choose the right spot to insert the member.

----------------*----------------*-----------------*------------

Name: SearchEntry
Synopsis: phEntry ← SearchEntry (lname)
Purpose: To retrieve an entry based on last name.
Visibility: public
Input parameters: String lname
Output parameters: PhoneEntry phEntry
Local variables: Boolean flag
Integer i
Pseudocode:

```
        flag ← false;
        i ← 1;
        while (NOT flag AND (i <= length(entries)) {
                flag ← entries[i].getLastname() = lname;
                i ← i + 1;
        }
        if (i <= length(entries)) phEntry ← entries[i];
        else phEntry ← null;
```

Exceptions: None
Remarks: None

------------*-------------*--------------*----------

Name: DeleteEntry
Synopsis: DeleteEntry (lname)
Purpose: To delete an entry; last name is provided
Visibility: public
Input parameters: String lname
Output parameters: None
Local variables: None
Pseudocode:

```
        if (SearchEntry (lname) = null) throw NameNotExistException;
        else   entries ← entries – SearchEntry (lname);
```

Exception:
NameNotExistException – display the error message and terminate the method

Remarks: The negative sign in the pseudocode indicates that the member is removed from the array. Implementer may choose to find the index of the corresponding entry and then update the array

-----------*-------------------*------------------*-----------

Name: ModifyEntry
Synopsis: ModifyEntry (lname, adr, phones)
Purpose: To modify an entry by overwriting the address and phone numbers
Visibility: public

Input parameters:	String	lname
	String	adr
	Array [1..P] of PhoneNumber	phones
Output parameters:	None	
Local variables:	PhoneEntry	phEntry
	Integer	i

Pseudocode:

```
        if (SearchEntry (lname) = null) throw NameNotExistExceptions;
        else {
                phEntry ← SearchEntry (lname);
                entries ← entries – SearchEntry (lname);
```

```
            phEntry.adr ← adr;
            for i = 1 to P  PhEntry.AddPhoneNumber (phones[i]);
            entries ← entries + phEntry;
        }
```

Exceptions:
NameNotExistException – display the error message and terminate the method

Remarks: The size of the array ‘P’ must be decided by the user.
The ‘-‘ sign in the pseudocode indicates deleting an element from an array.

---------------*----------------*-----------------*----------

Name: ListEntries
Synopsis: ListEntries (phone)
Purpose: To list all the entries which have the phone number that is passed as the input parameter
Visibility: public

Input parameters:	PhoneNumber	phone
Output parameters:	Array [1..K] of PhoneEntry	outEntries;
Local variables:	Integer	count, i

Pseudocode:

```
    count ← 0;
    for i = 1 to length (entries) {
        if (entries[i].isPhoneNumberPresent (phone)) {
            count ← count + 1;
            outEntries[count] ← entries[i];
        }
    }
```

Exceptions: None
Remarks: The output may be displayed by another method separately.

----------------*-----------------*-----------------*-------------

Name: AddPhoneNumber
Synopsis: AddPhoneNumber (lname, phone)
Purpose: To add a phone number to a particular entry
Visibility: public

Input parameters:	String	lname
	PhoneNumber	phone
Output parameters:	None	
Local variables:	PhoneEntry	phEntry

Pseudocode:

```
    if (SearchEntry(lname) = null) throw NameNotExistException;
    else {
        phEntry ← SearchEntry (lname);
        entries ← entries – SearchEntry (lname);
        phEntry.AddPhoneNumber (phone);
        entries ← entries + phEntry;
    }
Exceptions:
    NameNotExistException – display the error message and terminate
        the method
Remarks:    The '+' sign (and the '-' sign) in the pseudocode indicates
    adding (deleting) an element to (from) an array.
    ------------*----------------*-------------------*---------------
```

Name: DeletePhoneNumber
Synopsis: DeletePhoneNumber (lname, phone)
Purpose: To remove a phone number from a particular entry
Visibility: public

Input parameters:	String	lname
	PhoneNumber	phone
Output parameters:	None	
Local variables:	PhoneEntry	phEntry

```
Pseudocode:
    if (SearchEntry(lname) = null) throw NameNotExistException;
    else {
        phEntry ← SearchEntry (lname);
        entries ← entries – SearchEntry (lname);
        phEntry.DeletePhoneNumber (phone);
        entries ← entries + phEntry;
    }
Exceptions:
    NameNotExistException – display the error message and terminate
        the method
Remarks:    The '+' sign (and the '-' sign) in the pseudocode indicates
    adding (deleting) an element to (from) an array.
    ------------*----------------*-------------------*---------------
```

Class name: Appointment

Attributes:

private	Date	when
private	Hour	from, to
private	String	appointment
private	String	lastname

/* The class 'Date' will use the date class from the system.
The class 'Hour' is a rename of 'Integer'. The implementer may choose to add more constraints to this class, if desired.

All the three attributes have implicit getX() methods, but there are no setX() methods for any of the three attributes. The getX() methods will not be included in the design document.
*/

Methods:

Name:	CreateAppointment	
Synopsis:	CreateAppointment (date, start, end, appt,lname)	
Purpose:	To create a new appointment	
Visibility:	public	
Input parameters:	Date	date
	Hour	start, end
	String	appt
	String	lname
Output parameters:	Appointment	appoint
Local variables:	None	

Pseudocode:

```
If (NOT(validateDate (date)) throw InvalidDateException;
If (NOT(validateTime (start, end)) throw InvalidTimeException;
if (length(appt) > 256) throw AppointmentLengthLimitException;
if (lname ≠ null)
    if (length(lname) > 80) throw NameLengthLimitException;
appoint.when ← date;
appoint.from ← start;
appoint.to ← end;
appoint.appointment ← appt;
appoint.lastname ← lname;
```

Exceptions:

InvalidDateException – display the error message and ask the user to re-enter the date.

InvalidTimeException – display the error message and ask the user to re-enter the time.

AppointmentLengthLimitException – display a warning message to the user and ignore the information after 256 characters.

NameLengthLimitException – display a warning message to the user and Ignore the portion of the name beyond 80 characters.

Remarks: Last name is optional. If not specified in the input, a “null” value will be stored instead.

---------------*----------------*-----------------*----------------

Name:	ChangeAppointment	
Synopsis:	ChangeAppointment (newAppt)	
Purpose:	To change the appointment string	
Visibility:	public	
Input parameters:	String	newAppt

Output parameters: None
Local variables: None
Pseudocode:

```
appointment ← newAppt;
```

Exceptions: None
Remarks: None

------------*----------------*------------------*--------------

Class name: AppointmentCalendar

Attributes:

```
private Array [1..N] of Appointment            entries
/* ‘N’ must be chosen at the implementation time. */
```

Methods:

Name: InitializeAppointmentCalendar
Synopsis: InitializeAppointmentCalendar()
Purpose: To initialize the appointment calendar
Visibility: public
Input parameters: None
Output parameters: None
Local variable: Integer i
Pseudocode:

```
for i = 1 to N entries[i] ← null;
```

Exceptions: None
Remarks: Implementer must choose representation of “null”.

------------*---------------*-----------------*---------------

Name: SearchAppointment
Synopsis: SearchAppointment (date, start, end)
Purpose: To search for an appointment in the calendar
Visibility: public

Input parameters:	Date	date
	Hour	start, end
Output parameters:	Appointment	appt
Local variables:	Integer	i
	Boolean	flag

Pseudocode:

```
if (NOT(validateDate  (date)) throw InvalidDateException;
if (NOT(validateTime (start, end)) throw InvalidTimeException;
flag ← false;
i ← 0;
appt ← null;
while (NOT flag and (i <= length(entries)) {
    if (entries[i].when = date AND entries[i].from = start AND
        entries[i].to = end)  appt ← entries[i].appointment;
    i ← i + 1;
}
```

Exceptions:
InvalidDateException – display the error message and ask the user to re-enter the date.
InvalidTimeException – display the error message and ask the user to re-enter the time

Remarks: None

------------*---------------*-----------------*---------------

Name:	AddAppointment	
Synopsis:	AddAppointment (date, start, end, appt, lname)	
Purpose:	To add a new appointment to the calendar	
Visibility:	public	
Input parameters:	Date	date
	Hour	start, end
	String	appt
	String	lname
Output parameters:	None	
Local variables:	Appointment	appoint
	Integer	i

Pseudocode:

```
if (NOT(validateDate  (date)) throw InvalidDateException;
if (NOT(validateTime (start, end)) throw InvalidTimeException;
if (SearchAppointment (date, start, end) ≠ null) throw
        AppointmentAlreadyExistException;
appoint ← appoint.CreateAppointment (date, start, end, appt,lname);
entries ← entries + appoint;
```

Exceptions:
InvalidDateException – display the error message and ask the user to re-enter the date.
InvalidTimeException – display the error message and ask the user to re-enter the time
AppointmentAlreadyExistException – display the error message and terminate the method.

Remarks: The ‘+’ sign in the pseudocode indicates adding an element to n array.

----------*--------------------*----------------*------------

Name: DeleteAppointment
Synopsis: DeleteAppointment (date, start, end)
Purpose: To delete an appointment in the calendar
Visibility: public

Input parameters:	Date	date
	Hour	start, end
Output parameters:	None	
Local variables:	Integer	i

Pseudocode:

```
if (NOT(validateDate  (date)) throw InvalidDateException;
if (NOT(validateTime (start, end)) throw InvalidTimeException;
if (SearchAppointment (date, start, end) = null) throw
        AppointmentNotExistException;
entries ← entries -  SearchAppointment (date, start, end);
```

Exceptions:

InvalidDateException – display the error message and ask the user to re-enter the date.
InvalidTimeException – display the error message and ask the user to re-enter the time
AppointmentNotExistException –display the error message and terminate the method

Remarks: The '-' sign in the pseudocode indicates deleting an entry from an array.

----------*--------------------*----------------*-------------

Name: MoveAppointment
Synopsis: MoveAppointment (oldDate, oldStart, oldEnd, newDate, newStart, newEnd)
Purpose: Move the appointment at (oldDate, oldStart, oldEnd) to the place (newDate, newStart, newEnd)
Visibility: public

Input parameters:	Date	oldDate, newDate
	Hour	oldStart, oldEnd, newStart, newEnd
Output parameters:	None	
Local variables:	None	

Pseudocode:

```
if (NOT(validateDate  (oldDate)) throw InvalidDateException;
if (NOT(validateTime (oldStart, oldEnd)) throw InvalidTimeException;
if (SearchAppointment (oldDate, oldStart, oldEnd) = null) throw
        AppointmentNotExistException;
if (NOT(validateDate  (newDate)) throw InvalidDateException;
if (NOT(validateTime (newStart, newEnd)) throw InvalidTimeException;
if (SearchAppointment (newDate, newStart, newEnd) ≠ null) throw
        AppointmentAlreadyExistException;
        AddAppointment (newDate, newStart, newEnd,
                SearchAppointment (oldDate, oldStart, oldEnd));
        DeleteAppointment (oldDate, oldStart, oldEnd);
```

Exceptions:
InvalidDateException – display the error message and ask the user to re-enter the date.
InvalidTimeException – display the error message and ask the user to re-enter the time
AppointmentNotExistException –display the error message and terminate the method
AppointmentAlreadyExistException –display the error message and terminate the method

Remarks: None

----------*--------------------*----------------*-------------

Name: validateDate
Synopsis: validateDate (date)
Purpose: To check whether the input date is beyond the current date
Visibility private

Input parameters:	Date	date
Output parameters:	Boolean	answer
Local variables:	None	

Pseudocode:

```
        Answer ← (date >= currentDate());
```

Exceptions: None
Remarks: 'date' is chosen from the system and hence its format need not be verified.
currentDate() is a system function that returns the date from the system clock at the time of invocation.

----------------*-----------------*------------------*----------------

Name: validateTime
Synopsis: validateTime (start, end)
Purpose: To check the validity of time and the relationship between the two Parameters
Visibility: private

Input parameters:	Hour	start, end
Output parameters:	Boolean	answer
Local variables:	None	

Pseudocode:

```
        Answer ← (0 <= start <= 23) AND (0 <= end <= 23) AND (start < end);
```

Exceptions: None
Remarks: None

---------------*----------------*-----------------*-------------------

Class name: PhoneBook

Attributes:

```
public      PhoneDiary              phoneDiary
public      AppointmentCalendar     apptCalendar
/** There are no setX() or getX() methods for any of these attributes */
```

Methods

Name: InitializePhoneBook
Synopsis: InitializePhoneBook()
Purpose: To initialize the phone diary and the appointment calendar
Visibility: public
Input parameters: None
Output parameters: None
Local variables: None
Pseudocode:

```
phoneDiary.InitializePhoneDiary();
apptCalendar.InitializeAppointmentCalendar();
```

Exceptions: None
Remarks: None

---------*-------------------*---------------------*---------------

Name: SelectPhoneDiary
Synopsis: SelectPhoneDiary(phoneFilename)
Purpose: To read values of phone diary entries from file
Visibility: public
Input parameters: File phoneFilename
Output parameters: None
Local variables: Integer i
Pseudocode:

```
if (openfile(phoneFilename) = null) throw FileNotFoundException;
i ← 1;
while (NOT endOfFile (phoneFilename) AND (i <= N) {
    phoneDiary.entries[i] ← readRecord (phoneFilename);
    i ← i + 1;
}
close (phoneFilename);
```

Exceptions:
FileNotFoundException – display the error message to the user and Terminate the method.

Remarks:
'N' denotes the maximum number of entries the phone diary can hold; must be chosen by the implementer.

‘readRecord’ method assumes that the records stored in the file are in the same format as that of the entries in the phone diary. If there is a mismatch, the ‘readRecord’ method will display appropriate error messages; it will be left to the file handling mechanism of the chosen language and hence is the choice left to the implementer.

-------------*------------------*------------------*--------------

Name: SelectAppointmentCalendar
Synopsis: SelectAppointmentCalendar(calendarFilename)
Purpose: To read values of appointment calendar entries from file
Visibility: public
Input parameters: File calendarFilename
Output parameters: None
Local variables: Integer i
Pseudocode:

```
if (openfile(calendarFilename) = null) throw FileNotFoundException;
i ← 1;
while (NOT endOfFile (calendarFilename) AND (i <= N) {
    apptCalendar.entries[i] ← readRecord (calendarFilename);
    i ← i + 1;
}
close (calendarFilename);
```

Exceptions:

FileNotFoundException – display the error message to the user and Terminate the method.

Remarks:

‘N’ denotes the maximum number of entries the appointment calendar can hold; must be chosen by the implementer.
‘readRecord’ method assumes that the records stored in the file are in the same format as that of the entries in the appointment calendar. If there is a mismatch, the ‘readRecord’ method will display appropriate error messages; it will be left to the file handling mechanism of the chosen language and hence is the choice left to the implementer.

-------------*------------------*------------------*--------------

Name: WritePhoneDiary
Synopsis: WritePhoneDiary(phoneFilename)
Purpose: To write the phone diary back onto the file
Visibility: public
Input parameters: File phoneFilename
Output parameters: None
Local variables: Integer i
Pseudocode:

```
openfile (phoneFilename, write);       /* open for writing */
for i = 1 to length(phoneDiary.entries)
    writeRecord (phoneFilename, phoneDiary.entries[i]);
```

```
        close (phoneFilename);
```
Exceptions: None

Remarks:

‘opnefile (filename, write)’ will open a file for writing; it will re-initialize the file if it already exists.
‘writeRecord’ will write the entries in the same format as they exist in phoneDiary.

---------------*--------------------*---------------------*---------

Name: WriteAppointmentCalendar
Synopsis: WriteAppointmentCalendar(calendarFilename)
Purpose: To write the appointment calendar back onto the file
Visibility: public

Input parameters:	File	calendarFilename
Output parameters:	None	
Local variables:	Integer	i

Pseudocode:
```
        openfile (calendarFilename, write);   /* open for writing */
        for i = 1 to length(calendarDiary.entries)
                writeRecord (calendarFilename, apptCalendar.entries[i]);
        close (calendarFilename);
```
Exceptions: None

Remarks:

‘opnefile (filename, write)’ will open a file for writing; it will re-initialize the file if it already exists.
‘writeRecord’ will write the entries in the same format as they exist in appointment calendar.

---------------*--------------------*---------------------*---------

Name: RetrievePhoneEntry
Synopsis: RetrievePhoneEntry (date, start, end)
Purpose: To retrieve the phone diary entry corresponding to a name which is stored in one of the entries in the appointment calendar
Visibility: public

Input parameters:	Date	date
	Hour	start, end
Output parameters:	PhoneEntry	phEntry
Local variables:	String	name

Pseudocode:
```
        if (apptCalendar.SearchEntry (date, start, end) = null) throw
                AppointmentNotExistException;
        name ← (apptCalendar.SearchEntry (date, start, end)).lastname;
        if (name = null) throw NameFieldEmptyException;
        phEntry ← phoneDiary.SearchEntry (name);
```

Exceptions:
AppointmentNotExistException – display the error message to the user and terminate the method
NameFieldEmptyException – display the error message to the user and terminate the method
Remarks: None
--------------*----------------------*-------------------*-----------------

References:

1. Kasi Periyasamy, Tom Gendreau and Dave Riley, "Software Requirements Document for a Personal address and phone book - Part 1: Product Overview and Assumptions", September 2003.
2. Kasi Periyasamy, Tom Gendreau and Dave Riley, "Software Requirements Document for a Personal address and phone book - Part 2: Functional Requirements", October 2003.
3. Kasi Periyasamy, Tom Gendreau and Dave Riley, "Software Requirements Document for a Personal address and phone book - Part 3: GUI Requirements", October 2003.
4. Kasi Periyasamy, Tom Gendreau and Dave Riley, "Object-Oriented Design Document for a Personal address and phone book - Part 1: Architectural Design", November 2003.