UNIVERSITY AT ALBANY
State University of New York

COLLEGE OF ENGINEERING AND APPLIED SCIENCES
DEPARTMENT OF COMPUTER SCIENCE

ICSI311 Principles of Programming Languages

Assignment 03 Created by Qi Wang

Table of Contents

Part I: General information 02

Part II: Grading Rubric 03

Part III: Examples 04

Part IV: Description 06

Part I: General Information

- All assignments are individual assignments unless it is notified otherwise.
- All assignments must be submitted via Blackboard. No late submissions or e-mail submissions or hard copies will be accepted.
- Unlimited submission attempts will be allowed on Blackboard. **Only the last attempt will be graded.**
- Work will be rejected with no credit if
    - The work is late.
    - The work is not submitted properly (Blurry, wrong files, crashed files, files that can't open, etc.).
    - The work is a copy or partial copy of others' work (such as work from another person or the Internet).
- Students must turn in their original work. Any cheating violation will be reported to the college. Students can help others by sharing ideas and should not allow others to copy their work.
- Documents to be submitted:
    - **UML diagram**
    - **Java source file(s) with Javadoc style inline comments**
    - **Supporting files if any** (For example, files containing all testing data.)

  Note: Only the above-mentioned files are needed. Copy them into a folder, zip the folder, and submit the **zipped** file. We don't need other files from the project.
- Students are required to submit a design, all the error-free source files with Javadoc style inline comments and supporting files. Lack of any of the required items or programs with errors will result in a low credit or no credit.
- **Grades and feedback**: TAs will grade. Feedback and grades for properly submitted work will be posted on Blackboard. For questions regarding the feedback or the grade, students should reach out to their TAs first. Students have limited time/days from when a grade is posted to dispute the grade. Check email daily for the grade review notifications sent from the TAs. **Any grade dispute request after the dispute period will not be considered.**

Part II: Grading Rubric

The following includes, but not limited to, a list of performance indicators used for grading.

Performance Indicators		Max points	Earned points
#1: UML design	• Efficient and meet all requirements • Visibility, name, and type/parameter type/return type present for all classes with no issues • Class relationships are correct	5	
#2: Node	• Comments and naming conventions • A generic non-inner class and all methods present with no issues • No use of any node- related classes from any standard library • Meet all requirements	5	
#3: Generic ADT sorted list classes	• Comments and naming conventions • Specifications and implementation are separate • A generic class, a doubly linked list data structure and all methods present without issues • No use of any sort-related methods from any standard library • No use of any linked list related classes from any standard library • Meet all requirements	10	
#4: Generic ADT sorted list test	• Comments and naming conventions • Enough testing data from a file is used • Decompose method *main* • ADT-sorted list is completely tested • Meet all requirements	5	
#5: ADT address book classes	• Comments and naming conventions • Specifications and implementation are separate • A sorted list used as the data structure • No use of any sort-related methods from any standard library • All methods present without issues • Meet all requirements	10	
#6 Contact class	• Comments and naming conventions • All methods are included. For example, *constructors*, *getters*/*setters*, *toString*, and *equals.* • Meet all requirements	5	
#7: ADT address book test	• Comments and naming conventions • Enough testing data from a file is used • Decompose method *main* • ADT-address book is completely tested • Meet all requirements	5	
Total Points Awarded		/ 50	
Feedback to the student	Performance indicator #1: Performance indicator #2: Performance indicator #3: Performance indicator #4: Performance indicator #5: Performance indicator #6: Performance indicator #7:		

Part III: Examples

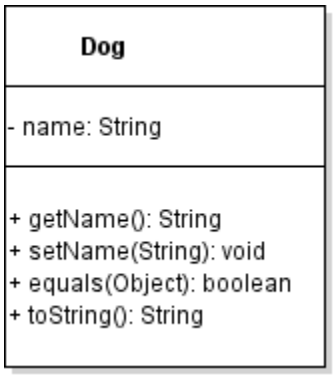
To complete a project, the following steps of a software development cycle should be followed. These steps are not pure linear but overlapped.

Analysis-design-code-test/debug-documentation.

1) Read project description to understand all specifications (**Analysis**)
2) Create a design (an algorithm or a UML class diagram) (**Design**)
3) Create programs that are translations of the design (**Code/Implementation**)
4) Test and debug(**test/debug)**
5) Complete all required documentation. (**Documentation**)

The following shows a sample design and the conventions.

- *Constructors* and *constants* should not be included in a class diagram.
- For each *field* (instance variable), include *visibility*, *name*, and *type* in the design.
- For each *method*, include *visibility*, *name*, *parameter type(s)* and *return type* in the design.
    - DON'T include *parameter names*, only *parameter types* are needed.
- Show class relationships such as *dependency*, *inheritance*, *aggregation*, etc. in the design. Don't include the *driver* program or any other testing classes since they are for testing purpose only.
    - Aggregation: For example, if Class A has an instance variable of type Class B, then, A is an aggregate of B.



The corresponding source codes with inline Javadoc comments are included on next page.

```
import java.util.Random;

/**
 * Representing a dog with a name.
 * @author Qi Wang
 * @version 1.0
 */
public class Dog{
    /**
     * The name of this dog
     */
    private String name;

    /**
     * Constructs a newly created Dog object that represents a dog with an empty name.
     */
    public Dog(){
        this("");
    }

    /**
     * Constructs a newly created Dog object wit
     * @param name The name of this dog
     */
    public Dog(String name){
        this.name = name;
    }

    /**
     * Returns the name of this dog.
     * @return The name of this dog
     */
    public String getName(){
        return this.name;
    }

    /**
     * Changes the name of this dog.
     * @param name The name of this dog
     */
    public void setName(String name){
        this.name = name;
    }

    /**
     * Returns a string representation of this dog. The returned string contains the type of
     * this dog and the name of this dog.
     * @return A string representation of this dog
     */
    public String toString(){
        return this.getClass().getSimpleName() + ": " + this.name;
    }

    /**
     * Indicates if this dog is "equal to" some other object. If the other object is a dog,
     * this dog is equal to the other dog if they have the same names. If the other object is
     * not a dog, this dog is not equal to the other object.
     * @param obj A reference to some other object
     * @return A boolean value specifying if this dog is equal to some other object
     */
    public boolean equals(Object obj){
        //The specific object isn't a dog.
        if(!(obj instanceof Dog)){
            return false;
        }
        //The specific object is a dog.
        Dog other = (Dog)obj;
        return this.name.equalsIgnoreCase(other.name);
    }
}
```

Class comments must be written in Javadoc format before the class header. A **description** of the class, author information, and version information are required.

Comments for fields are required.

open {

TAB

TAB

Method comments must be written in Javadoc format before the method header. The first word must be a verb in title case and in the **third** person. Use punctuation marks properly.

open {

TAB

TAB

A description of the method, comments on parameters if any, and comments on the return type if any are required.

A Javadoc comment for a formal parameter consists of three parts:

- parameter tag,
- a name of the formal parameter in the design ,
  (The name must be consistent in the comments and the header.)
- and a phrase explaining what this parameter specifies.

A Javadoc comment for return type consists of two parts:

- return tag,
- and a phrase explaining what this returned value specifies

More inline comments can be included in single line or block comments format in a method.

Part IV: Description
A contact organizer

Goals:

- Review and develop a deep and comprehensive understanding of the object-oriented paradigm
- Review and develop a deep and comprehensive understanding of abstract data types
- Review data structures and searching algorithms such as linked lists, linear search, etc.

For this assignment, you will create a contact organizer that organizes contacts in sorted order. You will design, implement and test two ADTs:

- A generic ADT Sorted List (implemented with a doubly linked list)
- An ADT address book (implemented with an ADT sorted list)

Java language is required for this assignment.

Part I: A generic ADT Sorted List:

A generic sorted list can store **comparable** objects in order. For example, in a sorted list, a collection of contacts can be stored in the order of their last names. When designing a generic ADT sorted list, type parameter, the deferred element type, needs to be constrained to be comparable type as shown below.

```
public class AGenericADTSortedLIst<E extends Comparable<E>>
```

The following functionalities must be provided, and this list must remain sorted after each operation. **It is not allowed to use any sort-related methods from any standard library; you must write everything needed by yourself.**

- Insert an element into this list.
    - When inserting a new element, a proper location must be determined so that the updated list remains sorted.
- Delete an element from this list.
- Delete all the elements from this list.
- Return a reference to the element at a specific position of this list.
- Determine if this list is empty.
- Determine the number of elements contained in this list.
- Return a string representation of this list (overridden *toString*).
- Compare this list with some other object for equivalency (overridden *equals*).

When implementing this ADT, ***a doubly linked list*** must be used as the data structure **(A generic node must be created. It is not allowed to use any linked list or node from any standard library. You must everything needed by yourself.).** In this ADT, there should be overloading constructors and two instance variables: a reference to the head of a doubly linked list (the data structure) and the count of elements of this list.

All classes must be tested individually (unit testing) before they are used with other classes. All methods must be called/tested for each class for all classes. There are no standard outputs for this assignment. Test this ADT before using it in the next ADT. To get started with the testing of this ADT, create an instance of the class, add a list of elements into the list, call other methods. To help you to get started, the following instructions are provided:

- Make a text file by including a list of integers (a file for input). Save the file in the project folder, the parent folder of folder *src*.
- Write a driver. In *main*, invoke the starting method in its helper class that is explained next.

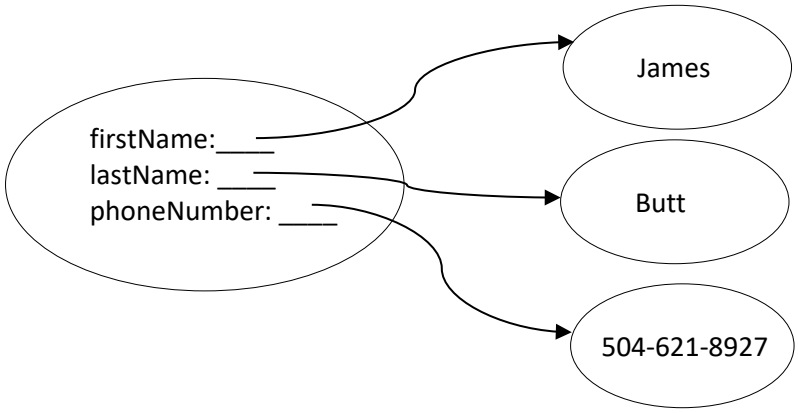
- Create a helper class for the driver. In the class, you may begin with the following three static methods:

```
//The starting method
public static void start(){
      //Create an instance of the ADT sorted list and save its reference.
      //Call method2 with the list reference passed to the method.
            //This method will add elements into the list.
      //Call method3 with the list reference passed to the method.
            //This method will delete and print the elements from the list.
      …
}
//Name this method properly
public static void method2(a reference to a list){
      //Read an integer from the input file, add a node to the sorted list.
      …
}
//Name this method properly
public static void method3(a reference to a list){
     //Delete and print the elements in order.
      …
}
```

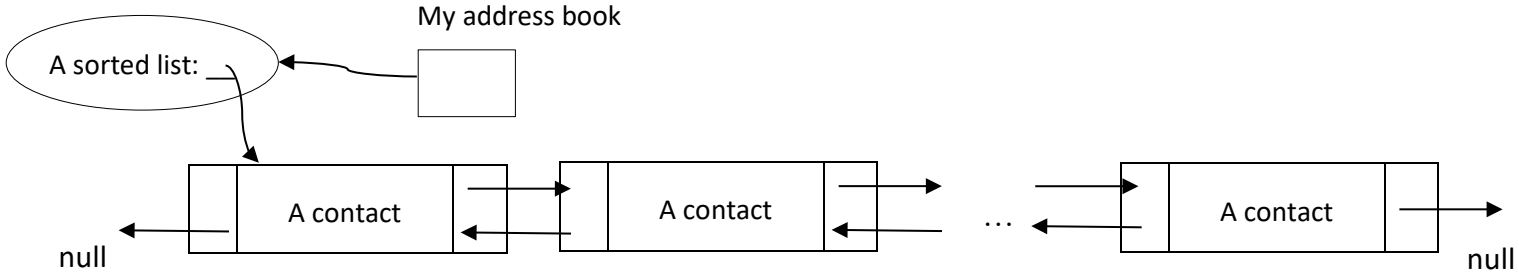
Other methods can be called/tested in one of the three methods or in an additional static method. All methods must be called/tested for each class for all classes.

Part II: ADT Address Book:

A contact is an object containing three values: first name, last name, and phone number. Assume that each contact has a unique name and a unique phone number, and all contacts are sorted by their last names in an address book. The following shows an instance of Contact class. Create *Contact* class, test it and use it for next part of the assignment.



An ADT Address book maintains a list of contacts in sorted order. The ADT is not a generic class. The element of a node is a contact. The following shows an instance of address book that organizes a list of contacts in sorted order.



To implement this ADT, an ADT sorted list (part I) must be used as the data structure. The following functionalities must be provided, and this list must remain sorted after each operation**. It is not allowed to use any sort-related methods from any standard library; you must write everything needed by yourself.**

- Search a contact in this address book.
- Insert a contact into this address book.
- Delete a contact from this address book.
- Delete all the contacts from this address book.
- Determine if this address book is empty.
- Determine the number of contacts contained in this address.
- Return a sorted doubly linked list by including all contacts whose phone numbers have the same area code.
- Return a sorted doubly linked list by including all contacts whose last names are the same.
- Return a string representation of this address book (overridden *toString*).
- Compare this address book with some other object for equivalency (overridden *equals*).

Notice that all operations are executed in the underlying sorted linked list of this address book. In this ADT, there should be overloading constructors and an instance variable: a reference to a sorted list (part I) with *Contact* as the element type.

All classes must be tested individually (unit testing) before they are used with other classes. All methods must be called/tested for each class for all classes. There are no standard outputs for this assignment. To test this ADT, create an instance of the class, add a list of contacts into the list, call other methods. You will use similar approach to part I testing. For the input file, you can prepare a text file using the data on the next page.

Here is the summary of the files to be submitted for this assignment. Save them into a folder, zip the folder, and submit the zipped file.

- A UML diagram
- Java source files with Javadoc style inline comments:
    - A generic ADT sorted list classes and its testing classes
    - Contact
    - ADT Address book and its testing classes
- Files for input:
    - Part I input file
    - Part II input file
- Any other supporting files

first_name	last_name	phone
James	Butt	504-621-8927
Josephine	Darakjy	810-292-9388
Art	Venere	856-636-8749
Lenna	Paprocki	907-385-4412
Donette	Foller	513-570-1893
Simona	Morasca	419-503-2484
Mitsue	Tollner	773-573-6914
Leota	Dilliard	408-752-3500
Sage	Wieser	605-414-2147
Kris	Marrier	410-655-8723
Minna	Amigon	215-874-1229
Abel	Maclead	631-335-3414
Kiley	Caldarera	310-498-5651
Graciela	Ruta	440-780-8425
Cammy	Albares	956-537-6195
Mattie	Poquette	602-277-4385
Meaghan	Garufi	931-313-9635
Gladys	Rim	414-661-9598
Yuki	Whobrey	313-288-7937
Fletcher	Flosi	815-828-2147
Bette	Nicka	610-545-3615
Veronika	Inouye	408-540-1785
Willard	Kolmetz	972-303-9197
Maryann	Royster	518-966-7987
Alisha	Slusarski	732-658-3154
Allene	Iturbide	715-662-6764
Chanel	Caudy	913-388-2079
Ezekiel	Chui	410-669-1642
Willow	Kusko	212-582-4976
Bernardo	Figeroa	936-336-3951
Ammie	Corrio	614-801-9788
Francine	Vocelka	505-977-3911
Ernie	Stenseth	201-709-6245
Albina	Glick	732-924-7882
Alishia	Sergi	212-860-1579
Solange	Shinko	504-979-9175
Jose	Stockham	212-675-8570
Rozella	Ostrosky	805-832-6163
Valentine	Gillian	210-812-9597
Kati	Rulapaugh	785-463-7829
Youlanda	Schemmer	541-548-8197
Dyan	Oldroyd	913-413-4604
Roxane	Campain	907-231-4722
Lavera	Perin	305-606-7291