King Saud University

College of Computer and Information Sciences

Department of Information Technology

Csc212: Data Structures

1st Semester 1445 H

Project title

Binary Search Tree Phonebook

Group#10

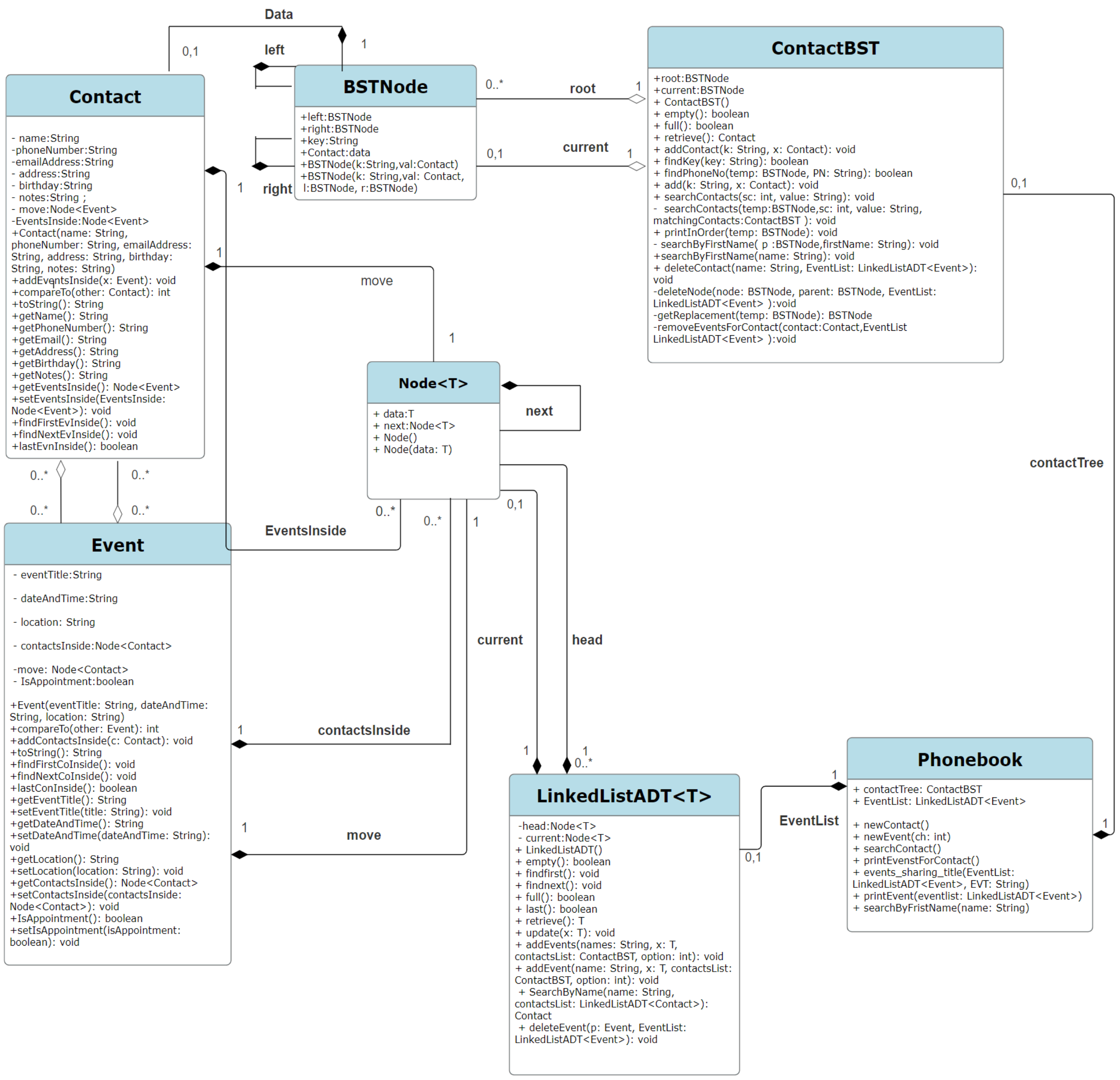
Section:41196		
Student Name	**ID**	**Tasks**
Leader:Wiam ahmed baalahtar	443200416	All method, UML, Big-O, report, (review, debugging).
Reema Alkhaldi	443201003	ContactBST , Phone book ,Event UML, Big-O, report, (review, debugging).

Supervised By: Dr. Abeer Alshaya.

Introduction:

In the first phase of our project, we will introduce a significant overhaul. The project revolves around the creation of a phone book program using key data structure concepts. Our aim is to construct a phone book comprising unique contacts and associating events with each contact. The program is built upon five main classes: “Phonebook,” “Contact,” “Event,”, “LinkedListADT.”, "ConatctBST" The foundational assumption is that the phone book contains BST of contacts and list of events. Within this structure, assuming each contact can have more than one event, and each event can have more than one contact.

Class Diagram:


Data
0,1
left
1
BSTNode
+left:BSTNode
+right:BSTNode
+key:String
+Contact:data
+BSTNode(k:String,val:Contact)
+BSTNode(k: String,val: Contact, l:BSTNode, r:BSTNode)
right
1
0..*
root
1
0,1
current
1
ContactBST
+root:BSTNode
+current:BSTNode
+ ContactBST()
+ empty(): boolean
+ full(): boolean
+ retrieve(): Contact
+ addContact(k: String, x: Contact): void
+ findKey(key: String): boolean
+ findPhoneNo(temp: BSTNode, PN: String): boolean
+ add(k: String, x: Contact): void
+ searchContacts(sc: int, value: String): void
- searchContacts(temp:BSTNode,sc: int, value: String, matchingContacts:ContactBST): void
+ printInOrder(temp: BSTNode): void
- searchByFirstName(p :BSTNode,firstName: String): void
+searchByFirstName(name: String): void
+ deleteContact(name: String, EventList: LinkedListADT<Event>): void
-deleteNode(node: BSTNode, parent: BSTNode, EventList: LinkedListADT<Event>):void
-getReplacement(temp: BSTNode): BSTNode
-removeEventsForContact(contact:Contact,EventList LinkedListADT<Event>):void
0,1
contactTree
Contact
- name:String
-phoneNumber:String
-emailAddress:String
- address:String
- birthday:String
- notes:String ;
- move:Node<Event>
-EventsInside:Node<Event>
+Contact(name: String, phoneNumber: String, emailAddress: String, address: String, birthday: String, notes: String)
+addEventsInside(x: Event): void
+compareTo(other: Contact): int
+toString(): String
+getName(): String
+getPhoneNumber(): String
+getEmail(): String
+getAddress(): String
+getBirthday(): String
+getNotes(): String
+getEventsInside(): Node<Event>
+setEventsInside(EventsInside: Node<Event>): void
+findFirstEvInside(): void
+findNextEvInside(): void
+lastEvnInside(): boolean
1
1
move
1
Node<T>
+ data:T
+ next:Node<T>
+ Node()
+ Node(data: T)
next
0..*
0..*
0..*
0..*
0..*
0..*
1
0,1
EventsInside
Event
- eventTitle:String
- dateAndTime:String
- location: String
- contactsInside:Node<Contact>
-move: Node<Contact>
- IsAppointment:boolean
+Event(eventTitle: String, dateAndTime: String, location: String)
+compareTo(other: Event): int
+addContactsInside(c: Contact): void
+toString(): String
+findFirstCoInside(): void
+findNextCoInside(): void
+lastConInside(): boolean
+getEventTitle(): String
+setEventTitle(title: String): void
+getDateAndTime(): String
+setDateAndTime(dateAndTime: String): void
+getLocation(): String
+setLocation(location: String): void
+getContactsInside(): Node<Contact>
+setContactsInside(contactsInside: Node<Contact>): void
+IsAppointment(): boolean
+setIsAppointment(isAppointment: boolean): void
current
head
1
contactsInside
1
move
1
1
0..*
LinkedListADT<T>
-head:Node<T>
- current:Node<T>
+ LinkedListADT()
+ empty(): boolean
+ findfirst(): void
+ findnext(): void
+ full(): boolean
+ last(): boolean
+ retrieve(): T
+ update(x: T): void
+ addEvents(names: String, x: T, contactsList: ContactBST, option: int): void
+ addEvent(name: String, x: T, contactsList: ContactBST, option: int): void
+ SearchByName(name: String, contactsList: LinkedListADT<Contact>): Contact
+ deleteEvent(p: Event, EventList: LinkedListADT<Event>): void
1
EventList
0,1
Phonebook
+ contactTree: ContactBST
+ EventList: LinkedListADT<Event>
+ newContact()
+ newEvent(ch: int)
+ searchContact()
+ printEvenstForContact()
+ events_sharing_title(EventList: LinkedListADT<Event>, EVT: String)
+ printEvent(eventlist: LinkedListADT<Event>)
+ searchByFristName(name: String)
1

Methods:

1.

- **Location**:ContactBST class.
- **Name**: newContact.
- **Description**: method that will add contact to the BST.
- **Input**: key - object of type Contact – information of new contact.
- **Output**: print message ("Contact added successfully!") OR ("Phone number already exists! ") OR ("Contact already exists!").
- **What they do**: The method will first determine whether the BST is empty or not. If it is, the contact will be added to the list and made it as a root to BST. If the BST is not empty, the method will check to see if the new contact's key or phone number already exists. If it does, the contact will not be added to the BST; If it does not, the contact will be added and sorted alphabetically using compareTo.
- *Big oh:*

Statements	S/E	Frequance
public void addContact(String k, Contact x) {	0	-
BSTNode q = current;	1	1
BSTNode newContact = new BSTNode(k, x);	1	1
if (empty()) {	1	1
root = current = newContact;	1	1
System.out.println("\nContact added successfully!");	1	1
return;}	1	1
if (findKey(k)) {	1	Log n
current = q;	1	1
System.out.println("\nContact already exists!");	1	1
return;}	1	1
if (findPhoneNo(root, (x.getPhoneNumber()))) {	1	n
System.out.println("\nPhone number already exists!");	1	1
return;}	1	1
if ((current.key).compareTo(k) > 0) {	1	1
current.left = newContact;	1	1
} else {	0	-
current.right = newContact;}	1	1
current = newContact;	1	1

System.out.println("\nContact added successfully!");	1	1
return;}	1	1
Total Big-O: O(n)		

2.

- **Location**:Phonebook class.
- **Name**: searchContact.
- **Description**: method that take information from user and give the user contacts information based on the information entered by the user.
- **Input**:nothing.
- **Output**:information of contacts OR print messages ("contact not found!")
- **What they do:** The method will ask the user to enter the search criteria and the value of his or her choice. It will then call method searchContacts(choice, value) to generate a new BST. Next, it will call another method to begin searching for contacts that match the value and number that the user entered (we use pre-order to traverse around all BST). Finally, it will call printInOrder to print the contacts method that was found (using in-order traverse). If the method does not find any contacts, it will inform the user.
- *Big oh:*

Statements	**S/E**	**Frequance**
public static void searchContact() {	0	-
String value;	0	-
int choice;	0	-
System.out.println("\nEnter search criteria: ");	1	1
System.out.println(""" 1. Name 2. Phone Number 3. Email Address 4. Address 5. Birthday\n""");	1	1
System.out.println("Enter your choice: ");	1	1
choice = input.nextInt();	1	1
input.nextLine();	1	1
switch (choice) {	1	1
case 1:	1	1
System.out.println("\nEnter the contact's name:");	1	1

value = input.nextLine();	1	1
break;	1	1
case 2:	1	1
System.out.println("\nEnter the contact's phone number:");	1	1
value = input.nextLine();	1	1
break;	1	1
case 3:	1	1
System.out.println("\nEnter the contact's email address:");	1	1
value = input.nextLine();	1	1
break;	1	1
case 4:	1	1
System.out.println("\nEnter the contact's address:");	1	1
value = input.nextLine();	1	1
break;	1	1
case 5:	1	1
System.out.println("\nEnter the contact's birthday:");	1	1
value = input.nextLine();	1	1
break;	1	1
default:	1	1
System.out.println("Invalid choice");	1	1
return;}	1	1
System.out.println();	1	1
contactTree.searchContacts(choice, value);}	1	n
Total Big-O: O(n)		

3.

- **Location**:ContactBST class
- **Name**: deleteContact
- **Description**: deleting Conatcts from Contacts tree and deleting event from events list.
- **Input**: name of contact that will remove , EventList
- **Output**:print message ("Contact Removed successfully!") OR ("Contact not found.") OR ("Empty List")
- **What they do**: The first step will check to see whether the root variable is not null and if the contact name the user input exists in the BST of contacts. whether the contact is discovered, the method will proceed to the next step; if not, it will return and notify the user that the contact was not found.

  In case a contact is detected, the method will call deleteNode. Immediately after, delete Node will call removeEventsForContact to access inside the contact that I have deleted and retrieve any events or appointments within the contact. If the contact contains any events or appointments, the method will forward the contact's information to the removeContactForEvent method, which will execute a while loop to remove the contact from the event/appointment. if the event/appointment contain no contacts left, we will delete it from the event list by calling deleteEvent.

  method deleteNode will check if the contact that i want to delete have one , two or no child then we will deleting based on aphabetical order

- *Big oh:*

Statements	S/E	Frequance
public void deleteContact(String name, LinkedListADT<Event> EventList) {	0	-
BSTNode temp = root;	1	1
BSTNode parent = null;	1	1
if (root == null) {	1	1
System.out.println("Empty List");	1	1
return;}	1	1
while (temp != null) {	1	n
if (temp.key.equals(name)) {	1	n-1
break;}	1	n-1
parent = temp;	1	n-1
if (temp.key.compareTo(name) > 0) {	1	n-1
temp = temp.left;	1	n-1
} else {	0	-

temp = temp.right;}}	1	n-1
if (temp == null) {	1	1
System.out.println("Contact not found.");	1	1
return;}	1	1
deleteNode(temp, parent, EventList);}	1	n
Total Big-O: O(n)		

4.

- **Location**:Phonebook class
- **Name**: newEvent
- Description: take information of new new appointment or event from user to added to the events list, add the contact within the appointment or event, and add the appointment or event inside the contact.
- **Input**: number user chose(is it one or two) – information of new event/ appointment.
- **Output**: print message ("Event scheduled successfully!") OR ("Appointment scheduled successfully!") OR ("The appointment you tried to add is alrady exists!") OR ("Conflict time! there is another event that have the same time and date. ").
- **What they do**:Since the user can enter multiple contacts separated by commas, method will create an array using split to hold the names of contacts without commas. If the user enters a single contact or decides to add appointments, the array will start at index zero and return. If the user enters multiple contacts, the method will send each contact to the addEvent method using a loop. method addEvent checks to see if the contact name the user entered is in the BST of contacts. (If it is not ,method will make a return) after that method sets the value of the event or appointment; if the eventList is null, it adds the event or appointment immediately; otherwise, it checks to see if the event or appointment already exists and adds the contact to the event that is already exists , if its appointment method does not add contact inside it, becase the appointment its allow to have one contact,then we will check if the new event/appointment have a conflict time with another event/appointment, if not method will add new event/appointment sorting alphabetically in eventList and add event to contact and contact to the event.
- *Big oh:*

Statements	S/E	Frequance
public static void newEvent(int ch) {	0	-
String nameOfContact;	0	-
System.out.println("Enter event title : ");	1	1
input.nextLine();	1	1

String title = input.nextLine();	1	1
if (ch == 1) {	1	1
System.out.println("Enter contact name sparated by a comma : ");	1	1
nameOfContact = input.nextLine();	1	1
} else {	0	-
System.out.println("Enter contact name : ");	1	1
nameOfContact = input.nextLine();}	1	1
System.out.println("Enter event data and time (Format: MM/DD/YYYY HH:MM) : ");	1	1
String dAndT = input.nextLine();	1	1
System.out.println("Enter event location : ");	1	1
String loc = input.nextLine();	1	1
EventList.addEvents(nameOfContact, new Event(title, dAndT, loc), contactTree, ch);}	1	n
Total Big-O: O(n)		

5a.

- **Location**:Phonebook class
- **Name**: printEvenstForContact
- **Description:** print all event inside contact by contact name.
- **Input:** nothing
- **Output:** ("There is no Event/Appointment for this contact!") OR ("Event found!") with a event information
- **What they do:** this methode for print events inside the contact that the user enter his/her name, if there is no event for the contact or contact not found,method will inform the user.
- *Big oh:*

Statements	S/E	Frequance
public static void printEvenstForContact() {	0	-
Node<Event> temp = new Node<Event>();	1	1
contactTree.retrieve().findFirstEvInside();	1	1
temp = contactTree.retrieve().getEventsInside();	1	1
if (temp == null) {	1	1
System.out.println("\nThere is no Event/Appointment for this contact!");	1	1

} else {	0	-
System.out.println("\nEvent found!");}	1	1
while (temp != null) {	1	n
System.out.println(temp.data);	1	n-1
if (temp.data.isIsAppointment()) {	1	n-1
System.out.println("type: " + "Appointment" + "\n");	1	n-1
} else {	0	-
System.out.println("type: " + "Event" + "\n");}	1	n-1
temp = temp.next;}}	1	n-1
Total Big-O : O(n)		

5b.

- **Location**:Phonebook class.
- **Name**: eventsSharingTitle.
- **Description**: print event and all contact inside it by event title.
- **Input**: EventList – Event title
- **Output**: print message ("Empty List") OR event information with contacts names inside it.
- **What they do**: if the event list not empty, method will print all events details and Contacts list inside it , that sharing the event title that recevie from user.
- *Big oh:*

Statements	S/E	Frequance
public static void eventsSharingTitle(LinkedListADT<Event> EventList, String EVT) {	0	-
if (EventList.empty()) {	1	1
System.out.println("Empty List");	1	1
} else {	0	-
Node<Contact> temp = new Node<Contact>();	1	1
boolean eventFound = false;	1	1
EventList.findfirst();	1	1
while (!EventList.last()) {	1	n
if (EventList.retrieve().getEventTitle().equalsIgnoreCase(EVT)) {	1	n-1
EventList.retrieve().findFirstCoInside();	1	n-1
temp = EventList.retrieve().getContactsInside();	1	n-1

System.out.println("\n" + EventList.retrieve());	1	n-1
System.out.print("Contacts names: ");	1	n-1
while (temp != null) {	1	n(n-1)
System.out.print(temp.data.getName());	1	(n-1)(n-1)
if (temp.next != null) {	1	(n-1)(n-1)
System.out.print(", ");}	1	(n-1)(n-1)
temp = temp.next;}	1	(n-1)(n-1)
System.out.println();	1	n-1
if (EventList.retrieve().isIsAppointment()) {	1	n-1
System.out.println("type: " + "Appointment");	1	n-1
} else {	0	-
System.out.println("type: " + "Event");}	1	n-1
eventFound = true;}	1	n-1
EventList.findnext();}	1	n-1
if (EventList.retrieve().getEventTitle().equalsIgnoreCase(EVT)) {	1	1
EventList.retrieve().findFirstCoInside();	1	1
temp = EventList.retrieve().getContactsInside();	1	1
System.out.println("\n" + EventList.retrieve());	1	1
System.out.print("Contacts names: ");	1	1
while (temp != null) {	1	n
System.out.print(temp.data.getName());	1	n-1
if (temp.next != null) {	1	n-1
System.out.print(", ");}	1	n-1
temp = temp.next;}	1	n-1
System.out.println();	1	1
if (EventList.retrieve().isIsAppointment()) {	1	1
System.out.println("type: " + "Appointment");	1	1
} else {	0	-
System.out.println("type: " + "Event");}	1	1
eventFound = true;}	1	1
if (!eventFound) {	1	1
System.out.println("\nNo events found with the title: " + EVT);}}}	1	1
Total Big-O :O(n^2)		

6.

- **Location**:Phonebook class
- **Name**: searchByFirstName
- **Description**: search of contacts that have same first name and print it to the user.
- **Input**: first name by user.
- **Output**: print message ("There is no contact have a first name " + name) OR List of contacts that have the same first name as the user entered.
- **What they do:** In order to print all contacts that share a first name, first the method call "searchByFirstName" method.

  Following a check to see if the tree is empty, the private "searchByfirstName" method will be called by giving in the parameters "root" and "first name." Once the parameter values are received, method will use indexOf to search the first space in the contacts name. The method will notify the user if it does not find any contacts with the first name they are looking for, otherwise it will print information of any contacts that have the same first name user entered .

Big oh:

Statements	S/E	Frequance
public static void searchByFristName(String name) {	0	-
contactTree.searchByFirstName(name);	1	n
}	0	-
Total Big-O :O(n)		

Statements	S/E	Frequance
public void searchByFirstName(String name) {	0	0
if (root == null) {	1	1
System.out.println("Empty List");	1	1
return;}	1	1
contactFound = false;	1	1
searchByFirstName(root, name);	1	n
if (!contactFound) {	1	1
System.out.println("There is no contact have a first name " + name);}}	1	1
Total Big-O :O(n)		

7.

- **Location**:Phonebook class.
- **Name**: printEvent.
- **Description**:method will print all event alphabetically.
- **Input**: eventList.
- **Output**: ("Empty list!") OR list of all Event in EventList.
- **What they do**: since the event insert alphabetically to the event list, the method will print all events information that exists in Eventlist, if there is no events in EventList method will inform the user.
- *Big oh:*

Statements	S/E	Frequance
public static void printEvent(LinkedListADT<Event> eventlist) {	0	-
if (eventlist.empty()) {	1	1
System.out.println("Empty list!");	1	1
return;}	1	1
eventlist.findfirst();	1	1
System.out.println("\nAll Events:");	1	1
while (!eventlist.last()) {	1	n
System.out.println();	1	n-1
System.out.println(eventlist.retrieve());	1	n-1
if (eventlist.retrieve().isIsAppointment()) {	1	n-1
System.out.println("type: " + "" + "Appointment");	1	n-1
} else {	0	-
System.out.println("type: " + "" + "Event");}	1	n-1
eventlist.findnext();}	1	n-1
System.out.println("\n " + eventlist.retrieve());	1	1
if (eventlist.retrieve().isIsAppointment()) {	1	1
System.out.println("type: " + "" + "Appointment");	1	1
} else {	0	-
System.out.println("type: " + "" + "Event");}}	1	1
Total Big-O: O(n)		

Resources:

- https://github.com/majohn17/UofUClasses
- https://github.com/BaluDeshamoni/DSA-Programs
- Data Structure book.pdf