DATA STRUCTURE PROJECT

LinkedList PhoneBook

Prepared By:
Faisal Alhaqbani 443101660
Mohammed Alzubaidi 44310700
Mohammed Almuhaitheef 443101706

Introduction :

Our Linked List Phonebook project is a comprehensive solution designed to manage contact information and schedule events efficiently.

Our main Classes :

- **Contact** : Represents individual contacts with attributes such as name, phone number, email address, etc.

- **Event** : Represents scheduled events or appointments associated with a contact. Contains details such as event title, date, time, and location. .

- **LinkedList** : Implements a custom linked list data structure. Provides methods for adding, deleting, searching, and updating contacts and events within the list.

- **Phonebook** : Serves as the main interface for user interactions. Manages instances of LinkedList for contacts and events. Provides methods for adding, searching, deleting contacts, scheduling events, and displaying event details. Includes a user-friendly command-line menu system.

The main tasks for each person

Task	Assigned To
Implement Event methods and calculate Big O , Write the report	Mohammed Almuhaitheef
Work on LinkedList, Contact, and Node , PHoneBook classes	Mohammed Alzubaidi
Lead the group and work on most Phonebook methods , and handle the exceptions	Faisal Alhaqbani

Project Specification:

Method	**addContact(Contact contact)**
Requires	**Contact name and phone number must be unique.**
Input	**A contact object to add.**
Output	**Message indicating success or failure.**
Results	**The contact is added if no existing contact with the same name and phone number is found.**

Method	**searchContact(int criteria)**
Requires	**No specific conditions.**
Input	**Multiple search criteria.**
Output	**Message indicating whether the contact was found or not.**
Results	**If found, displays all information about the contact.**

Project Specification:

Method	**deleteContact(String name)**
Requires	**No specific conditions.**
Input	**Contact's name.**
Output	**Message indicating whether the contact was deleted or not.**
Results	**Removes the contact from the phonebook and deletes associated events if the contact is found.**

Method	**scheduleEvent()**
Requires	**No conflicts between two events.**
Input	**Event's information such as title, date, time, etc.**
Output	**Message indicating whether the event was scheduled or not.**
Results	**If successful, the event will be stored in the phonebook's events list.**

Project Specification:

Method	**printEventDetails(int criteria)**
Requires	**No specific conditions.**
Input	**Contact's name or event title.**
Output	**Event information if successful; otherwise, a failure message.**
Results	**None.**

Method	**printAllEventsAlphabetically()**
Requires	**No specific conditions.**
Input	**None.**
Output	**Message indicating success or failure.**
Results	**Prints all events scheduled alphabetically by their titles.**

Project Specification:

Method	**printContactsByFirstName(String firstName)**
Requires	**No specific conditions.**
Input	**Contact's first name.**
Output	**Message indicating success or failure.**
Results	**Prints events belonging to contacts with the given first name.**

Time complexity of the methods

First LinkedList Methods

Method	Time Complexity
public boolean empty()	O(1)
public boolean last()	O(1)
public boolean full()	O(1)
public void findFirst()	O(1)
public void findNext()	O(1)
public T retrieve()	O(1)
public void update(T val)	O(1)
public void insert(T val)	O(1)
public void remove()	O(n)
public Boolean find(T val)	O(n)

Summary code complexity

Method Name	Code Complexity
addContact(Contact contact)	O(n)
searchContact(int criteria)	O(n)
deleteContact(String name)	O(n^2)
printContactsByFirstName(String firstName)	O(n)
printAllContacts(LinkedList<Contact> list)	O(n)
searchByName(String name)	O(n)
searchByPhoneNumber(String PhoneNumber)	O(n)
searchByEmailAddress(String EmailAddress)	O(n)
searchByAddress(String Address)	O(n)
searchByBirthday(String Birthday)	O(n)
contactExists(Contact contact)	O(n)
deleteContactEvents(String name)	O(n^2)
scheduleEvent()	O(n)
addEvent(Event tmp)	O(n)
PrintEventDetails(int criteria)	O(n)
printAllEventsAlphabetically()	O(n)
eventConflictExists(String date)	O(n)

You can see the details under

Time complexity of the methods

addContact()

Code	Big O
if(contacts.empty()) {	O(1)
contacts.insert(new Contact("","","","","", ""));	O(1)
contacts.insert(contact);	O(1)
return;	O(1)
}	
contacts.findFirst();	O(1)
Contact prev=contacts.retrieve();	O(1)
while(!contacts.last()) {	O(n)
if(contacts.retrieve().compareTo(contact)>0) {	O(n)
contacts.find(prev);	O(n)
contacts.insert(contact);	O(1)
return;	O(1)
}	
prev=contacts.retrieve();	O(n)
contacts.findNext();	O(n)
}	
if(contacts.retrieve().compareTo(contact)>0) {	O(1)
contacts.find(prev);	O(n)
contacts.insert(contact);	O(1)
return;	O(1)
}	
else {	
contacts.insert(contact);	O(1)
return;	O(1)
Total	O(n)

Time complexity of the methods

searchContact()

Code	Big O
String searchValue="";	O(1)
LinkedList<Contact> founded;	O(1)
Scanner input = new Scanner(System.in);	O(1)
contacts.findFirst();	O(1)
switch (criteria) {	O(1)
case 1:	
System.out.print("\nEnter the contact's name:");	O(1)
searchValue=input.nextLine();	O(1)
Contact contactByName=searchByName(searchValue);	O(n)
if(contactByName != null)	O(1)
contactByName.printInfo();	O(1)
else	
System.out.println("\ncontact not found");	O(1)
break;	O(1)
case 2:	
System.out.print("\nEnter the contact's PhoneNumber:");	O(1)
searchValue=input.next();	O(1)
Contact contactByPhoneNumber=searchByPhoneNumber(searchValue);	O(n)
if(contactByPhoneNumber != null)	O(1)
contactByPhoneNumber.printInfo();	O(1)
else	
System.out.println("\ncontact not found");	O(1)
break;	O(1)
case 3:	
System.out.print("\nEnter the contact's EmailAddress: ");	O(1)
searchValue=input.next();	O(1)
founded=searchByEmailAddress(searchValue);	O(n)
if(!founded.empty())	O(1)
printAllContacts(founded);	O(n)
else	
System.out.println("\ncontact not found");	O(1)
break;	O(1)
case 4:	
System.out.print("\nEnter the contact's Address:");	O(1)
searchValue=input.nextLine();	O(1)
founded=searchByAddress(searchValue);	O(n)
if(!founded.empty())	O(1)
printAllContacts(founded);	O(n)
else	
System.out.println("\ncontact not found");	O(1)
break;	O(1)
case 5:	
System.out.print("\nEnter the contact's Birthday:");	O(1)
searchValue=input.next();	O(1)
founded=searchByBirthday(searchValue);	O(n)
if(!founded.empty())	O(1)
printAllContacts(founded);	O(n)
else	
System.out.println("\ncontact not found");	O(1)
Total	O(n)

Time complexity of the methods

deleteContact()

Code	Big O
if (contacts.empty()) { System.out.print("\nPhonebook is empty.");	O(1)
if(searchByName(name)==null) { System.out.print("\nContact not found.");	O(n)
contacts.findFirst();	O(1)
while (!contacts.last()) {	O(n)
if (contacts.retrieve().getName().equalsIgnoreCase(name)) {	O(n)
deleteContactEvents(name);	O(n^2)
contacts.remove();	O(n)
System.out.print("\nContact deleted successfully.");	O(n)
return;	O(n)
contacts.findNext();	O(n)
}	
if (contacts.retrieve().getName().equalsIgnoreCase(name)) {	O(1)
deleteContactEvents(name);	O(n^2)
contacts.remove();	O(n)
System.out.print("\nContact deleted successfully.");	O(1)
}	
Total	O(n^2)

Time complexity of the methods

printContactsByFirstName ()

Code	Big O
LinkedList<Contact> founded = new LinkedList<>();	O(1)
if(contacts.empty()) System.out.println("\nthere are no contacts by this first name ");	O(1)
contacts.findFirst();	O(1)
firstName += " ";	O(1)
while(!contacts.last()) {	O(n)
if(contacts.retrieve().getName().startsWith(firstName))	O(n)
founded.insert(contacts.retrieve());	O(n)
contacts.findNext();	O(n)
}	
if(contacts.retrieve().getName().startsWith(firstName))	O(1)
founded.insert(contacts.retrieve());	O(1)
printAllContacts(founded);	O(n)
Total	O(n)

Time complexity of the methods

printAllContacts()

Code	Big O
if(list.empty())	O(1)
return;	
list.findFirst();	O(1)
if(list.last())	O(1)
System.out.print("\nContact found!\r");	O(1)
else	
System.out.print("\nContacts found!\r");	O(1)
while(!list.last()) {	O(n)
list.retrieve().printInfo();	O(n)
list.findNext();	O(n)
}	
list.retrieve().printInfo();	O(1)
}	
Total	O(n)

Time complexity of the methods

searchByName()

Code	Big O
if(contacts.empty())	O(1)
return null;	
contacts.findFirst();	O(1)
while(!contacts.last()) {	O(n)
if(contacts.retrieve().getName().equalsIgnoreCase(name))	O(n)
return contacts.retrieve();	O(1)
contacts.findNext();	O(n)
}	
if(contacts.retrieve().getName().equalsIgnoreCase(name))	O(1)
return contacts.retrieve();	O(1)
return null;	O(1)
}	
Total	O(n)

Time complexity of the methods

searchByPhoneNumber()

Code	Big O
if(contacts.empty())	O(1)
return null;	O(1)
contacts.findFirst();	O(1)
while(!contacts.last()) {	O(n)
if(contacts.retrieve().getPhoneNumber().equalsIgnoreCase(PhoneNumber))	O(n)
return contacts.retrieve();	O(1)
contacts.findNext();	O(n)
}	
if(contacts.retrieve().getPhoneNumber().equalsIgnoreCase(PhoneNumber))	O(1)
return contacts.retrieve();	O(1)
return null;	O(1)
Total	O(n)

Time complexity of the methods

searchByEmailAddress()

Code	Big O
LinkedList<Contact> founded = new LinkedList<>();	O(1)
if(contacts.empty())	O(1)
return founded;	O(1)
contacts.findFirst();	O(1)
while(!contacts.last()) {	O(n)
if(contacts.retrieve().getEmailAddress().equalsIgnoreCase(EmailAddress))	O(n)
founded.insert(contacts.retrieve());	O(n)
contacts.findNext();	O(n)
}	
if(contacts.retrieve().getEmailAddress().equalsIgnoreCase(EmailAddress))	O(1)
founded.insert(contacts.retrieve());	O(1)
return founded;	O(1)
Total	O(n)

Time complexity of the methods

searchByAddress()

Code	Big O
LinkedList<Contact> founded = new LinkedList<>();	O(1)
if(contacts.empty())	O(1)
return founded;	O(1)
contacts.findFirst();	O(1)
while(!contacts.last()) {	O(n)
if(contacts.retrieve().getAddress().equalsIgnoreCase(Address))	O(n)
founded.insert(contacts.retrieve());	O(n)
contacts.findNext();	O(n)
}	
if(contacts.retrieve().getAddress().equalsIgnoreCase(Address))	O(1)
founded.insert(contacts.retrieve());	O(1)
return founded;	O(1)
Total	O(n)

Time complexity of the methods

searchByEmailAddress ()

Code	Big O
LinkedList<Contact> founded = new LinkedList<> ();	O(1)
if(contacts.empty())	O(1)
return founded;	O(1)
contacts.findFirst();	O(1)
while(!contacts.last()) {	O(n)
if(contacts.retrieve().getEmailAddress().equalsIgnoreCase(EmailAddress))	O(n)
founded.insert(contacts.retrieve());	O(n)
contacts.findNext();	O(n)
}	
if(contacts.retrieve().getEmailAddress().equalsIgnoreCase(EmailAddress))	O(1)
founded.insert(contacts.retrieve());	O(1)
return founded;	O(1)
}	
Total	O(n)

Time complexity of the methods

searchByBirthday()

Code	Big O
LinkedList<Contact> founded = new LinkedList<>();	O(1)
if(contacts.empty())	O(1)
return founded;	O(1)
contacts.findFirst();	O(1)
while(!contacts.last()) {	O(n)
if(contacts.retrieve().getBirthday().equalsIgnoreCase(Birthday))	O(n)
founded.insert(contacts.retrieve());	O(n)
contacts.findNext();	O(n)
}	
if(contacts.retrieve().getBirthday().equalsIgnoreCase(Birthday))	O(1)
founded.insert(contacts.retrieve());	O(1)
return founded;	O(1)
Total :	O(n)

Time complexity of the methods

contactExists()

Code	Big O
return (searchByName(contact.getName()) !=null || searchByPhoneNumber(contact.getPhoneNumber()) !=null);	O(n)
Total :	O(n)

Time complexity of the methods

deleteContactEvents ()

Code	Big O
Events.empty()	O(1)
return;	O(1)
Events.findFirst()	O(1)
Events.findNext()	O(1)
while(!Events.last()) {	O(n)
if(Events.retrieve().getEventuser().getName().equalsIgnoreCase(name)) {	O(n)
Events.remove();	O(n^2) //inside a while loop
continue;	O(n)
}	
Events.findNext();	O(1)
}	
if(Events.retrieve().getEventuser().getName().equalsIgnoreCase(name))	O(1)
Events.remove();	O(n)
}	
Total:	O(n^2)

Time complexity of the methods

scheduleEvent()

Code	Big O
Scanner input = new Scanner(System.in);	O(1)
Event tmp = new Event();	O(1)
System.out.print("\nEnter event title:");	O(1)
tmp.setTitle(input.nextLine());	O(1)
System.out.print("Enter contact name:");	O(1)
tmp.setEventuser(searchByName(input.nextLine()));	O(n)
System.out.print("Enter event date and time (MM/DD/YYYY HH:MM):");	O(1)
tmp.setDate(input.nextLine());	O(1)
System.out.print("Enter event location:");	O(1)
tmp.setLocation(input.nextLine());	O(1)
(tmp.getEventuser()!= null)	O(1)
!eventConflictExists(tmp.getDate())	O(n)
addEvent(tmp);	O(n)
Total :	O(n)

Time complexity of the methods

addEvent()

Code	Big O
Event nullEvent = new Event("", "", "", null);	O(1)
if(Events.empty()) { ... }	O(1)
Events.findFirst();	O(1)
Event prev=Events.retrieve();	O(1)
prev=Events.retrieve();	O(n)
Events.findNext();	O(n)
while(!Events.last()){	O(n)
if(Events.retrieve().compareTo(tmp)>0) { ... }	O(n)
Events.find(prev);	O(n)
Events.insert(tmp);	O(1)
System.out.println("\nEvent scheduled successfully!");	O(1)
return; }	O(1)
prev=Events.retrieve();	O(n)
Events.findNext(); }	O(n)
if(Events.retrieve().compareTo(tmp)>0) {	O(1)
Events.find(prev);	O(n)
Events.insert(tmp);	O(1)
System.out.println("\nEvent scheduled successfully!");	O(1)
return; }	O(1)
else { Events.insert(tmp);	O(1)
System.out.println("\nEvent scheduled successfully!"); } }	O(1)
Total :	O(n)

Time complexity of the methods

PrintEventDetails ()

Code	Big O
if(Events.empty()) {	O(1)
System.out.println("\nEvent not found!");	O(1)
return;	O(1)
}	
Events.findFirst();	O(1)
Events.findNext();	O(1)
String result="";	O(1)
switch (criteria) {	O(1)
case 1: {	
System.out.print("Enter the contact's name:");	O(1)
String contactName= input.nextLine();	O(1)
while(!Events.last()) {	O(n)
if(Events.retrieve().getEventuser().getName().equalsIgnoreCase(contactName)) {	O(n)
result+=Events.retrieve().Info();	O(n)
}	
Events.findNext();	O(n)
}	
if(Events.retrieve().getEventuser().getName().equalsIgnoreCase(contactName)) {	O(1)
result+=Events.retrieve().Info();	O(1)
}	
if(result!="") {	O(1)
System.out.print("\nEvent found!");	O(1)
System.out.println(result);	O(1)
}	
break;	O(1)
case 2: {	
System.out.print("Enter the event title:");	O(1)
String eventTitle= input.nextLine();	O(1)
while(!Events.last()) {	O(n)
if(Events.retrieve().getTitle().equalsIgnoreCase(eventTitle)) {	O(n)
result+=Events.retrieve().Info();	O(n)
}	
Events.findNext();	O(n)
}	
if(Events.retrieve().getTitle().equalsIgnoreCase(eventTitle)) {	O(1)
result+=Events.retrieve().Info();	O(1)
}	
if(result!="") {	O(1)
System.out.print("\nEvent found!");	O(1)
System.out.println(result);	O(1)
}	
break;	O(1)
default:	O(1)
System.out.println("\ninvalid input");	O(1)
Total :	O(n)

Time complexity of the methods

printAllEventsAlphabetically()

Code	Big O
if(Events.empty())	O(1)
return;	O(1)
String result="";	O(1)
Events.findFirst();	O(1)
Events.findNext();	O(1)
while(!Events.last()) {	O(n)
result+=Events.retrieve().Info();	O(n)
Events.findNext();	O(n)
}	
result+=Events.retrieve().Info();	O(1)
System.out.println(result);	O(1)
Total :	O(n)

Time complexity of the methods

eventConflictExists()

Code	Big O
if(Events.empty()) {	O(1)
return false;	O(1)
}	
Events.findFirst();	O(1)
Events.findNext();	O(1)
while(!Events.last()) {	O(n)
if(Events.retrieve().getDate().equalsIgnoreCase(date))	O(n)
return true;	O(1)
Events.findNext();	O(n)
}	
if(Events.retrieve().getDate().equalsIgnoreCase(date))	O(1)
return true;	O(1)
return false;	O(1)
Total :	O(n)

Conclusion :

In conclusion,
the Linked List Phonebook project has been implemented successfully, meeting all specified requirements.

The project's user-friendly interface, robust functionalities, and adherence to best practices ensure a reliable and efficient phonebook application

We are grateful for the opportunity to work on this project, showcasing our programming skills and problem-solving abilities. Thank you for considering our submission.