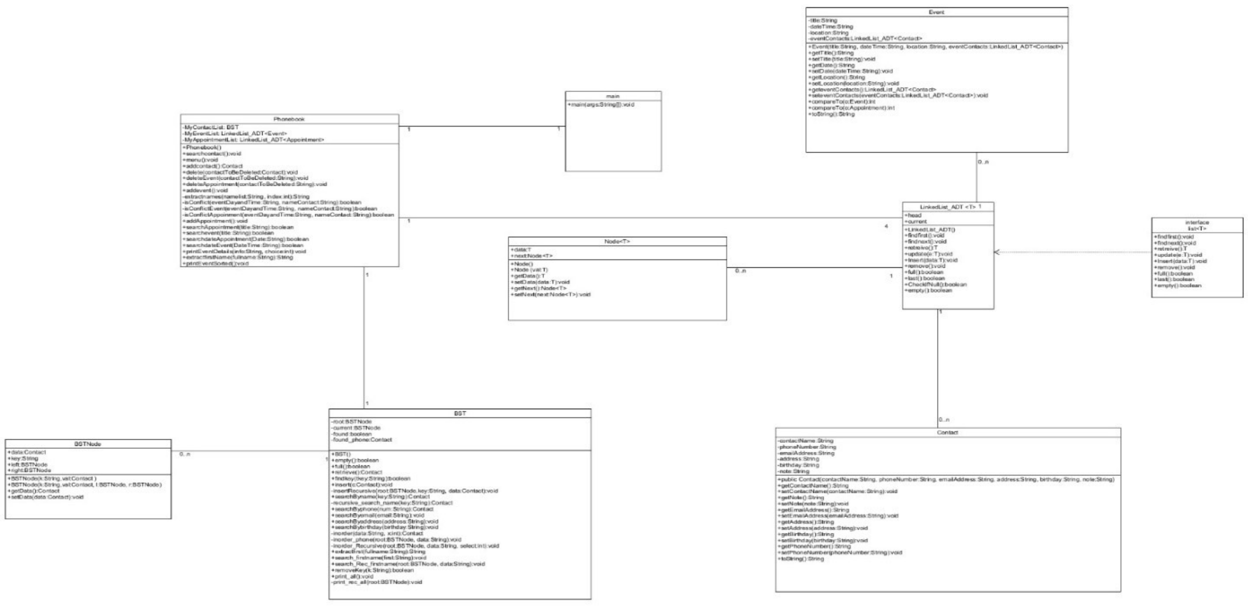
- **UML Class diagram:**


Event
-title:String
-dateTime:String
-location:String
-eventContacts:LinkedList_ADT<Contact>
+Event(title:String, dateTime:String, location:String, eventContacts:LinkedList_ADT<Contact>)
+getTitle():String
+setTitle(title:String):void
+getDate():String
+setDate(dateTime:String):void
+getLocation():String
+setLocation(location:String):void
+geteventContacts():LinkedList_ADT<Contact>
+seteventContacts(eventContacts:LinkedList_ADT<Contact>):void
+compareTo(o:Event):int
+compareTo(o:Appointment):int
+toString():String
main
+main(args:String[]):void
Phonebook
-MyContactList: BST
-MyEventList: LinkedList_ADT<Event>
-MyAppointmentList: LinkedList_ADT<Appointment>
+Phonebook()
+searchcontact():void
+menu():void
+addcontact():Contact
+delete(contactToBeDeleted:Contact):void
+deleteEvent(contactToBeDeleted:String):void
+deleteAppointment(contactToBeDeleted:String):void
+addevent():void
-extractnames(namelist:String, index:int):String
-isConflict(eventDayandTime:String, nameContact:String):boolean
-isConflictEvent(eventDayandTime:String, nameContact:String):boolean
-isConflictAppoinment(eventDayandTime:String, nameContact:String):boolean
+addAppointment():void
+searchAppointment(title:String):boolean
+searchevent(title:String):boolean
+searchdateAppointment(Date:String):boolean
+searchdateEvent(DateTime:String):boolean
+printEventDetails(info:String, choice:int):void
+extractfirstName(fullname:String):String
+printEventSorted():void
Node<T>
+data:T
+next:Node<T>
+Node()
+Node (val:T)
+getData():T
+setData(data:T):void
+getNext():Node<T>
+setNext(next:Node<T>):void
LinkedList_ADT <T>
+head
+current
+LinkedList_ADT()
+findfirst():void
+findnext():void
+retreive():T
+update(e:T):void
+Insert(data:T):void
+remove():void
+full():boolean
+last():boolean
+CheckIfNull():boolean
+empty():boolean
interface
list<T>
+findfirst():void
+findnext():void
+retreive():T
+update(e:T):void
+Insert(data:T):void
+remove():void
+full():boolean
+last():boolean
+empty():boolean
BSTNode
+data:Contact
+key:String
+left:BSTNode
+right:BSTNode
+BSTNode(k:String, val:Contact)
+BSTNode(k:String, val:Contact, l:BSTNode, r:BSTNode)
+getData():Contact
+setData(data:Contact):void
BST
-root:BSTNode
-current:BSTNode
-found:boolean
-found_phone:Contact
+BST()
+empty():boolean
+full():boolean
+retrieve():Contact
+findkey(key:String):boolean
+insert(c:Contact):void
-insertRecursive(root:BSTNode, key:String, data:Contact):void
+searchByname(key:String):Contact
-recursive_search_name(key:String):Contact
+searchByphone(num:String):Contact
+searchByemail(email:String):void
+searchByaddress(address:String):void
+searchBybirthday(birthday:String):void
-inorder(data:String, x:int):Contact
-inorder_phone(root:BSTNode, data:String):void
-inorder_Recursive(root:BSTNode, data:String, select:int):void
+extractfirst(fullname:String):String
+search_firstname(first:String):void
+search_Rec_firstname(root:BSTNode, data:String):void
+removeKey(k:String):boolean
+print_all():void
-print_rec_all(root:BSTNode):void
Contact
-contactName:String
-phoneNumber:String
-emailAddress:String
-address:String
-birthday:String
-note:String
+public Contact(contactName:String, phoneNumber:String, emailAddress:String, address:String, birthday:String, note:String)
+getContactName():String
+setContactName(contactName:String):void
+getNote():String
+setNote(note:String):void
+getEmailAddress():String
+setEmailAddress(emailAddress:String):void
+getAddress():String
+setAddress(address:String):void
+getBirthday():String
+setBirthday(birthday:String):void
+getPhoneNumber():String
+setPhoneNumber(phoneNumber:String):void
+toString():String
1
1
1
4
1
1
0..n
1
0..n
1
1
0..n
1
0..n

Contact class***:***

- Attributes:
  - ***contactname***: the name of the contact.
  - ***phoneNumber***: the phone number of the contact.
  - ***emailAddress***: the email address of the contact.
  - ***address*** : the address of the contact.
  - ***birthday***: the birthday of the contact.
  - ***notes***: the notes associated with the contact.
- Methods:
  - ***Contact(name:String, phoneNumber:String, emailAddress:String, address:String, birthday:String, notes:String*)**:** constructor with BigO(1) because constructors typically have a time complexity of O(1) because they involve a constant number of operations regardless of the size or complexity.
  - ***toString***():Returns a formatted string representation of the contact.
  - ***compareTo***(Object event):Compares two Contact objects based on their names in a case-insensitive manner and Returns a negative integer, zero, or a positive integer if the name of this contact is less than, equal to, or greater than the title of the specified event, respectively

- ***Getters/Setters*** BigO(1) because Getters and Setters typically have a time complexity of O(1) because they involve a constant number of operations regardless of the size or complexity.

Node<T> class***:***

- Attributes:
  - ***data***: the data stored in the node.
  - ***next***: the next node in the linked list.
- Methods:
  - ***Contact(name:String, phoneNumber:String, emailAddress:String, address:String, birthday:String, notes:String*)**:** constructor with BigO(1) because constructors typically have a time complexity of O(1) because they involve a constant number of operations regardless of the size or complexity.
  - ***Getters/Setters*** BigO(1) because Getters and Setters typically have a time complexity of O(1) because they involve a constant number of operations regardless of the size or complexity.

Event class***:***

- Attributes:
  - ***title***: the id of the contract.
  - ***dateTime:*** the weight of the contract.
  - ***location:*** *the location* of the contract.
  - ***eventContactS:*** *Linked list of contacts that involved in the event*
- Methods:
  - **Event(String title, Contact contact, String dateTime, String location):** constructor
  - ***Getters/Setters***
  - **toString:** return a formatted string representation of the contact.	Big O(1)
  - **compareTo(Event o)** : Compares two Event objects based on their titles in a case-insensitive manner and Returns a negative integer, zero, or a positive integer if the title of this event is less than, equal to, or greater than the title of the specified event, respectively
  - **compareTo(Appointment o)** : Compares Event and appointment objects based on their titles in a case-insensitive manner and Returns a negative integer, zero, or a

positive integer if the title of this event is less than, equal to, or greater than the title of the specified event, respectively

List Interface***:***

- o Methods:
  - **findFirst( ):**
  - **findNext( ):**
  - **retrieve( ):**
  - **update(T t):**
  - **full( ):**
  - **empty( ).**
  - **last().**

LinkedList_ADT<T> class***:***

- o Attributes:
  - ***head***: pointer for first element in list.
  - ***current:*** pointer for the current element in list.
- o Methods:
  - ***LinkedList_ADT(*)**:** constructor with BigO(1) because constructors typically have a time complexity of O(1) because they involve a constant number of operations regardless of the size or complexity.
  - ***empty():*** Check whether the list is empty or not , BigO(1) because it will return one Boolean value and it involve a constant number of operations .
  - ***checkIfNull():*** Check whether the current is null or not , BigO(1) because it will return one Boolean value and it involve a constant number of operations .
  - ***last():*** Check whether the current is in the last of the list or not, BigO(1) because it will return one Boolean value and it involve a constant number of operations .
  - ***full():*** check whether the list is full or not, BigO(1) because it will involve one operation which is returning false.

- ***findFirst():*** put current in the first element on the list, BigO(1) because it involve a constant number of operations.
- ***findNext():*** put current in the next element of the current ,BigO(1) because it involve a constant number of operations.
- ***retrieve():*** return the data of the Node , BigO(1) because it involve a constant number of operations.
- ***update(T Val):*** Update data in the given Node, BigO(1) because it involve a constant number of operations.
- ***Insert (T data):*** inserting a new node containing the specified data into a linked list while maintaining the order based on the data's natural ordering .O(n)

Phonebook class*:*

- Attributes:
  - ***MyContactList***: BST of contact.
  - ***MyEventList:*** Linked list of events
  - ***MyAppointmentList***: linked list of appointments
- Methods:
  - ***Phonebook():***constructor with BigO(1) because constructors typically have a time complexity of O(1) because they involve a constant number of operations regardless of the size or complexity. O(n), where n is the number of contacts.
  - ***searchcontact():*** Allows the user to search for a contact based on different criteria such as name, phone number, email, address, or birthday. O(n), where n is the number of contacts
  - ***menu():*** Displays a menu for various operations like adding, searching, and deleting contacts, scheduling events, printing event details, and exiting the program. O(n^3), where n is the number of contacts
  - ***addcontact()***:Takes user input to create a new contact and adds it to the contact list. O(n),where n is the number of contacts
  - ***delete()****:*Deletes a specified contact from the contact list. O(n^2) where n is the number of contacts
  - ***deleteEvent() :*** Deletes events associated with a specific contact from the event list. O(n^2) where n is the number of events
  - ***deleteAppointment()****:*Deletes appointments associated with a specific contact from the appointment list. O(n^2), where n is the number of appointments.

- ***addevent()**: Allows the user to schedule an event, associating it with one or more contacts. O(n), where n is the length of the input string*
- ***extractnames()**: Extr*acts individual contact names from a comma-separated string. O(n), where n is the number of appointments
- ***isConflict():***Checks if there is a scheduling conflict for a new event or appointment. *O(n), where n is the number of events or appointments*
- ***isConflictEvent():***Checks if there is a scheduling conflict for a new event. O(n), where n is the number of events
- ***isConflictAppointment()**:* Checks if there is a scheduling conflict for a new appointment. O(n), where n is the number of appointments
- ***addAppointment(): Allows the user to schedule an appointment with a specific contact. O(n), where n is the number of appointments***
- ***searchAppointment():**Searches for an appointment based on its title. O(n). where n is the number of appointments*
- ***searchEvent():***Searches for an event based on its title. O(n) ,where n is the number of events
- ***searchDateAppointment()**:* Searches for an appointment based on its date. O(n) ,where n is the number of appointments
- ***searchDateEvent():***Searches for an event based on its date. O(n) ,where n is the number of events
- ***printEventDetails()**:Prints details of events or appointments associated with a specific contact or title. O(n) ,where n is the total number of events and appointments*
- ***extractFirstName():***Extracts the first name from a full name. O(n), where n is the length of the input string
- ***printEventSorted():***Prints details of events and appointments in a sorted order. O(n) ,where n is the total number of events and appointments

BST class***:***

- Attributes:
  - ***root:*** Represents the root of the binary search tree
  - ***current:*** Represents the current node during tree traversal or search operations.
  - ***found***: A flag indicating whether a search operation found a specific element.
  - **found_phone:** Stores the contact information when searching by phone number.
- Methods:
  - ***BST():***Initializes an empty binary search tree.
  - ***empty():***Checks if the binary search tree is empty. O(1)
  - ***full():***Checks if the binary search tree is full
  - **retrieve*():*** return the data of the Node
  - **findkey(String key):** if a node with entered key found a flag raise true otherwise false

- ***insert(Contact c)****: Inserts a new contact into the binary search tree. O(log n) average case for a balanced tree*
- ***searchByname(String key):*** Searches for a contact by name in the binary search tree. O(log n) ,average case for a balanced tree
- ***searchByphone(String num)****: Searches for a contact by phone number in the binary search tree. O(log n) ,average case for a balanced tree*
- ***searchByemail(String email):*** *Searches for contacts by email in the binary search tree and prints the result. O(log n) ,average case for a balanced tree*
- ***searchByaddress(String address)****: Searches for contacts by address in the binary search tree and prints the result. O(log n) average case for a balanced tree*
- ***searchBybirthday(String birthday):*** *Searches for contacts by birthday in the binary search tree and prints the result. O(log n) average case for a balanced tree*
- ***search_firstname(String first)****: Searches for contacts by first name in the binary search tree and prints the result. O(log n) average case for a balanced tree*
- ***removeKey(String k):*** *Removes a contact with the given key from the binary search tree. O(log n) ,average case for a balanced tree*

- ***print_all():****Prints all contacts in the binary search tree. O(n) where n is the number of contacts in the tree.*
- ***insertRecursive(BSTNode root, String key, Contact data):*** *A recursive function to insert a new key and data into the BST. O(log n) in average case, O(n) in worst case.*
- ***recursive_search_name(String key):*** *A recursive helper method for searching a contact by name in the binary search tree. O(log n) in average case, O(n) in worst case.*
- ***inorder(String data, int x):*** *A wrapper method for initiating different types of inorder searches (phone, email, address, birthday). O(n) where n is the number of nodes in the tree.*
- ***inorder_phone(BSTNode root, String data):*** *A recursive helper method for searching a contact by phone number in the binary search tree. O(n) where n is the number of nodes in the tree.*
- ***inorder_Recursive(BSTNode root, String data, int select):*** *A recursive helper method for searching a contact by email, address, or birthday in the binary search tree. O(n) where n is the number of nodes in the tree.*
- ***search_Rec_firstname(BSTNode root, String data):*** *A recursive helper method for searching contacts by first name in the binary search tree. O(n) where n is the number of nodes in the tree.*
- ***print_rec_all(BSTNode root):*** *A recursive helper method for printing all contacts in the binary search tree. O(n) where n is the number of nodes in the tree.*
- ***extractfirst(String fullname):*** *Extracts the first name from a full name. O(n) where n is the length of the full name.*

Appointment *class:*

o ***Attributes:***

- ***title*** *: A string representing the title of the appointment.*
- ***date :*** *A string representing the date of the appointment.*
- ***location :*** *A string representing the location of the appointment.*
- ***appointmentContact :*** *A Contact object representing the contact involved in the appointment.*

o *Methods:*

- ***Appointment() :*** *This is a constructor method with a time complexity of O(1). It initializes an instance of the Appointment class with default values for the title , date , location , and appointmentContact attributes.*

- ***Appointment(title: String, date: String, location: String, appointmentContact: Contact) :*** *This is a constructor method with a time complexity of O(1). It creates an instance of the Appointment class with the provided title , date , location , and appointmentContact values.*

- ***getTitle() :*** *This method returns the title attribute of the Appointment object. It has a time complexity of O(1) because it simply returns the value stored in the title attribute.*

- ***setTitle(title: String) :*** *This method sets the value of the title attribute of the Appointment object. It has a time complexity of O(1) because it assigns the provided title value to the title attribute.*

- ***getDate() :*** *This method returns the date attribute of the Appointment object. It has a time complexity of O(1) because it simply returns the value stored in the date attribute.*

- ***setDate_Time(date: String) :*** *This method sets the value of the date attribute of the Appointment object. It has a time complexity of O(1) because it assigns the provided date value to the date attribute.*

- ***getLocation() :*** *This method returns the location attribute of the Appointment object. It has a time complexity of O(1) because it simply returns the value stored in the location attribute.*

- ***setLocation(location: String) :*** *This method sets the value of the location attribute of the Appointment object. It has a time complexity of O(1) because it assigns the provided location value to the location attribute.*

- ***getContactinvolved() :*** *This method returns the appointmentContact attribute of the Appointment object. It has a time complexity of O(1) because it simply returns the value stored in the appointmentContact attribute.*

- ***setContactinvolved(contact: Contact) :*** *This method sets the value of the appointmentContact attribute of the Appointment object. It has a time complexity of O(1) because it assigns the provided contact value to the appointmentContact attribute.*

- ***compareTo(Appointment o) :*** *This method compares the title attribute of the current Appointment object with the title attribute of the provided Appointment object (o). It returns an integer value based on the comparison result. It has a time complexity of O(1) because it performs a string comparison operation using the title attribute.*

- ***toString() :*** *This method returns a string representation of the Appointment object. It includes the title , date , location , and the name of the contact involved in the appointment. It has a time complexity of O(1) because it concatenates the values of the attributes and invokes the getContactName() method of the appointmentContact object, which also has a time complexity of O.(1)*

BooleanWrapper *class:*

- ***Attributes:***
- *value : A private boolean variable that holds the wrapped boolean value.*
- ***Methods:***
- ***BooleanWrapper(boolean value) :*** *This is a constructor method that initializes an instance of the BooleanWrapper class with the provided boolean value . It sets the internal value attribute of the object to the provided value.*

- ***get() :*** *This method returns the boolean value stored in the value attribute of the BooleanWrapper object. It retrieves and returns the current value of the object.*

- ***set(boolean value) :*** *This method sets the boolean value of the value attribute in the BooleanWrapper object. It takes a boolean value as a parameter and assigns it to the value attribute of the object, effectively updating the stored value.*

BSTNode class:

- ***Attributes:***

- ***key :*** *A string representing the key associated with the node.*
- ***data :*** *A Contact object holding the data associated with the node.*
- ***left :*** *A reference to the left child node.*
- ***right :*** *A reference to the right child node.*

- ***Methods:***

- ***BSTNode(String k, Contact val) :*** *This is a constructor method that creates a new instance of the BSTNode class. It takes a string k as the key and a Contact object val as the data to be stored in the node. It initializes the key and data attributes with the provided values, and sets the left and right child nodes to null .*

- ***BSTNode(String k, Contact val, BSTNode l, BSTNode r) :*** *This is another constructor method that creates a new instance of the BSTNode class. It takes a string k as the key, a Contact object val as the data, and two BSTNode objects l and r representing the left and right child nodes, respectively. It initializes the key , data , left , and right attributes with the provided values.*

- ***getData() :*** *This method returns the data attribute of the BSTNode object. It retrieves and returns the Contact object stored in the node. It has a time complexity of O(1) because it simply returns the value stored in the data attribute.*

- ***setData(Contact data) :*** *This method sets the data attribute of the BSTNode object. It takes a Contact object data as a parameter and assigns it to the data attribute. It updates the data stored in the node with the provided value. It has a time complexity of O(1) because it assigns the provided data value to the data attribute.*

main class:

- ***Methods:***
- ***main(String[] args):*** *main method will start the menu of Phonebook class*

CONCLUSION

In this phase, we successfully designed and implemented a phonebook tree using BST data structure. The phonebook tree provides a comprehensive set of functionalities for storing, searching, deleting contacts, as well as storing, searching, deleting, and printing events and appointment.