CSCI 260

CampusReserve

Anmolak Singh 1310189

Inderpreet Singh 1309142

Table of Contents

Problem Statement.....2

Flow Chart.....3

Array, LinkedList, and Binary tree.....4

Stack.....7

Queue.....8

Hash Table/Map.....9

Bubble Sort.....10

Screenshots.....11

Java Source Code.....13

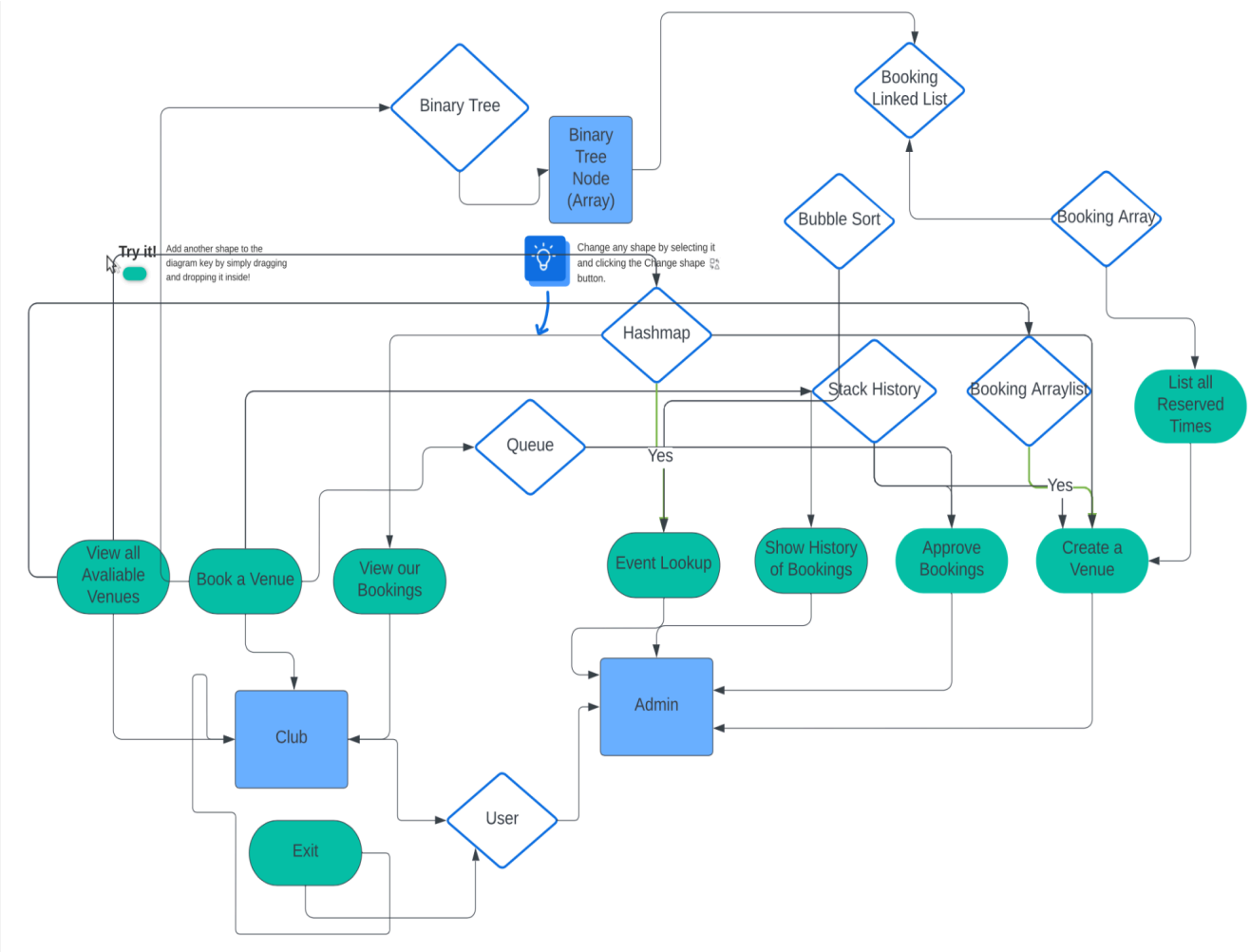
Problem Statement

- We are tackling the issue of scheduling conflicts and bad communication among university clubs when booking venues for events. By developing an efficient event management system, we streamlined the booking process and improved communication between hosts and club members. Now, clubs can easily check venue availability, submit requests, and receive instant confirmations, leading to smoother event planning and enhanced collaboration within the university.

Solution

- We addressed the challenge by implementing a sophisticated software solution leveraging a variety of data structures for efficiency. Our system utilizes arrays, array lists, hashmaps, binary trees, search algorithms, stacks, and queues to optimize the event management process. By strategically employing these data structures, we ensured quick access to venue availability, streamlined booking requests, and facilitated seamless communication between hosts and club members.

Flow Chart


Binary Tree
Binary Tree Node (Array)
Booking Linked List
Bubble Sort
Booking Array
Try it!
Add another shape to the diagram key by simply dragging and dropping it inside!
Change any shape by selecting it and clicking the Change shape button.
Hashmap
Stack History
Booking Arraylist
List all Reserved Times
Queue
Yes
Yes
View all Avaliable Venues
Book a Venue
View our Bookings
Event Lookup
Show History of Bookings
Approve Bookings
Create a Venue
Admin
Club
User
Exit

Array, LinkedList, Binary Tree

The binary tree, clubBT stores data in the form of a Object[] Array. The first element in the object array, is a String name, denoting the club's name. For example, if SASA logs into our portal clubBT creates a new node for SASA and make the first element of the array, "SASA". The second element denotes the current APPROVED bookings the club has access to in the form of a LinkedList<Booking>. Each new booking is added to the Linked List and it grows as long as the club keeps adding new bookings. For example if SASA books a 1200-1300 booking at NYIT it is stored in clubBT as a node with the following information: ["SASA", LinkedList<Booking>]. Inside the LinkedList there will be a booking object with the 1200-1300 NYIT Booking information. If SASA books another booking at Hofstra, the code will search through clubBT check if there is a node for SASA, access the linkedlist inside that node and add the following booking object to it.

ArrayList:

In the venue booking system project, the ArrayList data structure plays a crucial role in managing bookings for each venue. It serves as a container for storing Booking objects associated with each venue. This allows for easy addition, removal, and retrieval of bookings, providing flexibility in handling venue reservations. With constant-time access to elements by index, ArrayList ensures efficient management of booking information, enabling users to quickly view, update, and cancel bookings for various venues.

```
public class Venue {
    public String name;
    public int startTime;
    public int endTime;
    public ArrayList<Booking> bookings;
    public Stack<String> history;

    public Venue() {
        this.name = "";
        this.startTime = 0;
        this.endTime = 0;
        this.bookings = new ArrayList<Booking>();
        this.history = new Stack<String>();
    }
```

LinkedList:

LinkedLists are implicitly used within the project to manage collections of Booking objects. In this context, LinkedLists offer efficient insertion and deletion operations, particularly when modifying bookings or handling pending requests.

```
public class Booking {

    public Venue venue;
    public int startTime;
    public int endTime;
    public String clubName;
    public boolean approvedByAdmin;

    public Booking(Venue venue, int startTime, int endTime, String
clubName, boolean approvedByAdmin) {
        this.venue = venue;
        this.startTime = startTime;
        this.endTime = endTime;
        this.clubName = clubName;
        this.approvedByAdmin = approvedByAdmin;
    }
```

Binary Tree:

The Binary Tree data structure plays a pivotal role in organizing club information and associated bookings within the venue booking system. By structuring club data hierarchically, the Binary Tree class enables efficient search, insertion, and retrieval of club information and bookings. This facilitates streamlined administrative tasks such as approving bookings, managing club-specific reservations, and maintaining a structured overview of venue usage. Leveraging the hierarchical nature of Binary Trees enhances the scalability and performance of the application, ensuring smooth operation even with a large volume of club and booking data.

```
public class venueDriver {
    private static Scanner keyboard = new Scanner(System.in);
    public static HashMap<String, Venue> venues = new HashMap<String,
Venue>();
    public static Queue<Booking> bookingQueue = new
LinkedList<Booking>();
    public static BT clubBT = new BT();
```

Stack

- In the Venue class, the Stack<String> history is used to maintain a history of events related to the venue. When a new venue is created information about the venue creation, including its name, opening time, and closing time, is pushed onto the history stack using history.push The showHistory() method iterates over the elements in the history stack and prints out each event stored in it. This method allows users to view the history of events related to the venue.

```
System.out.println("Enter the venue name to show history:");

                    String venueName2 =
keyboard.nextLine().toUpperCase();

                    if (venues.containsKey(venueName2)) {

                        venues.get(venueName2).showHistory();

                    }

                    else {

                        System.out.println("Venue not found.");

                    }

                    break;
```

Queue

- The queue data structure is used to manage pending booking requests. When a club member books a venue, the booking request is added to the bookingQueue for admin approval. In the admin menu, admins can view pending bookings and decide whether to approve or deny them. If approved, the booking is removed from the bookingQueue and added to the club's list of bookings, indicating approval.

```
System.out.println(" - PENDING BOOKINGS - ");

                    for(Booking booking : bookingQueue) {

                        System.out.println(booking.toString());

                        System.out.println("Approve? (Y/N)");

                        String approve =
keyboard.nextLine().toLowerCase();

                        if(approve.equals("y")) {

                            booking.approvedByAdmin = true;

                            System.out.println("Booking approved.");

                            booking.venue.history.push("Booking
approved by admin: " + booking.toString());

                            BTNode currClubNode =
clubBT.getTargetNode(booking.clubName);


currClubNode.getBookingsList().add(booking);

                        }

                        else {

                            booking.approvedByAdmin = false;

                            booking.clubName = "";

                            booking.venue.history.push("Booking
denied by admin: " + booking.toString());

                            System.out.println("Booking denied.");

                        }
```

Hash Table/Map

In the venue Driver class, the HashMap is used to store venue objects, with their names as keys. When a new venue is added it is inserted into the venues hashmap. In the club menu the hashmap is used to display all available venues allowing club members to select and book venues. Admins can view all venues (case "1" in showAdminMenu()) by iterating over the keys of the venues hashmap and printing out each venue.

```
case "1":
                    System.out.println(" - ALL VENUES - ");
                    for (String key : venues.keySet()) {

System.out.println(venues.get(key).toString());
                    }
                    break;
                case "2":
                    Venue newVenue = new Venue();
                    newVenue.createVenue();
                    System.out.println("Venue created: " +
newVenue.toString());
                    venues.put(newVenue.name, newVenue);
                    break;
```

Bubble Sort

The bubble sort algorithm, integrated into your project, facilitates the sorting of approved booking times extracted from the booking queue. It iterates through the booking queue, identifying bookings approved by the admin, and extracts their corresponding start times. These times are then sorted chronologically using the bubble sort algorithm.

```
System.out.println(" - ALL RESERVED TIMES (SORTED) - ");
                List<Integer> approvedTimes = new ArrayList<>();

                // Extract approved times from bookingQueue
                for (Booking booking : bookingQueue) {
                    if (booking.approvedByAdmin) {
                        approvedTimes.add(booking.startTime);
                    }
                }

                // Bubble sort
                int n = approvedTimes.size();
                for (int i = 0; i < n - 1; i++) {
                    for (int j = 0; j < n - i - 1; j++) {
                        if (approvedTimes.get(j) >
approvedTimes.get(j + 1)) {
                            // Swap
                            int temp = approvedTimes.get(j);
                            approvedTimes.set(j, approvedTimes.get(j
+ 1));
                            approvedTimes.set(j + 1, temp);
```

Screenshots

```
 - LOGED IN AS ADMIN -
 - ADMIN MENU -
1. View All Venues
2. Add a Venue
3. Remove a Venue
4. Approve Bookings
5. Show History of a Venue
6. List All resererved times
7. Exit
```

```
Enter the venue name:
Party Hall 1
Enter the venue opening time:
1000
Enter the venue closing time:
2000
Venue created: PARTY HALL 1 (1000 - 2000)
Press Enter to continue...
```

```
 - LOGED IN AS NYIT -
 - CLUB MENU -
1. View All Available Venues
2. Book a Venue
4. View Our Bookings
5. Exit
1
 - ALL AVAILABLE VENUES -
PARTY HALL 1 (1000 - 2000)
Press Enter to continue...
```

```
Party Hall 1
CURRENT AVAILABLE TIMES FOR PARTY HALL 1:
BOOKING #1: 1000 - 1100 () @ PARTY HALL 1
BOOKING #2: 1100 - 1200 () @ PARTY HALL 1
BOOKING #3: 1200 - 1300 () @ PARTY HALL 1
BOOKING #4: 1300 - 1400 () @ PARTY HALL 1
BOOKING #5: 1400 - 1500 () @ PARTY HALL 1
BOOKING #6: 1500 - 1600 () @ PARTY HALL 1
BOOKING #7: 1600 - 1700 () @ PARTY HALL 1
BOOKING #8: 1700 - 1800 () @ PARTY HALL 1
BOOKING #9: 1800 - 1900 () @ PARTY HALL 1
BOOKING #10: 1900 - 2000 () @ PARTY HALL 1
Enter the booking #:
2
Booking pending: 1100 - 1200 (NYIT) @ PARTY HALL 1
Press Enter to continue...
```

```
 - PENDING BOOKINGS -
1100 - 1200 (NYIT) @ PARTY HALL 1
Approve? (Y/N)
Y
Booking approved.
Press Enter to continue...
```

```
Party Hall 1
Venue created: PARTY HALL 1 (1000 - 2000)
All slots created for PARTY HALL 1
Booking pending by NYIT for 1100 - 1200 (NYIT) @ PARTY HALL 1
Booking approved by admin: 1100 - 1200 (NYIT) @ PARTY HALL 1
```

```
 - OUR BOOKINGS -
1100 - 1200 (NYIT) @ PARTY HALL 1
Press Enter to continue...
```

Java Source Code

```
import java.util.*;
public class venueDriver {
   private static Scanner keyboard = new Scanner(System.in);
   public static HashMap<String, Venue> venues = new HashMap<String, Venue>();
   public static Queue<Booking> bookingQueue = new LinkedList<Booking>();
   public static BT clubBT = new BT();

   public static void clearTerminal() {
       System.out.print("\033[H\033[2J");
       System.out.flush();
   }

   public static void showClubMenu() {
       clearTerminal();
       System.out.println("Enter your club name:");
       String clubName = keyboard.nextLine().toUpperCase();
       //Create a new club node in the binary tree if it doesn't exist
       if(clubBT.countNodes() != 0) {
           if(clubBT.search(clubName)) {
           }
           else {
               clubBT.insert(clubName, new LinkedList<Booking>());
           }
       }
       else {
           clubBT.insert(clubName, new LinkedList<Booking>());
       }


       while (true) {
           clearTerminal();
           System.out.println(" - LOGED IN AS " + clubName + " - ");
           System.out.println(" - CLUB MENU - ");
           System.out.println("1. View All Available Venues");
```

```
            System.out.println("2. Book a Venue");
            //System.out.println("3. Cancel Booking");
            System.out.println("4. View Our Bookings");
            System.out.println("5. Exit");

            String option = keyboard.nextLine();
            switch (option) {
                case "1":
                    System.out.println(" - ALL AVAILABLE VENUES - ");
                    for (String key : venues.keySet()) {
                        System.out.println(venues.get(key).toString());
                    }
                    break;
                case "2":
                    System.out.println("Enter the venue name to book:");
                    String venueName = keyboard.nextLine().toUpperCase();
                    if (venues.containsKey(venueName)) {
                        System.out.println("CURRENT AVAILABLE TIMES FOR " + venueName +
":");
                        venues.get(venueName).showBookings();

                        System.out.println("Enter the booking #:");
                        int bookingNumber = keyboard.nextInt();


if(venues.get(venueName).bookings.get(bookingNumber-1).clubName.equals("")) {

venues.get(venueName).bookings.get(bookingNumber-1).clubName = clubName;

venues.get(venueName).bookings.get(bookingNumber-1).approvedByAdmin = false;

bookingQueue.add(venues.get(venueName).bookings.get(bookingNumber-1));
                            System.out.println("Booking pending: " +
venues.get(venueName).bookings.get(bookingNumber-1).toString());
                            venues.get(venueName).history.push("Booking pending by " +
clubName + " for " + venues.get(venueName).bookings.get(bookingNumber-1).toString());
                        }
                        else {
                            System.out.println("Booking failed. Slot already taken.");
                        }

                    }
```

```
                else {
                    System.out.println("Venue not found.");
                }

                keyboard.nextLine();

                break;
            case "3":
                //Cancel Booking
                break;
            case "4":
                //View Our Bookings
                System.out.println(" - OUR BOOKINGS - ");
                BTNode currClubNode = clubBT.getTargetNode(clubName);
                for(Booking booking : currClubNode.getBookingsList()) {
                    System.out.println(booking.toString());
                }
                break;
            case "5":
                return;
            default:
                System.out.println("Invalid input. Please try again.");
                break;
        }
        System.out.println("Press Enter to continue...");
        keyboard.nextLine();

    }
}

public static void showAdminMenu() {
    while (true) {
        clearTerminal();
        System.out.println(" - LOGED IN AS ADMIN - ");
        System.out.println(" - ADMIN MENU - ");
        System.out.println("1. View All Venues");
        System.out.println("2. Add a Venue");
        System.out.println("3. Remove a Venue");
        System.out.println("4. Approve Bookings");
        System.out.println("5. Show History of a Venue");
        System.out.println("6. List All resererved times");
        System.out.println("7. Exit");
```

```
            String option = keyboard.nextLine();
            switch (option) {
                case "1":
                    System.out.println(" - ALL VENUES - ");
                    for (String key : venues.keySet()) {
                        System.out.println(venues.get(key).toString());
                    }
                    break;
                case "2":
                    Venue newVenue = new Venue();
                    newVenue.createVenue();
                    System.out.println("Venue created: " + newVenue.toString());
                    venues.put(newVenue.name, newVenue);
                    break;
                case "3":
                    System.out.println("Enter the venue name to remove:");
                    String venueName = keyboard.nextLine().toUpperCase();
                    if (venues.containsKey(venueName)) {
                        venues.remove(venueName);
                        System.out.println("Venue removed: " + venueName);
                    }
                    else {
                        System.out.println("Venue not found.");
                    }
                    break;
                case "4":
                    System.out.println(" - PENDING BOOKINGS - ");
                    for(Booking booking : bookingQueue) {
                        System.out.println(booking.toString());
                        System.out.println("Approve? (Y/N)");
                        String approve = keyboard.nextLine().toLowerCase();
                        if(approve.equals("y")) {
                            booking.approvedByAdmin = true;
                            System.out.println("Booking approved.");
                            booking.venue.history.push("Booking approved by admin: " +
booking.toString());
                            BTNode currClubNode =
clubBT.getTargetNode(booking.clubName);
                            currClubNode.getBookingsList().add(booking);
                        }
                        else {
```

```
                            booking.approvedByAdmin = false;
                            booking.clubName = "";
                            booking.venue.history.push("Booking denied by admin: " +
booking.toString());
                            System.out.println("Booking denied.");
                        }
                        bookingQueue.poll();
                    }
                    break;
                case "5":
                    System.out.println("Enter the venue name to show history:");
                    String venueName2 = keyboard.nextLine().toUpperCase();
                    if (venues.containsKey(venueName2)) {
                        venues.get(venueName2).showHistory();
                    }
                    else {
                        System.out.println("Venue not found.");
                    }
                    break;
                case "6":
                System.out.println(" - ALL RESERVED TIMES (SORTED) - ");
                List<Integer> approvedTimes = new ArrayList<>();

                // Extract approved times from bookingQueue
                for (Booking booking : bookingQueue) {
                    if (booking.approvedByAdmin) {
                        approvedTimes.add(booking.startTime);
                    }
                }

                // Bubble sort
                int n = approvedTimes.size();
                for (int i = 0; i < n - 1; i++) {
                    for (int j = 0; j < n - i - 1; j++) {
                        if (approvedTimes.get(j) > approvedTimes.get(j + 1)) {
                            // Swap
                            int temp = approvedTimes.get(j);
                            approvedTimes.set(j, approvedTimes.get(j + 1));
                            approvedTimes.set(j + 1, temp);
                        }
                    }
                }
```

```
            // Print sorted times
            for (int time : approvedTimes) {
                System.out.println(time);
            }
            break;

            case "7":
                return;
            default:
                System.out.println("Invalid input. Please try again.");
                break;
        }
        System.out.println("Press Enter to continue...");
        keyboard.nextLine();
    }
}




public static void main(String[] args) {
    while(true){
        clearTerminal(); //Clear terminal
        //Show Options (welcome screen)
        System.out.println(" - WELCOME TO VENUE PORTAL - ");
        System.out.println("Login as Club (C) or Admin (A)?");
        String userType = keyboard.nextLine().toLowerCase();
        //Check if user input is valid
        while (!userType.equals("c") && !userType.equals("a")) {
            System.out.println("Invalid input. Please try again.");
            userType = keyboard.nextLine().toLowerCase();
        }
        //Show appropriate menu
        if (userType.equals("c")) {
            showClubMenu();
        }
        else if (userType.equals("a")) {
            showAdminMenu();
        }
        else {
            System.out.println("Invalid input. Please try again.");
```

```
            }
        }



    }
}
```

```
  public class Booking {
    public Venue venue;
    public int startTime;
    public int endTime;
    public String clubName;
    public boolean approvedByAdmin;

    public Booking(Venue venue, int startTime, int endTime, String clubName, boolean
approvedByAdmin) {
        this.venue = venue;
        this.startTime = startTime;
        this.endTime = endTime;
        this.clubName = clubName;
        this.approvedByAdmin = approvedByAdmin;
    }

    public String toString() {
        return this.startTime + " - " + this.endTime + " (" + this.clubName + ") @ " +
this.venue.name;
    }

}

import java.util.LinkedList;

class BTNode
{
    BTNode left, right;
    public Object[] data;

    public BTNode()
    {
        left = null;
        right = null;
        data = null;
    }
```

```
public BTNode(String club, LinkedList<Booking> bookings)
{
    left = null;
    right = null;
    data = new Object[]{club, bookings};
}

public void setLeft(BTNode n)
{
    left = n;
}

public void setRight(BTNode n)
{
    right = n;
}

public BTNode getLeft()
{
    return left;
}

public BTNode getRight()
{
    return right;
}

public void setData(String club, LinkedList<Booking> bookings)
{
    data = new Object[]{club, bookings};
}

public String getClub()
{
    return (String)data[0];
}

public LinkedList<Booking> getBookingsList()
{
    return (LinkedList<Booking>)data[1];
}
```

```
}

import java.util.*;

public class Venue {
   public String name;
   public int startTime;
   public int endTime;
   public ArrayList<Booking> bookings;
   public Stack<String> history;

   public Venue() {
       this.name = "";
       this.startTime = 0;
       this.endTime = 0;
       this.bookings = new ArrayList<Booking>();
       this.history = new Stack<String>();
   }

   public void createVenue() {
       Scanner keyboard = new Scanner(System.in);
       System.out.println("Enter the venue name:");
       this.name = keyboard.nextLine().toUpperCase();
       System.out.println("Enter the venue opening time:");
       this.startTime = keyboard.nextInt();
       System.out.println("Enter the venue closing time:");
       this.endTime = keyboard.nextInt();
       history.push("Venue created: " + this.name + " (" + this.startTime + " - " +
this.endTime + ")");

       for(int i = this.startTime; i < this.endTime; i+=100) {
           this.bookings.add(new Booking(this, i, i+100, "", true));
       }
       history.push("All slots created for " + this.name);
   }

   public String toString() {
       return this.name + " (" + this.startTime + " - " + this.endTime + ")";
   }
```

```
    public void showBookings() {
        for(int i = 0; i < this.bookings.size(); i++) {
            System.out.println("BOOKING #" + (i+1) + ": " +
this.bookings.get(i).toString());
        }
    }

    public void showHistory() {
        for(String event : this.history) {
            System.out.println(event);
        }
    }
}

import java.util.LinkedList;

class BT {
    private BTNode root;

    public BT() {
        root = null;
    }

    public boolean isEmpty() {
        return root == null;
    }

    public void insert(String club, LinkedList<Booking> bookings) {
        root = insert(root, club, bookings);
    }

    private BTNode insert(BTNode node, String club, LinkedList<Booking> bookings) {
        if (node == null)
            node = new BTNode(club, bookings);
        else {
            if (node.getRight() == null)
                node.right = insert(node.right, club, bookings);
            else
                node.left = insert(node.left, club, bookings);
        }
        return node;
    }
```

```
public int countNodes() {
    return countNodes(root);
}

private int countNodes(BTNode r) {
    if (r == null)
        return 0;
    else {
        int l = 1;
        l += countNodes(r.getLeft());
        l += countNodes(r.getRight());
        return l;
    }
}

public boolean search(String clubName) {
    return search(root, clubName);
}

private boolean search(BTNode r, String val) {
    if (r.getClub().equals(val))
        return true;
    if (r.getLeft() != null)
        if (search(r.getLeft(), val))
            return true;
    if (r.getRight() != null)
        if (search(r.getRight(), val))
            return true;
    return false;
}

public BTNode getTargetNode(String clubName) {
    if(!search(clubName)) {
        return null;
    }
    return getTargetNode(root, clubName);
}

private BTNode getTargetNode(BTNode r, String val) {
    if (r.getClub().equals(val))
        return r;
```

```
        if (r.getLeft() != null)
            if (getTargetNode(r.getLeft(), val) != null)
                return getTargetNode(r.getLeft(), val);
        if (r.getRight() != null)
            if (getTargetNode(r.getRight(), val) != null)
                return getTargetNode(r.getRight(), val);
        return null;
    }
}
```