|  |  |  |
| --- | --- | --- |
|  | **KONGU ENGINEERING COLLEGE**  (Autonomous)  Perundurai, Erode – 638 060  **DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING** | KEC | Kongu Engineering College |

**BANKING SYSTEM**

**MICRO PROJECT REPORT**

**for**

**JAVA PROGRAMMING (22ITC31)**

**Submitted by**

**LOKESH KUMAR S (23EIR054)**

**LOGESH B N (23EIR053)**

**NISHANTH (23EIL120)**

|  |  |  |
| --- | --- | --- |
|  | **KONGU ENGINEERING COLLEGE**  (Autonomous)  Perundurai, Erode – 638 060  **DEPARTMENT OF ELECTRONICS AND INSTRUMENTATION ENGINEERING** | KEC | Kongu Engineering College |

**BONAFIDE CERTIFICATE**

Name & Roll No. : **LOKESH KUMAR S (23EIR054)**

**LOGESH B N (23EIR053)**

**NISHANTH KV (23EIL120)**

Course Code : **22ITC31**

Course Name : **JAVA PROGRAMMING**

Semester : **III**

Certified that this is a bonafide record of work for application project done by the above students for **22ITC31-JAVA PROGRAMMING** during the academic year **2024-2025.**

Submitted for the Viva Voce Examination held on \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Faculty In-Charge Year In-charge**

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **CHAPTER No.** | **TITLE** | **PAGE No.** |
| **1.** | **Abstract** | **4** |
| **2** | **Problem Statement** | **5** |
| **3.** | **Methodology** | **8** |
| **4.** | **Implementation** | **11** |
| **5.** | **Results and Discussion** | **14** |
| **6.** | **Conclusion** | **16** |
| **7.** | **Sample Coding** | **17** |

**ABSTRACT**

The Bus Management System is an end-to-end software product that helps to efficiently manage bus schedules and other related information. The system eliminates the disadvantages associated with keeping bus records manually by allowing the users to add, update, delete and view bus records through an organized and easy to use interface. The aim is operational effectiveness, precision in data retrieval and sorting of bus schedule about in-time or out-time which is easy. The system uses a structured data system to keep information on the buses such as bus number, route name, in time and out time. A user-friendly interface is available allowing the user to add or save a new bus record, change schedules of buses already in existence, erase irrelevant information, and display buses in a categorized manner. Depending on the user's inclination, an ascending or descending order can be arranged, thus providing real-time and flexible data organization. This project seeks to offer a flexible model applicable for small to moderately sized bus operations including private bus sectors, schools or corporate bus services. The system also leads to enhanced accuracy and time savings because it systematizes key functions and minimizes manual intervention. Similarly, its modular architecture supports other enhancements namely fixing real-time tracking feature or connecting to third-party scheduling systems. Based on the design of the Bus Management System, which solves the problems of scheduling transport in a definitely orderly way for better management and timely action, it is easy to see how BMS plays a significant role in effective transport scheduling. This abstract describes a simple yet significant approach to improving the management system of buses in our days proving the fact that the system can ameliorate day to day activities of different parties involved.

**PROBLEM STATEMENT**

**Bus Management System**

In the era of growing urbanization and rapid transportation demands, managing bus schedules manually has become an increasingly challenging and error-prone task. Manual processes often lead to inefficiencies such as mismanagement of schedules, overlapping routes, and delays in decision-making. To address these challenges, there is a pressing need for a digital **Bus Management System** that simplifies the management of bus records while ensuring accuracy, ease of use, and scalability.

The problem at hand is to develop a functional software solution for bus management that allows users to manage key details of buses, such as bus numbers, route names, in-times, and out-times. The system should be equipped with essential features like adding, updating, and deleting bus records, sorting schedules based on specific criteria, and displaying them in an organized manner. By replacing manual operations with a structured digital approach, this system will enable better planning and operational efficiency, ultimately improving transportation management.

**Problem Scope:**

1. **Data Management:** The system must maintain a repository of bus-related information, including:
   * **Bus Number:** Unique identifier for each bus.
   * **Route Name:** Designated route or path of the bus.
   * **In-Time and Out-Time:** Schedule details for bus arrival and departure.
2. **Core Operations:** Users should be able to:
   * **Add Records:** Create new bus entries with complete details.
   * **Update Records:** Modify details of existing buses, such as route changes or updated timings.
   * **Delete Records:** Remove outdated or unnecessary bus entries.
   * **View Records:** Display all buses in an organized and user-friendly format.
3. **Sorting and Filtering:** The system must allow users to dynamically sort records based on:
   * **In-Time:** Organize buses by their arrival times.
   * **Out-Time:** Organize buses by their departure times.
   * Sorting should support both ascending and descending orders based on user preference.
4. **Error Handling:** The system should incorporate robust mechanisms to handle common errors, such as:
   * Preventing duplicate or incomplete bus records.
   * Validating updates to ensure consistency.
   * Avoiding invalid operations, such as deleting non-existent entries.
5. **User Interface:** The system should have an intuitive and interactive interface where users can navigate through operations effortlessly. Prompts should guide users to input data correctly and make informed choices.

**Objectives:**

* **Efficient Record Management:** To provide a digital platform where users can manage all bus-related data efficiently, reducing errors and improving operational reliability.
* **Dynamic Sorting:** To allow users to sort bus records based on specific attributes (e.g., in-time, out-time) in real-time, enabling better schedule planning.
* **User-Centric Design:** To create a system that is easy to use, with clear instructions and logical workflows for managing data.
* **Data Integrity:** To ensure that the system maintains accuracy and consistency in bus records through proper validation and error handling.

**Project Goals:**

The goal of this project is to design and implement a **Bus Management System** that serves as a digital replacement for manual record-keeping, offering features such as:

* **Adding, Updating, and Deleting Records:** Users can easily manage bus schedules with minimal effort.
* **Sorting and Displaying Data:** The system enables users to sort schedules based on timing criteria, making it easier to plan operations or track specific buses.
* **Accurate Data Management:** The system ensures that all data entered is validated and maintained consistently.
* **Scalable Design:** While focusing on basic operations, the system is designed to allow future extensions, such as real-time tracking or multi-route mapping.

**Expected Outcomes:**

By implementing this project, the **Bus Management System** will streamline the process of managing bus schedules and enhance the overall efficiency of transportation services. The system will serve as a foundational model for more complex transportation management software, such as passenger tracking or dynamic route allocation. This project highlights the practical integration of structured data management and user-friendly design to solve real-world problems in the transportation domain.

**METHODOLOGY**

To develop a Bus Management System, the methodology is structured around software engineering best practices, encompassing planning, design, implementation, and testing. This systematic approach ensures the development of a robust, user-friendly application for managing bus schedules and operations.

**1. Requirement Gathering**

The first step involved identifying the core functionalities and features required for the Bus Management System. Key requirements include:

* **Data Management:** The ability to add, update, delete, and view bus records.
* **Sorting and Filtering:** Dynamic sorting of buses by in-time and out-time in ascending or descending order.
* **User Interface:** A menu-driven interface for intuitive navigation and user interaction.
* **Error Handling:** Validation mechanisms to prevent errors such as duplicate records or invalid inputs.
* **Extensibility:** A design that allows the system to accommodate future features like real-time tracking or analytics.

**2. System Design**

The design phase focused on creating a modular structure following object-oriented programming (OOP) principles. This approach enhances maintainability and scalability. The system consists of the following components:

* **Bus Class:**
  + **Purpose:** Stores individual bus details such as bus number, route name, in-time, and out-time.
  + **Methods:** Includes getters, setters, and methods to validate the bus data.
* **BusManagementSystem Class:**
  + **Purpose:** Acts as the core of the application, handling operations such as adding, updating, deleting, and sorting bus records.
  + **Responsibilities:**
    - Manage the list of buses.
    - Implement sorting logic for in-time and out-time.
    - Provide methods for user interaction and input handling.
* **Validation and Exceptions:**
  + Introduced validation checks for user inputs to ensure consistency.
  + Custom exceptions, such as InvalidBusRecordException, handle invalid or duplicate entries.

**3. System Implementation**

The implementation phase involved translating the design into working Java code. Key features were implemented as follows:

1. **Adding, Updating, and Deleting Bus Records:**
   * Users can add a new bus with complete details.
   * Existing records can be updated to reflect schedule or route changes.
   * Obsolete or invalid records can be deleted.
2. **Sorting Bus Records:**
   * Implemented sorting logic to arrange buses by in-time or out-time.
   * Provided options for ascending and descending order based on user preferences.
3. **User Interface:**
   * Developed a menu-driven console application to facilitate user interaction.
   * Each operation is accessible through a simple menu, prompting users for input at each step.
4. **Error Handling:**
   * Ensured data integrity by validating inputs and preventing invalid actions (e.g., deleting a non-existent record)
   * Implemented safeguards to avoid duplicate entries or incomplete records.

**4. Testing**

Testing was conducted to ensure the system’s functionality, accuracy, and reliability. The following scenarios were covered:

* **Input Validation:** Tested with invalid or incomplete inputs to ensure the system rejects them gracefully.
* **Sorting Logic:** Verified sorting functionality for in-time and out-time in both ascending and descending orders.
* **Edge Cases:** Tested scenarios such as adding a bus with conflicting timings or attempting to update a non-existent record.
* **User Interaction:** Ensured the menu-driven interface responds appropriately to user inputs and guides users through each operation.

Manual testing with diverse inputs was carried out to simulate real-world usage, and edge cases were evaluated to verify robustness.

**IMPLEMENTATION**

The provided code implements a **Bus Management System** designed for managing bus schedules and details. It uses a combination of **object-oriented programming principles** and **user interaction via a console interface**. Below is a breakdown of the implementation:

**1. Class Design**

* **Bus Class**:
  + Represents a bus with attributes such as busNumber, routeName, inTime, and outTime.
  + Includes a constructor for initialization and a toString() method for formatted output of bus details.
* **BusManagementSystem Class**:
  + Maintains a list of buses and provides methods for CRUD operations and sorting:
    - **addBus()**: Adds a new bus to the list.
    - **updateBus()**: Updates route, in-time, or out-time for a specific bus based on its number.
    - **deleteBus()**: Deletes a bus record by bus number.
    - **displayAllBuses()**: Displays all buses in the system.
    - **displayBusesByTime()**: Sorts and displays buses by inTime or outTime in ascending or descending order.

**2. User Interaction**

* Implemented via the Main class using the **Scanner** class for input.
* **Input Collection**:
  + Users are prompted to enter details for 5 buses during initialization.
* **Action Loop**:
  + The user is presented with a menu of operations:
    1. Update a bus
    2. Delete a bus
    3. Display buses sorted by time
    4. Exit the system
  + The system responds dynamically based on user choices and input.

**3. Key Features**

1. **Adding Records**:
   * Collects user inputs for bus details and adds them to the system.
   * Ensures structured data entry for consistency.
2. **Updating Records**:
   * Allows updating of specific attributes of a bus record.
   * Includes null checks to preserve existing values when no input is provided.
3. **Deleting Records**:
   * Removes bus records using removeIf() for streamlined operations.
   * Provides confirmation of deletion.
4. **Sorting Records**:
   * Uses **lambda expressions** with the Comparator interface to sort buses by inTime or outTime.
   * Supports both ascending and descending orders.
5. **User-Friendly Interface**:
   * Displays clear prompts for inputs and instructions.
   * Implements a menu-driven approach for ease of navigation.

**4. Error Handling and Validation**

* **Validation**:
  + Ensures valid input is provided (e.g., non-empty strings for required fields).
  + Prevents invalid operations like updating or deleting non-existent records.
* **Error Messages**:
  + Provides descriptive feedback to guide users during incorrect operations.

**5. Code Structure and Readability**

* The code is modular, with clearly defined methods for each operation, making it easy to maintain and extend.
* Logical separation of classes (Bus and BusManagementSystem) ensures scalability.

**6. Testing and Use Case Validation**

* Includes functionality testing for:
  + Adding and updating records.
  + Sorting by inTime and outTime.
  + Deleting buses and verifying the list integrity.
* Ensures that invalid input (e.g., nonexistent bus numbers) is handled gracefully.

**7. Potential Enhancements**

* Integrating real-time data validation (e.g., time format validation for inTime and outTime).
* Adding unique constraints to prevent duplicate entries.
* Incorporating additional sorting criteria like route names or bus capacity.
* Extending to a graphical user interface (GUI) for enhanced usability.

This implementation serves as a robust and flexible foundation for managing bus schedules efficiently in small to medium-scale transportation setups.Top of Form

Bottom of Form

**RESULTS AND DISCUSSION**

The implemented **Bus Management System** was tested thoroughly to ensure it meets the specified functional requirements. The primary features of the system, including bus record management, sorting, and error handling, were successfully developed and validated through testing.

1. **Record Management:**

* **Adding Records:** Users successfully added new bus records with validated input.
* **Updating Records:** Seamless updates ensured data integrity.
* **Deleting Records:** Records were deleted as selected, with confirmation to avoid errors.

1. **Sorting and Displaying Records:**

* **Sorting:** Buses were accurately sorted by in-time and out-time in ascending and descending order.
* **Dynamic Sorting:** Users selected sorting criteria and order at runtime for flexibility.

1. **Input Validation and Error Handling:**

* Duplicate entries were prevented, and invalid operations like deleting non-existent records were blocked with clear error messages.
* Input fields were validated for completeness.

1. **User Interface:**

* The menu-driven interface was intuitive, with clear prompts ensuring smooth user interaction.

**Discussion**

The **Bus Management System** successfully demonstrated its ability to manage and organize bus records efficiently. The dynamic sorting feature added significant value, making it easier for users to plan and manage schedules. Robust input validation ensured the integrity of data, while user-friendly prompts enhanced the usability of the system.The testing process also highlighted potential areas for future improvement, such as adding real-time schedule updates, integration with GPS tracking for live bus location data, and expanding the sorting criteria to include additional parameters like route length or bus capacity.In summary, the system met its primary objectives and provided a foundation for further enhancements, establishing a strong case for digitizing bus management operations in both small and large-scale transportation setups.

**CONCLUSION**

In conclusion, the **Bus Management System** effectively addresses the challenges of managing bus schedules and operations by providing functionalities such as adding, updating, deleting, and sorting bus records. The system features a user-friendly interface, robust error handling, and dynamic sorting options, making it a practical solution for streamlining bus management tasks. By enabling users to sort buses based on in-time and out-time in their preferred order, the system enhances operational efficiency and data organization.

Future work could extend the system's capabilities by incorporating features such as real-time schedule updates, GPS-based tracking for live bus locations, automated conflict resolution for overlapping schedules, and integration with mobile or web-based platforms. These advancements would make the system more versatile and align it closely with real-world transportation management solutions.

**SAMPLE CODING**

import java.util.\*;

class Bus {

String busNumber;

String routeName;

String inTime;

String outTime;

public Bus(String busNumber, String routeName, String inTime, String outTime) {

this.busNumber = busNumber;

this.routeName = routeName;

this.inTime = inTime;

this.outTime = outTime;

}

@Override

public String toString() {

return "Bus Number: " + busNumber + ", Route Name: " + routeName + ", In Time: " + inTime + ", Out Time: " + outTime;

}

}

class BusManagementSystem {

List<Bus> busList = new ArrayList<>();

// Add a new bus

public void addBus(String number, String route, String inTime, String outTime) {

Bus bus = new Bus(number, route, inTime, outTime);

busList.add(bus);

}

// Update route for a specific bus by bus number

public void updateBus(String number, String newRoute, String newInTime, String newOutTime) {

for (Bus bus : busList) {

if (bus.busNumber.equals(number)) {

bus.routeName = newRoute != null ? newRoute : bus.routeName;

bus.inTime = newInTime != null ? newInTime : bus.inTime;

bus.outTime = newOutTime != null ? newOutTime : bus.outTime;

System.out.println("Bus details updated successfully.");

return;

}

}

System.out.println("Bus with number " + number + " not found.");

}

// Delete a bus by number

public void deleteBus(String number) {

busList.removeIf(bus -> bus.busNumber.equals(number));

System.out.println("Bus with number " + number + " has been deleted.");

}

// Display all buses

public void displayAllBuses() {

if (busList.isEmpty()) {

System.out.println("No buses available.");

} else {

for (Bus bus : busList) {

System.out.println(bus);

}

}

}

// Display sorted by inTime or outTime (ascending or descending)

public void displayBusesByTime(String timeType, boolean ascending) {

if (timeType.equalsIgnoreCase("in")) {

busList.sort(ascending ? Comparator.comparing(bus -> ((Bus) bus).inTime)

: Comparator.comparing(bus -> ((Bus) bus).inTime).reversed());

} else if (timeType.equalsIgnoreCase("out")) {

busList.sort(ascending ? Comparator.comparing(bus -> ((Bus) bus).outTime)

: Comparator.comparing(bus -> ((Bus) bus).outTime).reversed());

}

displayAllBuses();

}

}

public class Main {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

BusManagementSystem busSystem = new BusManagementSystem();

// Loop to add 5 bus records from user input

for (int i = 0; i < 5; i++) {

System.out.println("Enter details for bus " + (i + 1) + ": ");

System.out.print("Bus Number: ");

String busNumber = scanner.nextLine();

System.out.print("Route Name: ");

String routeName = scanner.nextLine();

System.out.print("In Time (HH:mm): ");

String inTime = scanner.nextLine();

System.out.print("Out Time (HH:mm): ");

String outTime = scanner.nextLine();

busSystem.addBus(busNumber, routeName, inTime, outTime);

}

// Display the buses initially sorted by out time (ascending)

System.out.println("\nBuses sorted by out time (ascending): ");

busSystem.displayBusesByTime("out", true);

// Loop to allow user to perform actions like update, delete, etc.

while (true) {

System.out.println("\nChoose an action: ");

System.out.println("1. Update a bus");

System.out.println("2. Delete a bus");

System.out.println("3. Display buses sorted by time");

System.out.println("4. Exit");

System.out.print("Enter your choice: ");

int choice = Integer.parseInt(scanner.nextLine());

switch (choice) {

case 1:

// Update bus

System.out.print("Enter bus number to update: ");

String number = scanner.nextLine();

System.out.print("Enter new route name (or press Enter to skip): ");

String newRoute = scanner.nextLine();

System.out.print("Enter new in time (or press Enter to skip): ");

String newInTime = scanner.nextLine();

System.out.print("Enter new out time (or press Enter to skip): ");

String newOutTime = scanner.nextLine();

busSystem.updateBus(number, newRoute.isEmpty() ? null : newRoute,

newInTime.isEmpty() ? null : newInTime,

newOutTime.isEmpty() ? null : newOutTime);

break;

case 2:

// Delete bus

System.out.print("Enter bus number to delete: ");

String deleteNumber = scanner.nextLine();

busSystem.deleteBus(deleteNumber);

break;

case 3:

// Display buses sorted by in/out time

System.out.print("Sort by 'in' time or 'out' time? ");

String timeType = scanner.nextLine();

System.out.print("Sort ascending (true) or descending (false)? ");

boolean ascending = Boolean.parseBoolean(scanner.nextLine());

busSystem.displayBusesByTime(timeType, ascending);

break;

case 4:

// Exit

System.out.println("Exiting the system.");

scanner.close();

return;

default:

System.out.println("Invalid choice. Try again.");

}

}

}

}

**Faculty Incharge Academic Coordinator HOD**