FPT UNIVERSITY
INFORMATION SYSTEMS MAJOR
FACULTY OF INFORMATION TECHNOLOGY

FPT UNIVERSITY

HOMESTAY BOOKING MANAGEMENT SYSTEM REPORT

SUPERVISOR : Ms. Than Thi Ngoc Van
SUBJECT : LAB211
CLASS : SE2043

STUDENT : Nguyen Hai Duong (SE203568)

HO CHI MINH CITY 2/2026

Contents

Chapter 1

Introduction

1.1 Project Overview

The Homestay Booking Management System is a console-based Java application developed to digitize and optimize the operational processes of a homestay business. The system serves as a centralized platform to manage tour packages and handle customer bookings.

1.2 Scope and Target Audience

- **Scope:** Manages the entire tour lifecycle (creation, listing, updates) and booking management (booking, modification, cancellation, searching).
- **Target Audience:** Administrators or homestay staff who require a powerful data entry tool without the complexity of a graphical user interface.

Chapter 2

System Overview

The system operates as a menu-driven application with 10 main functions:

1. **Add new tour:** Create the new tour with validation (tour's informations, homestay booking, validation between dates).
2. **Update a tour ID:** Modify the attributes of an existing tour.
3. **List the Tours with departure dates earlier than the current date:** Find tours with departure dates earlier than today.
4. **List total Booking amount for future tours:** List the total booking amount for tours with departure dates later than the current date.
5. **Add a new Booking:** Create a new booking.
6. **Remove a Booking by bookingID:** Delete an existing booking.
7. **Update a Booking by bookingID:** Modify an existing booking.
8. **List all Booking by fullName:** Find and display the booking information by full or partial name.
9. **Statistics:** Statistics on the total number of tourists who have booked each homestay.
10. **Quit program:** Upon exit, if any related data files have been modified, the system shall prompt the user to save changes.

Chapter 3

System Architecture

3.1 Layered Architecture

- **Viewer:** Main, Menu - Used to display the console menu for users.
- **Business:** TourManagement, BookingManagement - Include regex, logic, display, control data object layer.
- **Data Access Object (DAO):** BookingDao, FileManager, HomeStayDao, IgenericDao, ToursDao - Access to file information.
- **Model:** Tours, Homestays, Booking - Create a structure for objects.
- **Utilities:** Tools layers for all programs.

3.2 n-Layer Architecture Design

The system utilizes a multi-layered architecture (n-tier) to ensure modularity, maintainability, and scalability:

- **[Business Layer]**
  Components: TourManagement, BookingManagement
  Handles core business logic, performs calculations, and validates constraints.
- **[Data Access Layer]**
  Components: TourDAO, BookingDAO, HomeStayDAO
  Interacts directly with `.txt` files via the IDAO interface.
- **[Model Layer]**
  Components: HomeStay, Tour, Booking
  Represents the entity objects of the system.
- **[Utilities (Neutral Layer)]**
  Provides helper classes used across the system:
  - **Inputter:** Handles data entry and prevents input skipping.
  - **Acceptable:** Contains Regular Expressions (Regex) and format validation methods.

3.3 Class Diagram

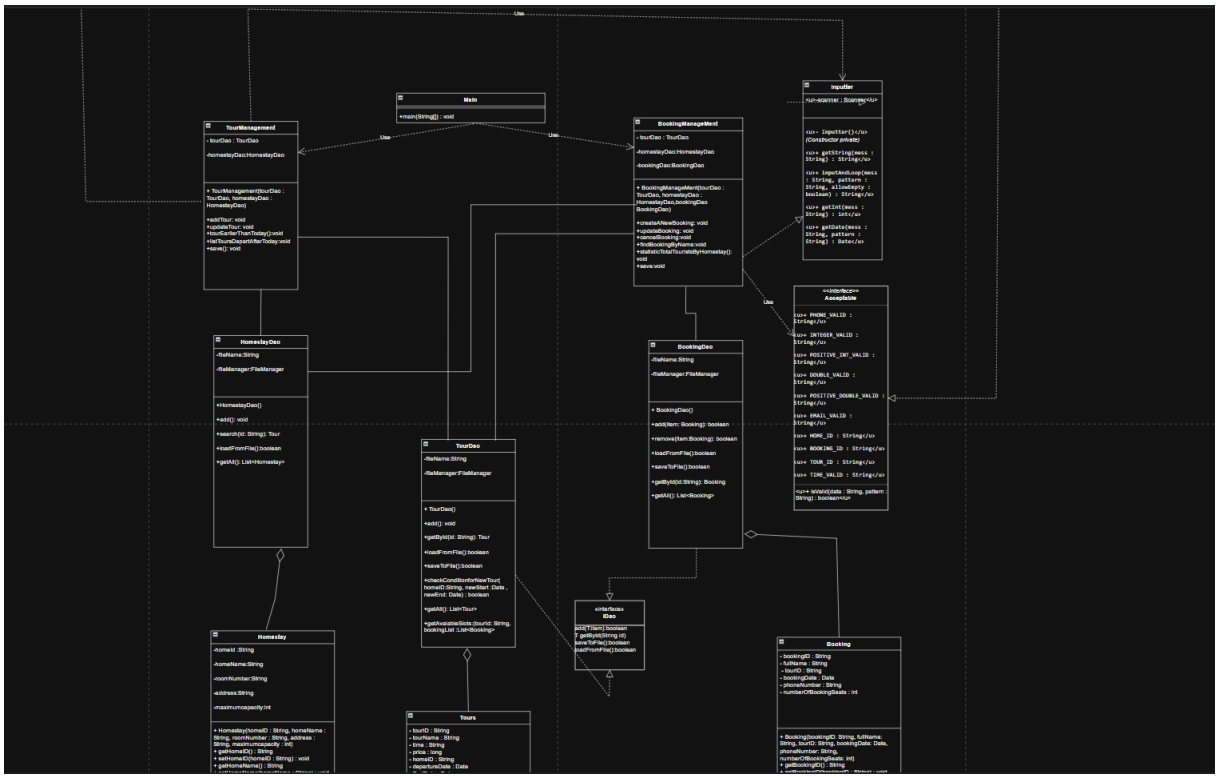

Figure 3.1: Detailed Class Diagram of the System

Chapter 4

Software Flow and Pseudocode

4.1 Function 1: Add a new Tour

```
ALGORITHM AddNewTour
BEGIN
    REPEAT
        INPUT tourID
        IF tourDao.getById(tourID) IS NOT NULL THEN
            DISPLAY [Error: TourID already exists]
        ELSE
            BREAK Loop
        END IF
    END REPEAT

    INPUT tourName
    INPUT timeString (Format: X days Y nights)
    EXTRACT days FROM timeString

    REPEAT
        INPUT price
        IF price <= 0 THEN
            DISPLAY [Error: Price must be positive]
        ELSE
            BREAK Loop
        END IF
    END REPEAT

    REPEAT
        INPUT homeID
        homestay = homestayDao.search(homeID)
        IF homestay IS NULL THEN
            DISPLAY [Error: Homestay not found]
        ELSE
            BREAK Loop
        END IF
    END REPEAT

    REPEAT
```

```
36          INPUT departureDate
37          CALCULATE endDate = departureDate + days
38          IF NOT tourDao.checkSchedule(homeID, departureDate,
                endDate) THEN
39              DISPLAY [Error: Date conflict or invalid]
40          ELSE
41              BREAK Loop
42          END IF
43      END REPEAT
44
45      REPEAT
46          INPUT numTourists
47          maxCap = homestay.getMaximumCapacity()
48          IF numTourists < 1 OR numTourists > maxCap THEN
49              DISPLAY [Error: Invalid number of tourists]
50          ELSE
51              BREAK Loop
52          END IF
53      END REPEAT
54
55      CREATE newTour(tourID, tourName, price, homeID, dates,
            numTourists)
56      CALL tourDao.add(newTour)
```

Listing 4.1: Add tour pseudocode

4.1.1 Function Workflow Description

The following steps describe the logical flow of the `addTour` function:

Step 1: Identity Validation
The system prompts the user to input a unique `Tour ID`. It iterates until a non-duplicate ID is provided by checking against the database (`tourDao`).

Step 2: Data Parsing and Input
The user enters the `Tour Name`, `Duration` (e.g., "3 days 2 nights"), and `Price`. The system uses Regex to extract the number of days from the duration string and validates that the price is a positive integer.

Step 3: Homestay Association
The system requests a `Home ID`. It queries the `homestayDao` to ensure the entered ID corresponds to an existing homestay. If invalid, the system prompts the user to retry.

Step 4: Schedule Logic and Conflict Check
The user inputs the `Departure Date`. The system calculates the `End Date` based on the duration extracted in Step 2. It then calls `checkConditionforNewTour` to verify that the homestay is available (not fully booked) during this specific period.

Step 5: Capacity Verification and Persistence
The user specifies the maximum number of tourists. The system compares this against the homestay's `Maximum Capacity`. If the number is valid, a new `Tour` object is instantiated and saved to the database.

4.2 Function 2: Update Tour by ID

```
ALGORITHM UpdateTour
BEGIN
    REPEAT
        DISPLAY [Enter Tour ID]
        INPUT tourID
        tour = tourDao.getById(tourID)

        IF tour IS NULL THEN
            DISPLAY [Error: Tour not found]
            EXIT
        ELSE IF tour.isBooked() IS TRUE THEN
            DISPLAY [Error: Tour is already booked, cannot update]
            EXIT
        ELSE
            DISPLAY [Current Tour Info]
            BREAK Loop
        END IF
    END REPEAT

    INPUT tourName
    IF tourName IS NOT EMPTY THEN
        tour.setName(tourName)
    END IF

    INPUT timeString (Format: X days Y nights)
    IF timeString IS NOT EMPTY THEN
        tour.setTime(timeString)
    END IF
    EXTRACT days FROM tour.getTime()

    REPEAT
        INPUT homeID
        IF homeID IS EMPTY THEN BREAK Loop

        homestay = homestayDao.search(homeID)
        IF homestay IS NOT NULL THEN
            tour.setHomeID(homeID)
            BREAK Loop
        ELSE
            DISPLAY [Error: Homestay not found]
        END IF
    END REPEAT

    REPEAT
        INPUT newDepartureDate
        IF newDepartureDate IS EMPTY THEN BREAK Loop

        CALCULATE newEndDate = newDepartureDate + days
        currentHomeID = tour.getHomeID()
```

```
50
51          IF tourDao.checkCondition(currentHomeID, newDepartureDate,
                newEndDate) THEN
52              tour.setDepartureDate(newDepartureDate)
53              tour.setEndDate(newEndDate)
54              DISPLAY [New End Date calculated]
55              BREAK Loop
56          ELSE
57          END IF
58      END REPEAT
59
60      REPEAT
61          INPUT numTourists
62          IF numTourists IS EMPTY THEN BREAK Loop
63
64          currentHome = homestayDao.search(tour.getHomeID())
65          maxCap = currentHome.getMaximumCapacity()
66
67          IF numTourists >= 1 AND numTourists <= maxCap THEN
68              tour.setNumberOfTourists(numTourists)
69              BREAK Loop
70          ELSE
71              DISPLAY [Error: Capacity must be between 1 and maxCap]
72          END IF
73      END REPEAT
74
75  END
```

Listing 4.2: Pseudocode for update tour by tourID

4.2.1 Update Function Workflow

The following steps describe the logical flow of the `updateTour` function:

Step 1: ID and Status Verification

The system requests the `Tour ID`. It verifies that the tour exists in the database. Crucially, it checks the `isBooked` status; if the tour is already booked, the system denies the update request to prevent data inconsistency.

Step 2: Basic Information Update

The system allows the user to input a new `Tour Name` and `Duration`. If the user leaves these fields empty, the existing values remain unchanged. If the duration is updated, the system re-extracts the number of days for upcoming date calculations.

Step 3: Homestay Re-assignment

The user can optionally input a new `Home ID`. The system validates this ID against the `homestayDao`. If valid, the tour is linked to the new homestay; otherwise, the user is prompted to retry or keep the old homestay.

Step 4: Schedule Recalculation

The user can input a new `Departure Date`. The system calculates the new `End Date`

based on the duration (from Step 2). It then invokes `checkConditionforNewTour` to ensure the target homestay is free during the new timeframe before applying changes.

Step 5: Capacity Re-validation
The user can update the maximum number of tourists. The system fetches the capacity of the currently assigned homestay (which might have changed in Step 3) and validates that the new tourist count fits within limits before saving the final state.

4.3 Function 3:List past tours

```
ALGORITHM TourEarlierThanToday
BEGIN
    SET today = GetCurrentDate()
    SET resultList = EMPTY
    SET allTours = tourDao.getAll()

    FOR EACH tour IN allTours DO
        departureDate = tour.getDepartureDate()

        IF departureDate < today THEN
            APPEND tour.toString() TO resultList
        END IF
    END FOR

    IF resultList IS NOT EMPTY THEN
        DISPLAY resultList

    ELSE
        DISPLAY [Not founded!!]
    END IF
END
```

Listing 4.3: List of past tour

4.3.1 Earlier Tours Function Workflow

The following steps describe the logical flow of the `tourEarlierThanToday` function:

Step 1: Initialization and Retrieval
The system initializes the current date timestamp (`today`) to serve as the comparison baseline. It then retrieves the complete collection of existing tours from the database via `tourDao.getAll()`.

Step 2: Temporal Filtering
The system iterates through every tour in the collection. For each tour, it compares the `Departure Date` against the current date. If the departure date is strictly earlier than today, the tour's details are formatted and appended to a result buffer.

Step 3: Conditional Output
The system checks if any tours were found in Step 2. If the result buffer is populated, it

renders a formatted table (Header, Body, Footer) displaying the past tours. If the buffer is empty, it outputs a "Not found" message to the user.

4.4 Function 4: Tour after current date

```
ALGORITHM ListToursDepartAfterToday
BEGIN
    SET today = GetCurrentDate()
    SET laterList = EMPTY LIST

    FOR EACH tour IN tourDao.getAll() DO
        IF tour.getDepartureDate() > today THEN
            ADD tour TO laterList
        END IF
    END FOR

    REM Sort by Total Revenue (Price * Tourists)
    SORT laterList BY (price * numberOfTourists) ASCENDING

    IF laterList IS NOT EMPTY THEN
        FOR EACH tour IN laterList DO
            DISPLAY tour.toString()
        END FOR
    ELSE
        DISPLAY [Not found]
    END IF
END
```

Listing 4.4: List Future Tours by Total Amount

4.4.1 List Upcoming Tours Workflow

The following steps describe the logical flow of the `listToursDepartAfterToday` function:

Step 1: Temporal Filtering
The system initializes the current date. It iterates through the entire tour collection, filtering and retaining only those tours with a `Departure Date` strictly after the current system date.

Step 2: Revenue-Based Sorting
The system defines a custom comparator to calculate the projected revenue for each tour (Price $\times$ Number of Tourists). The filtered list is then sorted in **ascending order** based on this calculated value.

Step 3: Data Formatting
The system iterates through the sorted list, converting each tour object into a string format suitable for tabular display.

Step 4: Conditional Output

If the filtered list contains data, the system prints a structured table (Header, Body, Footer) to the console. If the list is empty (no upcoming tours found), it outputs a notification message.

4.5 Function 5: Create a new booking

```
ALGORITHM CreateNewBooking
BEGIN
    REPEAT
        INPUT bookingId
        IF bookingDao.getById(bookingId) IS NOT NULL THEN
            DISPLAY [Error: Booking ID already exists]
        ELSE
            BREAK Loop
        END IF
    END REPEAT

    INPUT fullName

    REPEAT
        INPUT tourId
        selectedTour = tourDao.getById(tourId)
        IF selectedTour IS NULL THEN
            DISPLAY [Error: Tour not found]
        ELSE
            BREAK Loop
        END IF
    END REPEAT

    REPEAT
        INPUT bookingDate
        departureDate = selectedTour.getDepartureDate()
        IF bookingDate >= departureDate THEN
            DISPLAY Error
        ELSE
            BREAK Loop
        END IF
    END REPEAT

    REPEAT

        INPUT numPeople
        remainingSlots = tourDao.getAvailableSlots(tourId)

        IF numPeople <= 0 THEN
            DISPLAY [Error: Number must be positive]
        ELSE IF numPeople > remainingSlots THEN
            DISPLAY [Error: Not enough slots. Available:
               remainingSlots]
```

```
43          ELSE
44              BREAK Loop
45          END IF
46      END REPEAT
47
48      INPUT phoneNumber
49
50      CREATE bookingObj
51
52      IF bookingDao.add(bookingObj) IS TRUE THEN
53          selectedTour.setBooked(TRUE)
54          DISPLAY [Add successfully]
55      ELSE
56          DISPLAY [Add failed]
57      END IF
58  END
```

Listing 4.5: List Future Tours by Total Amount

4.5.1 Booking Creation Workflow

The following steps describe the logical flow of the `createANewBooking` function:

Step 1: Identification and Personal Data
The system validates the `Booking ID` to ensure uniqueness. Once a valid ID is secured, the system captures the customer's `Full Name`.

Step 2: Tour Association
The system requests a `Tour ID` and validates its existence in the database. This ensures the booking is linked to a valid tour object.

Step 3: Temporal Logic Verification
The user inputs the `Booking Date`. The system compares this against the selected tour's `Departure Date` to enforce the rule that bookings must occur strictly before the tour departs.

Step 4: Availability and Capacity Check
The system calculates the remaining slots by subtracting current bookings from the tour's capacity. It validates that the requested `Number of Tourists` is positive and does not exceed the available slots.

Step 5: Finalization and Status Update
After capturing the `Phone Number`, the system saves the new booking. Crucially, upon successful addition, it updates the associated Tour's status to `booked = true` to reflect active activity.

4.6 Function 6: Cancel booking

```
ALGORITHM CancelBooking
BEGIN
    REPEAT
        DISPLAY [Enter Booking ID]
        INPUT bookingId
        booking = bookingDao.getById(bookingId)

        IF booking IS NOT NULL THEN
            BREAK Loop
        ELSE
            DISPLAY [Error: Booking does not exist]
        END IF
    END REPEAT

    isRemoved = bookingDao.remove(booking)

    IF isRemoved IS TRUE THEN
        tourId = booking.getTourID()
        availableSlots = tourDao.getAvailableSlots(tourId)

        IF availableSlots > 0 THEN
            tour = tourDao.getById(tourId)
            tour.setBooked(FALSE)
        END IF

        DISPLAY [successfully]
    ELSE
        DISPLAY [Error: failed]
    END IF
END
```

Listing 4.6: Cancel Booking

4.6.1 Booking Cancelation Workflow

The following steps describe the logical flow of the `cancelBooking` function:

Step 1: Identification and Retrieval
The system prompts the user to input a `Booking ID`. It loops continuously until a valid ID corresponding to an existing booking is found in the database (`bookingDao`).

Step 2: Record Deletion
Upon confirmation of existence, the system executes the removal of the booking object from the database via the DAO.

Step 3: Tour Status Synchronization
After successful deletion, the system recalculates the available slots for the associated tour. If the available capacity is greater than zero (meaning the tour is no longer fully booked), the system updates the tour's status (`booked`) to `false`, effectively reopening it for new reservations.

4.7 Function 7: Update booking

```
ALGORITHM UpdateBooking
BEGIN
    REPEAT
        INPUT bookingId
        booking = bookingDao.getById(bookingId)

        IF booking IS NOT NULL THEN
            BREAK Loop
        ELSE
            DISPLAY [Error: Booking not found]
            EXIT
        END IF
    END REPEAT

    INPUT fullName
    IF fullName IS NOT EMPTY THEN
        booking.setFullName(fullName)
    END IF

    SAVE oldTourId = booking.getTourID()

    REPEAT
        INPUT newTourId
        IF newTourId IS EMPTY THEN BREAK Loop

        IF tourDao.getById(newTourId) IS NOT NULL THEN
            booking.setTourID(newTourId)
            BREAK Loop
        ELSE
            DISPLAY [Error: Tour not found]
            EXIT
        END IF
    END REPEAT

    REPEAT
        INPUT newDate
        IF newDate IS EMPTY THEN BREAK Loop

        tourDepDate = tourDao.getById(booking.getTourID()).
           getDepartureDate()
        IF newDate < tourDepDate THEN
            booking.setBookingDate(newDate)
            BREAK Loop
        ELSE
            DISPLAY [Error: Booking must be before departure]
        END IF
    END REPEAT

    INPUT numTourists
```

```
49      IF numTourists IS NOT EMPTY THEN
50          currentTourId = booking.getTourID()
51          availableSlots = tourDao.getAvailableSlots(currentTourId)
52
53          IF numTourists > availableSlots THEN
54              DISPLAY [Error: Not enough slots. Available:
                    availableSlots]
55              ROLLBACK booking.setTourID(oldTourId)
56              ROLLBACK booking.setSeats(oldSeats)
57              EXIT
58          ELSE
59              booking.setNumberOfBookingSeats(numTourists)
60              IF numTourists == availableSlots THEN
61                  tourDao.getById(currentTourId).setBooked(FALSE)
62              END IF
63          END IF
64      END IF
65
66      DISPLAY [ successfully]
67  END
```

Listing 4.7: Update booking

4.7.1 Booking Modification Workflow

The following steps describe the logical flow of the `updateBooking` function:

Step 1: Booking Identification
The system prompts for the `Booking ID`. It validates the ID against the `bookingDao`. If the booking does not exist, the process terminates immediately to prevent unauthorized access.

Step 2: Tour Re-assignment (With State Saving)
The user can optionally update the `Full Name` and the `Tour ID`. Before applying a new Tour ID, the system saves the `oldTourId`. This is crucial for the "rollback" mechanism in Step 5 if capacity checks fail.

Step 3: Temporal Validation
The user can input a new `Booking Date`. The system ensures this date strictly precedes the departure date of the currently assigned tour (whether valid from Step 2 or unchanged).

Step 4: Capacity Check and Transaction Logic
The user inputs the new number of tourists. The system calculates available slots for the target tour. If the requested number exceeds availability, the system triggers a **Rollback**: it reverts the Tour ID to the saved `oldTourId` and exits, cancelling the update to ensure data integrity.

Step 5: Finalization
If the capacity check passes, the system updates the seat count. It also checks if the new booking fills the remaining slots exactly, potentially updating the tour's status flags before displaying the final result.

4.8 Function 8: Search Booking

```
ALGORITHM FindBookingByName
BEGIN

    INPUT rawName

    REM Normalize input for case-insensitive search
    searchQuery = rawName.trim().toLowerCase()

    SET found = FALSE

    allBookings = bookingDao.getAll()

    FOR EACH booking IN allBookings DO
        storedName = booking.getFullName().toLowerCase()

        IF storedName CONTAINS searchQuery THEN
            formattedDate = FORMAT(booking.getBookingDate(), "dd/
               MM/yyyy")



            SET found = TRUE
        END IF
    END FOR

    IF found IS FALSE THEN
        DISPLAY [No booking found!]
    END IF
END
```

Listing 4.8: Search Booking

4.8.1 Search Booking Workflow

The following steps describe the logical flow of the `findBookingByName` function:

Step 1: Input Normalization
The system prompts the user to enter a full or partial name. It processes this input by trimming whitespace and converting the string to lowercase to ensure a case-insensitive comparison (e.g., matching "john" with "John").

Step 2: Iterative Search and Display
The system outputs a table header and iterates through the entire list of bookings. For each record, it checks if the stored full name contains the search query. If a match is found, the system formats the booking date and prints the booking details immediately as a table row, setting a flag to indicate success.

Step 3: Outcome Feedback
After checking all records, the system evaluates the success flag. If the flag remains false

(indicating no matches were found during the iteration), it outputs a “No booking found!” message to inform the user.

4.9 Function 9: StatisticTotalTourists

```
ALGORITHM StatisticTotalTourists
BEGIN

    FOR EACH tour IN tourDao.getAll() DO
        homeID = tour.getHomeID()

        maxCapacity = tour.getNumberOfTourists()
        available = tourDao.getAvailableSlots(tour.getID())

        REM Calculate currently booked seats
        bookedSeats = maxCapacity - available

        currentTotal = statisticMap.getOrDefault(homeID, 0)
        statisticMap.put(homeID, currentTotal + bookedSeats)
    END FOR


    IF statisticMap IS EMPTY THEN
        DISPLAY [No booking data]
        RETURN
    END IF

    FOR EACH homeID IN statisticMap.keySet() DO
        homestay = homeStayDao.search(homeID)

        IF homestay IS NOT NULL THEN
            homeName = homestay.getHomeName()
            totalGuests = statisticMap.get(homeID)

        END IF
    END FOR
END
```

Listing 4.9: StatisticTotalTourists

4.9.1 Statistics Workflow

The following steps describe the logical flow of the `statisticTotalTouristsByHomestay` function:

Step 1: Data Aggregation

The system initializes a Map to store the results. It iterates through every tour in the database, calculating the number of booked seats (Total Capacity - Available Slots) for each tour. These figures are accumulated into the map, grouped by the `Home ID`.

Step 2: Empty Data Validation

The system checks if the aggregation map is empty (meaning no bookings exist). If empty, it outputs a "No booking data" message and terminates the function to avoid processing an empty report.

Step 3: Entity Resolution and Reporting

The system iterates through the aggregated keys (`Home ID`). For each ID, it queries the `homestayDao` to retrieve the user-friendly `Home Name`. Finally, it prints a formatted row displaying the Homestay Name alongside the calculated total number of tourists.

Chapter 5

Deep Dive: Object-Oriented Analysis and Design

To elevate the system from a simple CRUD application to a robust management tool, several core Object-Oriented Programming (OOP) principles and advanced design patterns were applied, particularly in the booking modification logic.

5.1 Advanced OOP Implementation

The system's architecture demonstrates a high level of maturity through the following OOP characteristics:

- **Encapsulation & State Protection:** All entity attributes (e.g., `tourID`, `seats`) are hidden via private access modifiers. During the update process, the system uses local variables to store the "Original State" (`oldTourId`, `oldSeats`). This ensures that the object's integrity is maintained until the entire business transaction is validated[cite: 11, 23].
- **Transactional Integrity (Rollback Mechanism):** Unlike simple procedural updates, the `updateBooking` function treats the process as a pseudo-transaction. If the capacity check in the `TourDAO` fails, the system triggers a manual rollback by re-assigning the original values to the `Booking` object before exiting. This prevents the system from entering an "Inconsistent State" where a booking exists for a non-existent or over-capacity tour.
- **Status Synchronization (Observer-like Logic):** The relationship between `Booking` and `Tour` objects is dynamic. When a booking's seat count changes, the `Business Layer` automatically invokes a recalculation of the `isBooked` flag in the `Tour` object. This ensures that data remains synchronized across different layers without manual intervention from the end-user.
- **Abstraction via DAO Pattern:** By using the `IDao` interface, the business logic remains decoupled from the specific storage implementation (Text files). This allows for future scalability, such as migrating to a SQL database without rewriting the core business rules.

Chapter 6

Validation and Business Logic

6.1 Validation

6.1.1 Tour Constraints

- **Tour ID:** Must be unique across the system.
- **Mandatory Information:** `tourName` and `time` cannot be empty.
- **Price:** Must be a positive integer.
- **Homestay ID:** Must follow the format "HS0000" and exist in the HomeStay list.
- **Dates:** Departure and End dates must follow "dd/MM/yyyy". End date must be after departure date.

6.1.2 Booking Constraints

- **Booking ID:** Must be unique and follow format "B00000".
- **Tour ID:** Must correspond to an existing Tour.
- **Booking Date:** Must be strictly earlier than the Tour's Departure Date.
- **Phone Number:** Must consist of exactly 10 digits.

6.2 Business Logic Architecture

6.2.1 Tour Management Logic

The system enforces the following business rules for managing tours:

- **Uniqueness:** Every tour must be identified by a unique ID. Attempting to duplicate an ID triggers a validation error.
- **Homestay Association:** A tour cannot exist in isolation; it must be linked to a valid, existing Homestay. The system validates the `HomeID` against the Homestay database during creation and updates.

- **Temporal Logic:** The `End Date` of a tour is never manually input. Instead, it is automatically calculated based on the `Departure Date` and the duration (parsed from a string like "3 days 2 nights"). This prevents date inconsistencies.
- **Immutability of Booked Tours:** To preserve the integrity of customer contracts, a tour's details cannot be updated if it already has active bookings (`isBooked = true`).

6.2.2 Booking Management Logic

The booking subsystem handles the reservation process with the following logic:

- **Availability Check:** Before accepting a booking, the system calculates the `Available Slots` for the specific tour ($Available = TourCapacity - CurrentBookings$). If the requested number of tourists exceeds this value, the booking is rejected.
- **Chronological Constraints:** A booking can only be placed if the `Booking Date` is strictly earlier than the tour's `Departure Date`.
- **Status Synchronization:** The system automatically updates the status of a Tour.
  - When a booking is added, the Tour is marked as `booked`.
  - When a booking is cancelled, the system recalculates availability. If slots open up, the Tour's status is reverted to `available`.
- **Transactional Updates:** When updating a booking (e.g., changing the number of people), the system employs a "Rollback" mechanism. If the new request exceeds capacity, the system reverts all changes to the previous valid state to prevent data corruption.

6.2.3 Reporting and Analytics

The system provides analytical capabilities to support decision-making:

- **Temporal Filtering:** The system can segregate tours into "Past" (Earlier than today) and "Upcoming" (After today) categories. Upcoming tours are sorted by potential revenue (Price $\times$ Tourists) to prioritize high-value packages.
- **Search Capabilities:** Users can locate specific bookings using case-insensitive partial name matching.
- **Statistical Aggregation:** The system generates a summary report calculating the total number of tourists served by each Homestay, aggregating data across multiple tours.

Chapter 7

Conclusion

The Homestay Booking Management System successfully meets the functional requirements through a robust Object-Oriented design. The application of layered architecture ensures data integrity and maintainability. The system uses file storage to ensure portability and simplicity.