Project Proposal: Travel Agency Management System

Peter Jiang
ID: 2430026079

April 30, 2025

1 Introduction

This proposal outlines the implementation of a Travel Agency Management System (TAMS). The system streamlines operations for travel agencies by providing a comprehensive platform for managing travel bookings, itineraries, customer information, and other critical business operations. TAMS employs Object-Oriented Programming (OOP) principles, utilizes file I/O for data persistence, and features a user-friendly graphical interface built with Java Swing.

2 Real-World Scenario and Business Logic

2.1 Scenario Overview

The Travel Agency Management System serves the needs of modern travel agencies that offer various travel packages to different destinations. The agency handles customer bookings, maintains trip itineraries, manages payments, collects customer reviews, and organizes activities. This system digitizes these processes, enhancing operational efficiency and customer satisfaction.

2.2 Business Logic

The core business logic of the Travel Agency Management System includes:

- **Package Management**: Create, update, and delete travel packages with details such as destination, duration, accommodations, activities, and pricing.
- **Customer Management**: Register new customers, update customer information, and track customer booking history.
- **Booking Process**: Allow customers to browse packages, make reservations, specify travel dates, and process payments.
- **Itinerary Management**: Create and modify detailed day-by-day itineraries for travel packages and custom trips.

- **Activity Management**: Maintain a database of activities that can be added to packages or itineraries.
- **Review System**: Collect and display customer reviews and ratings for travel packages.
- **Reporting**: Generate reports on bookings, revenue, customer statistics, and popular destinations.

3 Object-Oriented Design

3.1 Class Hierarchy

The system implements a comprehensive class structure that leverages key OOP concepts:

- **Model Package**: Contains all data entities and business logic
  - `TravelService`: Abstract class providing a base for travel offerings
  - `TravelPackage`: Concrete implementation of a pre-defined travel package
  - `CustomTrip`: Customized trip built for a specific customer
  - `Customer`: User data and contact information
  - `Booking`: Reservation details linking customers to travel packages
  - `Payment`: Financial transaction details
  - `Itinerary` and `ItineraryDay`: Trip timeline structure
  - `Activity`: Individual activities that can be part of packages
  - `Review`: Customer feedback and ratings
  - `BookingStatus`, `PaymentMethod`, `PaymentStatus`: Enums for tracking status
  - Interfaces: `Bookable`, `Reviewable`
- **View Package**: GUI components
  - `MainWindow`: Application container with tabbed navigation
  - `BasePanel`: Abstract base for all panels
  - `PackagesPanel`: Travel package management interface
  - `TravelPackagesPanel`: Alternative package management view
  - `CustomersPanel`: Customer data management
  - `BookingsPanel`: Reservation processing
  - `ActivitiesPanel`: Activity management
  - `ReviewsPanel`: Customer feedback management
  - `ReportsPanel`: Analytics and reporting
- **Controller Package**: Application logic
  - `TravelAgencyController`: Coordinates between views and model

- **Util Package**: Utility services
  - `DataManager`: Handles file I/O for data persistence
- **Exceptions Package**: Custom exceptions
  - `BookingException`: Booking-related errors
  - `PaymentProcessException`: Payment processing errors

3.2 UML Class Diagram

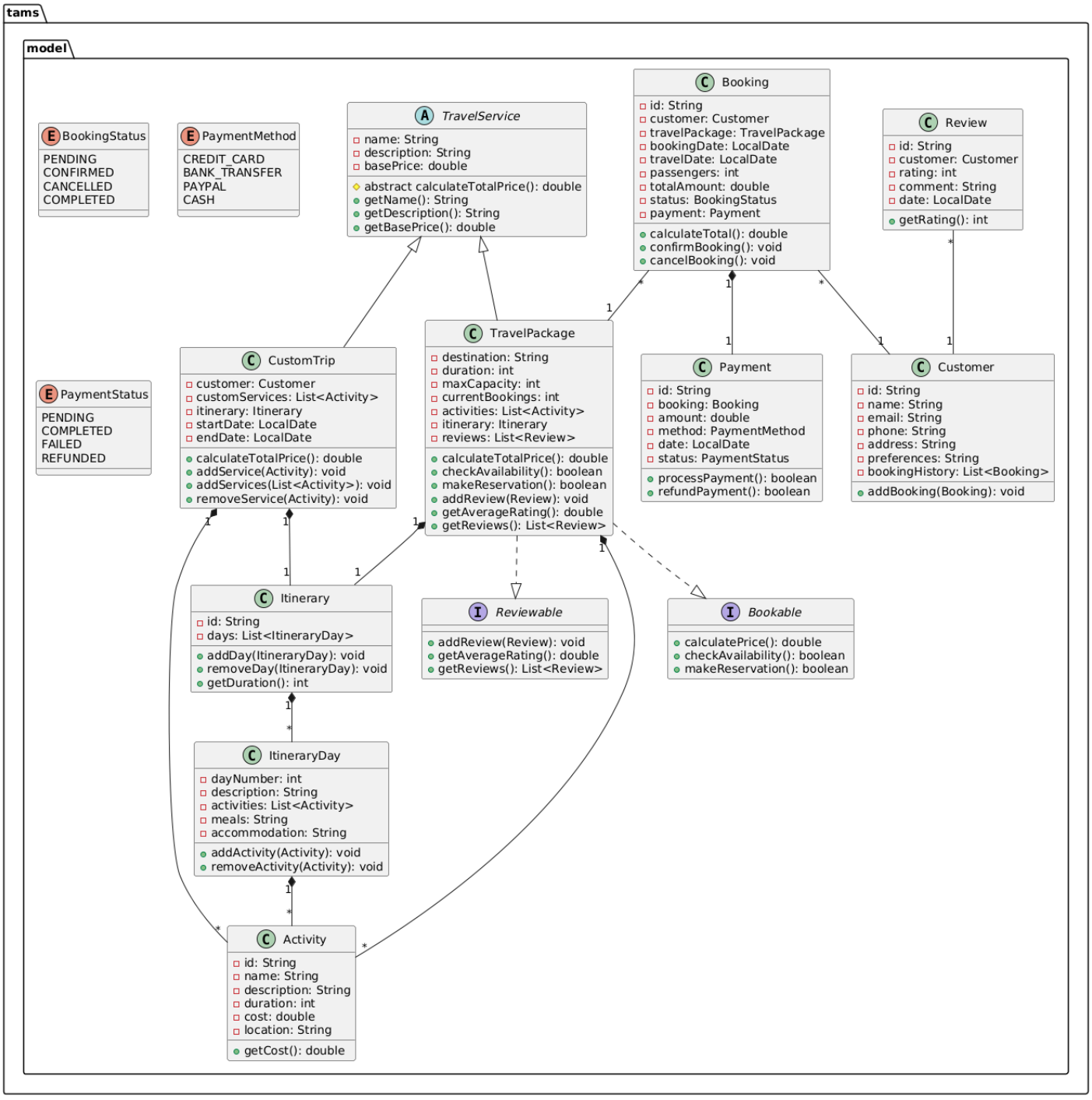

Figure 1: Model

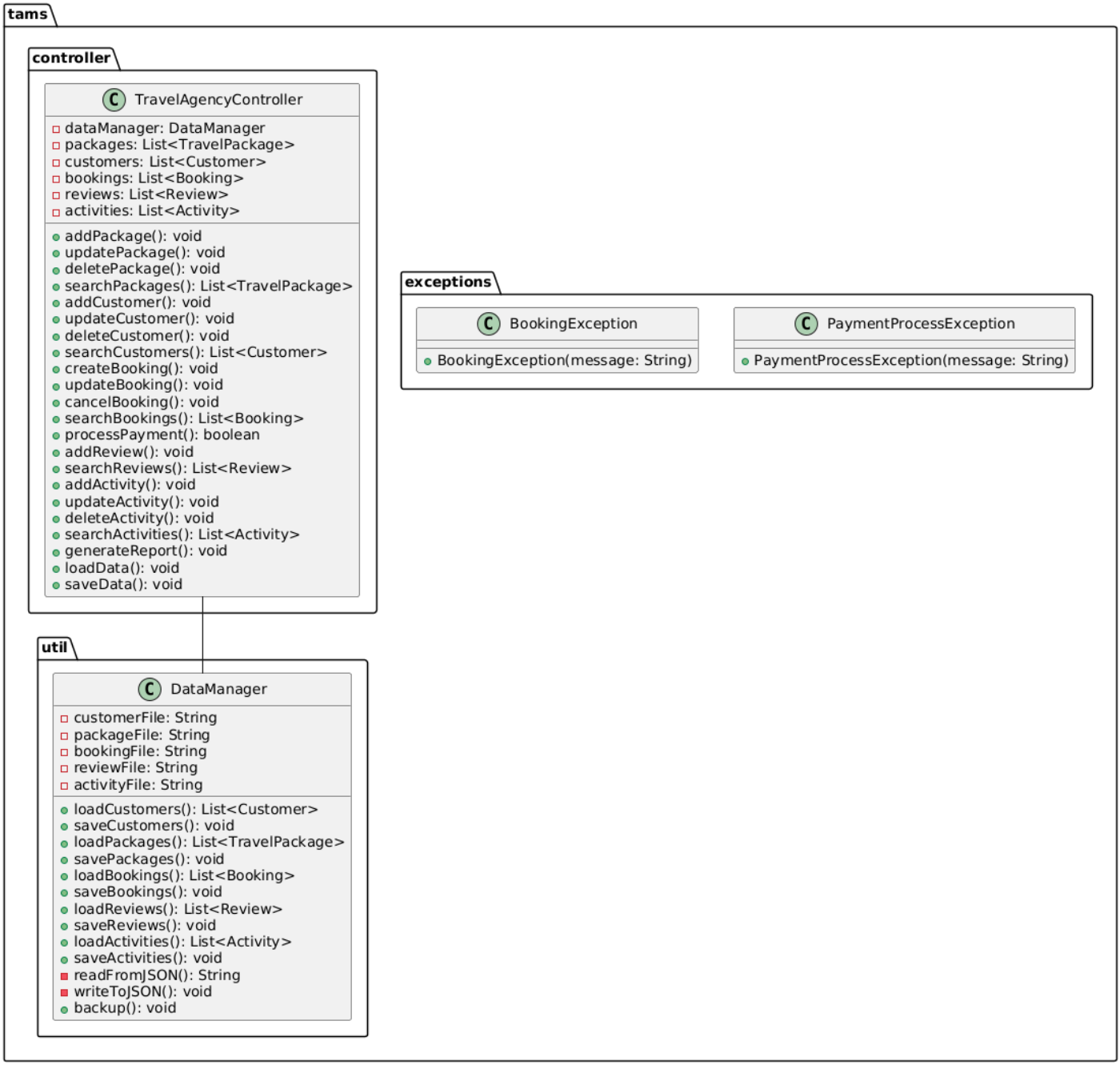

Figure 2: Controller, Utilizes and Exceptions

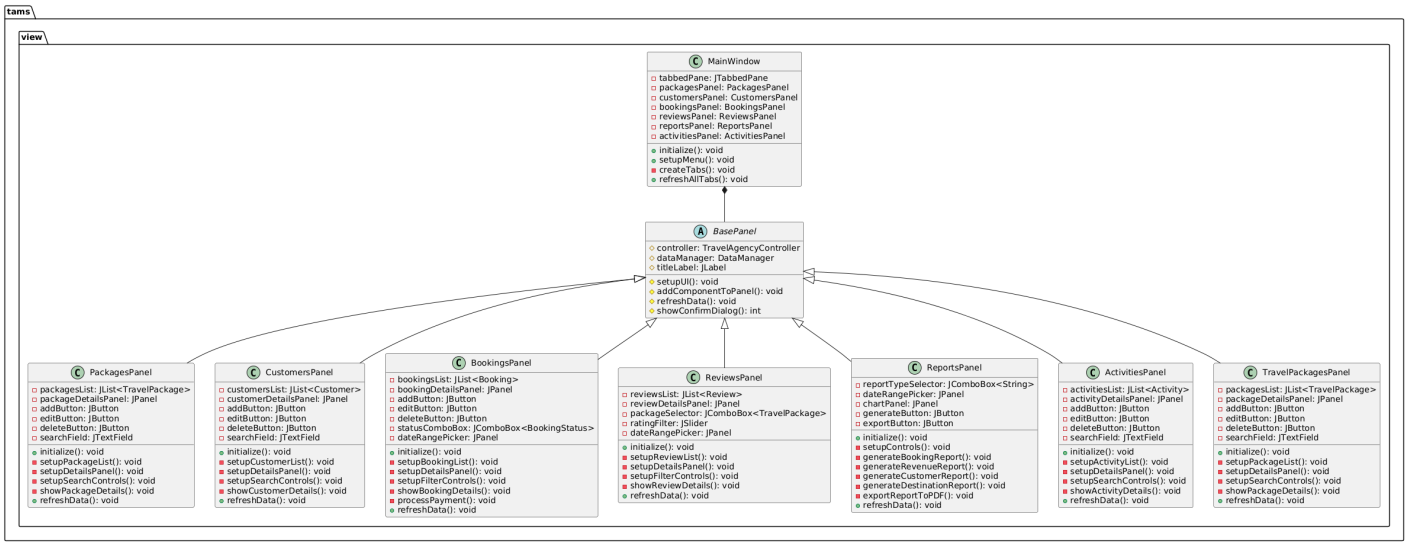

Figure 3: View

3.3 OOP Concepts Implementation

3.3.1 Interfaces

The system implements two main interfaces:

- **Bookable**: Defines methods that any bookable service must implement, including `calculatePrice()`, `checkAvailability()`, and `makeReservation()`.
- **Reviewable**: Defines methods for entities that can receive customer reviews, including `addReview()`, `getAverageRating()`, and `getReviews()`.

3.3.2 Abstract Classes

- **TravelService**: An abstract class that provides common attributes and methods for all travel services, including name, description, and base price. It includes abstract methods like `calculateTotalPrice()` that must be implemented by subclasses.
- **BasePanel**: Abstract class for all GUI panels, providing common functionality like setup methods, data refresh capabilities, and dialog handling to ensure consistent UI patterns.

3.3.3 Inheritance

The inheritance hierarchy includes:

- `TravelService` serves as the parent class for `TravelPackage` and `CustomTrip`, allowing shared behavior while enabling specialized implementation.
- `BasePanel` is extended by all specific panels (`PackagesPanel`, `CustomersPanel`, etc.), providing common UI functionality.
- Each subclass inherits common attributes and methods while implementing specific behaviors.

3.3.4 Polymorphism

Polymorphism is demonstrated through:

- Different implementations of `calculateTotalPrice()` in `TravelPackage` and `CustomTrip`.
- Use of the `Bookable` interface allowing different travel services to be treated uniformly.
- Panel-specific implementations of `refreshData()` and `setupUI()` methods.
- Ability to handle different types of travel services through a common parent class.

3.3.5 Method Overloading

- Multiple constructors in classes like `Booking` and `Customer` with different parameter sets.
- `addService()` method in `CustomTrip` with versions for adding single services or collections.
- `searchPackages()` method in `TravelAgencyController` with different parameter combinations.

3.3.6 Method Overriding

- Override of `calculateTotalPrice()` in subclasses of `TravelService`.
- Override of `toString()` in various model classes for customized string representation.
- Override of `refreshData()` in panel classes to update their specific content.

3.3.7 Exception Handling

Custom exceptions include:

- `BookingException`: For handling errors in the booking process, such as unavailable packages or invalid booking details.
- `PaymentProcessException`: For managing payment processing errors, like failed transactions or invalid payment methods.

4 Graphical User Interface Design

The GUI is implemented using Java Swing, providing an intuitive interface for users to interact with the system.

4.1 GUI Examples

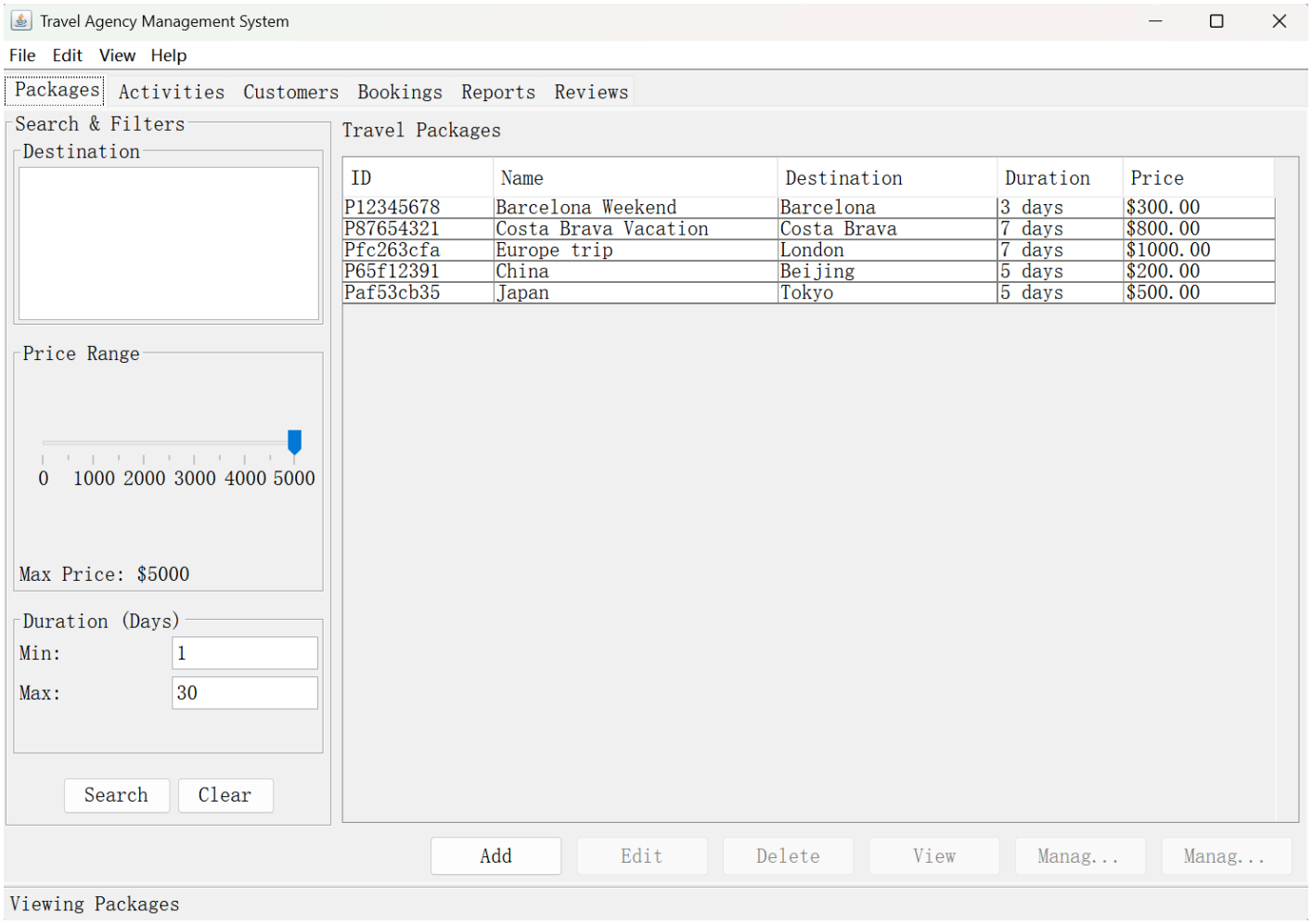

Figure 4: Main Window (Packages Tab)

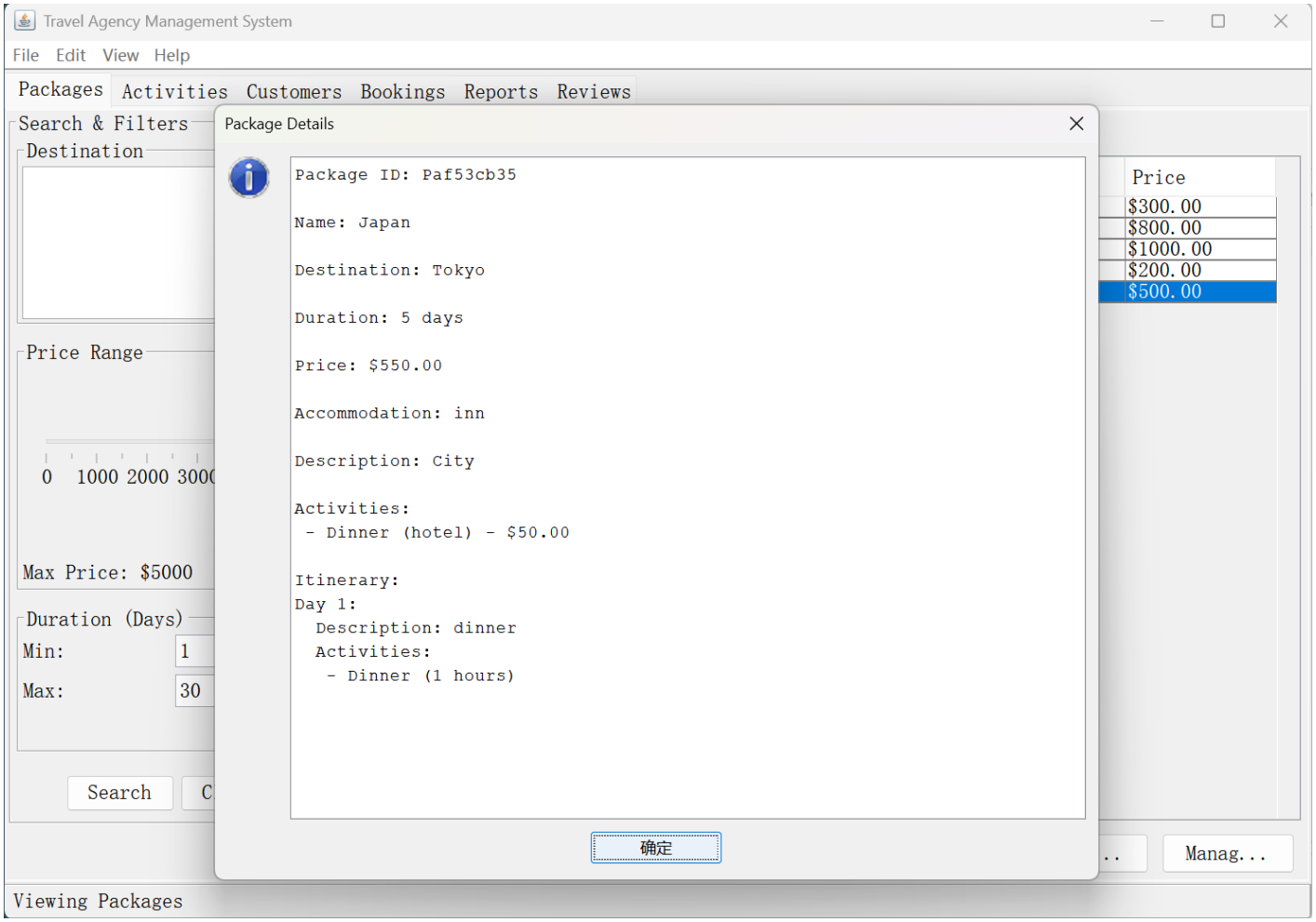

Figure 5: Packages Panel (View packages detail)

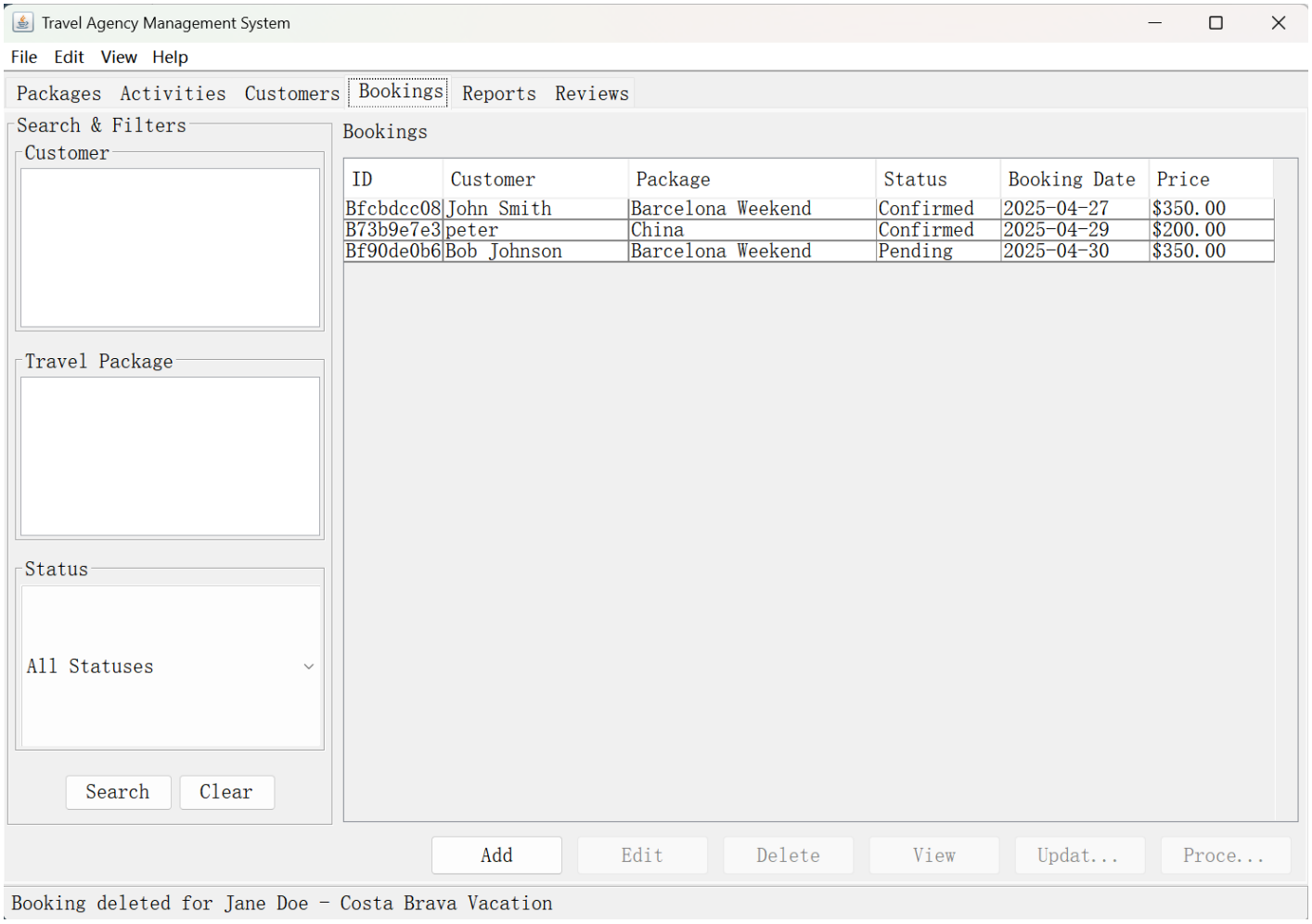

Figure 6: Bookings Panel

The GUI implements the following features:

- **Tab-based Navigation**: Easy navigation between different system functions through a tabbed interface:
  - Packages Tab: For managing travel packages
  - Customers Tab: For managing customer data
  - Bookings Tab: For processing bookings
  - Activities Tab: For managing activities
  - Reviews Tab: For handling customer feedback
  - Reports Tab: For analytics and reporting
- **Search and Filtering**: Ability to search and filter data across all modules using text fields and dropdowns.
- **CRUD Operations**: Interface elements for creating, reading, updating, and deleting records:
  - **Create**: Forms for adding new packages, customers, bookings, and activities

  - **Read**: Tables and detailed views to display information
  - **Update**: Forms to modify existing records
  - **Delete**: Options to remove records with confirmation dialogs
- **Reports Section**: Visual representation of data including:
  - Booking statistics
  - Revenue analysis
  - Customer demographics
  - Popular destinations and packages
- **Interactive Elements**: Form controls including:
  - Text fields for data entry
  - Dropdown menus for selecting options
  - Date pickers for scheduling
  - Buttons for actions
  - List components for displaying records

5 Data Storage Design

5.1 File I/O Architecture

The system uses file I/O to store and retrieve data persistently. The data storage is organized as follows:

- **JSON File Format**: Data is stored in JSON format for readability and ease of handling.
- **File Structure**:
  - `customers.json`: Customer information and preferences.
  - `packages.json`: Travel package details and availability.
  - `bookings.json`: Booking records and status.
  - `reviews.json`: Customer reviews and ratings.
  - `activities.json`: Available activities for packages.
- **Data Loading**: When the application starts, it loads data from these files into memory using the DataManager class.
- **Data Saving**: Changes are saved back to the files automatically when data is modified and when the application closes.
- **Data Validation**: Inputs are validated before being written to ensure data integrity.

5.2 Data Processing

The system includes:

- **Data Validation**: Validating user inputs before saving to ensure data integrity.
- **Sorting and Searching**: Algorithms to efficiently sort and search through data collections.
- **Data Analysis**: Functions to analyze booking trends, popular destinations, and revenue statistics.
- **Data Export**: Capability to export reports for external use.

6 Additional Features

6.1 Advanced Search Functionality

The system includes advanced search options to:

- Search packages by destination, date range, price range, or activities
- Filter customer records by name, email, or contact information
- Search bookings by status, date, or package type
- Find activities by location, type, or price range

6.2 Data Analysis

The system offers data analysis features:

- Booking statistics and trends
- Popular destinations and activities
- Customer demographics and preferences
- Revenue analysis and projections

7 Conclusion

The Travel Agency Management System (TAMS) provides a comprehensive solution for travel agencies to manage their operations efficiently. By implementing the outlined features and following object-oriented design principles, the system offers a robust, user-friendly platform that enhances the agency's ability to serve customers and track business performance.

The system successfully meets all project requirements by implementing:

- Multiple interfaces and abstract classes
- A robust inheritance hierarchy

- Polymorphic behavior for flexible operations
- Method overloading and overriding
- Custom exception handling
- Persistent data storage through file I/O
- A comprehensive Java Swing GUI

The modular design ensures that the system can be easily maintained and extended to accommodate future requirements and enhancements.

GitHub Repository

The source code for this project is available at Github and I will keep updating it. If you have any question or feedback, please feel free to contact me at:
https://github.com/jytpeterjiang/Travel-Agency-Management-System