COM534 – Group A: User Ticket Machine Application

Student Name: Md Mashfe Islam

GitHub Repository: 6islam

Overview of the User Ticket System

The aim of the Group A component is to design and build a simple ticket-purchasing system that represents what a customer would interact with on a standard ticket machine. This version does **not** include any administrative controls, instead, it focuses entirely on the buying experience from a user's perspective. The program enables customers to view destinations, select a destination, choose a ticket type, and receive a calculated ticket price. The structure is intentionally lightweight so that the system is easy to extend in future stages.

System Requirements

The design of the program followed an object-oriented structure. The **Destination** class represents each travel location and holds the price of a single ticket, a return ticket and a counter that records how many tickets have been purchased for that destination. The **Ticket** class represents the actual ticket a user buys, containing information about the destination chosen, whether the ticket is single or return and the final calculated price. The **TicketMachine** class forms the main processing unit of the program. It stores a list of destinations, allows them to be displayed to the user and creates a Ticket object whenever a purchase is made. Finally, the **Main** class contains the program loop. This is where the user is repeatedly shown the menu and the program reads their choice until they decide to exit. These classes work together in a simple and logical way that reflects real-world behaviour in a ticket machine.

Functional Description

The requirements for this version of the system were straightforward. The program needed to show a list of destinations, display prices for single and return tickets, allow the customer to choose a destination, calculate the correct ticket price and finally record that a ticket had been sold. All these functions were successfully implemented. The system also needed to be simple for a user to navigate, so a numbered menu was used to guide the customer through each available option. Good usability was essential; therefore, the layout of the console and responses to user selections were designed to be clear and immediate. The code was stored in a private GitHub repository to ensure proper version control throughout development.

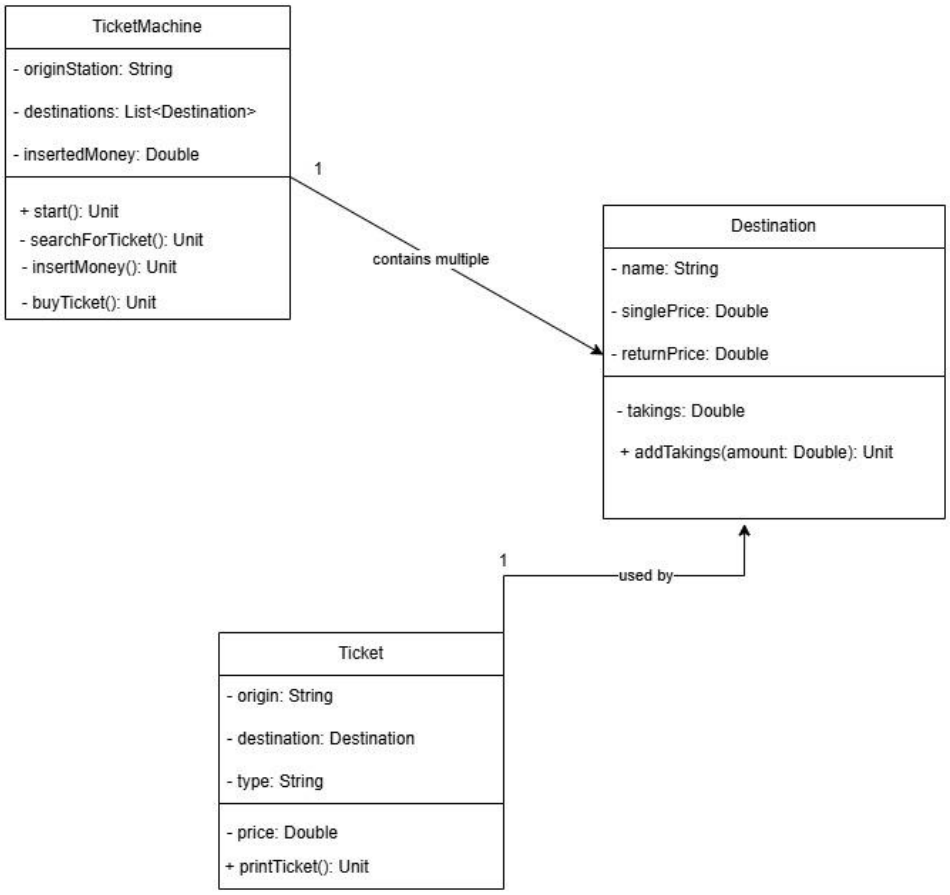
Design Approach

The application was built using an object-oriented design. Each class represents a single concept that contributes to the behaviour of the user-side ticket machine.

Class Descriptions:

The system is built around four main classes. The *Destination* class stores the prices for single and return tickets and keeps track of how many tickets have been sold. The *Ticket* class represents a user's purchase, recording the destination, ticket type and final price. The *TicketMachine* class manages all destinations, displays them to the user and creates Ticket objects when a purchase is made. The *Main* class controls the program loop and handles menu choices, allowing the user to navigate the system easily.

UML Representation



The design shows `TicketMachine` holding several `Destination` objects, and producing `Ticket` objects when the user buys a ticket.

Implementation Summary

Kotlin was chosen for the implementation due to its clear syntax and compatibility with the JVM.
The application was developed and executed in IntelliJ IDEA.
The program consists of multiple classes inside a `src/` folder, adhering to a clear logical structure.

The interaction flow is based on a simple loop:

1. Show menu
2. Read the user's choice
3. Perform the selected action
4. Return to menu

This cycle continues until the user selects "Exit".

Testing and Outcome

A range of tests were performed to validate the system:

- Destinations are displayed correctly.
- Prices match the values stored in the Destination class.
- Ticket purchases update the sales counter accurately.
- Invalid menu options are rejected gracefully.
- Both ticket types return correct price calculations.

The system behaved as expected in all test scenarios.

Reflection and Possible Improvements

Although the system meets all requirements for Group A, there are several areas where it could be enhanced:

- Introducing error handling for invalid text input.
- Saving sales information to a file so that data persists after closing the program.
- Adding a payment simulation.
- Connecting this user system with the Group B admin system in the future.