Travel-Reservation System

Ashutosh Jha, 2016A3PS0115G
Priyal Chhatrapati, 2016A8TS0378G
Shivam Agarwal, 2016A8PS0332G

Project Component

Object-Oriented Programming

(BITS F213)

Design Problem

Design and implement an application to book bus/taxi based on available source, destination, and Time of Boarding. The application should take care of the following aspects:

- Repository of bus and taxi information containing bus company name, bus number, source, destination, start time, end time, the total number of seats in the bus (capacity), and whether it is AC/Non AC.

- Reserve 'n' number of seats on a particular bus based on the availability of seats.

- Reserve taxi based on source, time of boarding, and the number of hours for which a taxi is required.

- Users should be able to create new accounts, log in to their existing accounts, change the volatile details pertaining to their account, create new bookings from their account, view previous bookings related to their account, and delete bookings in their account.

- Users should also be informed about the fare for their booking based on their preferences.

Object-Oriented Programming : Concepts Used.

Classes Description

1. Vehicle: An **Abstract Class**, it is used as a blueprint for the classes Bus and Taxi. It contains the ‘get’ and ‘set’ functions for variables common to child classes and also an abstract method.

2. Bus: This is the class for instantiation of Bus object, it extends Vehicle class and implements ACBus and NonACBus **Interfaces**. Apart from ‘get’ and ‘set’ functions for variables unique to Bus class, it also implements **Constructor Overloading** and overrides the abstract method from Vehicle class.

3. Taxi: This is the class for instantiation of the Taxi object, it extends Vehicle class. It includes **two static functions** ACTaxi and NonACTaxi. Apart from the ‘get’ and ‘set’ functions unique to Taxi class, it also implements **Constructor Overloading** and overrides the abstract method from Vehicle class.

4. Booking: This class deals with all the travel booking operations in the application. Apart from all ‘get’ and ‘set’ functions for its own private variables, it implements **Method Overloading** and **Vararg Overloading**.

5. User: This class deals with all the activities related to user account in the application. Apart from all ‘get’ and ‘set’ functions for its own private variables, it implements **Method Overloading** and **Vararg Overloading**.

6. Driver: This class contains the driver code which helps take inputs from user and implement methods from other classes, forming the API.

Nested Classes

- Vehicle - static ACTaxi
- Vehicle - static NonACTaxi

1. ACTaxi:

   We have implemented a **Static Nested Class, ‘public static class ACTaxi**’ which helps set and retrieve the rate for an AC Taxi.

2. NonACTaxi:

   We have implemented a **Static Nested Class, ‘public static class NonACTaxi**’ which helps set and retrieve the rate for a No AC Taxi.

Overloaded Methods

- Booking Class
- User Class

1. Booking Class:

   We implement **Method Overloading** with the two methods **Booking(int nSeats, String ... s) and Booking(String ... nSeats)**, where the variable nSeats differentiates the two methods for Bus and Taxi booking respectively.

2. User Class:

   We implement **Method Overloading** with the two methods **User() and User(String ... u)**, where the vararg argument differentiates between the two functions for null user object and valued user object.

Overloaded Constructors

- Bus Class
- Taxi Class

1. Bus Class:

   We implement **Constructor Overloading** with the two methods:

   Bus(String vehicleID, String companyName, String source, String destination, String startTime, String endTime, int distance, int capacity, boolean[] frequency) and

   Bus(String vehicleID, String companyName, String source, String destination, String startTime, String endTime, int distance, int capacity, boolean[] frequency, boolean ac)

   Which differ on the variable ‘ac’ which specifies whether the user wants or does not want an air conditioned vehicle, it’s absence is assumed as non AC vehicle.

2. Taxi Class:

We implement **Constructor Overloading** with the two methods:

Taxi(String vehicleID, String companyName, String source, String startTime) and

Taxi(String vehicleID, String companyName, String source, String startTime, boolean ac)

Which differ on the variable ‘ac’ which specifies whether the user wants or does not want an air conditioned vehicle, it’s absence is assumed as non AC vehicle.

Variable Argument Overloading

- Booking Class
- User Class

1. Booking Class:

   We implement **Variable Argument Overloading** with the two methods **Booking(int nSeats, String ... s) and Booking(String ... s)**, where the variable nSeats differentiates the two methods for Bus and Taxi booking respectively. And depending on first input being an integer or string, the method which will be used is decided.

2. User Class:

   We implement **Variable Argument Overloading** with the two methods **User() and User(String ... u)**, where the vararg argument differentiates between the two functions for null user object and valued user object.

Hierarchical Inheritance

- Vehicle - Bus and Taxi

1. Vehicle - Bus and Taxi Classes:

   Both classes Bus and Taxi are children of the Vehicle Class, this is an implementation of **Hierarchical Inheritance.**

Multiple Inheritance

- ACBus & NonACBus - Bus

1. ACBus and NonACBus - Bus:

   The class Bus implements both interfaces ACBus and NonACBus, this is an implementation of **Multiple Inheritance.**

Abstract Class

- Vehicle Class

1. Vehicle Class:

   The Vehicle class is an **Abstract Class** and acts as the blueprint for the classes Bus and Taxi, as they inherit from Vehicle.

Wrappers

- Integer
- Long

Package

- Automotive
- Sys

1. Automotive: We have used this **package** to contain the Vehicle, Taxi and Bus Classes, where physical real world objects have a representation.

2. Sys: We have used this **package** for service related classes Booking and User, where the user interface and activities are implemented.

Exception Handling

- User
- Driver

1. User: In the User class we have used **Exception Handling** in the methods **setPassword(String password)** and **setContactNo(String contactNo)**.

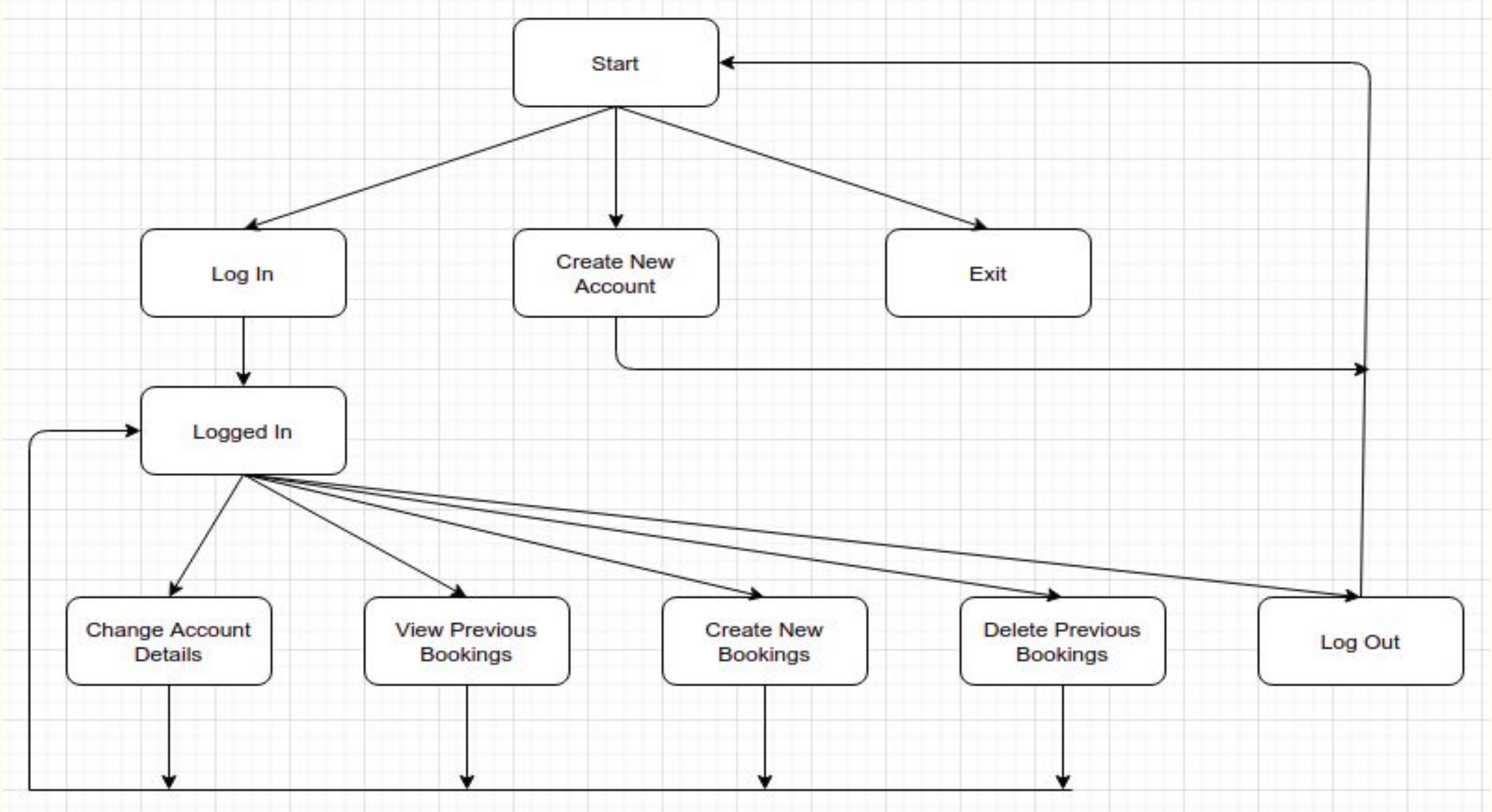
2. Driver: In the Driver class we have used **Exception Handling** in the **main** function to guide the user to give the correct input.

Program Flow

The program starts with 3 options, Login, Create new account and Exit. If the user decides to create a new account he is asked about the basic details and an account is created. If the user Log in instead he is given 5 options, to change volatile account details (password, contact number), View previous bookings, Create new booking, Delete booking or log out.

The user may book Taxi or Bus, and based on his preferences a history is maintained, along with a booking id which is used for viewing and deleting previous bookings.

The next slide includes a flowchart which gives a basic overview of the program flow.


Start
Log In
Create New Account
Exit
Logged In
Change Account Details
View Previous Bookings
Create New Bookings
Delete Previous Bookings
Log Out