Smart Vehicle Booking System - Code Flow and Description

Vilal , Hanuma, Madan, Shriom, Amal

Overview

The Smart Vehicle Booking System is designed to facilitate the reservation of smart bikes for on-campus transportation at IIIT-H. The system allows users to book bikes via a mobile app, initiate and conclude trips by scanning QR codes on bikes, and handle payment transactions manually or through an auto-deduct feature.

Description and Assumptions

Main.java

The entry point of the application. Creates a user, wallet, and payment management instance. Initiates a payment using the `makePayment` method.

PaymentMethod (Abstract Class)

Represents a generic payment method with a unique transaction ID. Subclasses: `UPIPayment`, `CardPayment`, `DeductFromSalary`, `DeductFromStudentFee`, `InAppWalletPayment`. Each class implements the `processPayment` method with specific payment logic. `InAppWalletPayment` extends `PaymentMethod` with additional properties (`walletId` and `walletBalance`). Implements specific payment logic for in-app wallet payments.

User (Class)

The `User` class is the base for all the users. Represents a system user with properties like `userId`, `name`, `email`, `password`, and `wallet`. Methods include `createAccount`, `uploadId`, `addMoneyToWallet`, `viewTripHistory`, `getUserDetails`. Staff, Students, Teachers inherit properties from this class.

Wallet (Class)

Extends `PaymentMethod`. Manages wallet-related operations such as `addMoney`, `deductMoney`, and `makePayment`. `addMoney` method allows users to add money to their wallet. `deductMoney` method deducts money unless it is less than zero.

VehicleType (Abstract Class)

Represents a generic vehicle type with a unique ID. 'Bicycle', 'Bicycle', 'Moped' extend this abstract class and provide specific details for each vehicle type. Each class has specific details and implements `getVehicleDetails`.

TripManagement (Class)

Manages user trips with specific vehicles. Methods include `startTrip`, `endTrip`, `calculateDistance`, `calculateAmountDue`, `applyFine`. Uses a unique trip ID generated by `generateTripId`. This class assumes a standard base rate, a rate per 100 meters, and a daily fine of Rs.50 for exceeding the time limit.

SmartVehicleBookingSystem (Class)

The `SmartVehicleBookingSystem` class is intended for a centralized management system for users, vehicles, parking spaces, trips, payments, and feedback. Methods include `registerUser`, `addVehicle`, `addParkingLot`, `createTrip`, `processPayment`, `addFeedback`.

Feedback (Class)

Represents user feedback with properties like `feedbackId`, `user`, `rating`, and `comment`. Method `provideFeedback` is a placeholder for processing and storing feedback.

ParkingLot (Class)

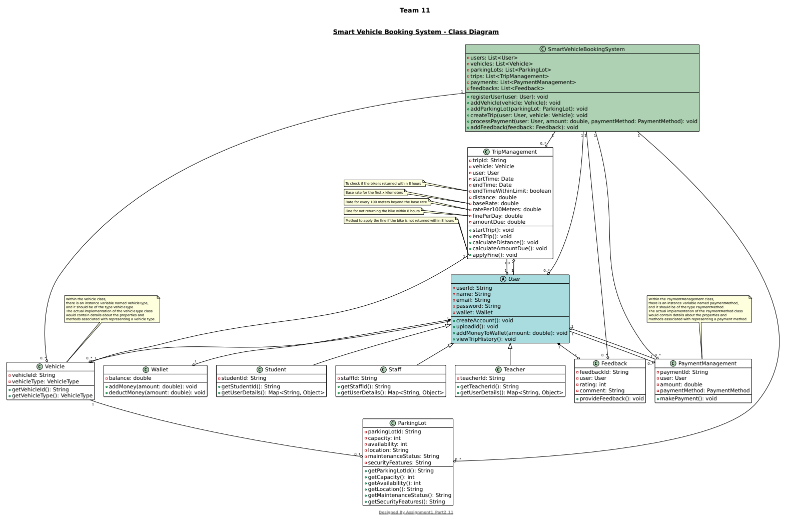
Represents a parking lot with properties like `parkingLotId`, `capacity`, `availability`, `location`, `maintenanceStatus`, `securityFeatures`. Methods include `updateAvailability`, `setMaintenanceStatus`, `setSecurityFeatures`.

PaymentManagement (Class)

The `PaymentManagement` class assumes that a user pays a specific amount using a specific payment method. Manages payments with properties like `paymentId`, `user`, `amount`, and `paymentMethod`. Method `makePayment` invokes the specific payment method's `makePayment` logic. The `makePayment` method is a general call to payment methods.

Code Structure

The code is logically organized into class modules representing different system components. Classes and methods are well-commented to explain their functionality. Relationships between classes are represented through composition and inheritance. This skeletal code is made based on the UML diagram submitted in Part 1 assignment. For a visual representation of the system structure, the complete UML diagram can be viewed through the encompassing figure 1.


Team 11
Smart Vehicle Booking System - Class Diagram
SmartVehicleBookingSystem
users: List<User>
vehicles: List<Vehicle>
parkingLots: List<ParkingLot>
trips: List<TripManagement>
payments: List<PaymentManagement>
feedbacks: List<Feedback>
registerUser(user: User): void
addVehicle(vehicle: Vehicle): void
addParkingLot(parkingLot: ParkingLot): void
createTrip(user: User, vehicle: Vehicle): void
processPayment(user: User, amount: double, paymentMethod: PaymentMethod): void
addFeedback(feedback: Feedback): void
TripManagement
tripId: String
vehicle: Vehicle
user: User
startTime: Date
endTime: Date
endTimeWithinLimit: boolean
distance: double
baseRate: double
ratePer100Meters: double
finePerDay: double
amountDue: double
startTrip(): void
endTrip(): void
calculateDistance(): void
calculateAmountDue(): void
applyFine(): void
To check if the bike is returned within 8 hours
Base rate for the first x kilometers
Rate for every 100 meters beyond the base rate
Fine for not returning the bike within 8 hours
Method to apply the fine if the bike is not returned within 8 hours
User
userId: String
name: String
email: String
password: String
wallet: Wallet
createAccount(): void
uploadId(): void
addMoneyToWallet(amount: double): void
viewTripHistory(): void
Within the Vehicle class, there is an instance variable named VehicleType, and it should be of the type VehicleType. The actual implementation of the VehicleType class would contain details about the properties and methods associated with representing a vehicle type.
Within the PaymentManagement class, there is an instance variable named paymentMethod, and it should be of the type PaymentMethod. The actual implementation of the PaymentMethod class would contain details about the properties and methods associated with representing a payment method.
Vehicle
vehicleId: String
vehicleType: VehicleType
getVehicleId(): String
getVehicleType(): VehicleType
Wallet
balance: double
addMoney(amount: double): void
deductMoney(amount: double): void
Student
studentId: String
getStudentId(): String
getUserDetails(): Map<String, Object>
Staff
staffId: String
getStaffId(): String
getUserDetails(): Map<String, Object>
Teacher
teacherId: String
getTeacherId(): String
getUserDetails(): Map<String, Object>
Feedback
feedbackId: String
user: User
rating: int
comment: String
provideFeedback(): void
PaymentManagement
paymentId: String
user: User
amount: double
paymentMethod: PaymentMethod
makePayment(): void
ParkingLot
parkingLotId: String
capacity: int
availability: int
location: String
maintenanceStatus: String
securityFeatures: String
getParkingLotId(): String
getCapacity(): int
getAvailability(): int
getLocation(): String
getMaintenanceStatus(): String
getSecurityFeatures(): String
Designed By Assignment1_Part2_11

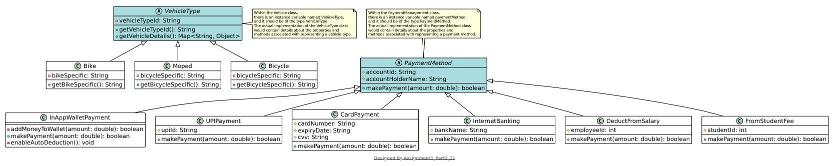

VehicleType
vehicleTypeId: String
getVehicleTypeId(): String
getVehicleDetails(): Map<String, Object>
Within the Vehicle class, there is an instance variable named VehicleType, and it should be of the type VehicleType. The actual implementation of the VehicleType class would contain details about the properties and methods associated with representing a vehicle type.
Within the PaymentManagement class, there is an instance variable named paymentMethod, and it should be of the type PaymentMethod. The actual implementation of the PaymentMethod class would contain details about the properties and methods associated with representing a payment method.
Bike
bikeSpecific: String
getBikeSpecific(): String
Moped
bicycleSpecific: String
getBicycleSpecific(): String
Bicycle
bicycleSpecific: String
getBicycleSpecific(): String
PaymentMethod
accountId: String
accountHolderName: String
makePayment(amount: double): boolean
InAppWalletPayment
addMoneyToWallet(amount: double): boolean
makePayment(amount: double): boolean
enableAutoDeduction(): void
UPIPayment
upiId: String
makePayment(amount: double): boolean
CardPayment
cardNumber: String
expiryDate: String
cvv: String
makePayment(amount: double): boolean
InternetBanking
bankName: String
makePayment(amount: double): boolean
DeductFromSalary
employeeId: int
makePayment(amount: double): boolean
FromStudentFee
studentId: int
makePayment(amount: double): boolean
Designed By Assignment1_Part2_11

Figure 1: Class diagram illustrating all the components