Prashant Kumar (2022201058)

Assumptions:

- In the class diagram, I have defined all the data members as **private** but there are getters and setters to access the members, I didn't mention it.

Class diagram explanations:

1. **User:** Represents a generic user with essential attributes such as userId, userName, email, and password. It includes composition with a Document class representing user identification, and a userWallet for transactions composite with Wallet class.
   A user may or may not have a document, hence the "1" to "0..1" cardinality, and the same with the wallet class.
2. **Document:** Encapsulates information about user identification documents, including documentType, and documentData.
3. **Staff, Teacher, and Student:** These classes inherit common user class attributes and have additional attributes.
4. **UserAccountManager:** Manages user accounts, providing functionalities to create accounts, login, upload identification documents, and view trip history. Contains an association with the User class.
   A user account manager can manage one or more than one user's data, hence the cardinality will be 1 to many.
5. **Wallet:** Represents a user's wallet with attributes for walletBalance and paymentHistory. It provides a method to retrieve the current balance.
   a user may or may not have a wallet. So cardinality will be "1" to "0..1".
6. **AddMoneyIntoWallet:** Manages the process of adding money to a wallet, which uses Wallet and Payment method classes.
7. **PaymentMethod:** Represents a generic payment method with attributes like paymentId and paymentGateway. It serves as a base class for more specific payment methods.
8. **creditCardOrDebitCard, and upiApp:** These classes are payment methods to add money into wallets and pay vehicle fares.
9. **SmartVehicle**: Represents a generic smart vehicle with attributes such as vehicleNum, noOfRides, rating, isParked, and basePrice. It includes associations with QRCode, DockingStation.
10. **Bike, bicycle, moped:** Three types of vehicles are inherited from smartVehicle class and the bike and moped bike have some additional properties like distance traveled and battery/ fuel percentage.
11. **QRCode:** Represents a QR code with an attribute for the wayToConnect, likely used for connecting with or identifying SmartVehicles.
    A smart vehicle has one QR code associated with it. Cardinality will be "1" to "1".
12. **DockingStation:** Represents a docking station for smart vehicles with attributes like dockingStationId, dockeingstationLocation, capacity, availableVehicleList, dataLogs, and maintenanceStatus. SmartVehicles.
    One docker-station can have multiple vehicles so cardinality will be 1 to many with smart vehicles.

13. **ParkingLotManager:** Manages parking lots, including a parkingLotId and uses smart vehicle and docking station class. can Provide methods to update vehicle status, manage vehicle locations, and track vehicle availability.
14. **RatingAndFeedback**: Manages user feedback and ratings for a SmartVehicle, with attributes for feedbackByUser and ratingByUser.
15. **PaymentManager:** Manages payments for all types of trips, containing attributes like paymentId, and autoDeductedEnabled. It provides methods to make payments both by wallet and other payment methods. This uses three classes trip manager, wallet, and paymentmethods.
16. **TripManagement:** Represents a trip with attributes such as tripId, vehicleType (linked to SmartVehicle), user (linked to User), distanceCovered, timeTakenByTrip, and a connection to RatingAndFeedback. It also provides methods to scan a QR code, start a trip, and end a trip.
    Cardinality with rating and feedback class will be 1 to 1, with a smart vehicle will be many to many, and with the user will be many to 1.

Sequence diagram explanations:

1. **Create an account/ login, book a vehicle, and start the trip:**
    - A new user creates an account and uploads identification documents with the UserAccountManager. If the user already exists, then directly log in.
    - The user checks available smart vehicles at a docking station by querying with availableVehicleList(). The docking station responds with a list of available vehicles.
    - The user scans a QR code at the docking station to select a smart vehicle (scanQR()), then starts and ends the trip using the startTrip() and endTrip() methods.
    - After the trip ends, the user provides feedback and a rating through the provideFeedbackAndRating(String, int) method, and this feedback is associated with the specific smart vehicle used during the trip.
2. **After finishing a trip make the payment:**
    - The user ends a trip using endTrip() in the TripManagement class and fare calculation is performed.
    - Payment by wallet is initiated with makePaymentByWallet(), and then the PaymentManager checks the wallet's current balance.
    - In case of sufficient funds, the PaymentManager deducts the fare from the wallet.
    - If there are insufficient funds, the user is notified, and an attempt is made to add money to the wallet (addMoneyToWallet(double)) before retrying the payment.
    - Payment using a different method (makePaymentByPaymentMethod()) involves communication with the chosen PaymentMethod (e.g., card or UPI apps) to complete the payment.
    - The payment ID is returned to the PaymentManager after finalizing the payment process.