

**ICT340**

**Application Analysis and Design**

**JAN 2024**

**ECA**

|  |  |
| --- | --- |
| Name: | Yang Xian Wei Shawn |
| T-Group | T01 |
| Date Submitted | 6 April 2024 Saturday |

**Answer all questions. (Total 100 marks)**

**Question 1**

**Question 1(a)**

**Question 1(a)(i) (5 marks)**

**ANS:**

Primary Actors:

Drivers, Customers, Administrator, Premium Customers (Inherited From Customers)

Supporting Actors:

Bank System

**Question 1(a)(ii) (13 marks)**

**ANS:**

A diagram of a diagram

Description automatically generated

**Question 1(b) (4 marks)**

**ANS:**

The system design is inconsistent and ambiguous.

1. Inconsistent Cancellation Policy:

It is inconsistent because the requirements excerpt mentions that customers can always cancel without any penalty which contradicts the information in the appendix which stated that customers would be penalized with a $4 surcharge fee for cancelling after 5 minutes of car bookings.

2. Ambiguous Vehicle Selection:

It is ambiguous because the requirements excerpt mentions that customers can choose a destination and wait for a taxi to arrive which is ambiguous because the information in the appendix stated that PickMeNow offers customers 3 different types of vehicles for their ride which are cars, vans and excursion buses. Meanwhile, the requirements excerpt does not clarify if taxi referred to all vehicle types or 1 of the 3 vehicle types that customers can choose from.

**Question 2**

**Question 2(a) (15 marks)**

**ANS:**

|  |  |  |
| --- | --- | --- |
| **1. Class:** | User, superclass of Driver, Customer, Premium Customer and Administrator | **Relationship:** |
| **Attributes:** | userID | The unique identifier of the user |
|  | name | The name of the user |
|  | contact\_number | The contact number of the user |
|  | email\_address | The email address of the user |

|  |  |  |
| --- | --- | --- |
| **2. Class:** | Driver | **Relationship:** |
| **Attributes:** | accountNum | The bank account number of the driver |
|  | bank\_name | The bank name of the driver |

|  |  |  |
| --- | --- | --- |
| **3. Class:** | Customer | **Relationship:** |
| **Attributes:** | points | The number of points accumulated from the ride |
|  | isPremium | Whether customer is premium or not |

|  |  |  |
| --- | --- | --- |
| **4. Class:** | Administrator | **Relationship:** |
| **Attributes:** | Refer to super class User |  |

|  |  |  |
| --- | --- | --- |
| **5. Class:** | Vehicle, superclass of Car, Van and ExcursionBus | **Relationship:** |
| **Attributes:** | licensePlateNumber | The license plate number of the vehicle |
|  | brand | The brand of the vehicle |
|  | model | The model of the vehicle |

|  |  |  |
| --- | --- | --- |
| **6. Class:** | Car | **Relationship:** |
| **Attributes:** | Refer to super class Vehicle |  |

|  |  |  |
| --- | --- | --- |
| **7. Class:** | Van | **Relationship:** |
| **Attributes:** | depositAmt | The deposit amount for the vehicle |
|  | bookingFee | The booking fee for the vehicle |

|  |  |  |
| --- | --- | --- |
| **8. Class:** | ExcursionBus | **Relationship:** |
| **Attributes:** | depositAmt | The deposit amount for the vehicle |

|  |  |  |
| --- | --- | --- |
| **9. Class:** | Ride | **Relationship:** |
| **Attributes:** | referenceNum | The reference number of the ride |
|  | fare | The fare of the ride |
|  | pickUpPoint | The pick up point of the ride |
|  | destination | The destination of the ride |
|  | distance | The distance of the ride |
|  | date | The date of the ride |
|  | startTime | The start time of the ride |
|  | endTime | The end time of the ride |

|  |  |  |
| --- | --- | --- |
| **10. Class:** | Feedback | **Relationship:** |
| **Attributes:** | rating | The rating of the ride |
|  | description | The feedback description of the ride |
|  | followUpAction | The follow up action of the ride |

**Question 2(b) (15 marks)**

**ANS:**

A diagram of a company

Description automatically generated

**Question 3**

**Question 3(a) (8 marks)**

**ANS:**

A screenshot of a computer

Description automatically generated

**Question 3(b)**

**Question 3(b)(i) (4 marks)**

**ANS:**

class OrchestratingClass:

def \_\_init\_\_(self):

self.rideDict = {}

self.driverDict = {}

def findRide(self, rideReference):

return self.rideDict.get(rideReference)

def findDriver(self, driverID):

return self.driverDict.get(driverID)

def acceptRide(self, driverID, rideReference):

aRide = self.findRide(rideReference)

aDriver = self.findDriver(driverID)

return aRide.setDriver(aDriver)

**Question 3(b)(ii) (4 marks)**

**ANS:**

class Ride:

def \_\_init\_\_(self, rideReference):

self.\_rideReference = rideReference

self.\_rideStatus = "Open"

self.\_driver = None

@property

def updateRideStatus(self):

return self.\_rideStatus

@updateRideStatus.setter

def updateRideStatus(self, newrideStatus):

return newrideStatus

def setDriver(self, driver):

self.\_driver = driver

self.\_rideStatus = "Driver Assigned"

return driver.acceptRide()

**Question 4 (14 marks)**

**ANS:**

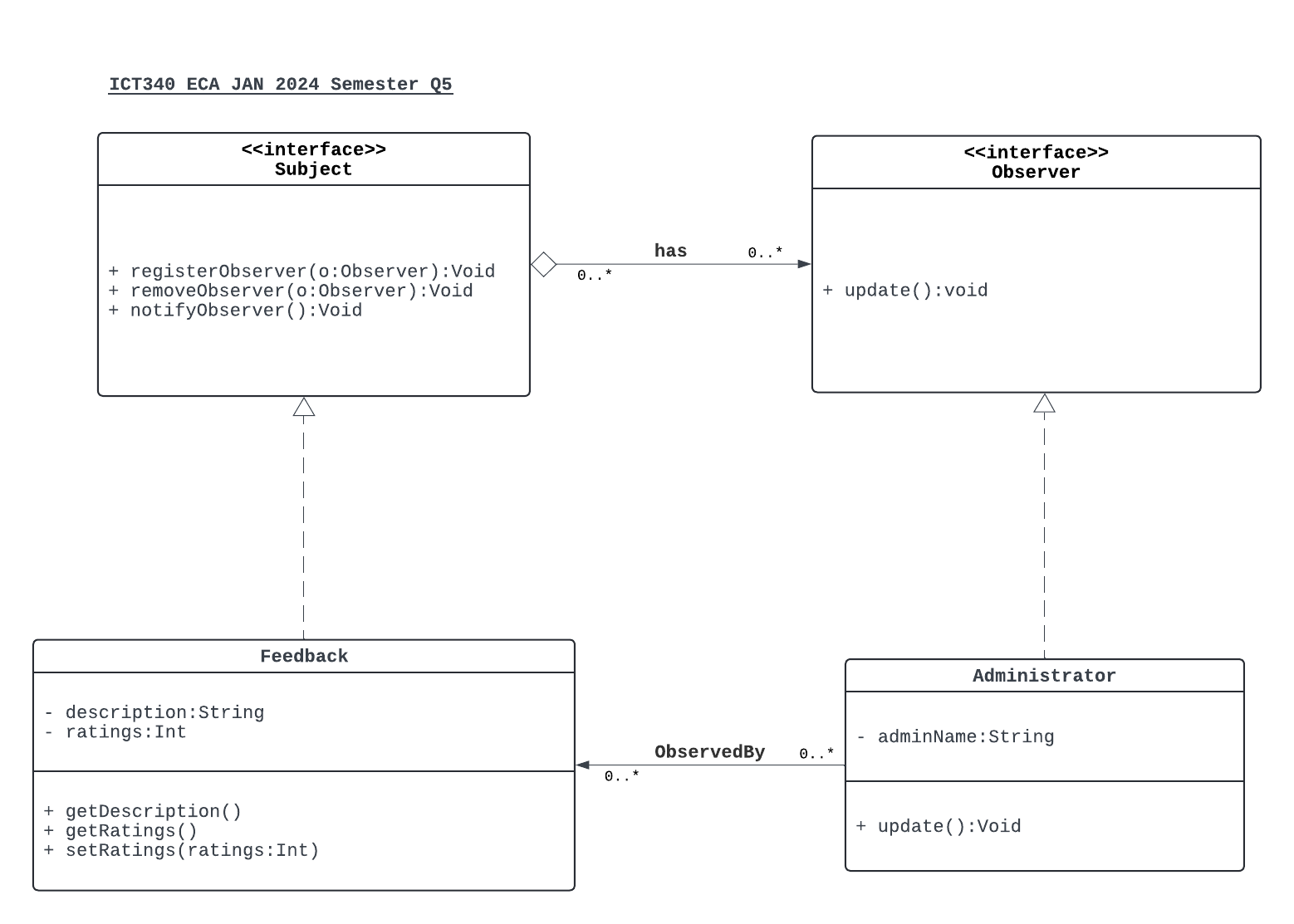
A diagram of a customer service

Description automatically generated

**Question 5**

**Question 5(a) (8 marks)**

**ANS:**



**Question 5(b) (10 marks)**

**ANS:**

class Observer():

def update(self, subject):

pass

class Subject:

def \_\_init\_\_(self):

self.\_observers = []

def registerObserver(self, observer):

if observer not in self.\_observers:

self.\_observers.append(observer)

def removeObserver(self, observer):

try:

self.\_observers.remove(observer)

except ValueError:

pass

def notifyObservers(self, modifier = None):

for observer in self.\_observers:

if modifier != observer:

observer.update(self)

class Feedback(Subject):

def \_\_init\_\_(self, description):

Subject.\_\_init\_\_(self)

self.\_description = description

self.\_ratings = 0

@property

def description(self):

return self.\_description

@property

def ratings(self):

return self.\_ratings

@ratings.setter

def ratings(self, ratings):

self.\_ratings = ratings

self.notifyObservers()

class Administrator(Observer):

def \_\_init\_\_(self, name):

self.\_adminName = name

def update(self, subject):

print(f"Feedback: {subject.description}, Rating: {subject.ratings}. \n The Administrator {self.\_adminName} has been notified of the feedback. \n")

#### Additional Test Results:

def main():

# Create an administrator

admin\_1 = Administrator("Anson Goh")

admin\_2 = Administrator("Jason Wang")

# Create some feedback

feedback1 = Feedback("Great ride!")

feedback2 = Feedback("Car was dirty")

# Register the administrator to receive notifications from both feedbacks

feedback1.registerObserver(admin\_1)

feedback2.registerObserver(admin\_2)

# Set the description and ratings for the feedbacks (This will trigger notification)

# feedback1.description = "Excellent service by the driver"

feedback1.ratings = 5

feedback2.ratings = 3

if \_\_name\_\_ == "\_\_main\_\_":

main()

**----- END OF ECA PAPER -----**