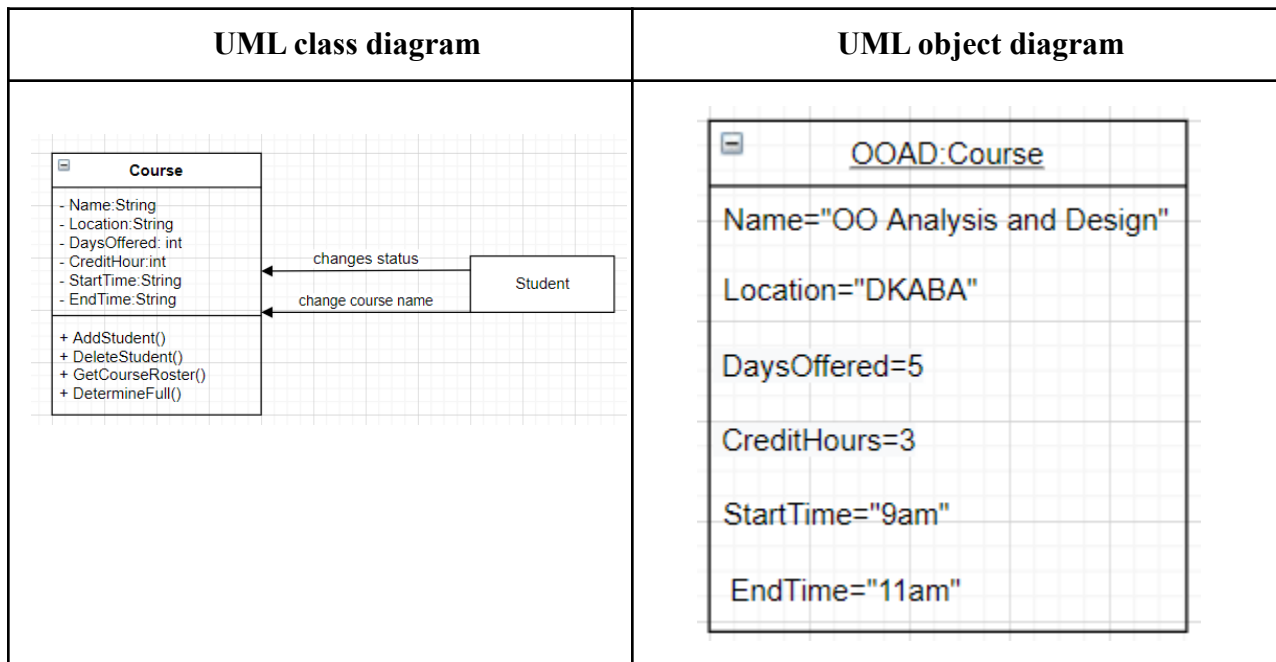


Tutorial 1: Object Orientation

- Describe the relationship between classes and objects.
 - A class is an abstract definition of an object. To illustrate, it defines the structure, behavior and characteristic of each object in the class. Besides, it serves as a template for creating objects.
 - Similar objects are grouped into the same classes. For example in the student class, we can construct a “Heck Ken” and “Man Yee” object.
- Assuming that the Course class is part of a Course Registration System:
 - Construct an **UML class diagram** for the Course class and include some messages that a Student class may wish to send to the object of the Course class.
 - Construct an **UML object diagram** for the Course object.



- Define the term **polymorphism**. Explain how a message passes to different objects by applying the polymorphism concept. You may explain with an example. (pg54)
 - Allow one message to be passed by calling function and passing in parameter (method overriding)
 - Sending object is the caller which need not know what kind of object will receive the message
 - For example: *Animal sounds* - cat will meow, dog will bark, horse will neigh

4.

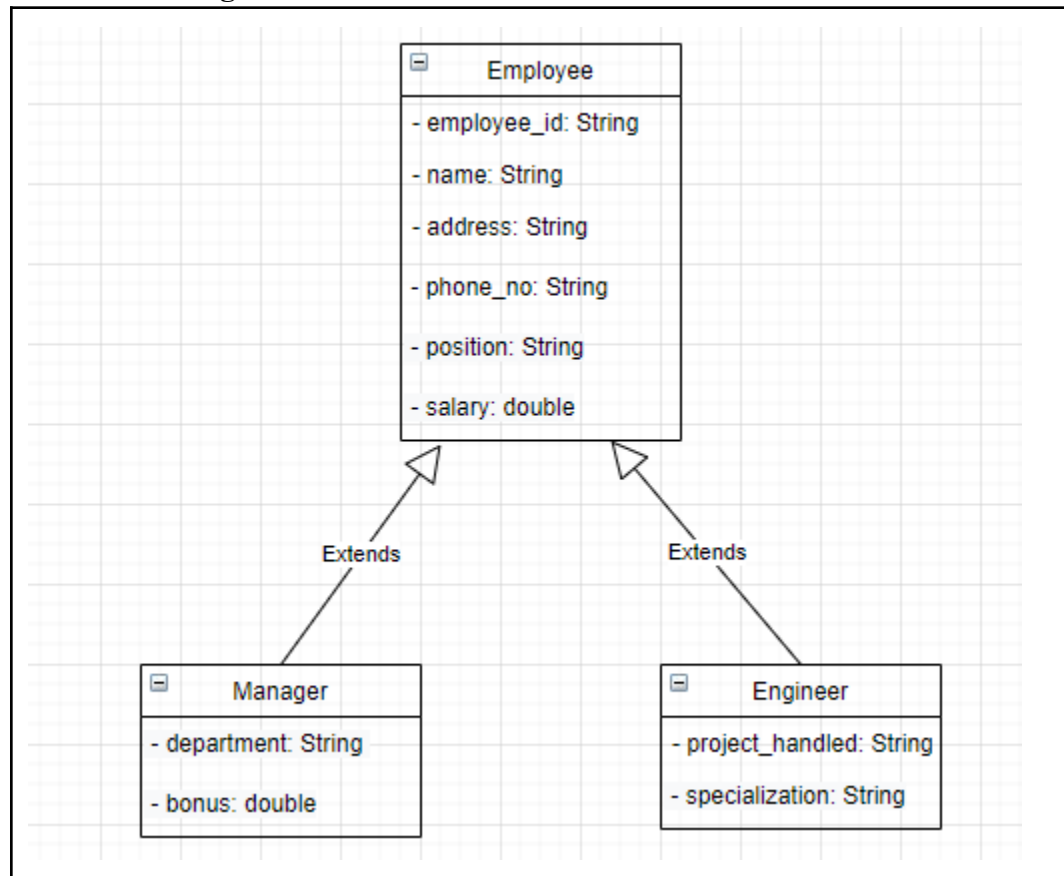
Identify and describe the relationship for the classes below with the correct association.

Employee, Manager and Engineer

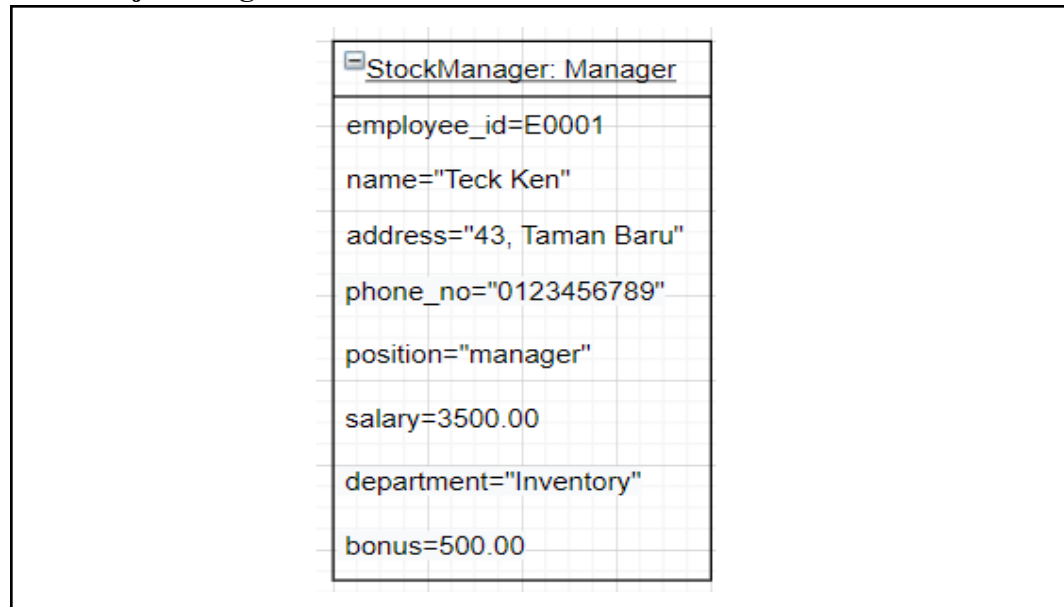
- Construct UML class diagram for the classes above with the appropriate class name and properties.
- Based on the above solutions, construct an UML object diagram from one of the classes above.

- These classes are in a “is-a” relationship which is inheritance. The Manager and Engineer class inherit characteristics from the Employee class and they can have their own additional attributes.

UML Class Diagram



- UML Object Diagram

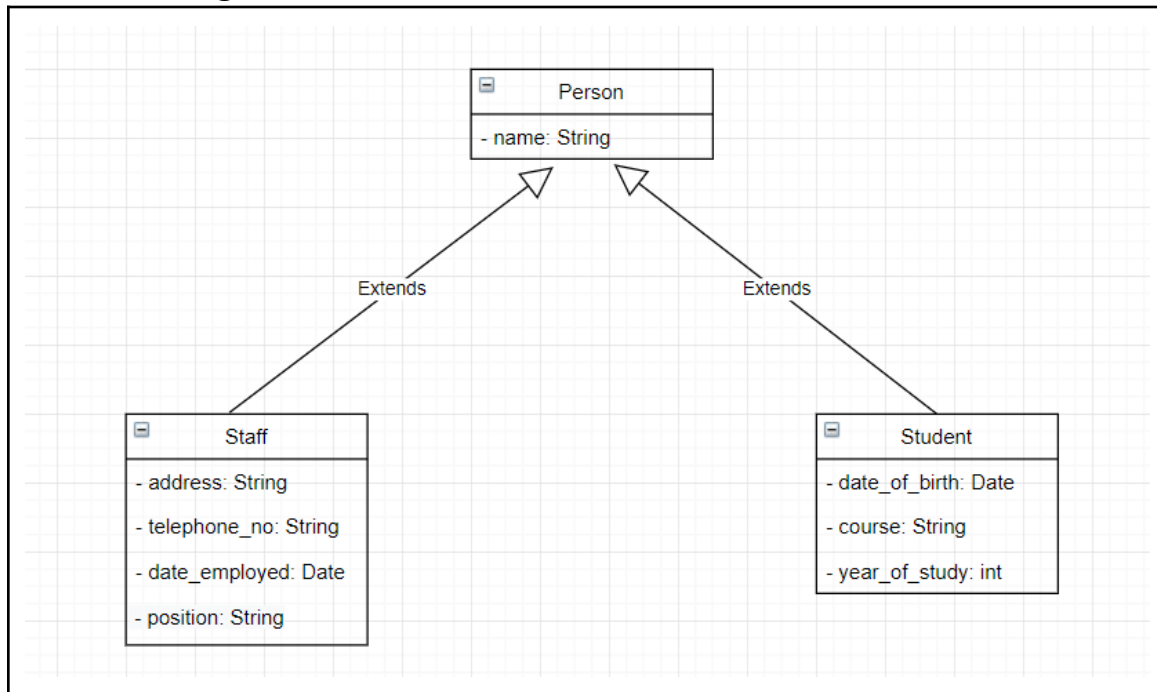


5.

Suppose you work for a university that needs to manage basic information about staff and students. The information about each staff is such as name, address, telephone number, date employed and position. The information about each student is name, date of birth, course and year of study.

- Draw an UML class diagram with generalization/ inheritance relationship between the classes based on the scenario above. (Note: Do not include the operations)
- Discuss the benefits of using *inheritance* in object-oriented.

UML Class Diagram;



- Inheritance allows us to use classes as standard templates from which other classes can be built. It leads to:
 - ❖ Less redundancy
 - ❖ Faster creation of new classes
 - ❖ Standards and consistency within and across development efforts
 - ❖ Ease in supporting exceptions

Telephone number data type:

- String as some phone number might need country code
- Not suitable as phone number start with zero
- We do not need calculation on telephone number

Tutorial 2: Modeling Concepts

Question 1

Unified Software Development Process (USDP) is a generic software development process for object oriented analysis and design. Describe the **best practices of the USDP** approach in software development.

1. Develop iteratively and incrementally

- The project should be broken into smaller modules in which develop functionality incrementally until an adequate system has been fully developed.

2. Use component-based development

- Build a system by focusing on integrating components that have been built before rather than developing a system from scratch.

3. Requirements-driven development

- Extensively involve users in validating user requirements to give them a concrete impression about the developed system.

4. Configurability

- Since the system has many release version, thus project manager need a version control mechanism to control each version of the system

5. Architecture-centrism

- Structure the system into packages, components and deployment.

6. Model visually

- Use UML (Unified modeling language) to express a system in Structured Oriented Analysis and Design (SAD).

Question2

Identify the major criticisms of the information system projects that are developed using the traditional waterfall approach.

Major criticisms for waterfall approach:

1. Real projects rarely follow a linear flow
2. Iterations is almost impossible
3. Unresponsive to changes during project
4. User or business needs / requirements not addresses
5. Modules not integrating
6. Difficulties with maintenance
7. Late discovery of flaws
8. Poor quality of end-user experience

Sample answer:

This often results in the discovery, during testing activities when the different pieces of the system are integrated, of quality related problems that have remained hidden during the design and implementation activities. Because such problems are discovered late in the development process, it may be too late to resolve them or they may be too costly to resolve.

Question 3

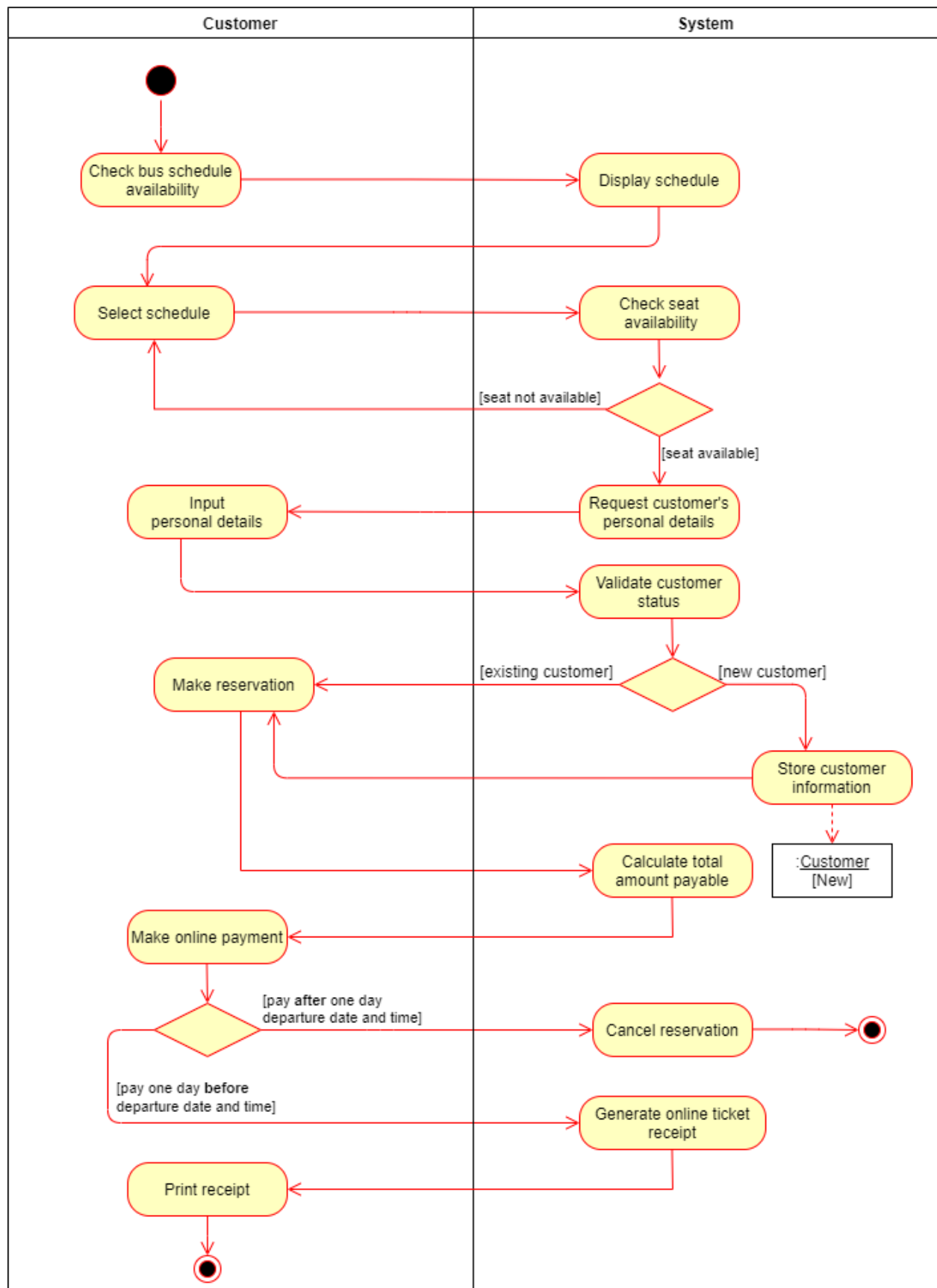
Construct an *activity diagram* for the **bus tickets reservation processes** as described below:

The customer is allowed to place bus tickets reservation based on the bus schedule availability via online. If the bus schedule is available, the system will check the seat availability. If the seat is available, the system will request customer's personal details. The system will check the customer's status. If he or she is a new customer, the system will register the customer details into the system; else the customer can proceed to make the reservation.

Once the reservation transaction is done, the system will display the total amount payable and the customer should make online payment one day before the departure date and time; else the system will automatically cancel the reservation. Once the payment transaction is done, the system should be able to generate the online ticket receipt and the customer is required to print out the receipt as a proof.

You are required to include the relevant object flows and swim lanes in your activity diagram.

Area	Marks
Correct swim lanes and object creation	3
Correct notation	2
Logic and correct object flows	6



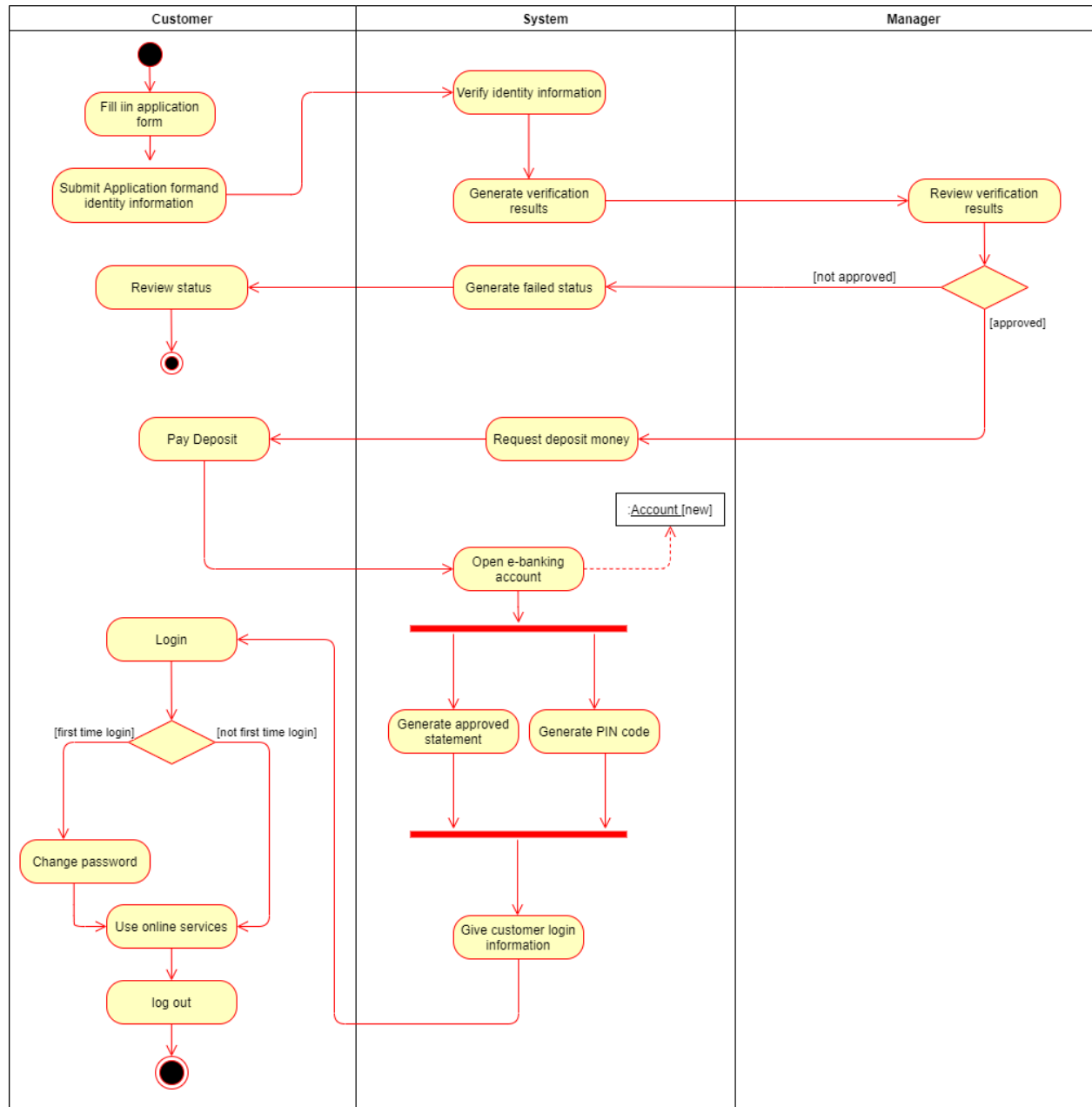
Question 4

Commerce Bank provides an e-banking services such as e-billing, e-statement, online share trading, fund transfer, etc. The customers can apply for online services by filling in an application form to open a new account at customer service counter. Then, the application form and the photocopy of the customer's identity card with finger print verification will be submitted to the manager for approval purpose. The application will be reviewed and processed within 20 minutes. If the application is approved, the customer needs to deposit RM300 as deposit for opening an e-banking account. After that, an approved statement will be given to the customer including a 6 digit PIN code for online service access.

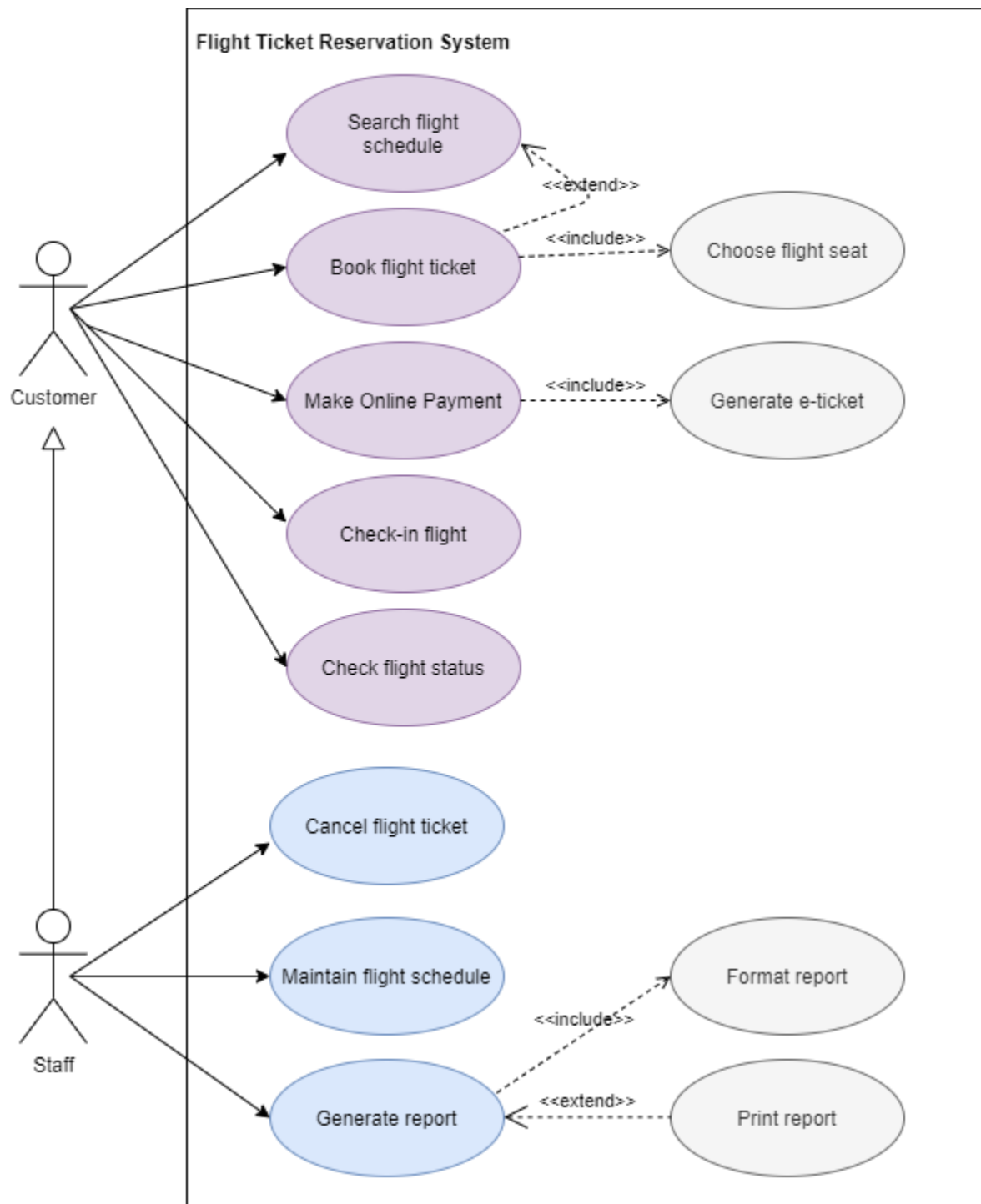
On the same day, the customer is required to login to the system with the PIN code given. For first time login, a customer needs to change his / her own password. After that, the customer can use the online services. At the end, he/she must log out the system and return to the main menu.

Analyze the above scenario and construct an activity diagram for the online customer e-banking account opening process as described above. You are required to include the relevant object flows and swim lanes in your activity diagram.

Area	Marks
Correct swim lanes and object creation	3
Correct notation	2
Logic and correct object flows	6



Tutorial 3



Question 2

Delicious Pizza Sdn. Bhd is a fast-growing pizza delivery company in Malaysia. The pizza chain has growth to more than 200 restaurants ll over the country. The company has implemented an Online Pizza Ordering System and allowing customers to order pizza from a wide variety of selections. The company also provide free delivery service when the order has reached the minimum order amount. The Online Pizza Ordering System consists of the following system functionalities:

The system should allow customer to:

- Register user account
- Search store locations
- Place order with add on beverages option
- Make payment
- Generate receipt
- Track order status

Question 1

Asia Pacific Berhad is an airline company which operates scheduled domestic and international flights to more than 150 destinations spanning over 25 countries. The company has implemented a Flight Ticket Reservation System and allowing the customers to book flight tickets and manage their booking online. The targeted users of this system are customers and staff. The Flight Ticket Reservation System consists of the following system functionalities:

The system should allow customers to:

- Search flight schedule
- Book flight ticket with seat selection option
- Make online payment
- Generate e-ticket
- Check-in flight
- Check flight status

The system show allow staff to:

- Cancel flight ticket
- Maintain flight schedule
- Generate report
- Access all functions that are accessible by customers.

Construct a *use case diagram* that depicts the functional requirements for the Flight Ticket Reservation System.

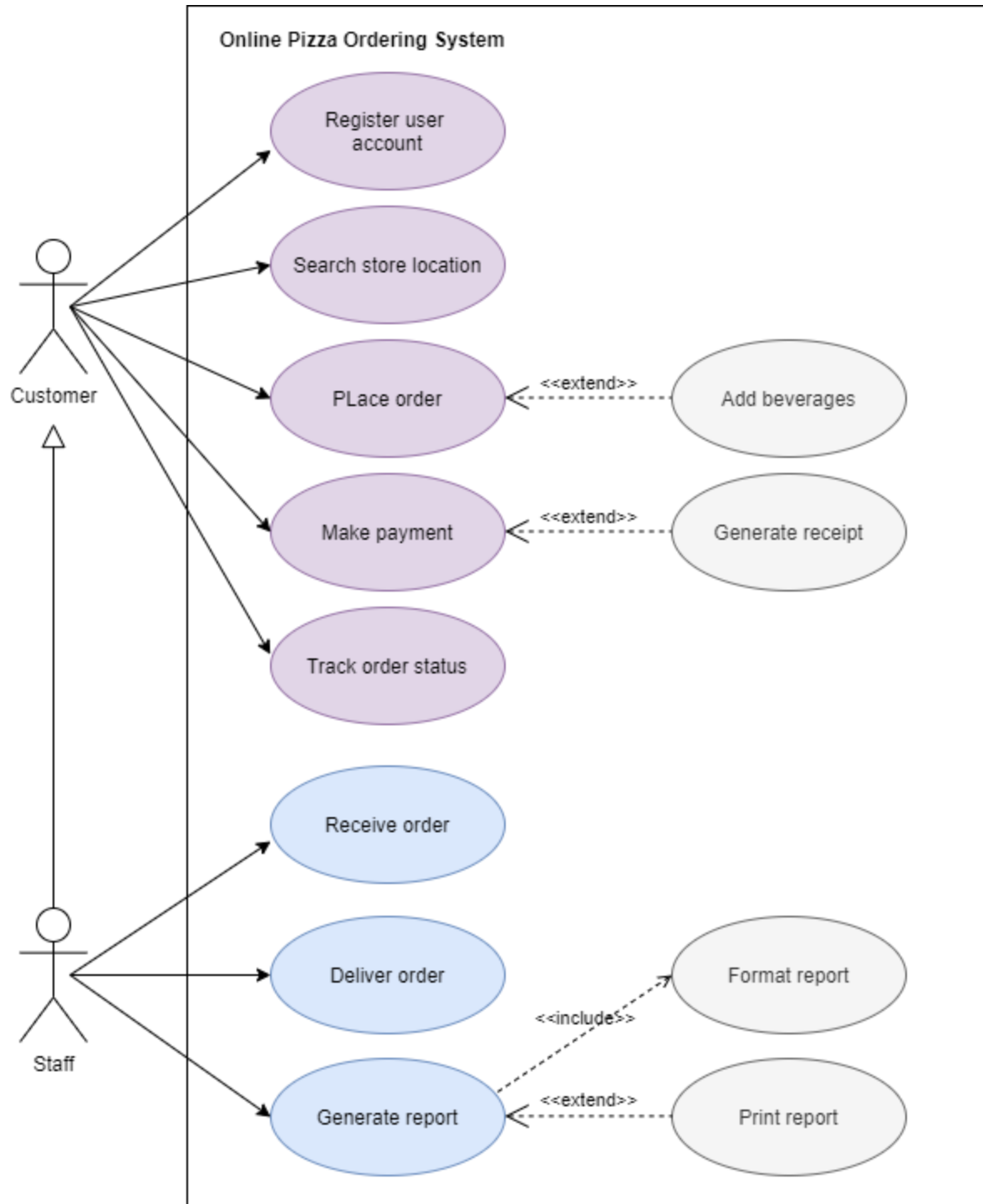
Area	Marks Allotted
Correct actors	2
Define system boundary and Name	1
Correct use of include and extend relationship	2
Correct use cases	8
Correct generalization relationship	1
Correct notation	1

The system should allow staff to:

- Receive order
- Deliver order
- Generate report
- Access all functions that are accessible by customers

- a) Construct a *use case diagram* that depicts the functional requirements for the Online Pizza Ordering System.

Area	Marks Allotted
Correct primary and supporting actors	2
Define system boundary	1
Correct use of include and extend relationship	2
Correct use cases	8
Correct generalization relationship	1
Correct notation	1



- b) Write a *use case description* to document the basic scenario for the main flow and alternative flow of events for “*Register user account*” use case.

Use Case Name: Register user account	
Actor: Customer	
Brief Description: This use case allows customers without an account to create a new user account.	
Precondition: The user does not have an account	
Main Flows of Events	
Actor Action	System Responses
	1. Display main menu
2. Select “register account”	3. Display registration form
4. Enter name, ic, email, phone number, address, gender	5. Validate the details entered by user
	6. If the value entered by user is correct, the system will prompt for confirmation
7. Select “confirm”	8. If the user select “confirm”, the system will create and store the information of the user into the database
	9. The system will display a successful message - “Successfully registered”
<u>Alternative Flow:</u> <u>A1. Step 6</u> If the value entered by the user is incorrect, the system will prompt an error message to the user and request the user to enter again. <u>A2. Step 8</u> If the user selects “cancel”, the system will redirect the user back to main menu	
Postcondition: An account has been successfully created.	

- c) Recommend **TWO** suitable non-functional requirements for the Online Pizza Ordering System and provide reasons to justify why these non-functional requirements should be included.

- **Efficiency** - The online pizza ordering system should be efficient which is able to handle simultaneous user orders within the peak load period without facing downtime.
- **Secure information** - The online pizza ordering system should provide secure access by encapsulating the customer credential data to prevent credential theft from happening. All the sensitive information, especially password has to be encrypted before it is stored in the user record/table/database. For example, Oracle has MD5, SHA-1, SHA-256.
- **Performance** - System is able to take 1000 orders at the same time and maintain the system smooth operation without bugs.
- **Availability** - System can quickly recover and make available 24 by 7 without downtime. Hence, a backup server is needed in case of system failure.

Question 3

A local college has started a project of college Bus Tracking System. Registered college students/staffs are able to view the bus daily schedule with details such as destination, departure time, expected arrival time, and bus registration number (plat number). Users can also set notification when a particular bus is arriving. A bus is assigned to one specific route and a driver (part time or full time) is in-charge of one bus only to ensure service effectiveness. The following questions are based on the college bus tracking system:

- a) List the actors and identify the suitable functional requirements for college Bus Tracking System.
- b) Based on the solutions given in a), Construct a **use case diagram** that depicts the functional requirements for college Bus Tracking System.

Area	Marks
Correct actors	1
Define system boundary and name	1
Correct use of include and extend relationship	2
Correct main use cases	6

a.

Actor : users (staffs/students), bus

Functional requirements

The system allow users:

- View bus daily schedule
- View the bus position in a map view
- Set bus arrival notifications

The system allows staffs:

- Maintain bus schedule
- Assign specific route and driver to one bus

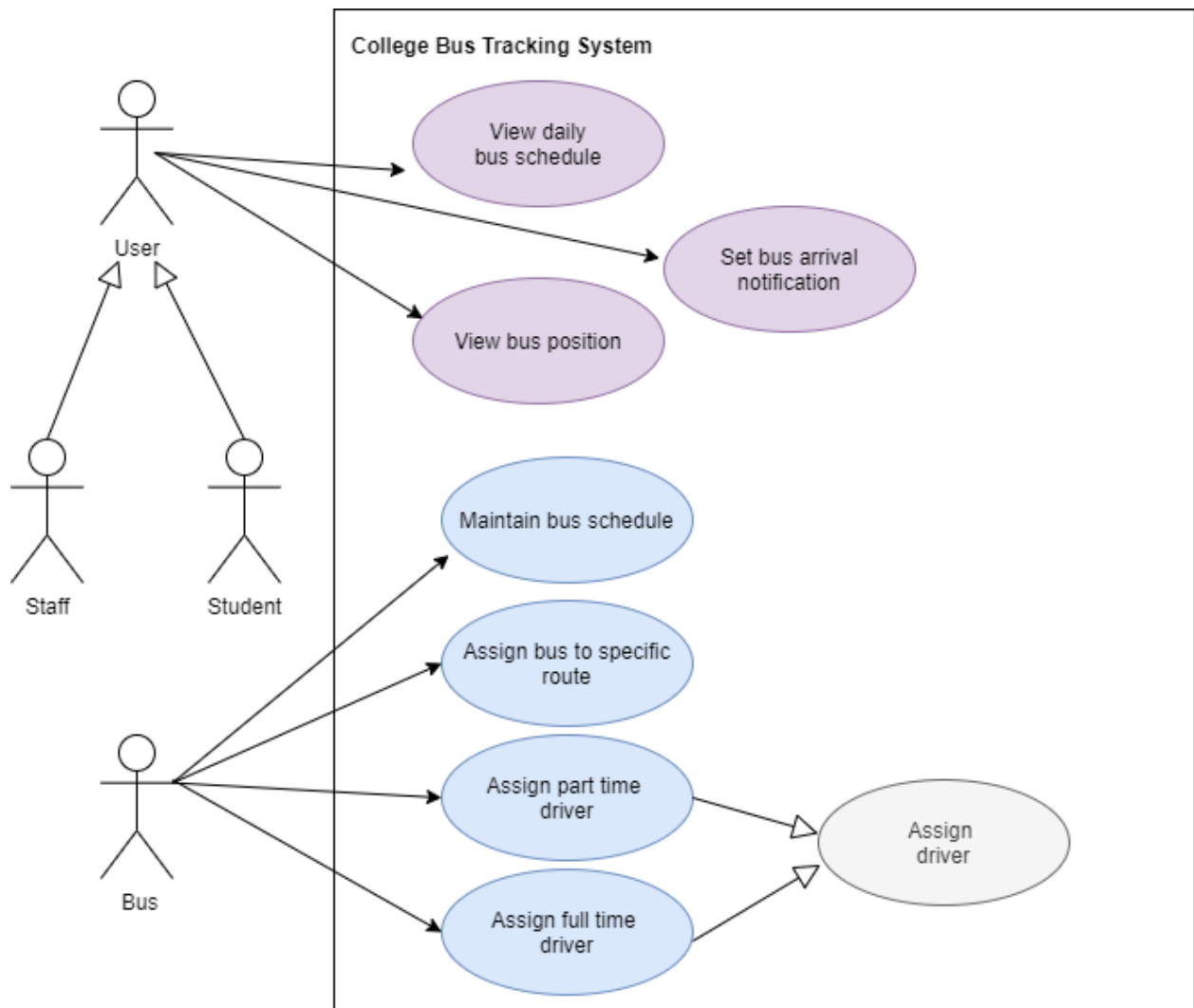
Sample Answer:

Actors - Students and Staff, Driver

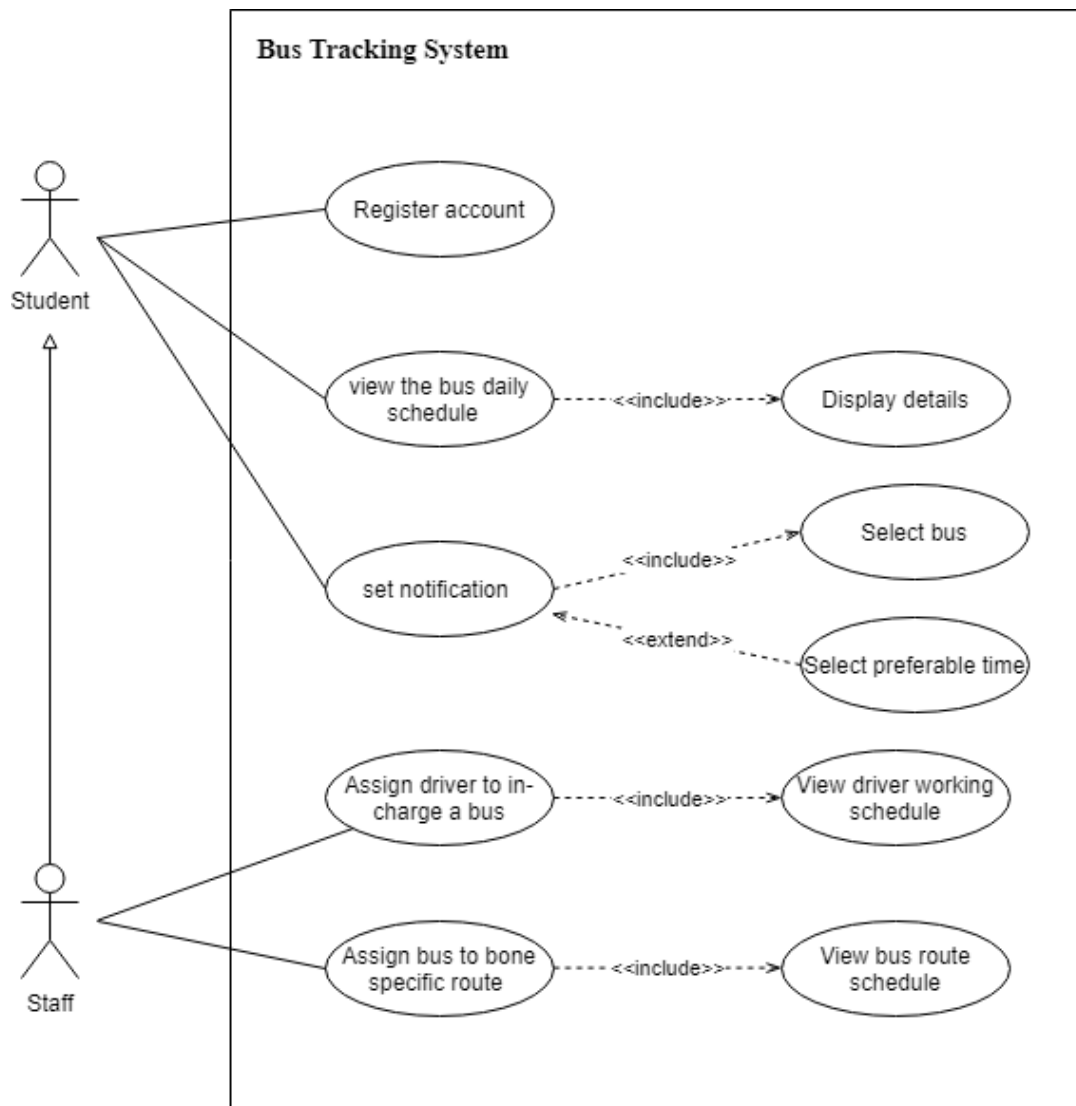
Functional Requirements:-

- The system allows registered staff and students to view daily bus schedules.
- The system allows users to set notifications.
- Assign a driver to handle a specific route.
- The system allows staff to edit bus schedules.
- The system allows students to print bus schedules.

b.

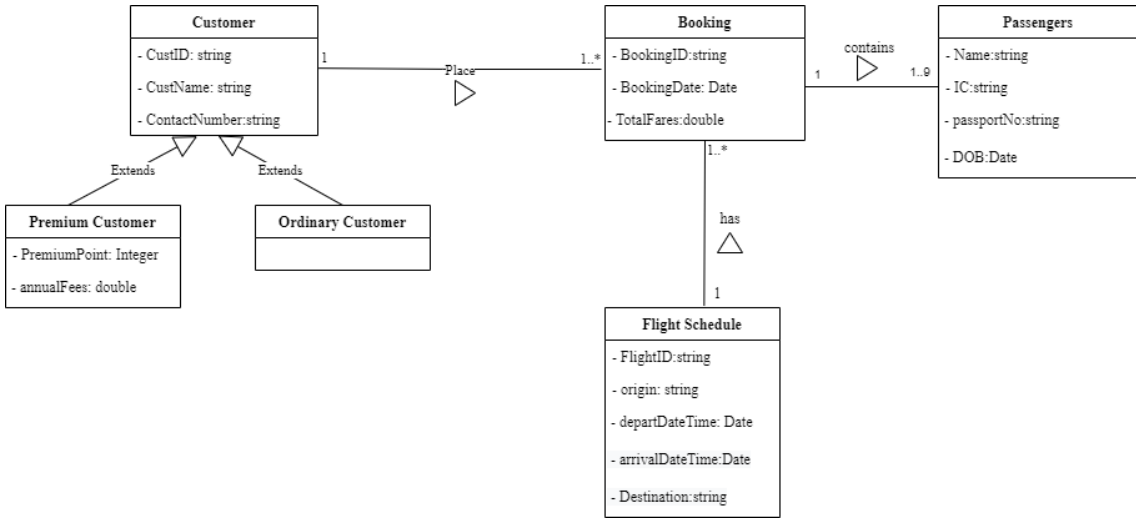
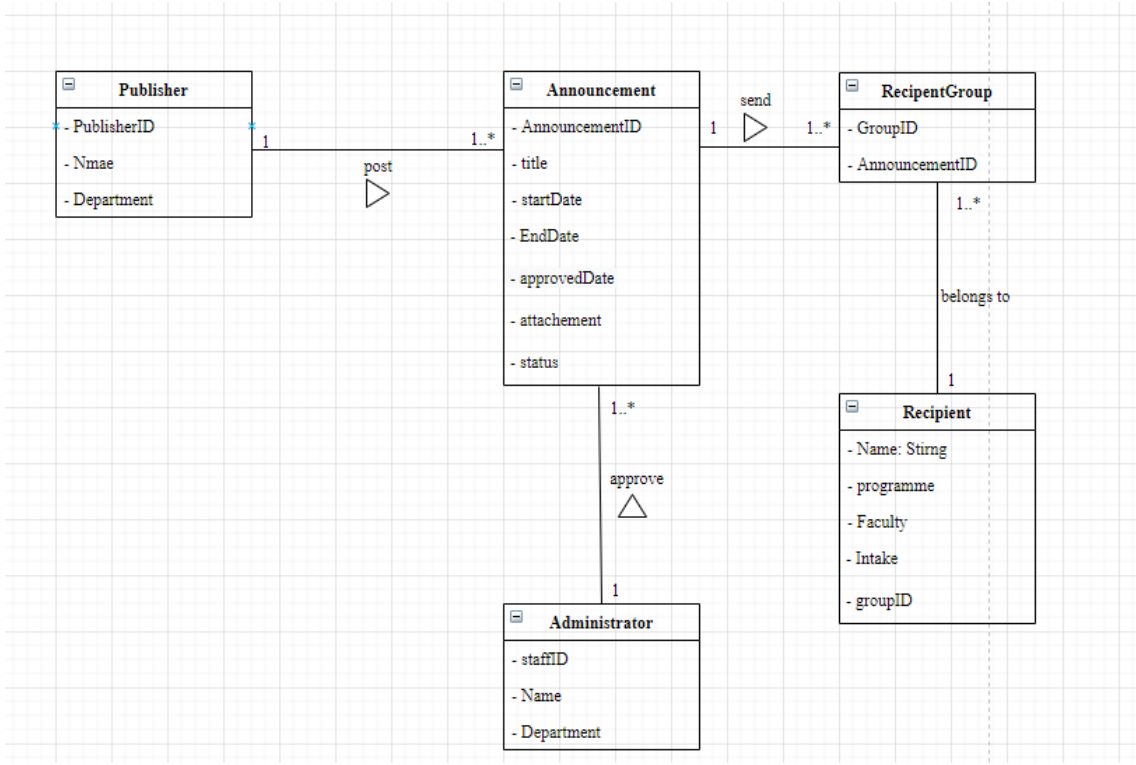


Sample Answer:



Tutorial 4

Draw.io : <https://app.diagrams.net/#G1NE0QDyM2Z469y-h-GhkUU0Mns5Sy-wVZ>

	Answer
T4Q1	 <pre> classDiagram class Customer { -CustID: string -CustName: string -ContactNumber: string } class PremiumCustomer { -PremiumPoint: Integer -annualFees: double } class OrdinaryCustomer { } class Booking { -BookingID: string -BookingDate: Date -TotalFares: double } class FlightSchedule { -FlightID: string -origin: string -departDateTime: Date -arrivalDateTime: Date -Destination: string } class Passengers { -Name: string -IC: string -passportNo: string -DOB: Date } Customer "1" -- "1..*" Booking : Place Customer < -- PremiumCustomer Customer < -- OrdinaryCustomer Booking "1..*" -- "1" FlightSchedule : has Booking "1" -- "1..9" Passengers : contains </pre> <p>The diagram shows a Customer class with attributes CustID, CustName, and ContactNumber. It has two subclasses: Premium Customer (with PremiumPoint and annualFees) and Ordinary Customer. A Booking class has attributes BookingID, BookingDate, and TotalFares. It has a 1-to-many relationship with Flight Schedule (has) and a 1-to-many relationship with Passengers (contains). Premium Customer and Ordinary Customer both extend the Customer class.</p>
T4Q2	 <pre> classDiagram class Publisher { -PublisherID -Nmae -Department } class Announcement { -AnnouncementID -title -startDate -EndDate -approvedDate -attachement -status } class RecipientGroup { -GroupID -AnnouncementID } class Recipient { -Name: Stiring -programme -Faculty -Intake -groupID } class Administrator { -staffID -Name -Department } Publisher "1" -- "1..*" Announcement : post Announcement "1" -- "1..*" RecipientGroup : send RecipientGroup "1..*" -- "1" Recipient : belongs to Administrator "1" -- "1..*" Announcement : approve </pre> <p>The diagram shows a Publisher class with attributes PublisherID, Nmae, and Department. It has a 1-to-many relationship with Announcement (post). Announcement has attributes AnnouncementID, title, startDate, EndDate, approvedDate, attachement, and status. It has a 1-to-many relationship with RecipientGroup (send) and a 1-to-many relationship with Administrator (approve). RecipientGroup has attributes GroupID and AnnouncementID. It has a 1-to-many relationship with Recipient (belongs to). Recipient has attributes Name, programme, Faculty, Intake, and groupID. Administrator has attributes staffID, Name, and Department.</p>
T4Q3(a)	<p>Use case realization - identify domain classes and represent how a use case will be implemented in terms of collaborating objects.</p> <p>Domain - Knowledge scope Example: Customer Service chatbot, Domain can be Covid-19, Educational</p>

Institution (Open Day, find out a suitable programme to enroll.) **Product knowledge.**

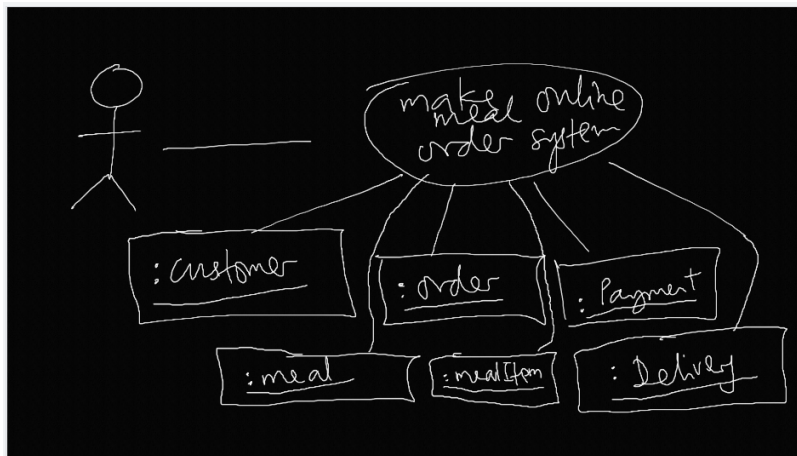
Realize - self-awareness

Example: I realize that I am weak in English sentence structure. A good thing. Now, you will start to plan.

Read newspaper - construct sentences

Watch english comedy - Friends / Malcolm in the middle - Read the subtitles.

Make online order systems - Domain classes would be Customer, Order, Payment, Delivery



T4Q3(b) draw.io

T4Q4

CRC card is a brainstorming tool that is used in object-oriented design which models interaction between objects. Analysts should create them because they can explore multiple alternative interactions quicker and can avoid a lot of drawing and erasing.

Example:

Class Name: Client	
Responsibilities	Collaborations
Provide client information Provide list of campaigns	Campaign provides campaign details.

Class Name: Campaign	
Responsibilities	Collaborations
Provide campaign information Provide list of adverts Add a new advert	Advert provides advert details. Advert constructs new object.

Tutorial 5

Draw.io URL: [Flowchart Maker & Online Diagram Software](#)

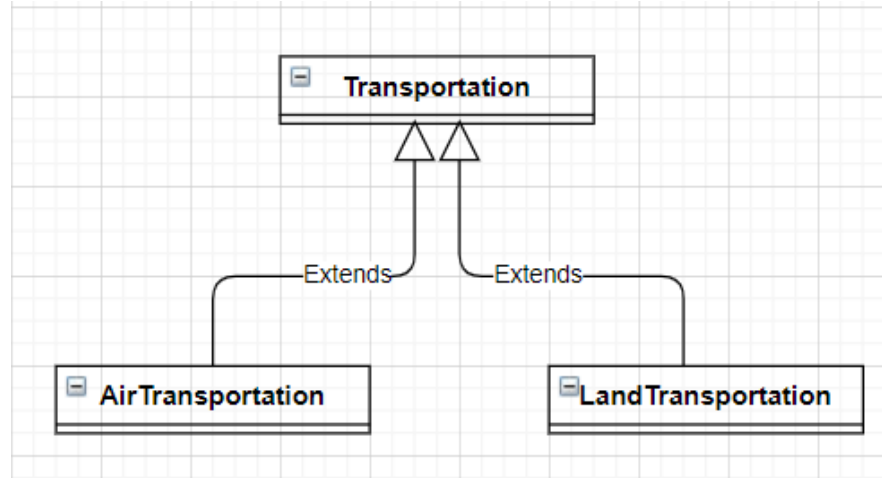
Q	Answer
T5Q1a	<p>Aggregation implies a relationship where the child can exist independently of the parent. For example, courses and students. If the course enrolled by the student is removed, the student will still exist and they can join for another course.</p> <p>Composition is a relationship where the child cannot exist independent of the parent. For example, Order and Order details. If the Order has been canceled, the order details will not be able to be seen inside the system anymore. Since the order details cannot exist while the order has been canceled.</p> <p>Class diagram has classes with attributes and operations. Classes are related to each other via association. Association can be aggregation / composition or normal relationship line with relationship name and also multiplicity.</p>
T5Q1b	<p>Abstract</p> <ul style="list-style-type: none">- An abstract class is a class that cannot instantiate an object. <p>Concrete</p> <ul style="list-style-type: none">- A concrete class is a class that can instantiate an object. <p>Example 1: An abstract class called GeometricObject with an abstract method getArea() that has no method body / implementation. A concrete class called Circle Another concrete class called Rectangle</p> <p>Elaborate: If the GeometricObject is not an abstract class, in this case, we can be able to create an object of the GeometricObject class and call the getArea(). This will be hit with a problem, because you will proceed to ask the next question. May I know what GeometricObject you are referring to?</p> <p>Example 2: An abstract class called Person with an abstract method called calculateSalary() without implementation. A concrete class called Part Time Employee, which has the method body of calculateSalary(). Another concrete class called Full Time Employee, which has the method body of calculateSalary().</p> <p>Is it a must that an Abstract class must have at least one abstract method to make the class as Abstract class? FALSE</p> <p>Which other classes/components only have CONSTANTS and / or behaviors? INTERFACE</p>

Do you think Abstract class can have data members? Yes, I can.

Q2

Transportation, Air transportation and Land transportation

Generalization: Air transportation and Land transportation “is a ” transportation

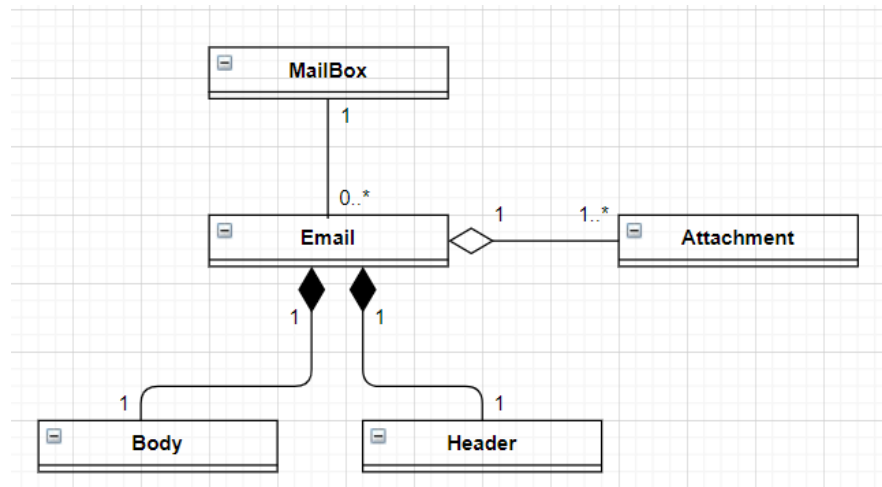


Hotel campus, Hotel buildings and Hotel rooms

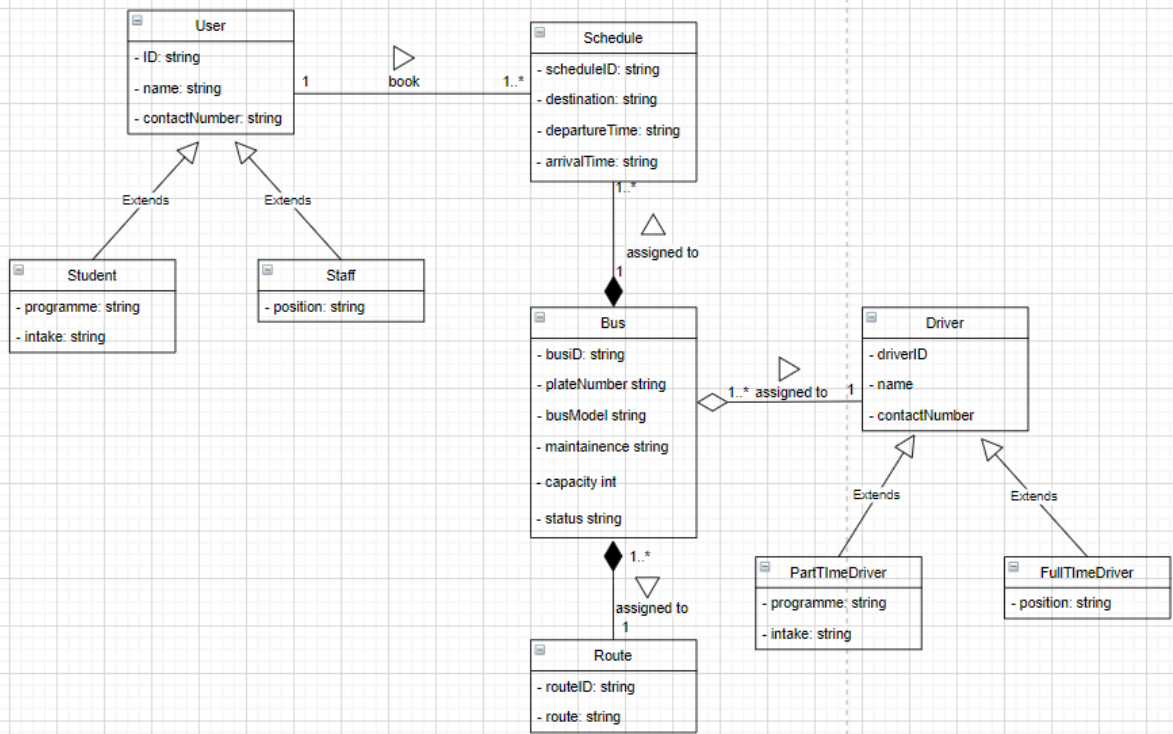
Composition



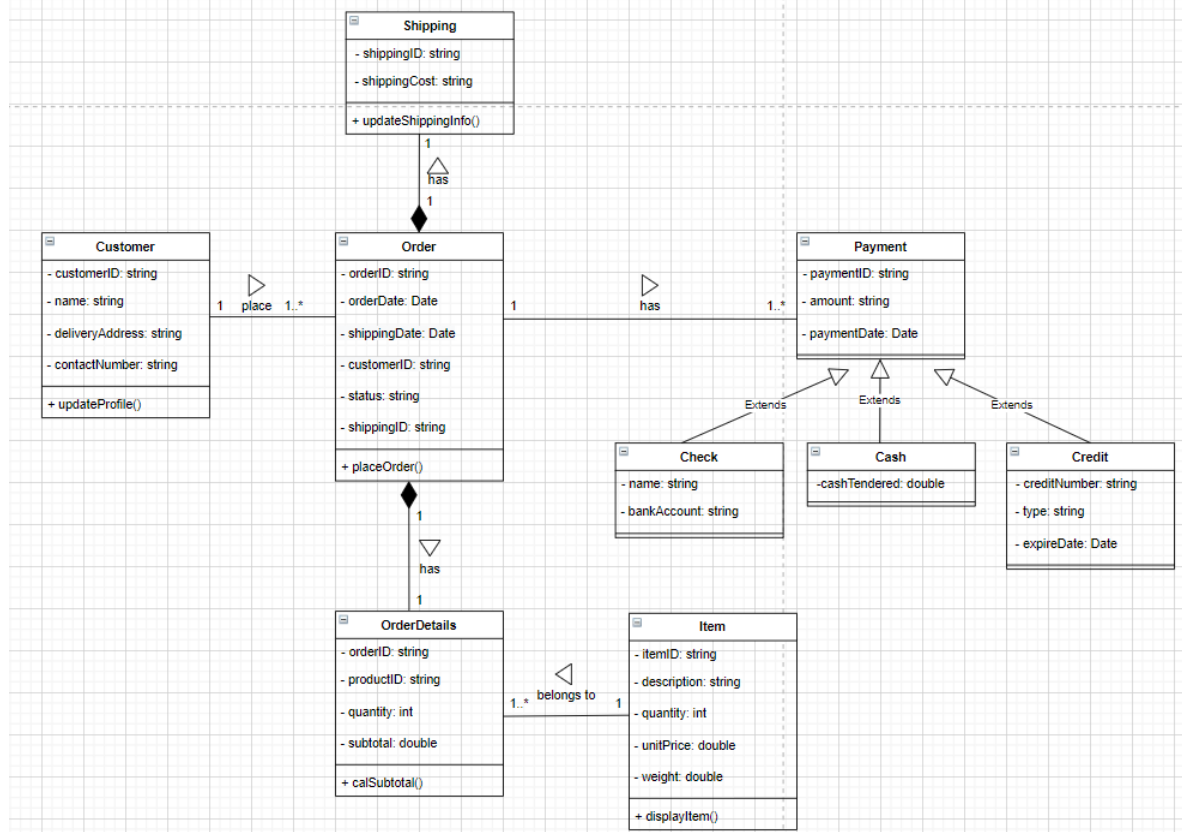
Q3



Q4



Q5

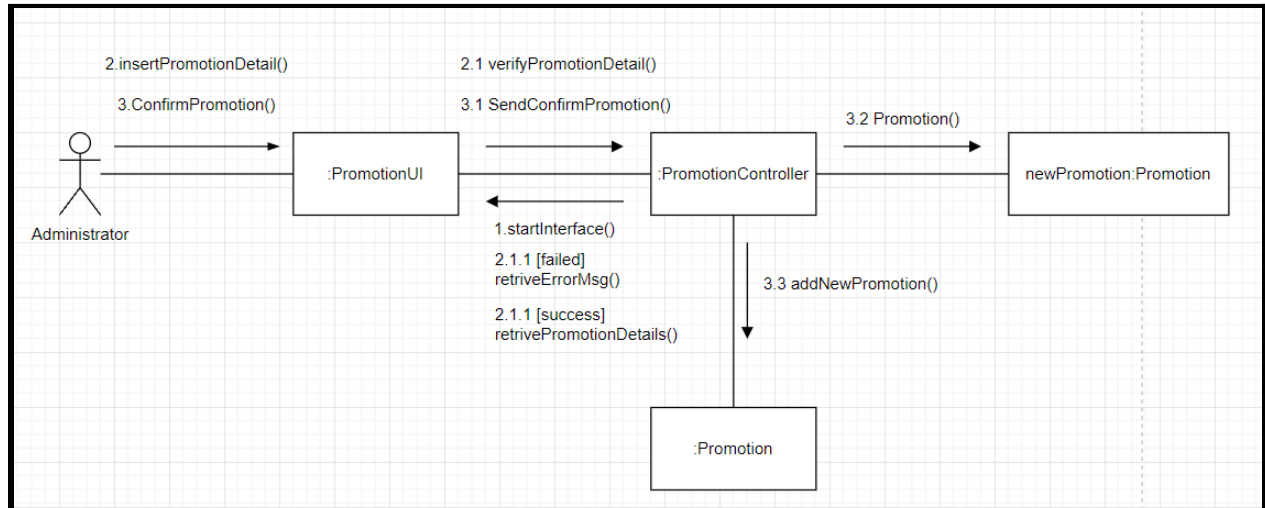


Tutorial 6

Draw.io URL: [Flowchart Maker & Online Diagram Software](#)

Question 1 & 2 : Draw.io

Question 3 (Collaboration Diagram)



Tutorial 7 : Specifying Operations

Question 1

Explain **TWO (2)** reasons Object Constraint Language (OCL) is needed to specify operations in a class.

- From the analysis perspective, OCL ensures the user's needs can be understood.
- From the design perspective, OCL can guide the programmer to develop an appropriate implementation such as the constraints from a method in a class.
- **Computational complexity** – e.g. the bubble sort algorithm has an execution time: $N \times N$, N =number of items being sorted.
- **Ease of implementation and understandability**: better to sacrifice performance to simplify implementation
- **Flexibility**: flexible to change
- **Fine-tuning the object model**: some adjustment to the object model may simplify the algorithm and should be considered

OCL is not a programming language, it is a formal language used to express the business rules of an operation which cannot be shown in a class diagram. Example of OCL refers to pre-conditions or post-condition of an operation. In other words, if we refer to stack operation, it follows the last in first out fashion of arranging objects in a container, in which there is no way for the user to specify a remove method to take out any object anywhere from within a stack.

Question 2



Refer to the diagram above, write OCL based on the following assumptions:

- a) A set of all departments in the company
- b) The staff whose payroll number is 12345
- c) Get the age of an employee
- d) All employee are aged 18 or above

- a. context Company inv : self.departments
- b. context Staff inv : self.payrollNumber = 12345
- c. context Staff inv : self.getAge()
- d. context Staff inv : self.age -> select (age >= 18)

***Notes : the context must follow the class name**

Description:

Logical operator implies consist of two parts. Whenever the first part is true, the second part has to be true as well

expr1 implies expr2: returns true when both expressions are true

expr1 implies expr2: when expr1 is not true, true is returned (in other words, expr2 value matters only when expr1 applies)

Context: TransactionType1

OCL:

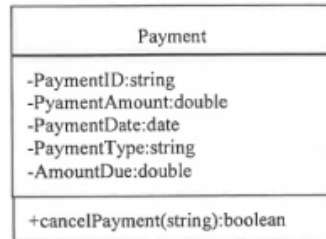
```
self.Amount > 200 implies
self.Debtor.Name->size() = 1 and self.Creditor.Name->size() = 1
```

Description: The example rule mandates the usage of both Creditor and Debtor name when Amount exceeds 200

Question 3

The following diagram shows **Payment** design class diagram for the Online Furniture Sales Ordering System

Construct **TWO (2)** Object Constraint Languages (OCL) for the Payment class.



- **context Payment inv: self.AmountDue < PaymentAmount**
 - If the PaymentAmount is more than AmountDue, the excess will be kept for the next charges/bill
- **context Payment inv: self.AmountDue = 0**
 - If the AmountDue is equal to 0 which means, the customer is clear from debt, the system will not chase the customer for payment. The system will update the customer payment status to clear

Question 4

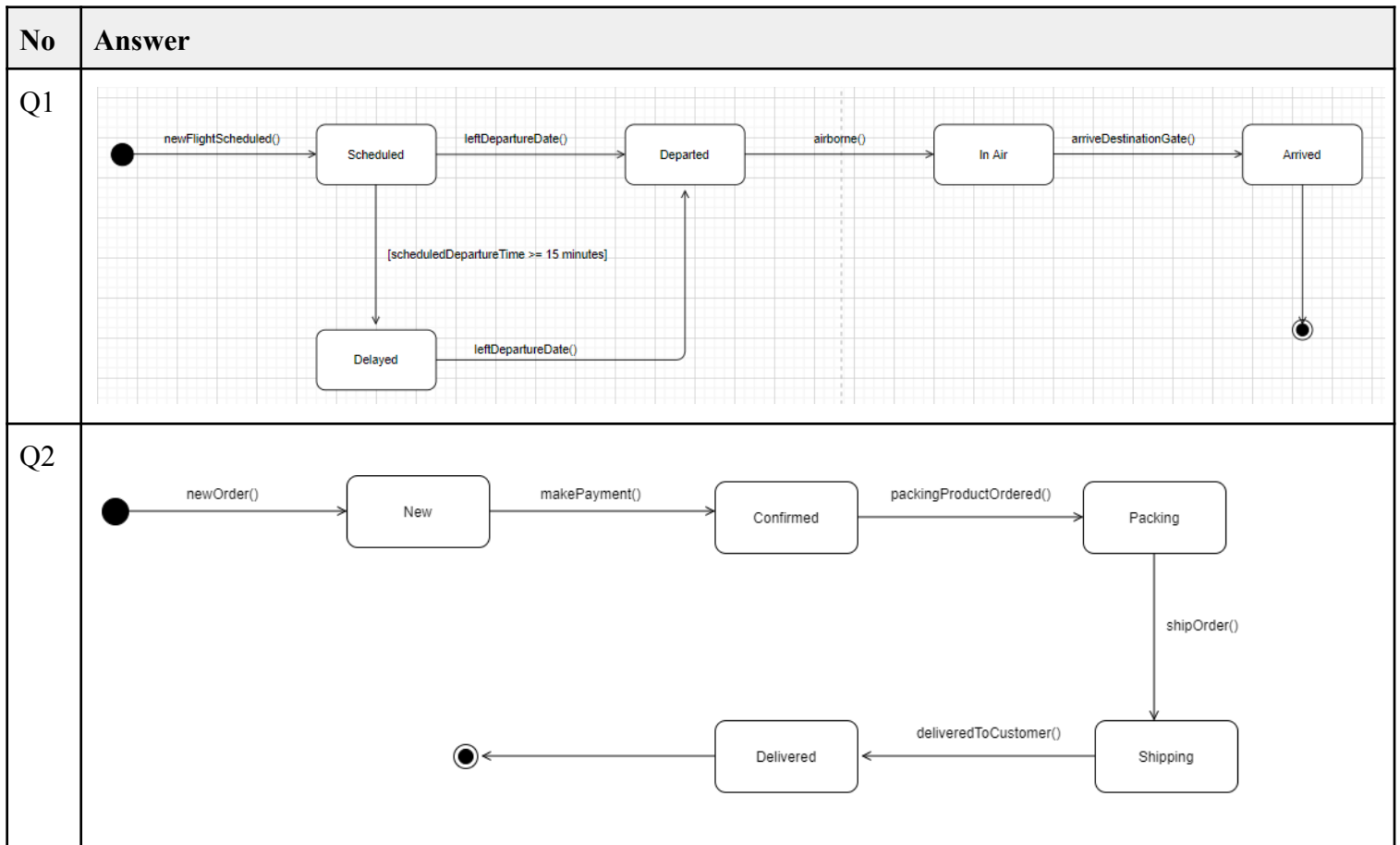
Write Object Constraint Language (OCL) equivalent for the following constraints:

- A company only hires employee that is age 20 and above.
- An employee who registers one car sticker in the company must possess at least two years of driving experience.
- If an employee is promoted to senior level, the monthly bonus is at least RM500.
- A manager must possess at least 5 years of managerial experience.

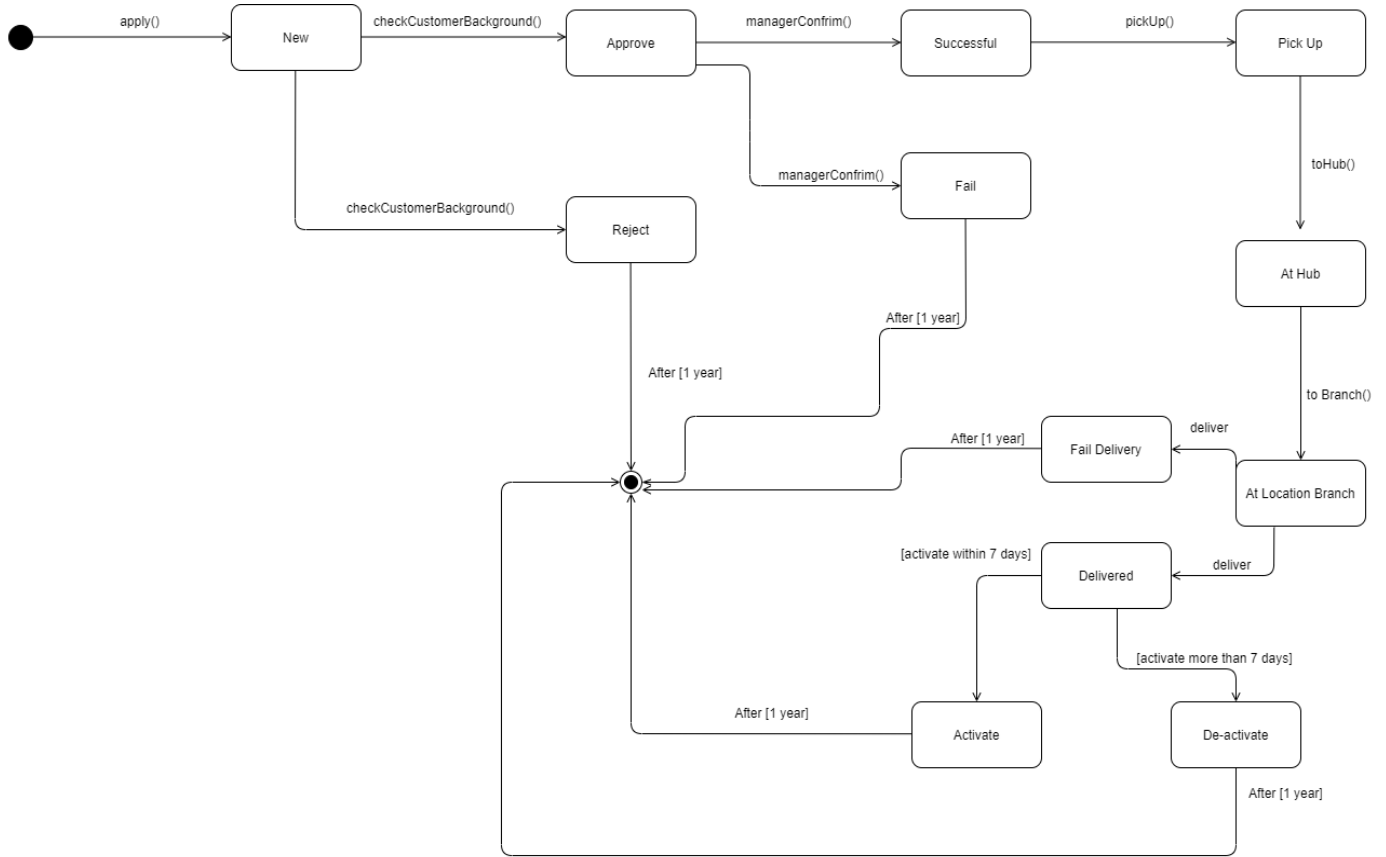
- context Company inv : self.employee.age>=20
context Staff inv : self.age>=20 *-- better answer (never go too far from class diagram)*
- context Employee inv : self.carSticker -> notEmpty() implies self.driveExp >= 2
- context Employee inv : self.level="Senior" implies self.monthlyBonus>=500
- context Employee inv : self.position="Manager" implies self.workingExp >=5
context Manager inv : self.managerialExp>=5

Tutorial 8 (done)

Draw.io URL: [Tutorial 8 OOAD](#)

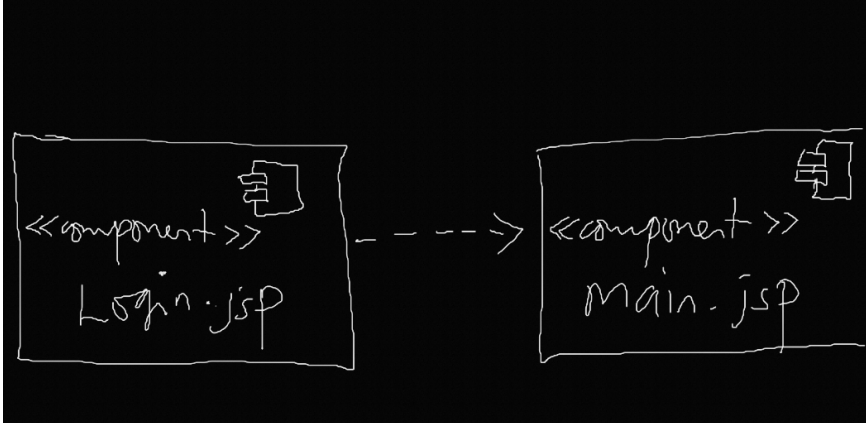


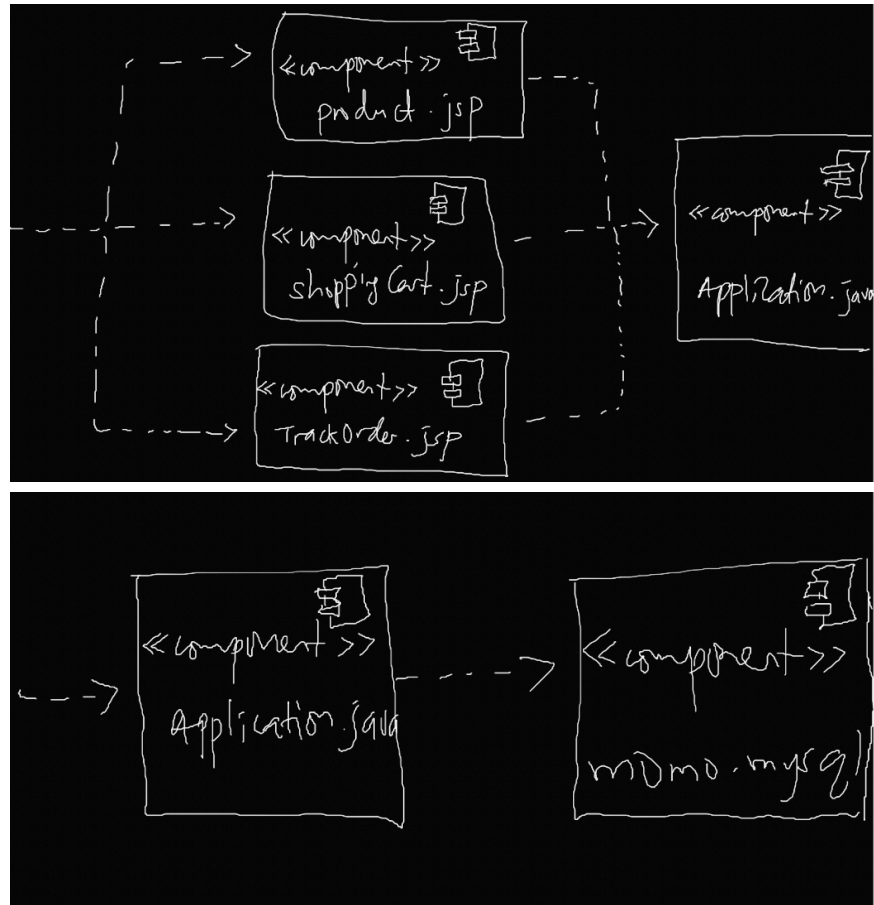
Q3



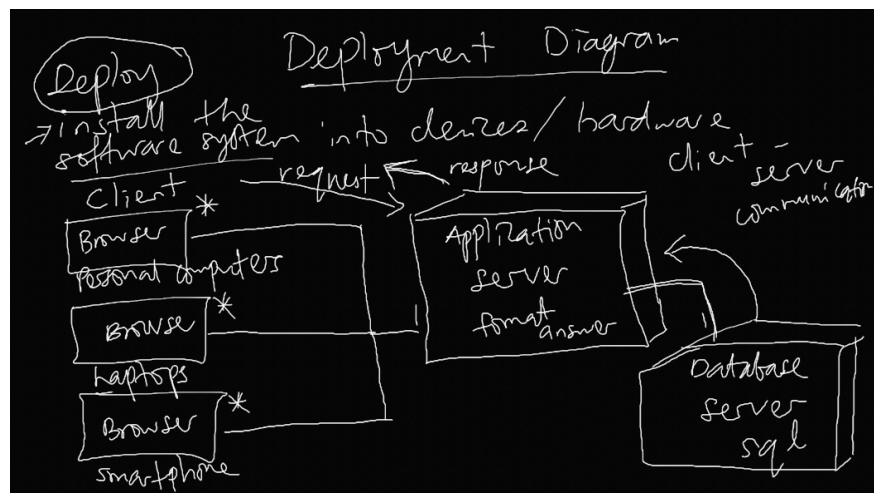
Tutorial 9

Draw.io URL: https://app.diagrams.net/#G1S_lcxI-i133qH38iGY5_aXIU6KZathDP

Q	A
T9Q1	<p>Partitioning is to divide a system into few partitions/divisions, where each partition has all the resources to perform its functions appropriately. Examples of resources are processing power, memory and storage.</p> <p>Example: Hard disk that is partitioned into C and D drives/directories, where both C and D drives are having the same architecture, serve with the same capability.</p> <p>Layering is to divide a component into few independent layers. Example: Three-tier architecture with View/Presentation layer, Controller/Business Logic layer, Model/Database layer. When any layer has any bugs, developers only required to fix the bugs present a layer without touching the rest.</p> <p>Software developers love layering of system or software architecture because of ease of maintenance/to achieve software maintainability. In other words, it also cuts down the maintenance time and cost.</p>
T9Q2	 <p>The diagram is a UML Component Diagram on a black background. It features two rectangular boxes representing components. The left box is labeled with the stereotype <<component>> and the name Login.jsp. The right box is also labeled with the stereotype <<component>> and the name Main.jsp. Both boxes have a small component icon in the top right corner. A dashed arrow points from the Login.jsp component to the Main.jsp component, indicating a dependency.</p>



Component Diagram



Deployment Diagram

Tutorial 10

Draw.io URL: <https://app.diagrams.net/#G1Spa7YVtBI2VaNH2nIX-qx72zmGU2lij>

Question		
T10Q1	<p>Coupling</p> <ul style="list-style-type: none"> - Undesirable - High coupling increase maintenance cost - Example: Inside Function A, there is a call to execute Function B. Inside Function B, there is a call to execute Function C. This kind of design is of very high dependency. If Function B has bugs, programmers are required to check Function C and Function A to find out the root cause of the problem. - One change requires changes to multiple components in a system that have high coupling. <p>Cohesion</p> <ul style="list-style-type: none"> - Desirable - High cohesion decrease maintenance cost - Example: Payment modules that are high in cohesiveness will only have functions or tasks such as calculate subTotal and calculate grandTotal. Because these functions are highly relevant to achieve the same purpose. - Examples of undesirable low cohesiveness systems such that the Student register course module involve functions or tasks such as register and calculate CGPA. - Modules that have high cohesion perform a specialized function to increase maintainability because a scope of functions can be easily found in one module. <p>Programmers desire low coupling and high cohesion when designing information systems.</p>	
T10Q2a	<p>LSP Liskov Substitution Principles Barbara Liskov</p> <p>The meaning of substitute and its example - Supermarket - baking cake - anchor butter - Reach - tak ada stock - if no anchor butter can be replaced with lurpak butter. Another example: Planta can be replaced with Margarine.</p> <p>What is LSP? How to use it? When to use it? Apply in which situation? Where to use it? Why do we have to use it? What about application of LSP in real life as an example?</p>	<p style="text-align: right;">Liskov</p>

	<p>Adhering to the Liskov Substitution Principle.</p> <p>It is an inheritance concept with parent-child relationship.</p> <p>Principle says “When class S is a subtype of class T then an object of type T can be replaced by an object of type S without affecting functionality/correctness of the implementation or program”.</p> <p>Consume a service of a base class, must work correctly when the base class object is replaced by a child class (derived class) object.</p> <p>Based on Figure 1, since HourlyPaidWorker is the subclass of ContractWorker, in this case, HourlyPaidWorker also will be inheriting attributes of superclass such as BasicSalary, Allowances and ContractDuration. In addition, HourlyPaidWorker also will be inheriting methods/functionality of superclass such as getContractDuration() and calContractWorkerSalary(). This shows that the inheritance hierarchy has violated the Liskov Substitution Principles. It is because the HourlyPaidWorker should not be having any BasicSalary, Allowances and also not they are not a Contract worker.</p>	Substitution Principle
T10Q2b	<pre> classDiagram class Worker { WorkerID Name IdentityCard No } class ContractWorker { Basic Salary Allowances Contract Duration cal Contract Worker Salary() get Contract Duration() } class HourlyPaidWorker { Total Hours Rate per Hour set Rate per Hour() get Rate per Hour() cal Daily Paid() } Worker < -- ContractWorker Worker < -- HourlyPaidWorker </pre>	
Q3	draw.io	
Q4	<p>Design qualities :</p> <ul style="list-style-type: none"> ● Functional - the online online sales order system should work as it requires which allows customers to view the furniture online, add to cart and place an order without any issue. ● Efficient - When the customer places an order using this online sales order system, it should process the payment in a short period of time such as within 30 seconds in order to let the user have a good user 	

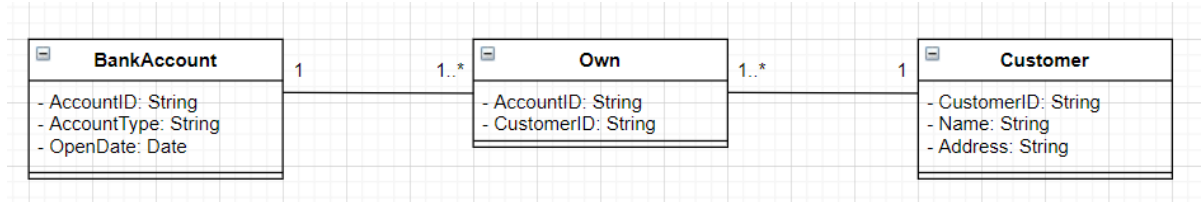
	<p>experience while using this system.</p> <ul style="list-style-type: none"> ● Reliable - System should be able to take 1000 orders at the same time and maintain the system operation smoothly without any bugs. ● Secure - The online sales order system should provide secure access by encapsulating the customer credential data to prevent credential theft from happening. All the sensitive information, especially password has to be encrypted before it is stored in the user record/table/database. For example, Oracle has MD5, SHA-1, SHA-256. 	
Q5	draw.io	

Tutorial 11

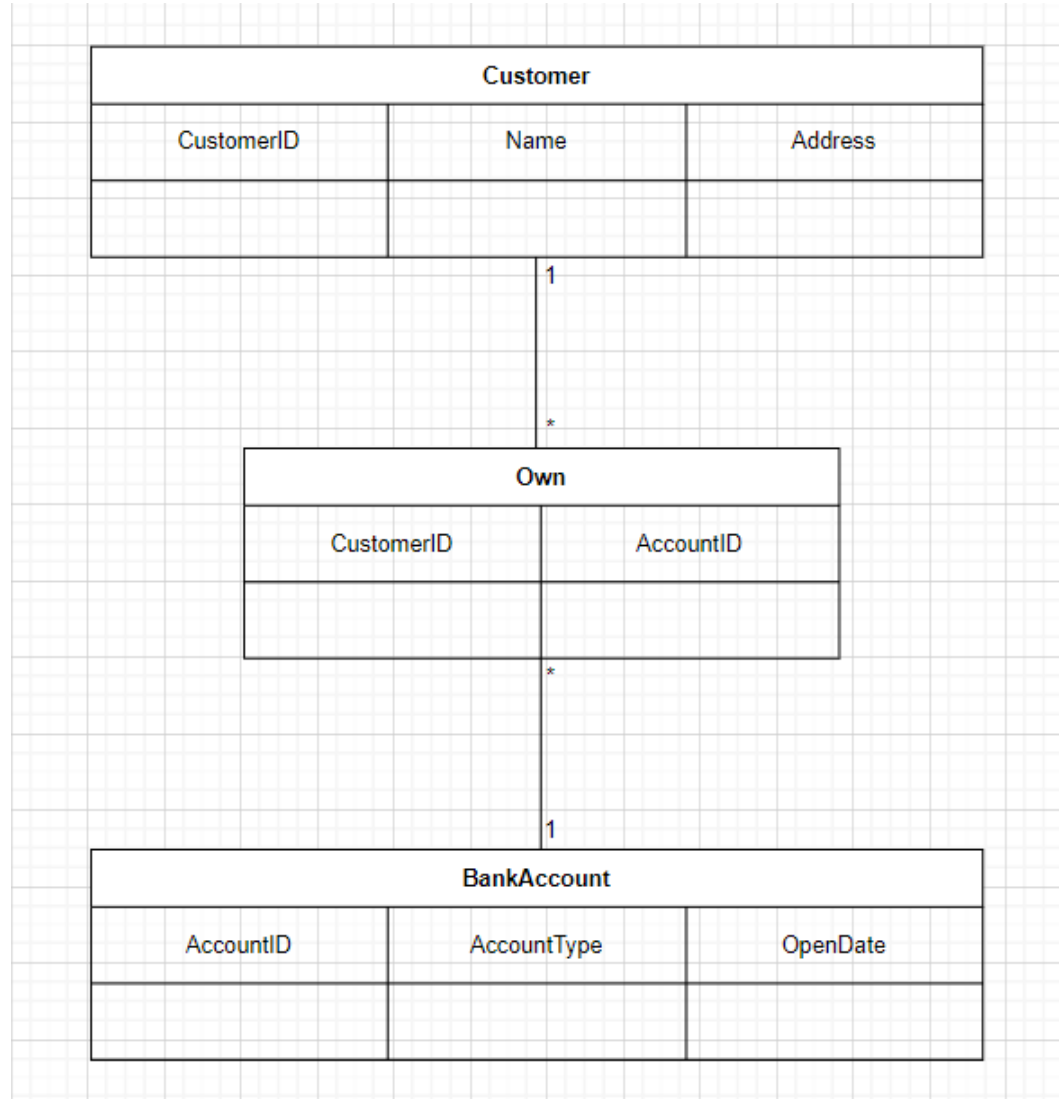
Draw.io : [Flowchart Maker & Online Diagram Software](#)

Q	A
Q1	<p>What is the key difference between a relational DBMS and an object DBMS</p> <ul style="list-style-type: none"> ● Relational DBMS is a collection of tables with fields, primary key, foreign key, column, row (records) and relationships between tables are established using primary key. If the system is designed using OOAD, classes have to be mapped to tables in Relational DBMS and some design decision trades off have to be taken care of. ● In relational DBMS, the data type is simple. For example, Text, int, char ● Object-oriented DBMS is differ from relational DBMS in that they are capable of storing objects with all their complex structure. It is not necessary to transform the classes in the design model of the system in order to map them to storage objects. Designing for an object database will have a minimal impact on the design of the system. ● In OODB, we can use complex data types. For example, address type, arraylist, interface, student type, string, date
Q2	<pre> erDiagram STAFF --o{ FLIGHT_SCHEDULE : "has" STAFF --o{ MAINTAIN : "has" FLIGHT_SCHEDULE --o{ MAINTAIN : "has" MAINTAIN --o{ STAFF : "has" MAINTAIN --o{ FLIGHT_SCHEDULE : "has" STAFF { string staffID PK string staffName string staffContactNo string position float salary } FLIGHT_SCHEDULE { string flightID PK string origin string departDateTime string arrivalDateTime string Destination } MAINTAIN { string staffID FK string flightID FK } </pre> <p>The diagram illustrates a many-to-many relationship between the Staff and FlightSchedule tables. The Staff table has attributes: staffID (primary key), staffName, staffContactNo, position, and salary. The FlightSchedule table has attributes: flightID (primary key), origin, departDateTime, arrivalDateTime, and Destination. An associate table named Maintain is used to resolve this relationship, with foreign keys staffID and flightID linking back to the primary keys of the Staff and FlightSchedule tables, respectively.</p>
Q3	<ul style="list-style-type: none"> ● Many to many relationship ● Resolve many to many relationship by using using associate table

Class diagram Diagram:



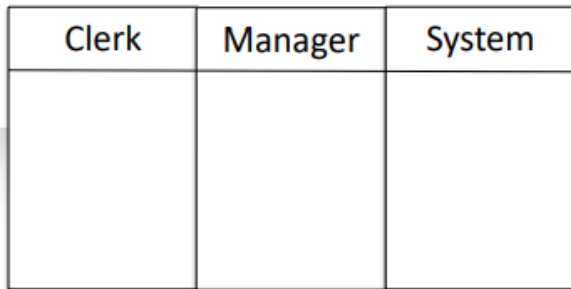
Relational DBMS Diagram:



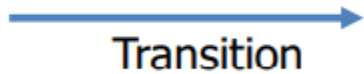
Review from all chapters

- C1:** Basic OO concepts
- C2:** Waterfall, USDP, **Activity Diagram**
- C3:** Functional & non-functional requirements, **use case diagram** and description
- C4:** **Association, multiplicity, association class**, use case realizations, CRC card
- C5:** **Composition and aggregation**
- C6:** **Sequence diagram** & collaboration diagram
- C7:** **OCL**
- C8:** **State chart**
- C9:** Client server, peer to peer, layering, partitioning, MVC
- C10:** Architecture type, Package diagram, **component diagram, deployment diagram**
- C11:** Logical & Physical design, quality of design, cohesion, coupling & its type, LSP
- C12:** Class specification, Collection class, 1 way & 2 way association
- C13:** Data management layer, RDBMS

Activity Diagram



Swim lane



start



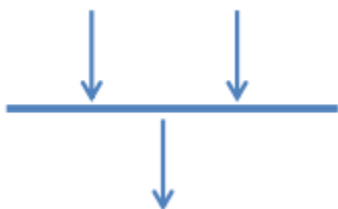
end



Decision

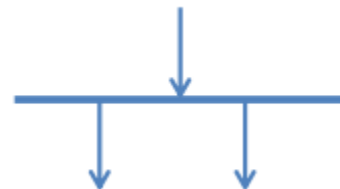
- contain guard condition to determine which path to take

Join



- Waiting for all subtasks to finish before proceeding

Fork



- Starting several subtasks in parallel (the sequence irrelevant)