**UML Diagrams**

* **What is UML?**

**-**UML stands for Unified Modeling Language.

-UML is popular for its diagrammatic notations. We all know that UML is for visualizing, specifying, constructing and documenting the components of software and non-software systems. Hence, visualization is the most important part which needs to be understood and remembered.

UML notations are the most important elements in modeling. Efficient and appropriate use of notations is very important for making a complete and meaningful model. The model is useless, unless its purpose is depicted properly.

Hence, learning notations should be emphasized from the very beginning. Different notations are available for things and relationships. UML diagrams are made using the notations of things and relationships. Extensibility is another important feature which makes UML more powerful and flexible.

* **Why Do We Use UML?**

A complex enterprise application with many collaborators will require a solid foundation of planning and clear, concise communication among team members as the project progresses.

Visualizing user interactions, processes, and the structure of the system you're trying to build will help save time down the line and make sure everyone on the team is on the same page.

* **What are the Types of UML Diagrams?**

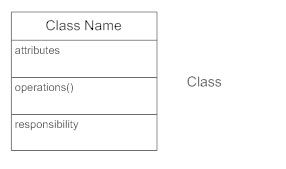
1. Class diagrams
2. Use Case Diagrams.
3. Entity Relationship Diagrams.
4. Sequence Diagrams.
5. State Transition Diagrams.
6. Data Flow Diagrams.
7. Collaboration Diagrams.
8. Activity Diagrams.
9. Component Diagrams.
10. Deployment Diagrams.

**Practical No. 1**

**CLASS DIAGRAM**

**What is a Class diagram?**

* A class diagram is a static structural diagram.
* It describes the attributes, operations of the class, constraints imposed on them and the relationships among objects.
* Classes are represented with boxes that contains three compartments.



**VISIBILITY**

**Use visibility markers to signify who can access the information contained within a class.**

**Private visibility, denoted with a - sign, hides information from anything outside the class partition.**

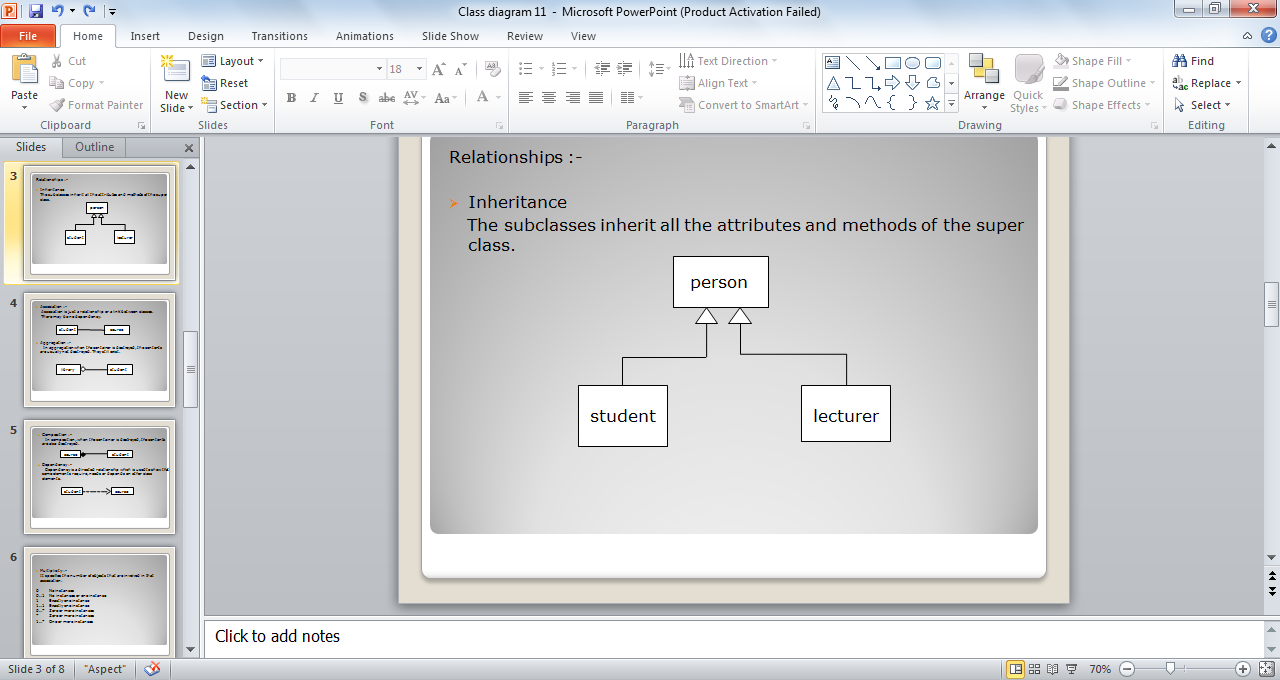
**Public visibility, denoted with a + sign, allows all other classes to view the marked information.**

**Protected visibility, denoted with a # sign, allows child classes to access information they inherited from a parent class.**



**RELATIONSHIPS:-**

* **Inheritance**

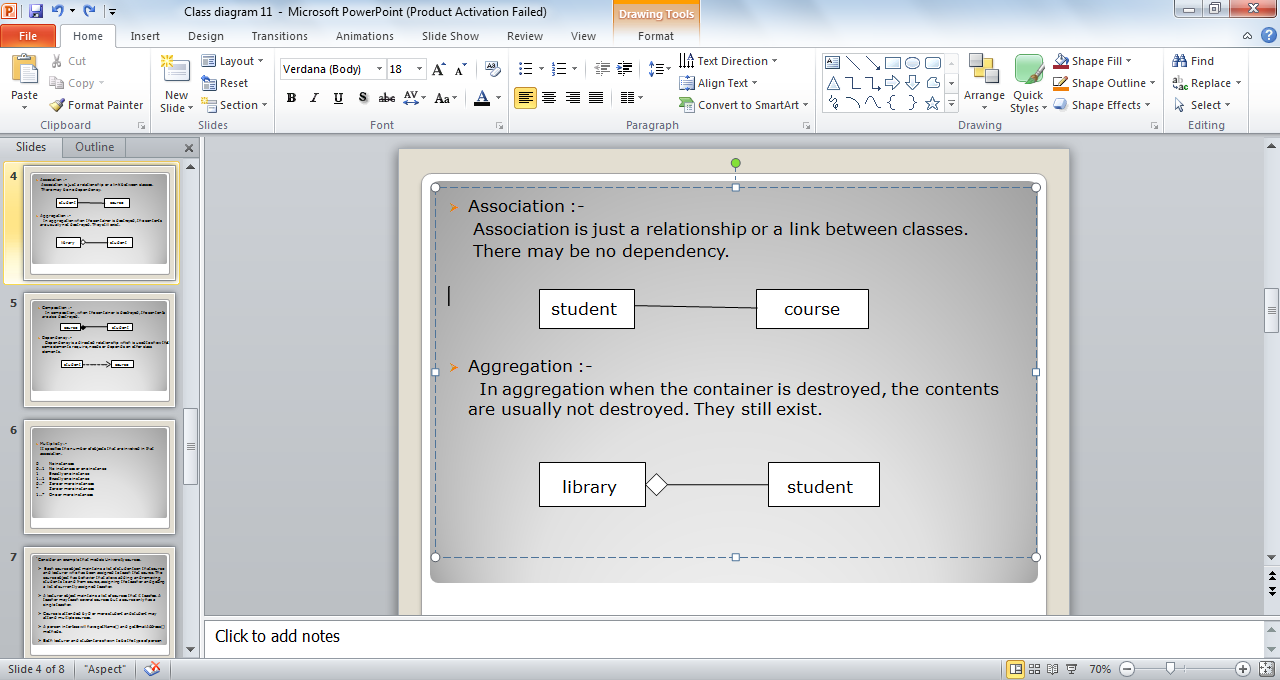
The subclasses inherit all the attributes and methods of the super class.

Generalization is another name for inheritance or an "is a" relationship. It refers to a relationship between two classes where one class is a specialized version of another. For example, Honda is a type of car. So the class Honda would have a generalization relationship with the class car.

* **Association :-**

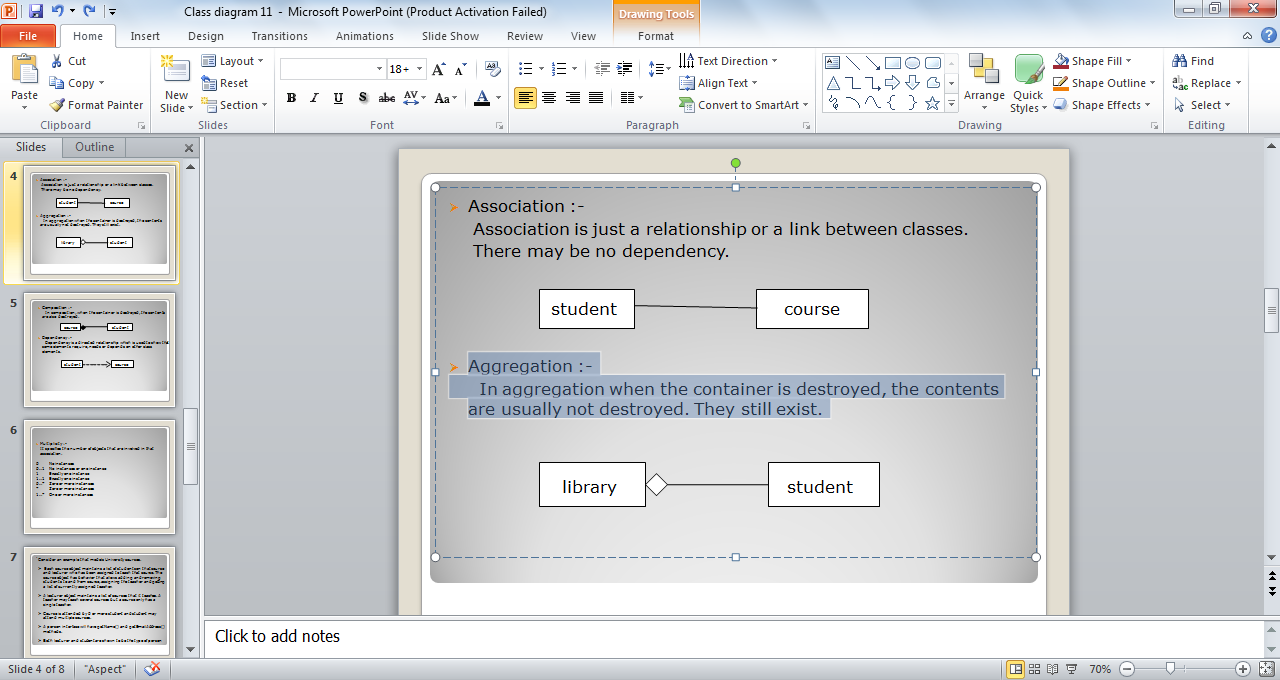
Association is just a relationship or a link between classes.

There may be no dependency.

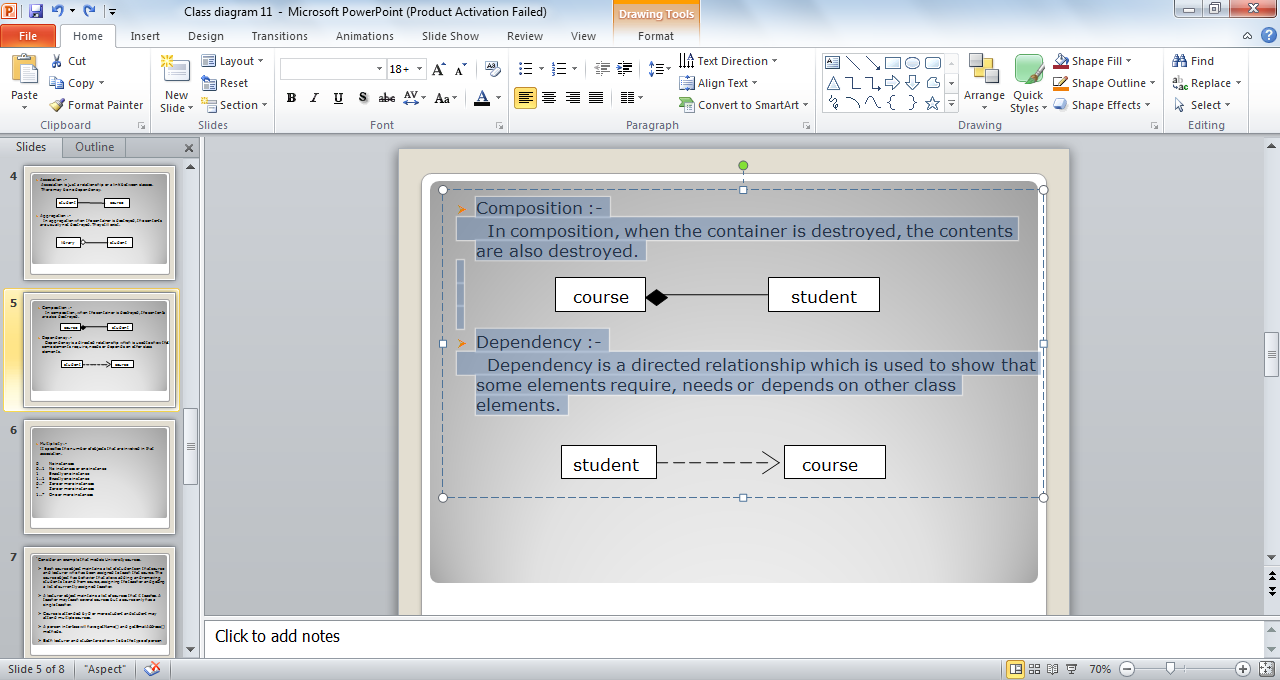


* **Aggregation :-**

In aggregation when the container is destroyed, the contents are usually not destroyed. They still exist.

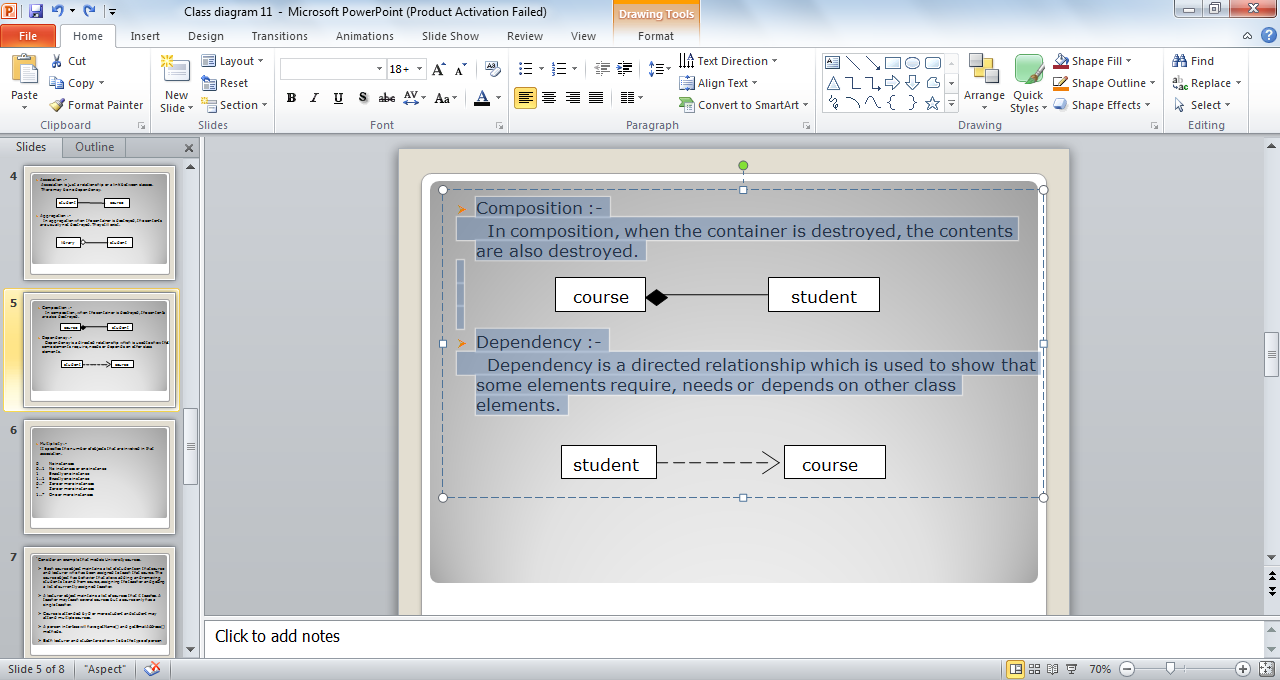


* **Composition :-**

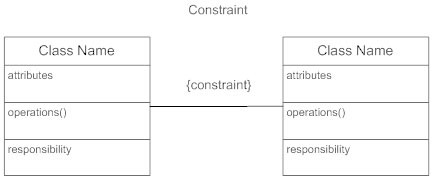
 In composition, when the container is destroyed, the contents are also destroyed.

* **Dependency :-**

Dependency is a directed relationship which is used to show that some elements require, needs or depends on other class elements.



* **Constraint :Place constraints inside curly braces {}.**



* **Multiplicity :-**

It specifies the number of objects that are involved in that association.

0 No instances

0…1 No instances or one instance

1 Exactly one instance

1…1 Exactly one instance

0…\* Zero or more instances

\* Zero or more instances

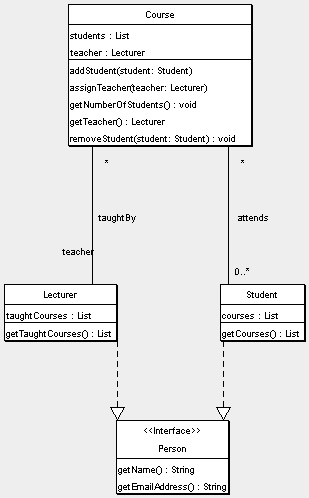
1…\* One or more instances

**In real life coding examples, the difference between inheritance and aggregation can be confusing. If you have an aggregation relationship, the aggregate (the whole) can access only the PUBLIC functions of the part class. On the other hand, inheritance allows the inheriting class to access both the PUBLIC and PROTECTED functions of the superclass.**

**PRACTICAL NO.1 (A)**

**Draw a class diagram for the scenario given below.**

This is an example that models University Courses. Assume three classes’ such as course, lecturer, student and an interface person. Each course objects maintains a list of student on that course and lecturer who has been assigned to teach that course. The course object has behavior that allows adding and removing student to and from course, assigning the teacher and getting a list of currently assigned student and currently assigned teacher. A teacher may teach several courses but a course only has a single teacher .A lecturer object maintains a list of courses that it teaches, course is attended by 0 or more student and student may attend multiple courses. A person interface will have getName() and getEmailAddress () methods both lecturer and student are shown to be the type of person.



**Explanation :**

The below class diagram is an example that models university courses.

In this example there are 3 classes such as **course, lecturer, student** and an interface **person.**

Where each course objects maintains a list of students on that course and lecturer who has been assigned to teach that course.

**Class Course:-**

* The course objects has some behavior that allows adding and removing student to and from course.
* And also assigning the teachers and getting a list of currently assigned student and currently assigned teachers.
* A teacher may teach several courses but a course only has a single teacher.

**Class Lecturer:-**

* A lecturer object maintains a list of courses that it teaches.
* Course is attended by zero or more student and student may attend multiple courses.

**Interface Person:-**

* It consists of two methods getName() and getEmailAddress() that helps

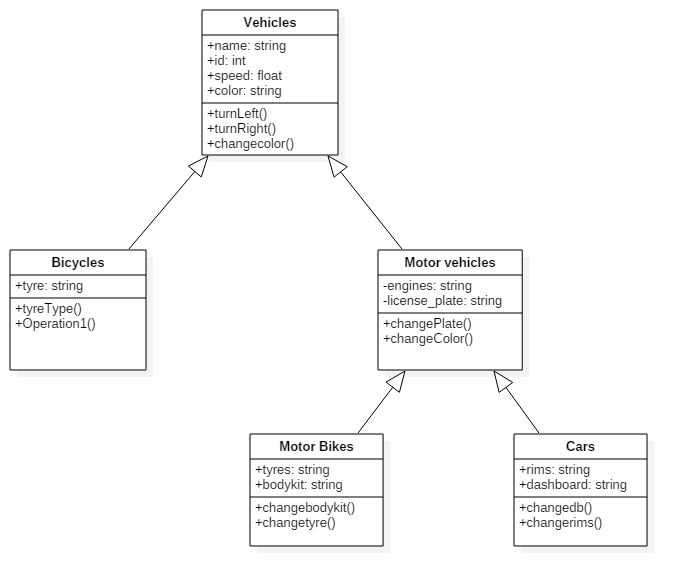
to get the names and email address of the persons that is the part of the university courses.

* Both the lecturer and student are shown to be the type of person.

**PRACTICAL NO. 1(B)**

**Draw a class diagram using StarUML for the scenario given below.**

This scenario shows an *inheritance hierarchy* of a series of classes and their subclasses. It’s for an imaginary application that must model different kinds of vehicles such as bicycles, motor bike and cars. All Vehicles have some common attributes (speed and color) and common behavior (turnLeft, turnRight). Bicycle and MotorVehicle are both kinds of Vehicle and are therefore shown to inherit from Vehicle. To put another way, Vehicle is the superclass of both Bicycle and MotorVehicle. In our model MotorVehicles have engines and license plates. Attributes have been added accordingly, along with some behavior that allows us to examine those attributes. MotorVehicles is the base class of both MotorBike and Car; therefore these classes not only inherit the speed and color properties from Vehicle, but also the additional attributes and behavior from MotorVehicle. Both MotorBike and Car have additional attributes and behavior which are specific to those kinds of object.



**Explanation :**

The above diagram shows complete inheritance between different object that are called vehicles.

**Class vehicle** is the topmost class that is inherited by classes bicycle and motor vehicle.

The attributes and functions of Vehicle are inherited by Bicycle and Motor vehicle that are

Attribute : name, id , speed, color

Functions :turnleft (), turnright (), chgcolor ().

**Class Bicycle** has additional information like

Attributes: tyre

Methods :tyretype ()

**Class Motor vehicle** also has additional properties like

Attributes : engines, license\_plate

methods :changeplate (), chgcolor ()

Futher Class MotorBikes and Cars are inherited from Class Motor Vehicles

having additional properties like

**Class MotorBikes**

Attributes :tyres, bodykit

Methods :chgbodykit (), chgtyre ()

**Class Cars**

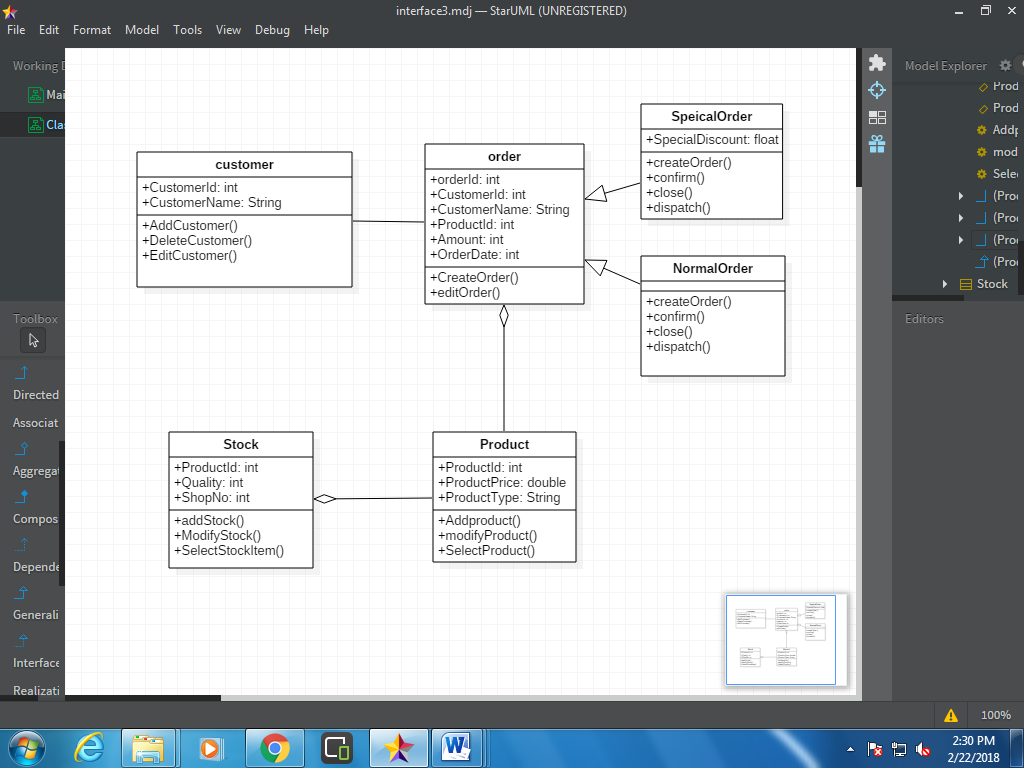
Attributes : rims, dashboard

Methods :chagDb (), chgrims ()

**PRACTICAL NO. 1(C)**

**DRAW A CLASS DIAGRAM FOR THE SCENARIO GIVEN BELOW:**

This is an example that models “**ORDER MANAGEMENT”**. The **Customer** object has properties such as CustomerId, CustomerName, Address and Phone and methods such as AddCustomer(),DeleteCustomer() and EditCustomer(). **Order**object includes OrderId, CustomerId, CustomerName, ProductId, Amount and OrderDate as its property and CreateOrder() and EditOrder(OrderId) as its behavior. A customer can place one or many orders. Further there are**SpecialOrder** object and **NormalOrder** object which have same methods  CreateOrder(), confirm(), close(), dispatch() whereas the SpecialOrder object also has one property named SpecialDiscount. SpecialOrder and NormalOrder objects are both kinds of order and are therefore shown to inherit from order entity. Moreover the system also has Product entity having attributes such as ProductId, ProductPrice, ProductType and methods such as AddProduct(), ModifyProduct() and SelectProduct(ProductId). Stock object has properties like ProductId, Quality and ShopNo and behavior such as addStock(), ModifyStock(ProductId) and slectStockItem(ProductId). Note that specialOrder and NormalOrder has 1 or more product whereas stock has many products.



**Explanation :**

The above diagram shows how the Orders from a number of customers are managed.

There is a **Class Order** with Attributes that take user information and methods that allow fo create as well as confirm orders

It is inherited by 2 classes :**Class specialOrder** and **Class NormalOrder.**

The Customer class is dependent on Order and the multiplicity shows there can be more than many customers placing orders.

Class order and **product** are in Composition relation where there can be no order if there is no product.

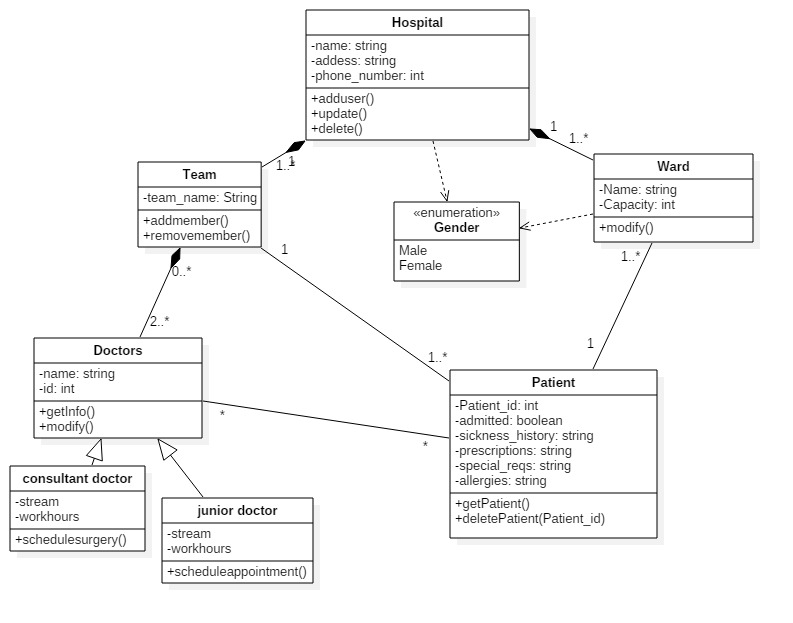
There can be more than 1 product in a single order

**Class stock** is in association relation I.e a static relatiom with product Class.

**PRACTICAL NO. 1(D)**

**DRAW A CLASS DIAGRAM FOR THE SCENARIO GIVEN BELOW:**

This is an example that models “**Hospital Management**”. The **ward** object of this system has attributes such as name, patient-gender and capacity. Note that patient-gender is a gender type which is an enumeration containing enums male and female. The system also has **Patient** entity with attributes such as patient\_id, admitted, sickness \_history, prescriptions, special\_reqs and allergies and gender which is again a gender enumeration type. And operations such as getPatient() and deletePatient(Patient\_id).Ward is a division of a **hospital** object having attributes such as name address and phone number. In hospital there are number of wards each of which may be empty or have one or more patient. Each ward has unique name. This ward is shared by patients who need a similar kind of care. Each patient is on a single ward. The system also has **Doctor** entity which is further classified into **Consultant Doctor**and**Junior Doctor**. The doctors in the hospital are organized into **Teams**entity with attribute team\_name. Each team can have two or more doctors. Each patient is under the care of a single team of doctors. A patient may be treated by any number of doctors but all the doctors must belong to same team that cares for the patient. Note that team is own by the hospital.



**Explanation :**

The above class diagram shows relationship between different entities of a hospital.

It starts with the **Class Hospital** that contains attributes needed to take users information and Methods to manipulate changes.

It is in composition relationship with 2 classes

1) **Class team** : where there can be more than 1 team in a hospital

2) **Class Ward** : which shows there can be more than 1 ward in a hospital

and both these classes cannot exist without the Hospital Class

Futher the Class team is in Composition relation with **Class doctors** where there can be 2 or more doctors in a team

Class doctors is inherited by **Consultant doctor** and **junior Doctor.**

**Class patien**t is in association relation with Class Ward, Class team and Class Doctors.

where the multiplicity says:

1. There can there can be many patients under single doctor

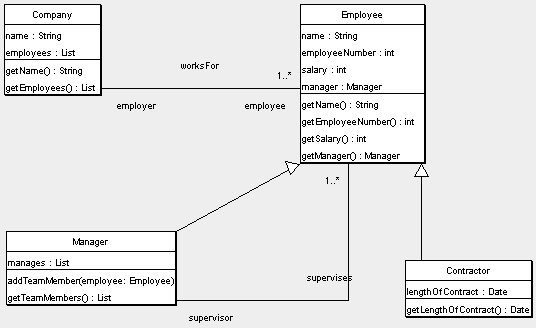
2.There can be many patients under single team

3. There can be many patients in a single ward.

**PRACTICAL NO. 1 (E)**

**Draw a class diagram for the scenario given below.**

This scenario is from system that models companies for a payroll or reporting system. Company object has properties such as name and employees\_list and getName and getEmployees as its behavior. Employee object includes employee no, name, salary and manager as its properties getName (), getEmplyoeeNo () ,getSalary() and getManager() as its methods. getManager() accepts object of manager. Company may have one or more employees. A manager object keeps manages as list property and add TeamMember(employee\_list) and getTeamMember() as its behaviors. One or more employee can be managed by manager objects. Some employees are contractual employees who are within a lieu of a contractor object. A contractor object may have length\_ of \_contract as its property and getLength() as its behavior.



**Explanation :**

The above scenario is of a class diagram reflecting a payroll or reporting system

**Class Employee** is in directly associated with **Class Company** and **Class manager** where there can more than 1 employee in a company as well as under a manager.**Class Contractual Employees** are dependent on Class Employee's attributes and methods.

**PRACTICAL NO. 1 (F)**

**Q) Create a class diagram(Use Star UML)  for “library management” using the classes with their attributes and operation given below. Also set the appropriate relationship between the classes using the relationship tools from the toolbox following the overview of the system given below.**

Overview of the system:-

a)  It has a class “Book”. Book has authors so it has an “Author” class.

b)  In order to collect book information it has “BookItem” class which uses some of the properties from book class.

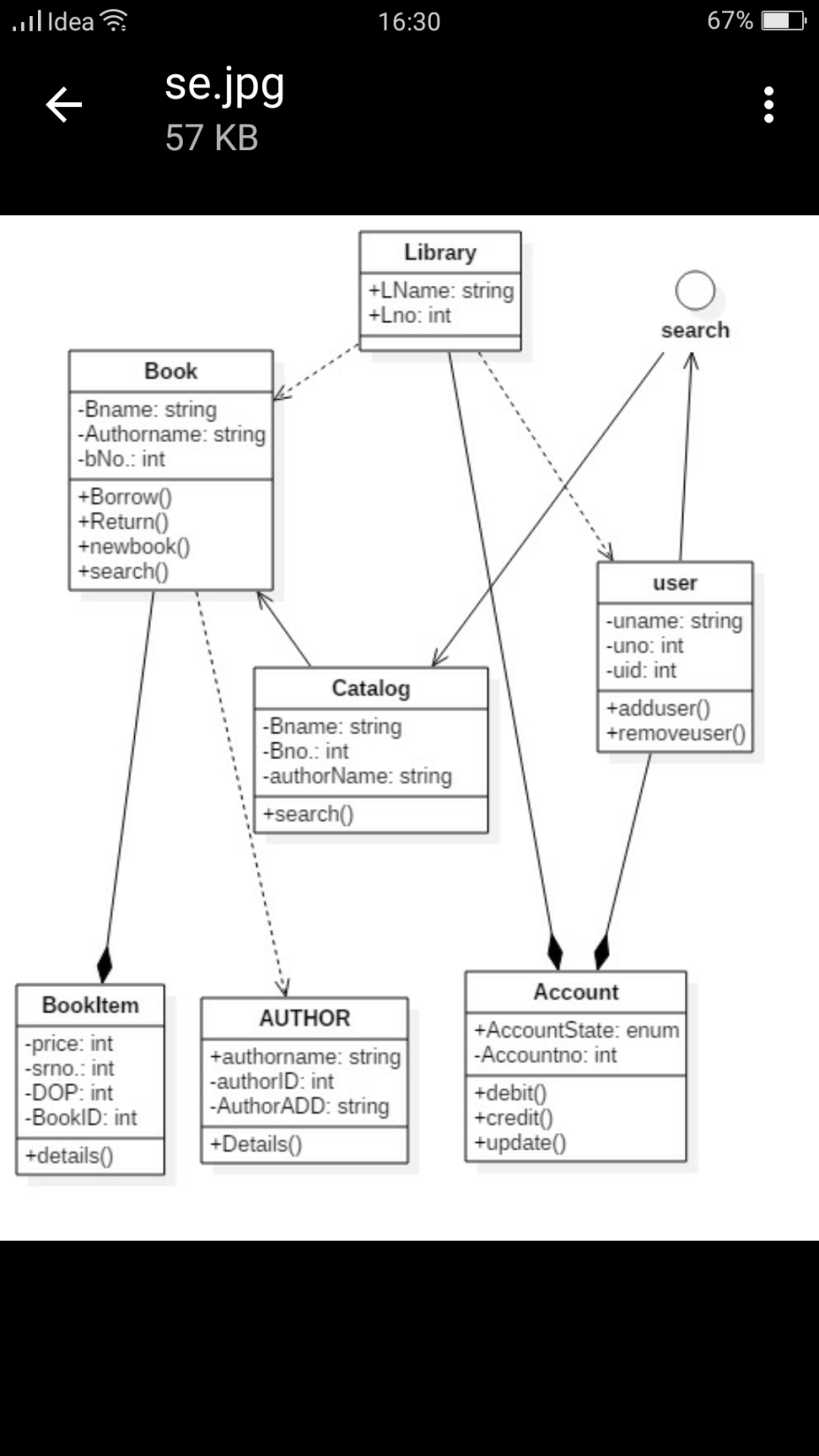
c)  It needs an account for reserving book by the user so it has an

“Account class.”

d)  In account class there is an attribute named state which uses an enumeration named “AccountState”.

e)  It also has a class “Library” to manage the account, user and the books.

f)  It has a user class to manage the user detail that has an account in the library and he can borrow and return books to library.

g)  The system also has an interface “Search” where the user searches the book he needed from the “Catalog” class.

**Explanation :**

The class diagram is of Library management system.

The main **class library** is dependent on **Class Book** and **Class user**

Class Library and Class user is in composition with **Class account.**

The Class Book is futher dependent on Author and is in composition with **Class Book iTem.**

The Class user is directly associated with **The search interface** which in turn is associated with **Class catalog** Followed by Class Book.

**Practical No. 2**

**ACTIVITY DIAGRAMS**

## **What is an Activity Diagram?**

An activity diagram visually presents a series of actions or flow of control in a system similar to a [flowchart](https://www.smartdraw.com/flowchart/) or a [data flow diagram](https://www.smartdraw.com/data-flow-diagram/). Activity diagrams are often used in business process modeling. They can also describe the steps in a [use case diagram](https://www.smartdraw.com/use-case-diagram/). Activities modeled can be sequential and concurrent. In both cases an activity diagram will have a beginning and an end.

## **Basic Activity Diagram Notations and Symbols**

##### **Initial State or Start Point**

A small filled circle followed by an arrow represents the initial action state or the start point for any activity diagram. For activity diagram using swimlanes, make sure the start point is placed in the top left corner of the first column.

Start point symbol - Activity diagram

##### **Activity or Action State**

An action state represents the non-interruptible action of objects. You can draw an action state in SmartDraw using a rectangle with rounded corners.



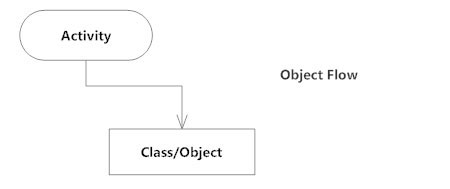
##### **Action Flow**

Action flows, also called edges and paths, illustrate the transitions from one action state to another. They are usually drawn with an arrowed line.

Action flow - Activity diagram

##### **Object Flow**

Object flow refers to the creation and modification of objects by activities. An object flow arrow from an action to an object means that the action creates or influences the object. An object flow arrow from an object to an action indicates that the action state uses the object.

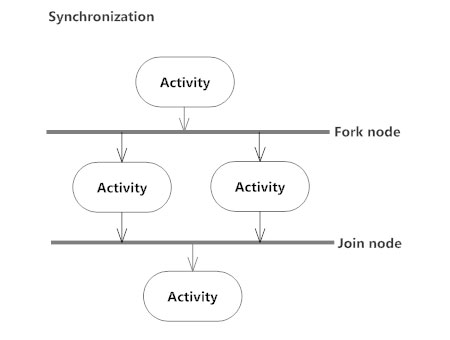


##### **Decisions and Branching**

A diamond represents a decision with alternate paths. When an activity requires a decision prior to moving on to the next activity, add a diamond between the two activities. The outgoing alternates should be labeled with a condition or guard expression. You can also label one of the paths "else."

##### Guard symbol - Activity diagram**Guards**

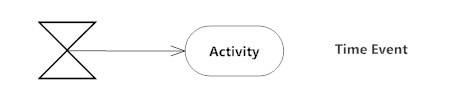
In UML, guards are a statement written next to a decision diamond that must be true before moving next to the next activity. These are not essential, but are useful when a specific answer, such as "Yes, three labels are printed," is needed before moving forward.

**Synchronization**

A fork node is used to split a single incoming flow into multiple concurrent flows. It is represented as a straight, slightly thicker line in an activity diagram.A join node joins multiple concurrent flows back into a single outgoing flow.A fork and join mode used together are often referred to as synchronization.

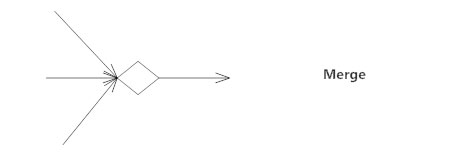
##### **Time Event**

This refers to an event that stops the flow for a time; an hourglass depicts it.



##### **Merge Event**

A merge event brings together multiple flows that are not concurrent.



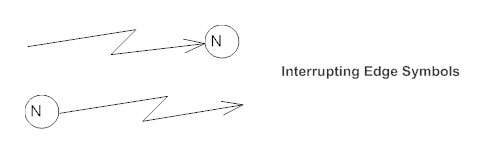
##### **Sent and Received Signals**

Signals represent how activities can be modified from outside the system. They usually appear in pairs of sent and received signals, because the state can't change until a response is received, much like synchronous messages in a [sequence diagram](https://www.smartdraw.com/sequence-diagram/). For example, an authorization of payment is needed before an order can be completed.



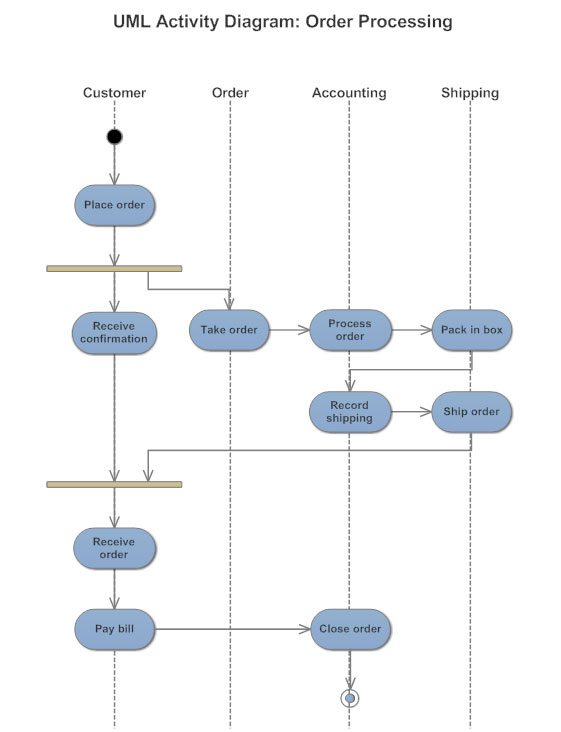
##### **Interrupting Edge**

An event, such as a cancellation, that interrupts the flow denoted with a lightning bolt.



##### **Swimlanes**

Swimlanes group related activities into one column.



##### **Final State or End Point**

An arrow pointing to a filled circle nested inside another circle represents the final action state.

End point symbol - Activity diagram

**PRACTICAL NO. 2 (A)**

**DERIVE AN ACTIVITY DIAGRAM FROM THE NARRATIVE TEXT ON “BANK ATM MACHINE FOR WITHDRAWING**

**CASH”.**

**Summary**:

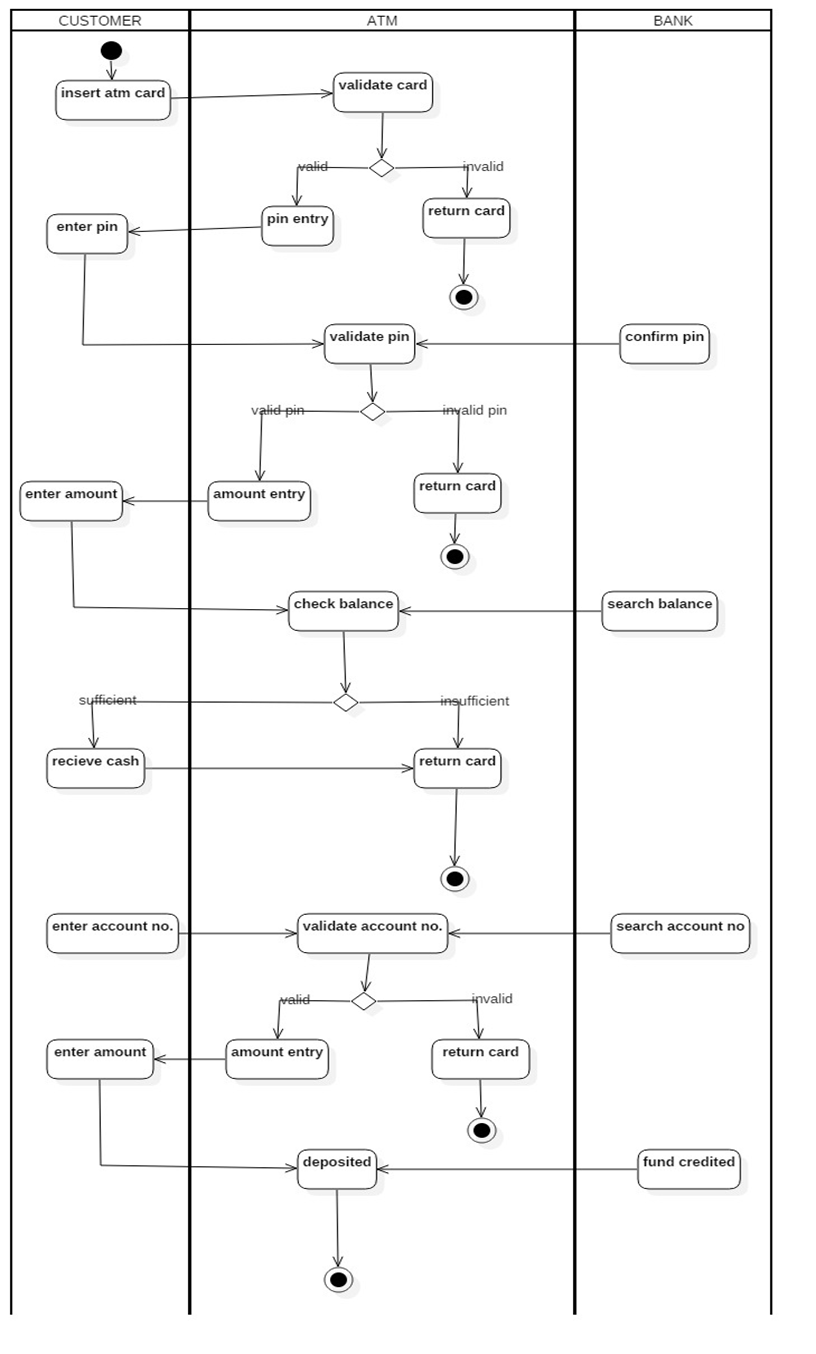
An automated teller machine (**ATM**) or the automatic banking machine (**ABM**) is a banking subsystem that provides bank **customers** with access to financial transactions in a public space without the need for a cashier, clerk or bank teller.

**Customer** uses bank ATM to **check** balances of his/her bank accounts deposit **funds**,    **withdraw** cash and/or transfer   funds which   are   th generalization  alternative of ATM transaction use case.

**FOR WITHDRAWING CASH**

On most bank ATMs, the customer is authenticated by inserting a plastic ATM card and entering a personal identification number (PIN). Bank will than authenticate the customer’s pin number. Only authenticated customer can request the system for withdrawing money while the unauthenticated customer will get back his ATM card as the system will reject the card.

Then the system will request the authenticated customer to enter the amount be de withdrawn. The bank will check the balance amount of the customer if it is sufficient bank will provide the requested amount to the customer and debit the respective amount from the balance. The customer will collect or take the amount from the slot. In case of insufficient amount the system will show the balance and reject the card. At the end of all the process the customer



**Explanation:**

**FOR WITHDRAWAL:**

The activity diagram shows the processes to withdraw cash from ATM.

The initial process starts by **inserting the card** at the customer end that is **validated by the ATM .**

If validated ,**asks for Pin** from the customer.

**Validates** the pin by comparing with the bank.

If successful, asks for the **withdrawal amount**.

**Checks balance** from the bank account and if found sufficient **Cash is received** and **card is returned**.

At any point if the process shows **invalid result, the card is returned**.

**FOR DEPOSITING:**

**Asks for account no**. from customer and **validates** the same with the bank.

If valid the amount is asked from the customer which is then **deposited** in the ATM and the **fund is credited** in the bank.

**PRACTICAL NO. 2(B)**

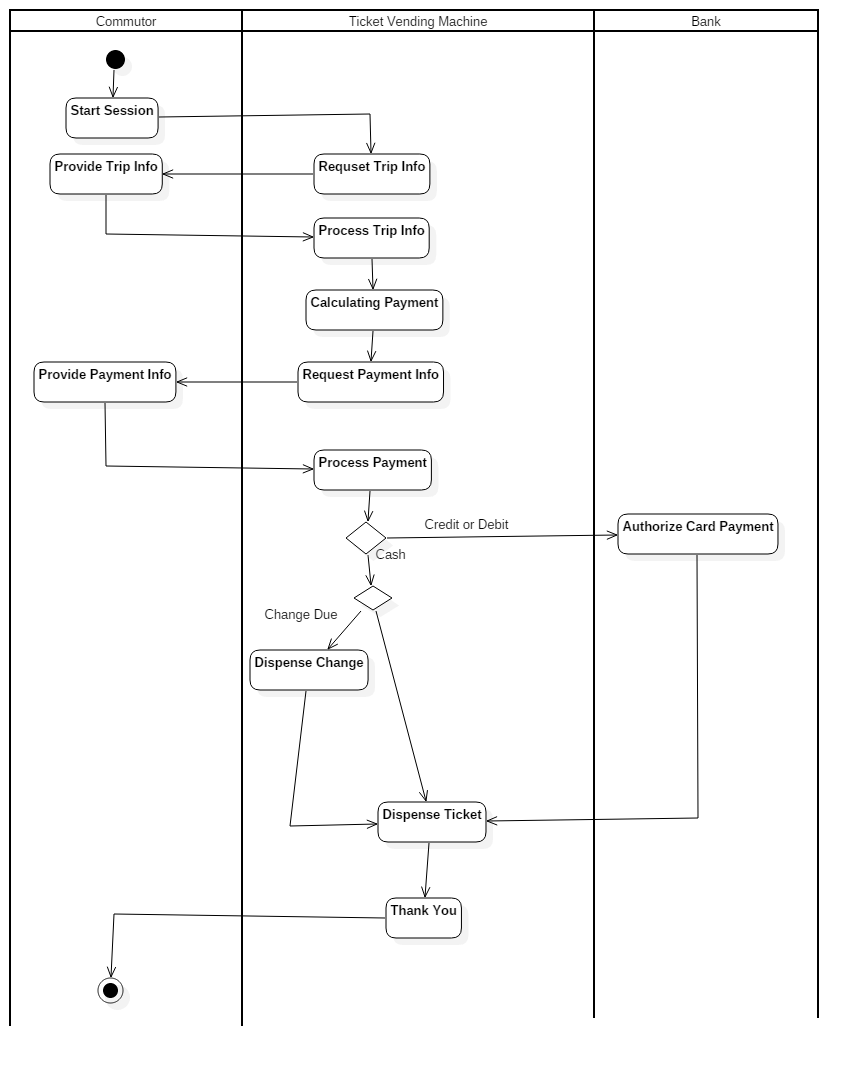
**DERIVE AN ACTIVITY DIAGRAM FROM THE**

**NARRATIVE TEXT ON “TICKET VENDING MACHINE”**

The scenario provided below describes the behavior of the purchase ticket use case.

 Activity is **started** by Commuter actor who needs to buy a ticket. Ticket vending machine will **request trip information** from Commuter. The commuter will **provide trip information** to the machine. Based on the provided trip information, the ticket vending machine will **process the trip information** and calculate payment due by **requesting payment**. Commuter will **provide payment information**tothe machine**.**The machine will**process the payment**on the basis of payment by cash or credit or debit card. If payment by card was selected by Commuter, another actor, Bank will participate in the activity by authorizing the payment. After payment is complete, ticket is dispensed to the Commuter. And the commuter will get the ticket.

Cash payment might result in some **change due**, so the change is dispensed to the Commuter in this case by the machine. Ticket vending machine will show some "Thank You" screen at the end of all the activity.



**Explanation:**

The above diagram depicts the activities in a ticket vending machine

The process starts with Requesting the trip info from the user which is then processed to calculate the payment.

The payment info is then provided to the commutor.

The process of payment is then started.

1. If the payment is by card ,authororization is done at the bank and ticket is dispensed. the process is terminated.

2. If the payment is in Cash, The Change due is returned if required and ticket is dispensed.The process is terminated.

**PRACTICAL NO. 2(C)**

**DRAW THE ACTIVITY DIAGRAM FOR THE GIVEN PROBLEM OF USE CASE.**

**DESCRIPTION OF THE EXAMINATION PAPER PREPARATION SUPPORT SYSTEM.**

**Use case name**: submit question

**Participant**: lecturer

**Entry conditions:**

1. The question is ready and stored in a file

2. The lecturer is assigned to the module

**Exit conditions:**

1.  The file is uploaded to the system

2.  The module leader is notified of the availability of the question

3. The event is logged by the system

**Flow of Events:**

1.  The lecturer logs into the system by entering his/her username and password;

2.  The system checks the username and password;

3.  The system displays the list of modules of which he/she is the lecturer, module leader

And/or internal examiner;

4.  The lecturer selects a module and his/her role in the module as a lecturer;

5. The system prompts the user to enter the file name and location on his/her computer, and Additional information if any.

**Exceptional conditions and alternative flow of events:**

When the username and password is not correct: 3.1: display error message, go back to step 1; When the lecturer is not listed on the module: 4.1: quit the system;

**Special requirements:**

1.  The file should be encrypted when transmitted from lecturer’s computer to the server

2.  The notification of success in uploading the file should be within 20 seconds

3.  The event should be recorded in a log file to contain the following information:

a)  Name of the lecturer,

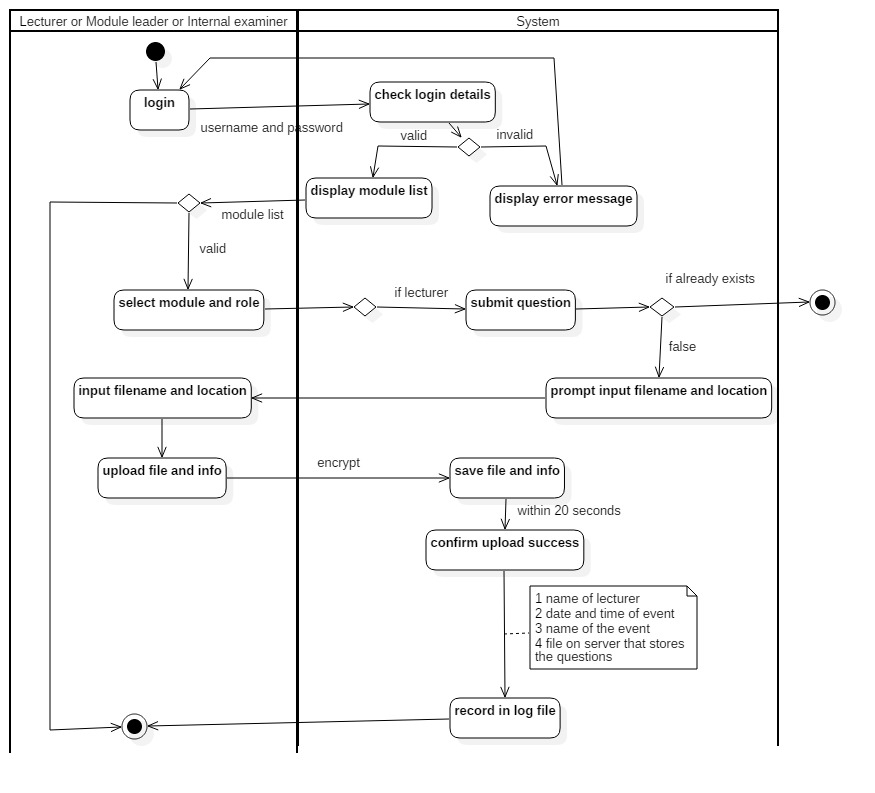
b)  Date and time of the event,

c)  The name of the event (upload exam question).

d)  The file on the server that stores the questions.

**.**

**0000000000000000000000000000000**



**Explanation:**

The diagram is the process to submit examination paper questions.

The process starts with the login that checks the details entered and displays module list on success.

If failed returns to login.

For a valid module list, there will be selection for a module or role .

If invalid, then terminated.

Further if the lecturer submits a question and it already exists, the session is terminated else the file name and location is asked.

On approval of file it is encrypted and saved.

Within 20 seconds the user is notified of confirmation with the details placed in a record file.

And the session is terminated.

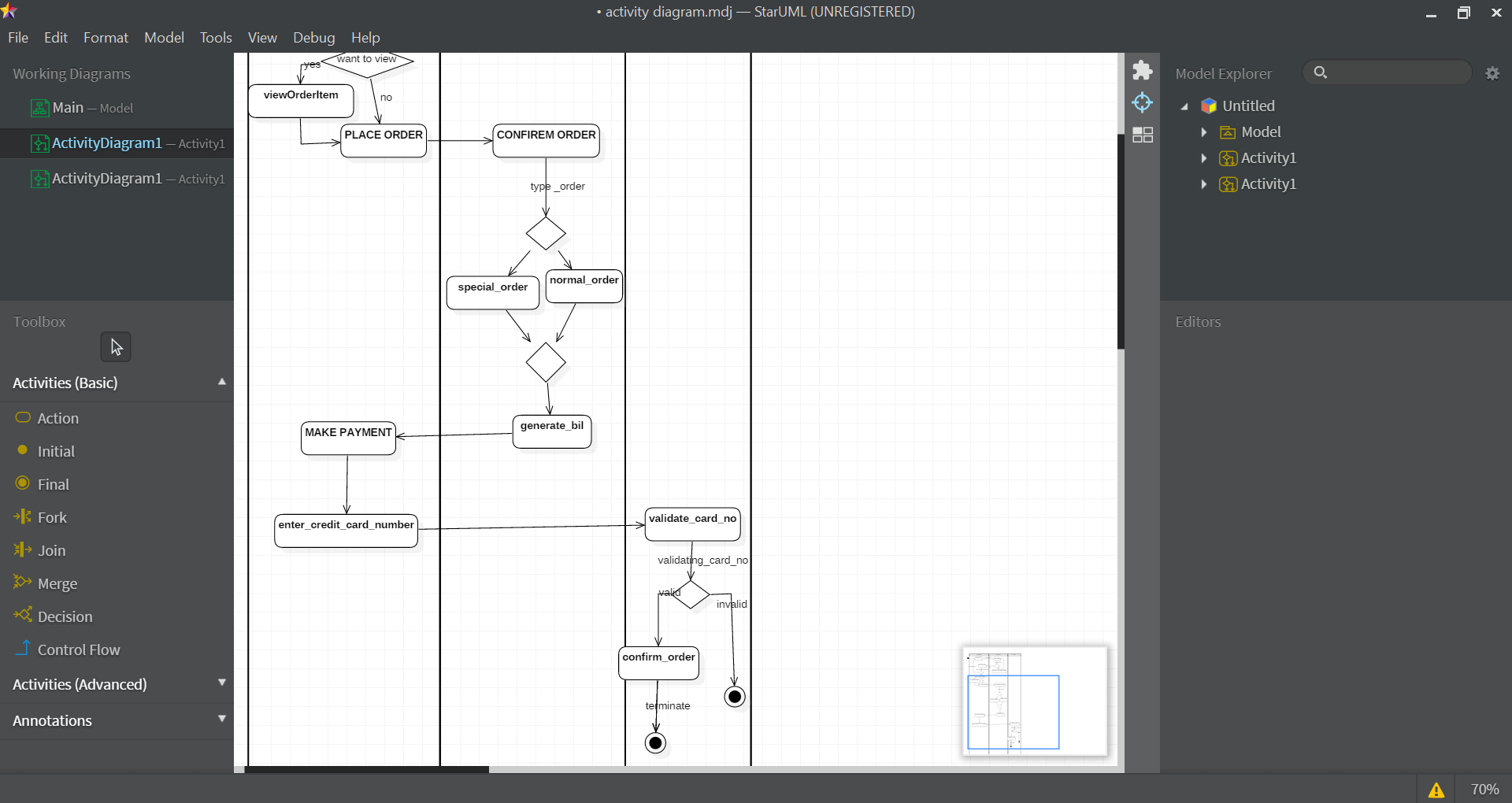
**PRACTICAL NO. 2(D)**

**DERIVE AN ACTIVITY DIAGRAM FROM THE CASE**

**GIVEN BELOW ON “ORDER PROCESSING SUBSYSTEM“**

Web **Customer**uses some web site to make purchases online. Where customer can search item, View item, **add item to the cart, place order and make payment.**For placing an order the customer first searches the required items from the system. As and when the customer finds the item available in the system he starts adding item to the chart. The System provides facility to the Customer to add any number of items to the chart. Customer can also view his shopping chart containing items.Ones the customer finishes his shopping he can place the order by requesting system to confirm the order.The system will then check whether the order is normal order or any special order and according to that the system will generate the bill and request for payment.

After getting the bill the customer can make payment.The bank will validate the credit card number. If the credit card number is valid the system will confirm the order. Otherwise the process will get terminated.



**Explanation:**

The activity diagram is for order processing system.

It starts with the customer searching for a specific product which is viewed by the

system.

Of the product is available it is added to the cart or an option to select another item is displayed.

After adding to the cart, customer can go to search another item or can view the details of selected item.

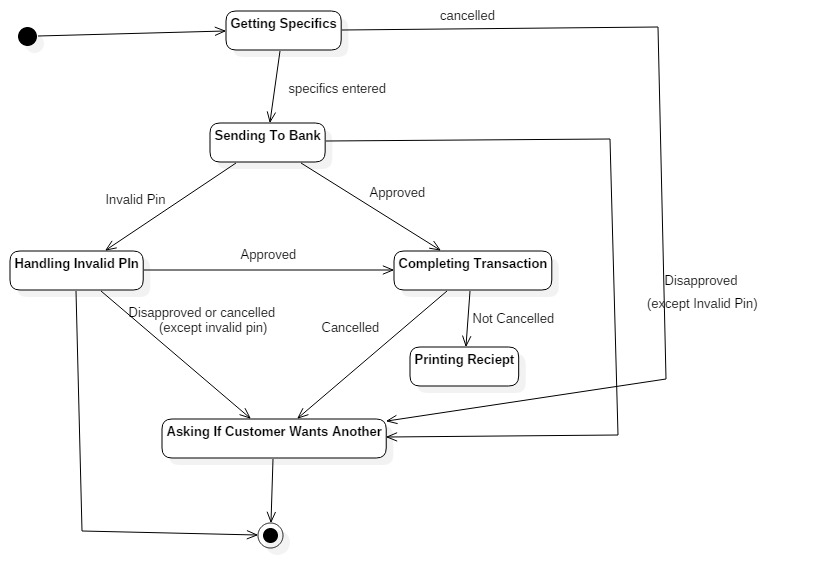
Finally the customer can place the order which is then confirmed by the system

Depending on the type of order, the bill is generated.

For payment, Customer enters card details which are validated.

If valid , Order is placed and terminates

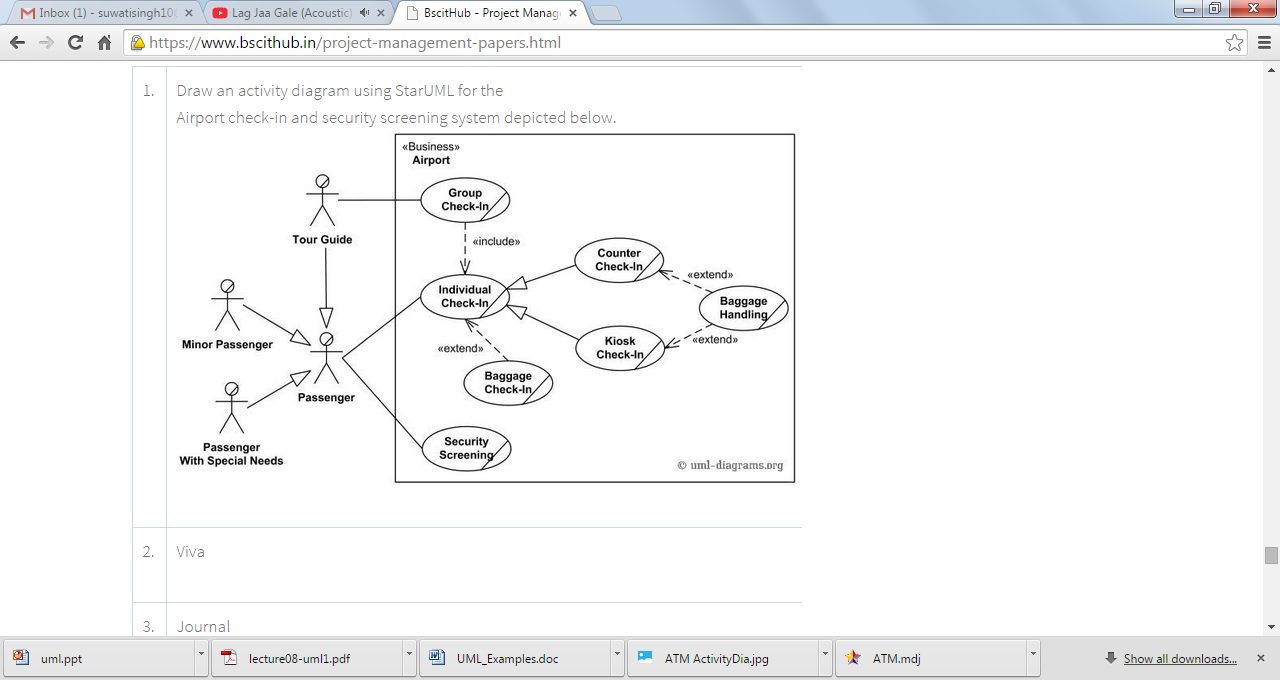
If invalid, terminates.

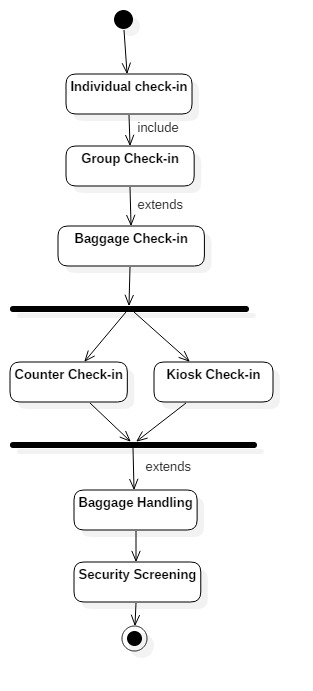
****

**PRACTICAL NO. 2(E)**

**Draw an activity diagram using StarUML for the**

**Airport check-in and security screening system depicted below.**

****



**Explanation:**

This activity diagram shows the process for airport check-in and security screening.

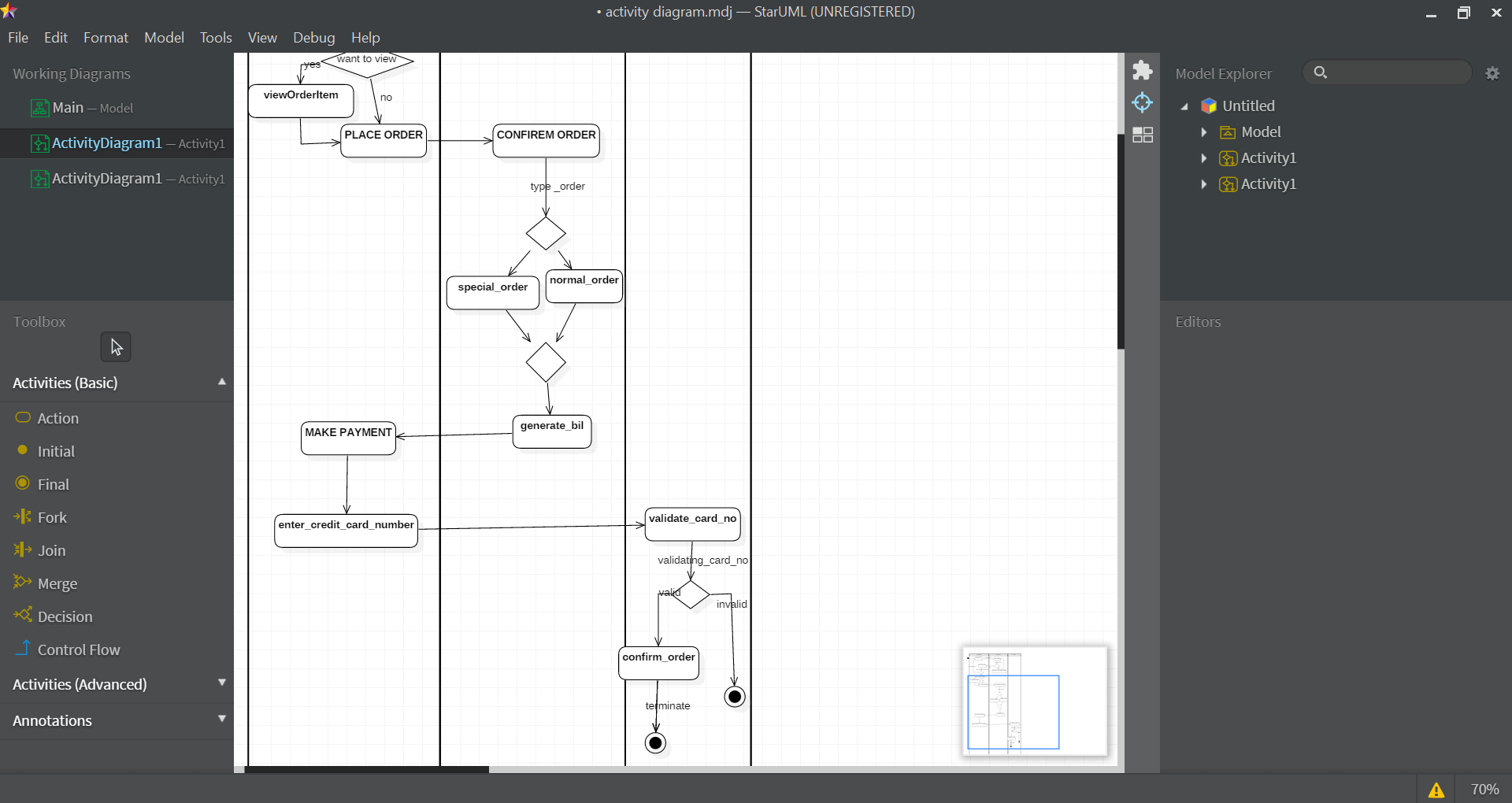
Check-in starts with either individual or group check-in which also includes baggage check-in.

There is a choice to either go for Counter check-in or kiosk check-in.

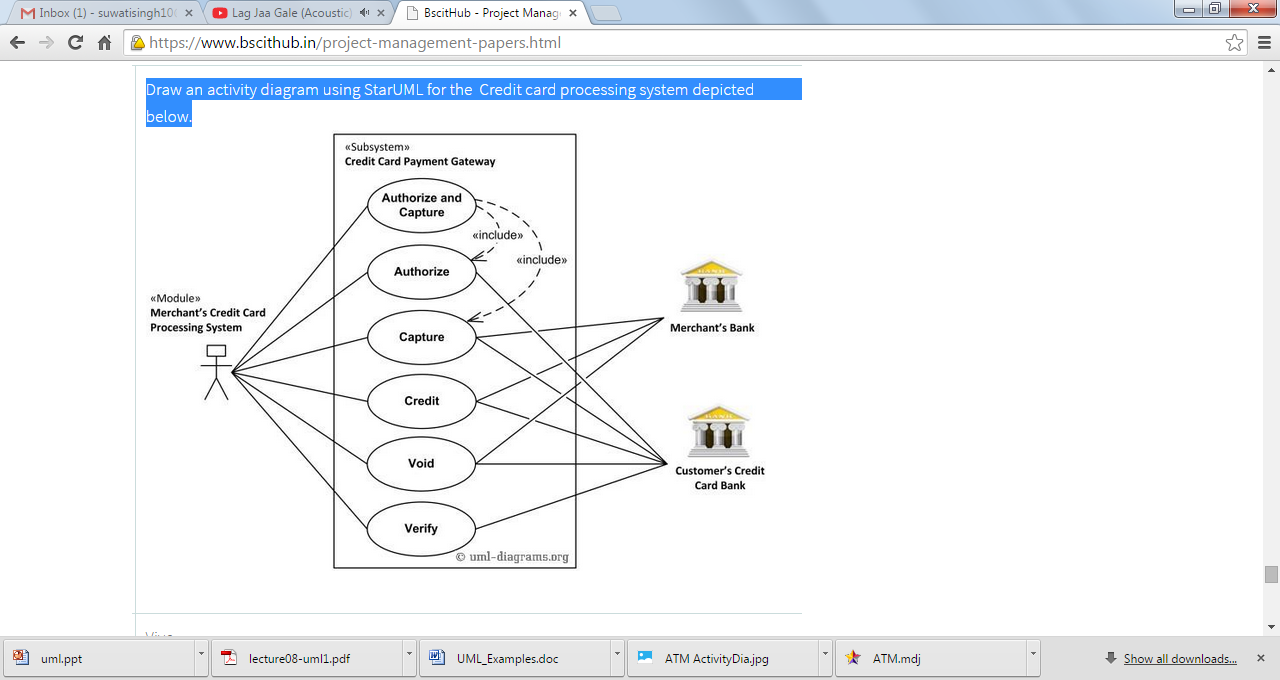
It further extends baggage handling followed by security screening.

Terminated

**PRACTICAL NO. 2(F)**



**Draw an activity diagram using StarUML for the  Credit card processing system depicted below.**

****

**PRACTICAL NO. 2(G)**

The author completes an online form that requests the user to input author name , Correspondence address , email and title of paper. The system validates this data and if it is correct , asks the author to submit the paper.

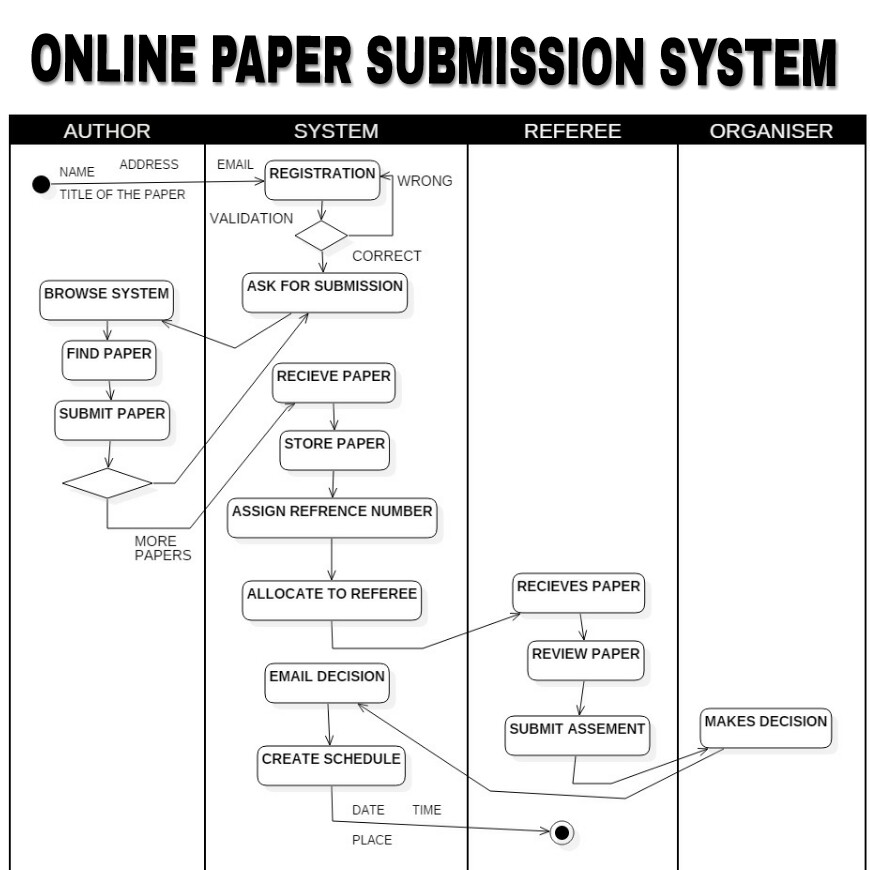
The author then browses to find the correct paper on their system and submit it.

Once received and stored , the system returns to the author a reference number for the paper.

Authors may submit as many papers as they like to be considered for acceptance to the conference up until the deadline date for submissions.

Papers are allocated to referees for assessment. They review each paper and submit to the system their decision.

Once the program organizer has agreed the decisions authors are informed by email. Accepted papers are then schedule to be delivered at a conference. This involves allocating a date , time and place for the presentation of the paper.



**EXPLANATION:**

Above is the activity diagram for online paper submission method.

It starts with taking the registration details.

If validates, ask for submission.

The author browses the system and finds paper.

If appropriate paper then submits else for more papers goes to receive papers.

It then stores the paper by assigning a reference number which is allocated to the referee.

The refree receives the paper, reviews and assesses it.

It is then given to organiser for the decision which is emailed and a schedule is created by the system.

The session is then terminated.

Practical No. 3

**USE CASE DIAGRAMS**

**ACTORS:**

An actor portrays any entity (or entities) that perform certain roles in a given system.

The different roles the actor represents are the actual business roles of users in a given system.

An actor in a use case diagram interacts with a use case.

**USE CASE:**

A use case in a use case diagram is a visual representation of a distinct business functionality in a system.

The key term here is "distinct business functionality”.

To choose a business process as a likely candidate for modelling as a use case,

you need to ensure that the business process is discrete in nature.

**SYSTEM BOUNDARY:**

A system boundary defines the scope of what a system will be.

A system cannot have infinite functionality.

So, it follows that use cases also need to have definitive limits defined.

A system boundary of a use case diagram defines the limits of the system.

The system boundary is shown as a rectangle spanning all the use cases in the system.

**System Boundary**

**Use Case**

**INCLUDE:**

When a use case is depicted as using the functionality of another use case in a diagram,

this relationship between the use cases is named as an include relationship.

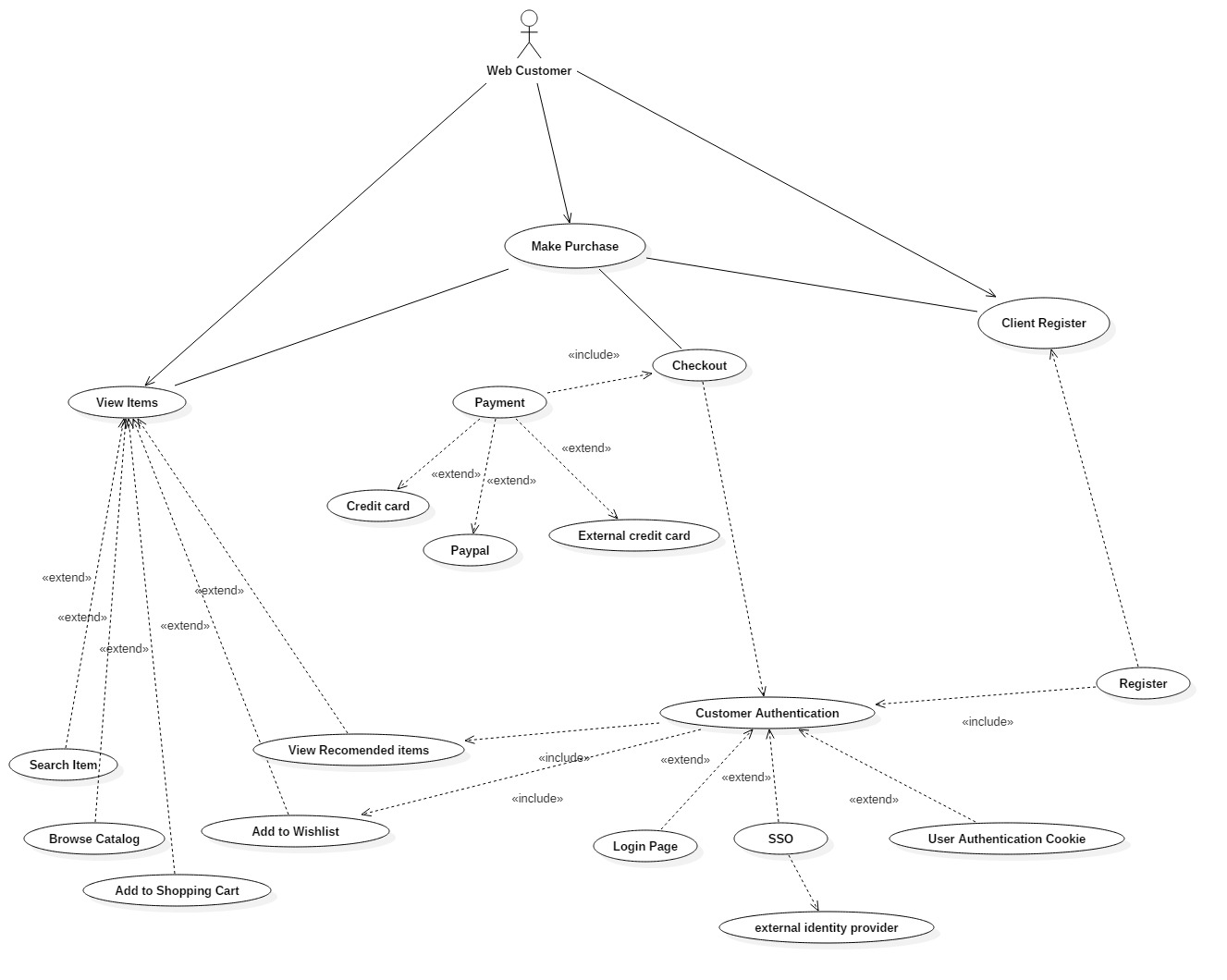
Literally speaking, in an include relationship, a use case includes the functionality

described in another use case as a part of its business process flow.

In this use case, Register <<includes>> Customer Authentication i.e. the Web Customer

cannot Register without providing his/her authentication.

It has to be done in order to complete the Registration.



**EXTEND:**

In an extend relationship between two use cases, the child use case adds

to the existing functionality and characteristics of the parent use case.

An extend relationship is depicted with a directed arrow having a dotted shaft,

similar to the include relationship.

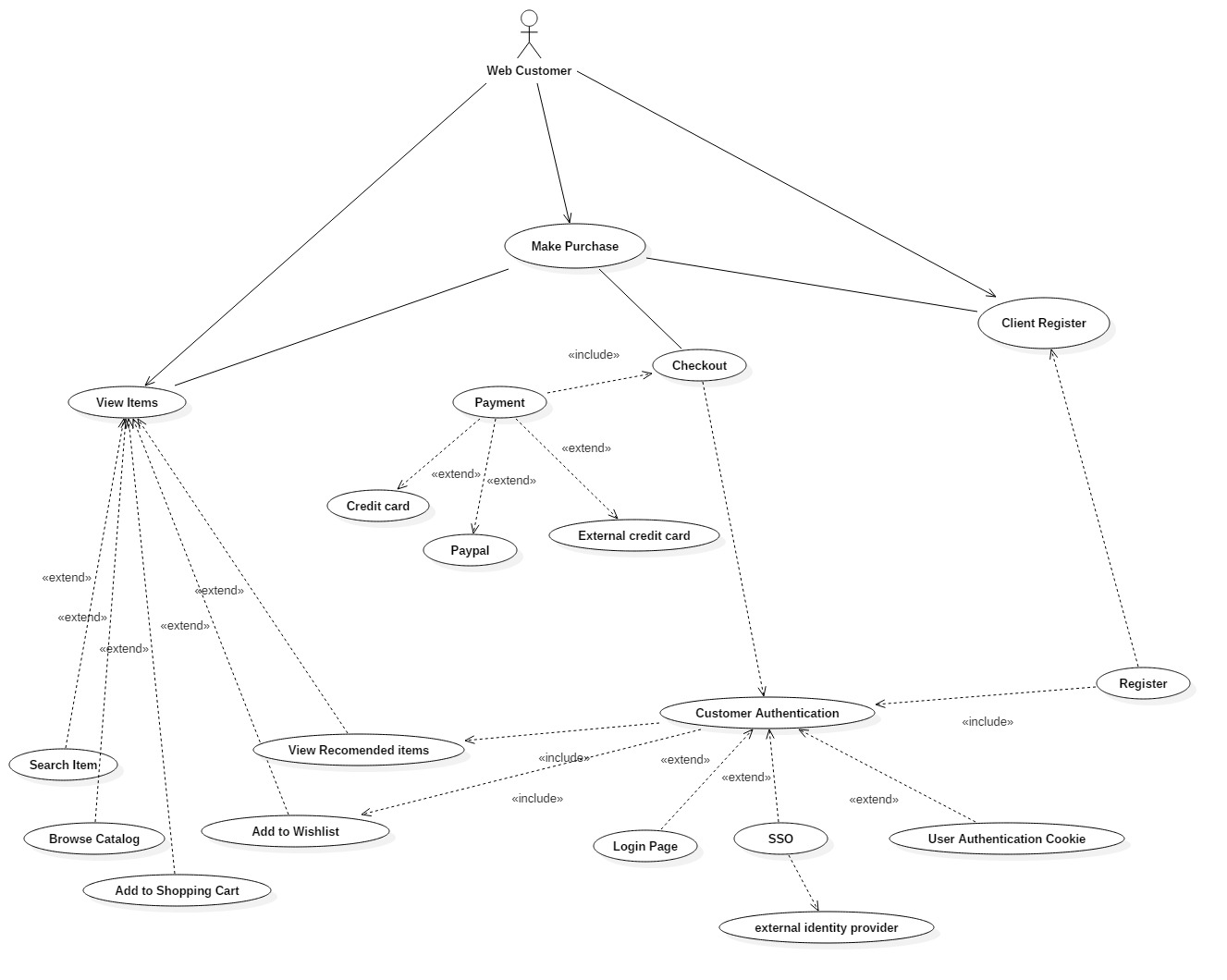
In this case, Payment <<extends>> Credit Card, SOS and External Credit Cards.

This means that Credit Card, SOS and external Credit Cards give extra functionality to

Payment methods.

Payment can be done using these methods and Payment

depends on these sources.

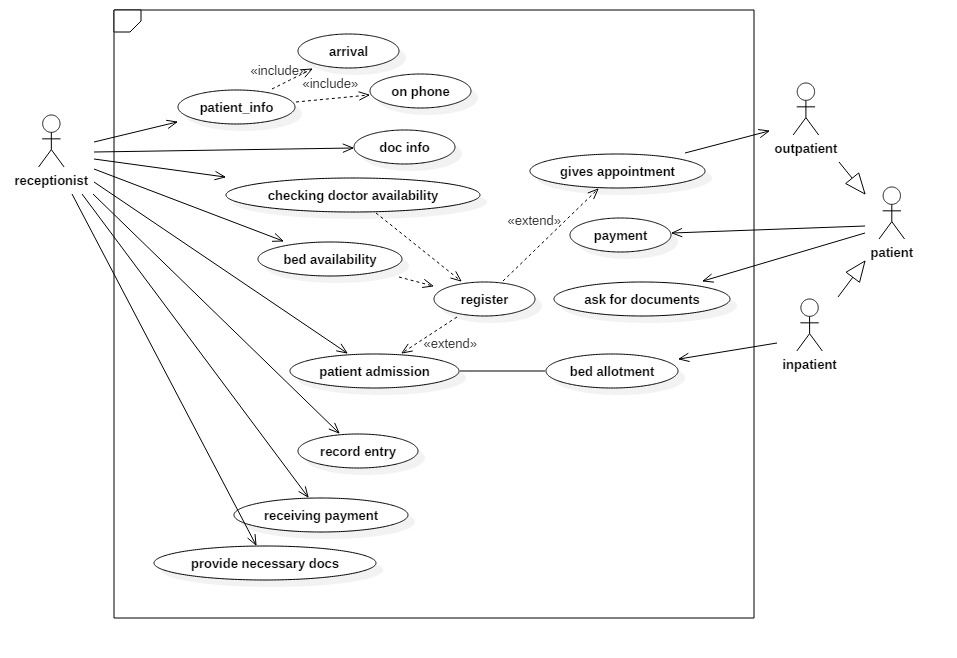


**PRACTICAL NO. 3(A)**

**DRAW A USE-CASE DIAGRAM USING STARUML FOR THE**

“**Hospital Reception Subsystem**” explained below:

 “Hospital Reception Subsystem” supports some of the many job duties of hospital receptionist. Receptionist schedule patient’s appointments with the doctor and also schedule patient hospital admission. If doctor is available and admission to the hospital is possible then receptionist can extend the service to patient registration by collecting the patient information on patient arrival or over the phone. Patient registration is an integral part of patient Hospital Administration use case. Hospital administration use case is further generalized into outpatient hospital admission and inpatient hospital administration. Note that for the patient that will stay in the hospital, he or she should have a bed allotted in a ward. Receptionists might also receive patient’s payments, record them in a database and provide receipts, file insurance claims and medical reports.



**Explanation:**

The above usecase depicts the functions of a receptionist and the patients in a hospital.

The primary duty of the receptionist is taking patient info either from the arrived patient or on phone.

Then a document is prepared for the same.

It then checks for the doctors availability for outpatients and doctors as well as bed availability for inpatient.

If availability is there in both cases, registration is done and compulsorily the appointment is fixed.

For the inpatient it admits the patient and allots a bed.

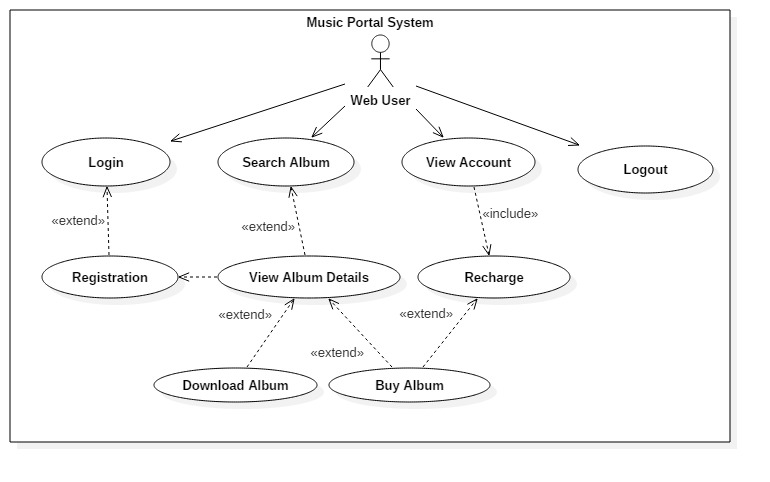
It records the entry, receives the payment and provide the documents necessary.

The outpatient and inpatient are a generalization of patients that can pay and request for documents.

**PRACTICAL NO. 3(B)**

**DRAW A USE-CASE DIAGRAM USING STARUML FOR THE “MUSIC PORTAL SYSTEM**”**DEPICTED BELOW.**

The following narration describes some of the use cases for “**Music Portal System**”. This system has **web user** as its main actor. The web user can perform first level uses cases namely **SearchAlbum, login, logout**and**ViewAccount**.**Registration** use case extends login i.e. if the user doesn’t have a login and wishes to create a new one, he or she can register to get a login. Moreover the **ViewAlbumDetail** use case is extending the SearchAlbum use case. Further the ViewAlbumDetail use case is extended by two more services viz**. DownloadAlbum**and**BuyAlbum**. Note that to download or to buy an album the user must be a registered member. Moreover the buyalbum and ViewAccount is further extended with R**echarge** use case**.**



**Explanation :**

The main actor for the above usecase is the Web user.

It includes login where registration is compulsory.

For downloading or buying a album, the user has to register and recharge.

Then the user can view album details and search for a particular album.

For recharge information, View account action can be performed.

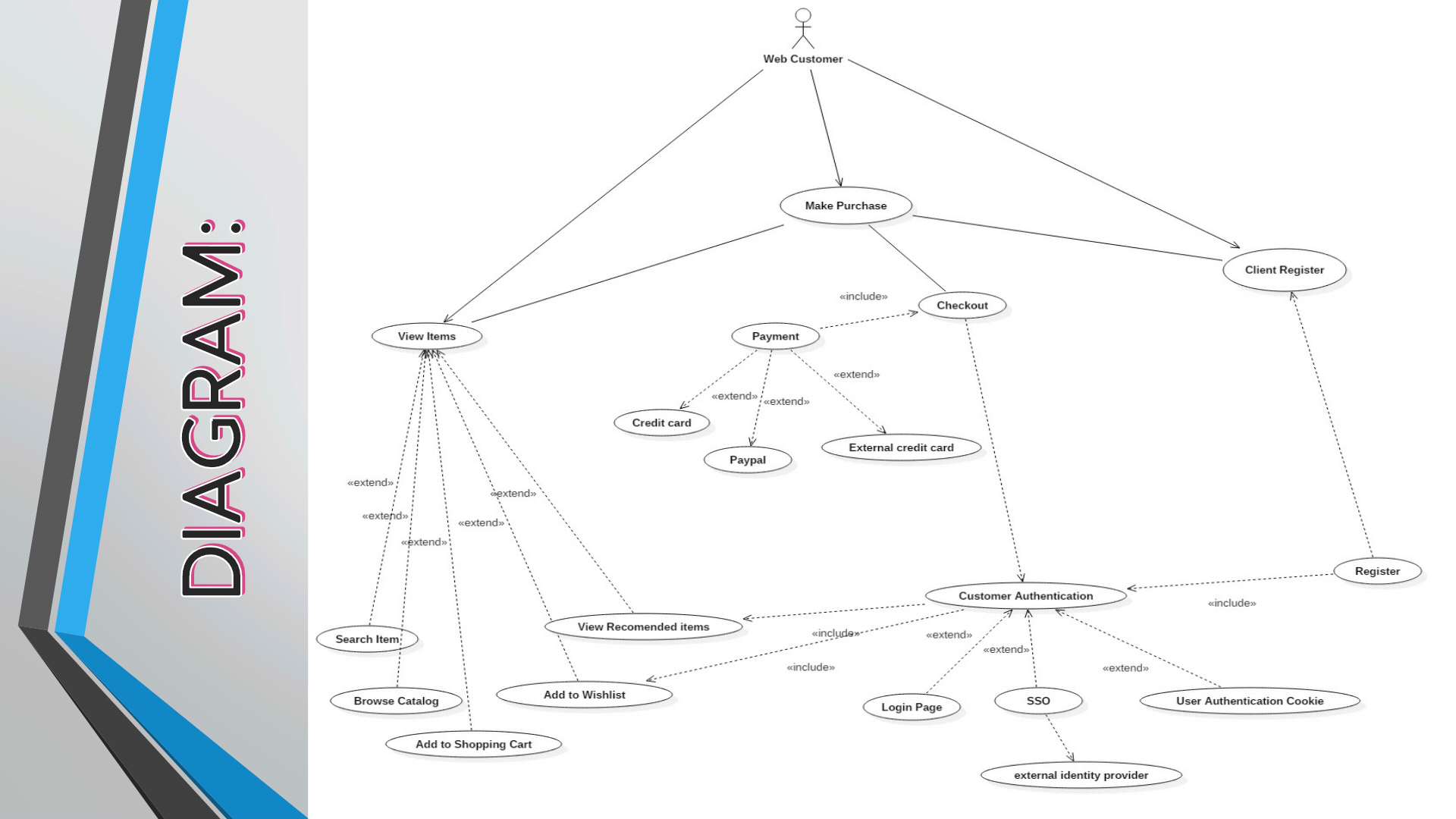
Logout to terminate.

**PRACTICAL NO. 3 (c)**

**Draw a use-case diagram using StarUML for the scenario given below**.

**Web Customer** **actor** uses some web site to make purchases online. Top level **use cases** are **View Items**,**Make Purchase** and **Client Register**. View Items use case could be used by customer as top level use case if customer only wants to find and see some products. This use case could also be used as a part of Make Purchase use case. Client Register use case allows customer to register on the web site, for example to get some coupons or be invited to private sales. Note that **Checkout** use case is **included use case** not available by itself - checkout is part of making purchase. Except for the **Web Customer** actor there are several other actors which will be described below with detailed use cases. **View Items** use case is **extended** by several optional use cases - customer may search for items, browse catalog, view items recommended for him/her, add items to shopping cart or wish list. All these use cases are extending use cases because they provide some optional functions allowing customer to find item. **Customer Authentication** use case is **included** in **View Recommended Items** and **Add to Wish**

**List** because both require customer to be authenticated. At the same time, item could be added to the shopping cart without user authentication.**Checkout** use case includes several required uses cases. Web customer should be authenticated. It could be done through user login page, user authentication cookie ("Remember me") or Single Sign-On (SSO). Web site authentication service is used in all these use cases, while SSO also requires participation of external identity provider. **Checkout** use case also includes**Payment** use case which could be done either by using credit card and external credit payment service or with PayPal.



**Explanation:**

The Above diagram shows the activities of a Web Customer.

The main functions include registration, Viewing items and making a purchase.

Registration includes customer authentication that extends to Login page, SSO that provides external identity and a user authentication code.

If the user is registered and authentified then the functions are extended to View the items.

They can search or browse for a particular item and add it in the wishlist or shopping cart.

If there is a Purchase,the checkout includes payment that can be bY various methods

**PRACTICAL NO. 3(d)**

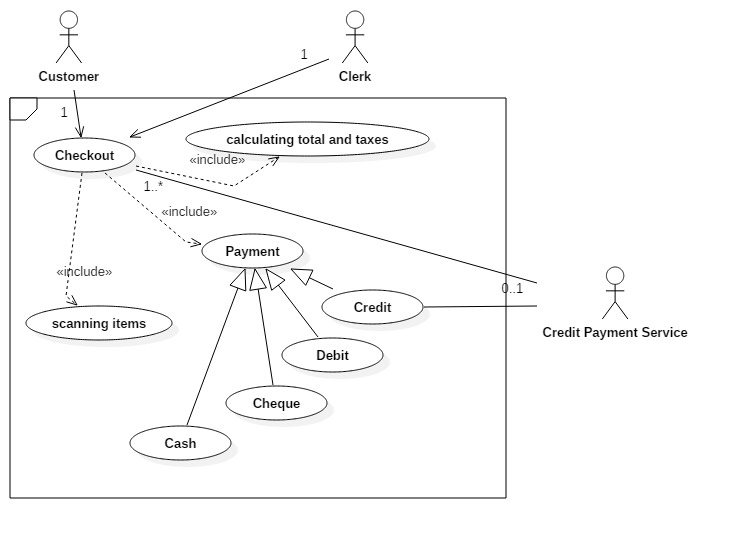
**Draw a use-case diagram using StarUML for the retail Point-ofsalesystem depicted below.**

A retail POS system typically includes a computer, monitor, keyboard, barcode scanners, weight scale, receipt printer, credit card processing system, etc. and POS terminal software.

Checkout **use case** involves Customer, Clerk and Credit Payment Service **actors** and **includes** scanning items, calculating total and taxes, payment use cases.

Checkout use case requires Customer actor, hence the 1 multiplicity of Customer. Clerk can only participate in a single Checkout use case. Credit Payment Service can participate with many Checkout use cases at the same time. Checkout use case may not need Credit Payment Service (for example, if payment is in cash), thus the 0..1 multiplicity.

Checkout use case is an example of a large and complex use case split into several use cases each describing some logical unit of behavior. Note, that including use case becomes incomplete by itself and requires the included use cases to be complete. Payment use case is represented using**generalization** relationship. It means that only one specific type of payment is accepted - either by cash, or by credit, debit, or with check. An alternative to such representation could be to use **include** relationship so that not just single but several forms of payment could be accepted from the same client during checkout.

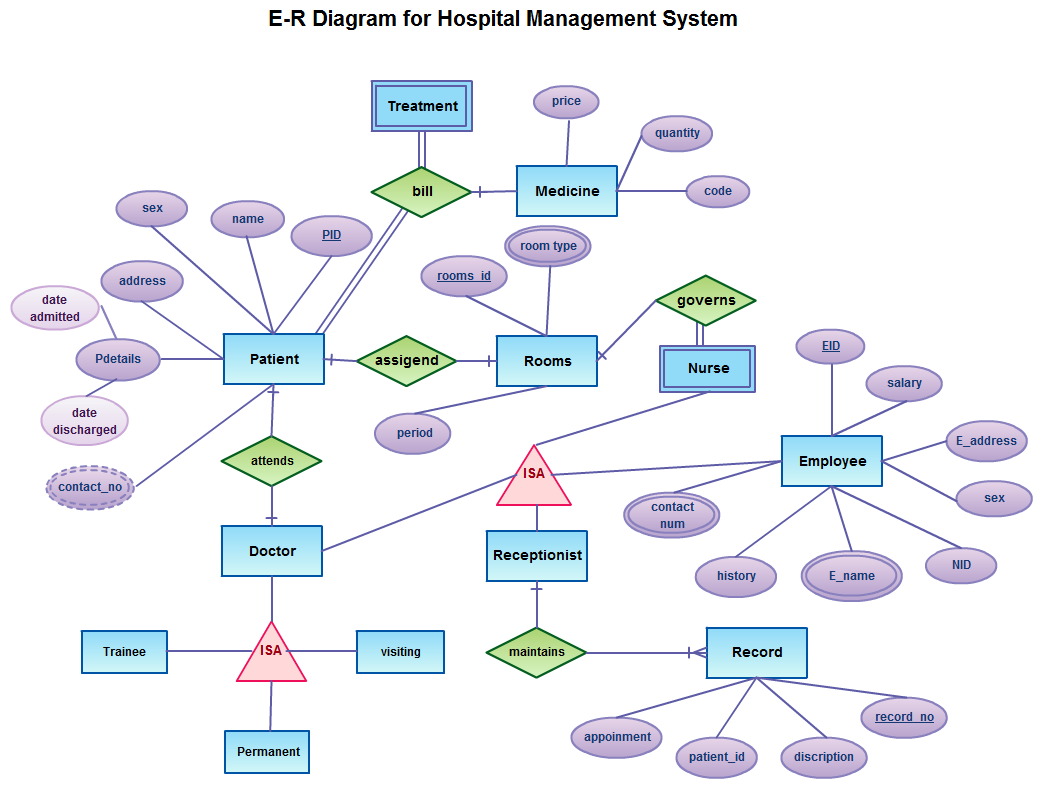


**Explanation:**

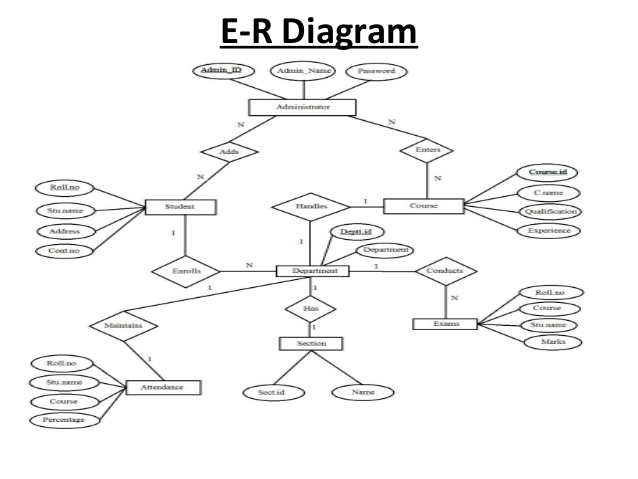
The retail point of sale system in this usecase has 3 actors that are clerk, Customer and Credit payment service provider.

The checkout mechanism by the actors includes calculating taxes, scanning items, and payment.

The payment is futhergeneralised as Credit, Debit, Cheque or Cash



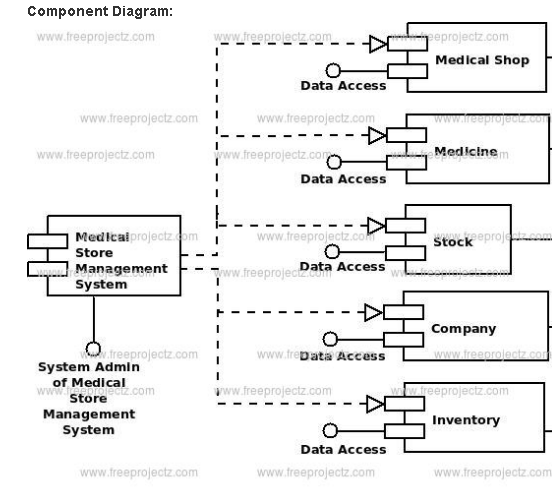
**Student management ERD**



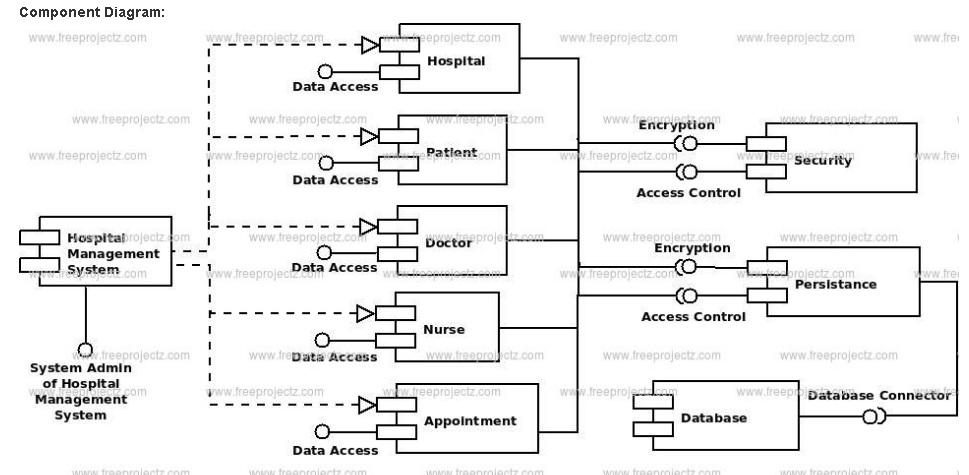
|  |
| --- |
| Suppose you are given the following requirements for a simple database for the National Hockey League (NHL):   1.  The NHL has many teams, each **team** has a name, a city, a coach, a captain, and a set of players, 2.  Each **player** belongs to only one team, each player has a name, a position (such as left wing or goalie), a skill level, and a set of **injury records**, a team captain is also a player, 3.  A game is played between two teams (referred to as host\_team and guest\_team) and has a date (such as May 11th, 1999) and a score (such as 4 to 2).   **Construct a clean and concise ER diagram for the NHL database.** |

|  |
| --- |
| Create an ERD using the following requirements:   1.  An INVOICE is written by a SALESREP. Each sales representative can write many invoices, but each invoice is written by a single sales representative. 2.  The INVOICE is written for a single CUSTOMER. However, each customer can have many invoices. 3.  An INVOICE can include many detail lines (LINE), each of which describes one product bought by the customer. 4.  The product information is stored in a PRODUCT entity. 5.  The product’s vendor information is found in a VENDOR entity. |

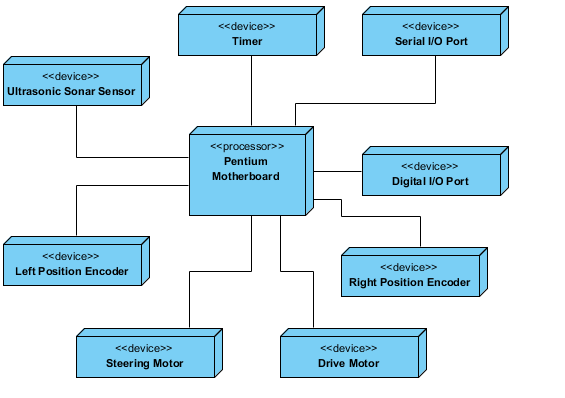
|  |
| --- |
| Draw an **ER diagram** for the given scenario of buying an article.  Entities: Article, Source, Order, Copyright Agency, Country, Buyer  Attributes:  Article: Title(PK),authors, pdf file, fee  Source: Title(PK), publisher,issue,date, pages  Order: Order number(PK),total payment, date, tax status  Copyright Agency: name, address  Country: copyright from, taxrate  Buyer: name, address, email(PK), billing info  Following relationships are to be set:  a. Article is published in source. Many articles can be published in many sources.  b. Buyer places order. He can place zero or more orders.  c. Orders deliver articles. One article can be delivered in many orders and one order can deliver many articles as well.  d. Source pays fees to Copyright agency for every article published.  e. Every country has a single source of publication. |

****

**Component diagram for medical store management**

****

**Component diagram for hospital management system**

****

**Deployment diagram for embedded system**