R16

Unified Modeling Language Lab

Department of CSE

Vasireddy Venkatadri Institute of Technology

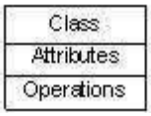
Nambur(V), Peda kakani(M), Guntur Dt.- 522508

UML Building Blocks

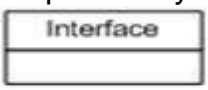
1. Things

Structural things define the static part of the model. They represent the physical and conceptual elements. Following are the brief descriptions of the structural things.

1. **Class** – Class represents a set of objects having similar responsibilities.



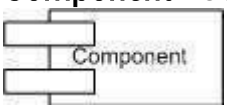
2. **Interface** – Interface defines a set of operations, which specify the responsibility of a class.



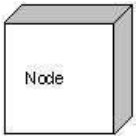
3. **Collaboration** –Collaboration defines an interaction between elements.

4. **Use case** –Use case represents a set of actions performed by a system for a specific goal.

5. **Component** –Component describes the physical part of a system.

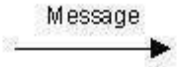


6. **Node** – A node can be defined as a physical element that exists at runtime.



Behavioral Things consists of the dynamic parts of UML models

1. **Interaction** – Interaction is defined as a behavior that consists of a group of messages exchanged among elements to accomplish a specific task.



2. **State machine** – State machine is useful when the state of an object in its life cycle is important. It defines the sequence of states an object goes through in response to events. Events are external factors responsible for state change

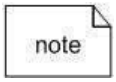
Grouping things can be defined as a mechanism to group elements of a UML model

together.

1. **Package –** Package is the only one grouping thing available for gathering structural and behavioral things.

Annotational things can be defined as a mechanism to capture remarks, descriptions, and comments of UML model elements.

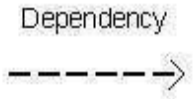
1. **Note** - It is the only one Annotational thing available. A note is used to render comments, constraints, etc. of an UML element.



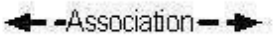
2 Relationship

Relationship is another most important building block of UML. It shows how the elements are associated with each other and this association describes the functionality of an application.

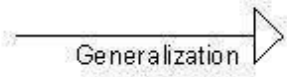
1. **Dependency** is a relationship between two things in which change in one element also affects the other.



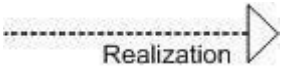
2. **Association** is basically a set of links that connects the elements of a UML model. It also describes how many objects are taking part in that relationship.



3. **Generalization** can be defined as a relationship which connects a specialized element with a generalized element. It basically describes the inheritance relationship in the world ofobjects.



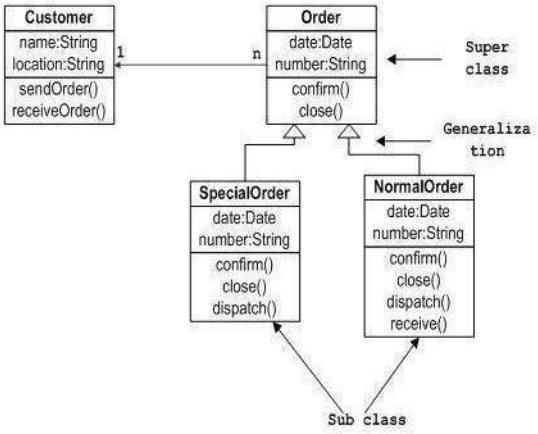
4. **Realization** can be defined as a relationship in which two elements are connected. One element describes some responsibility, which is not implemented and the other one implements them. This relationship exists in case of interfaces.



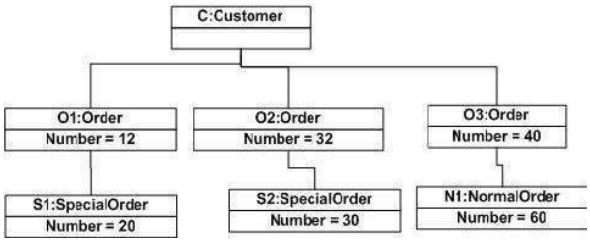
3 UML Diagrams

Structural Diagrams represent the static aspect of the system. These static aspects represent those parts of a diagram, which forms the main structure and are therefore stable.

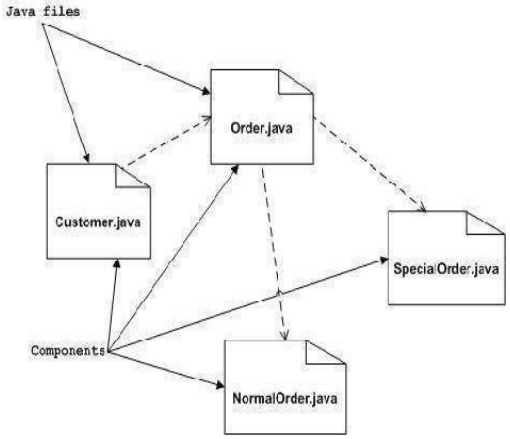
1. Class diagrams consist of classes, interfaces, associations, and collaboration.


Customer
name:String
location:String
sendOrder()
receiveOrder()
1
n
Order
date:Date
number:String
confirm()
close()
Super class
Generaliza tion
SpecialOrder
date:Date
number:String
confirm()
close()
dispatch()
NormalOrder
date:Date
number:String
confirm()
close()
dispatch()
receive()
Sub class

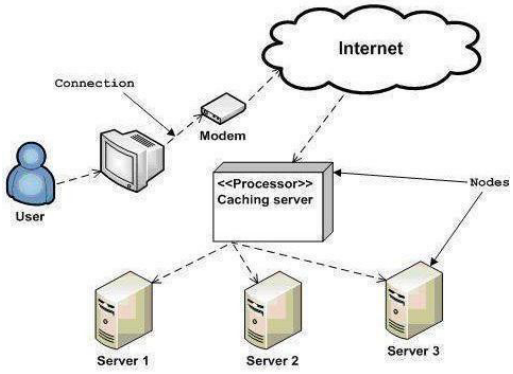
2. Object diagrams are a set of objects and their relationship is just like class diagrams.



3. Component diagrams represent a set of components and their relationships.

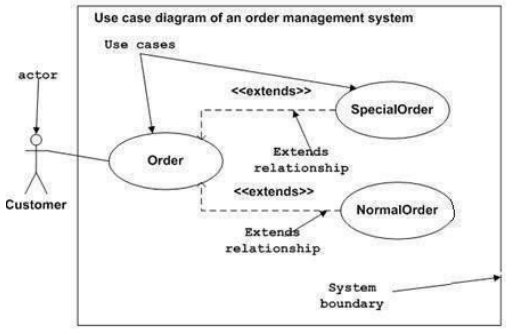


4. Deployment diagrams are a set of nodes and their relationships. These nodes are physical entities where the components are deployed.

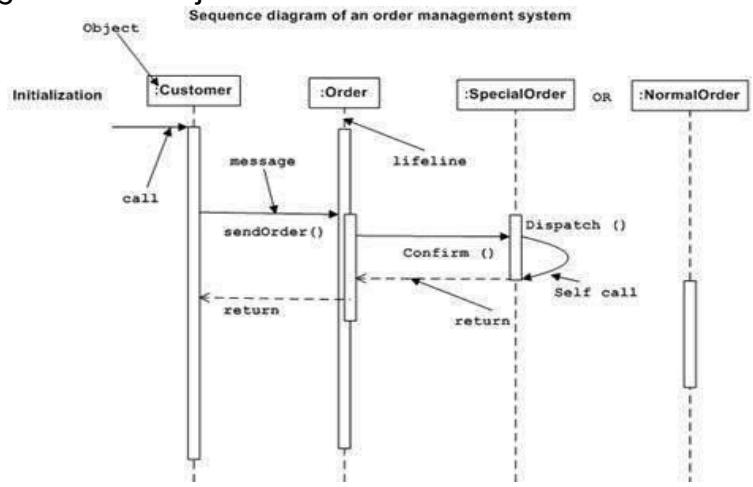


Behavioral Diagrams basically capture the dynamic aspect of a system. Dynamic aspect can be further described as the changing/moving parts of a system.

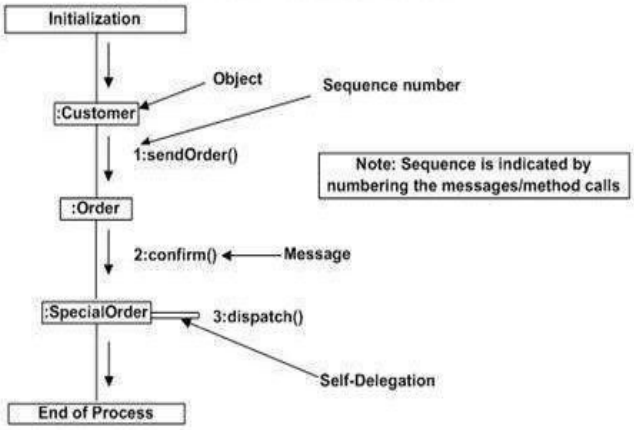
5. Use case Diagrams are a set of use cases, actors, and their relationships. They represent the use case view of a system.



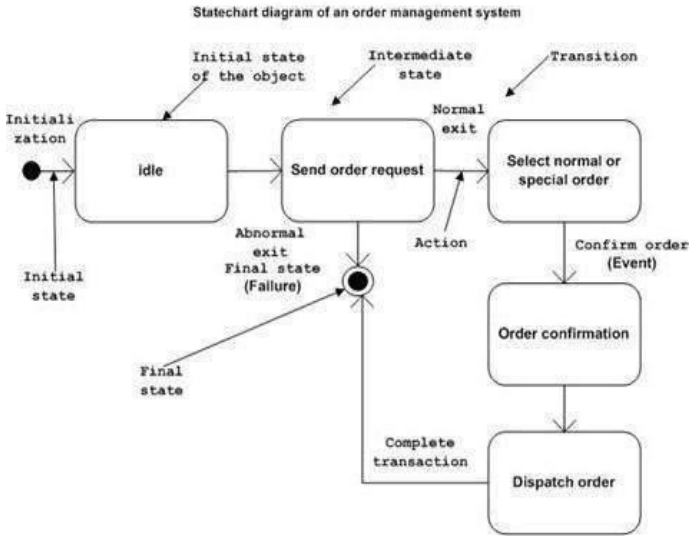
6. Sequence diagram deals with some sequences, which are the sequence of messages flowing from one object to another.



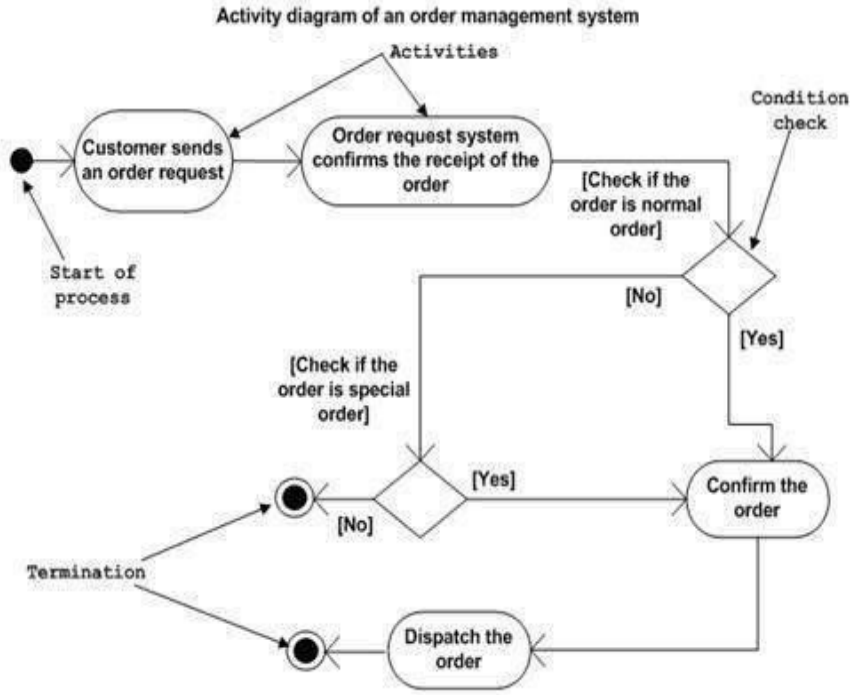
7. Collaboration diagram represents the structural organization of a system and the messages sent/received. Structural organization consists of objects and links.



8. State-chart Diagram describes the state change of a class, interface, etc. State chart diagram is used to visualize the reaction of a system by internal/external factors.



9. Activity diagram describes the flow of control in a system. It consists of activities and links.



DESIGN PATTERNS

Design Patterns: Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the problem

Design Patterns are the best solutions for the re-occurring problems in the application programming environment.

Design Patterns are very much about choices and alternatives. Frequently more than one design pattern could be used to solve a given problem. E.g. Template Method vs Strategy, Strategy vs State.

Design Pattern Elements

1. PatternName

Handle used to describe the design problem Increases vocabulary

Eases design discussions

Evaluation without implementation details

2. Problem

Describes when to apply a pattern

May include conditions for the pattern to be applicable Symptoms of an inflexible design or limitation

3. Solution

Describes elements for the design

Includes relationships, responsibilities, and collaborations Does not describe concrete designs or implementations

A pattern is more of a template

4. Consequences

Results and Trade Offs

Critical for design pattern evaluation Often space and time trade offs Language strengths and limitations

(Broken into benefits and drawbacks for this discussion)

Design patterns can be subjective.

One person's pattern may be another person's primitive building block. The focus of the selected design patterns are:

Object and class communication

Customized to solve a general design problem Solution is context specific

Describing design patterns

➤ Graphical notations, while important and useful, aren't sufficient. They capture the end product of the design process as relationships between classes and objects.
By using a consistent format we describe the design pattern . Each pattern is divided into sections according to the following template.

Pattern Name and Classification:

➤ it conveys the essence of the pattern succinctly good name is vital, because it will become part of design vocabulary.
Intent: What does the design pattern do?

➤ What is its rational and intend?
➤ What particular design issue or problem does it address?

Also Known As: Other well-known names for the pattern, if any. Motivation:

➤ A scenario that illustrates a design problem and how the class and object structures in the pattern solve the problem.
➤ The scenario will help understand the more abstract description of the pattern that follows.

Applicability:

• Applicability: What are the situations in which the design patterns can be applied?
• What is example of the poor designs that the pattern can address?
• How can recognize situations?
• Structure: Graphical representation of the classes in the pattern using a notation based on the object Modeling Technique (OMT).
• Participants: The classes and/or objects participating in the design pattern and their responsibilities.

Structure:

➤ Graphical representation of the classes in the pattern using a notation based on the object Modeling Technique(OMT).

Participants:

The classes and/or objects participating in the design pattern and their responsibilities.

Collaborations:

- How the participants collaborate to carry out their responsibilities.

Consequences:

- How does the pattern support its objectives?
- What are the trade-offs and result of using the pattern?
- What aspect of the system structure does it let vary independently?

Implementation:

- What pitfalls, hints, or techniques should be aware of when implementing the pattern?
- Are there language-specific issues?

Sample Code:

- Code fragments that illustrate how might implement the pattern in c++ or Smalltalk.

Related Patterns:

What design patterns are closely related to this one? What are the imp differences? With Which other patterns should this one be used?

The Catalog of Design Pattern Abstract Factory:

Provide an interface for creating families of related or dependent objects without specifying their concrete classes.

Adaptor: Convert the interface of a class into another interface clients expect.

Bridge:

Decouple an abstraction from its implementation so that two can vary independently.

Builder:

Separates the construction of the complex object from its representation so that the same constriction process can create different representations.

Chain of Responsibility:

Avoid coupling the sender of a request to it's receiver by giving more than one object a chance to handle the request. Chain the receiving objects and pass the request along the chain until objects handles it.

Command:

Encapsulate a request as an object, thereby letting parameterize clients with different

request, queue or log requests, and support undoable operations.

Composite:

Compose objects into three objects to represent part-whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly.

Decorator:

Attach additional responsibilities to an object dynamically. Decorators provide a flexible alternative to sub classing for extending functionality.

Façade:

Provide a unified interface to a set of interfaces in a subsystem's Facade defines a higher-level interface that makes the subsystem easier to use.

Factory Method:

Defines an interface for creating an object, but let subclasses decide which class to instantiate. Factory Method lets a class defer instantiation to subclasses.

Flyweight:

Use sharing to support large numbers of fine-grained objects efficiently.

Interpreter:

Given a language, defining a representation of its grammar along with an interpreter that uses the representation to interpret sentences in the language.

Iterator:

Provide a way to access the element of an aggregate object sequentially without exposing its underlying representation.

Mediator:

Define an object that encapsulate how a set of objects interact. Mediator promotes loose coupling by keeping objects from referring to each other explicitly, and let’s you very their interaction independently.

Memento:

Without violating encapsulation, capture and externalize an object’s internal state so that object can be restored to this state later.

Observer:

Define a one-to-many dependency between objects so that when one object changes state, all it’s dependents are notified and updated automatically.

Prototype:

Specify the kinds of objects to create using a prototypical instance, and create new objects by copying this prototype.

Proxy: Provide a surrogate or placeholder for another object to control access to it.

Singleton:

Ensure a class has only one instance, and provide a point of access to it.

State:

Allow an object to alter its behavior when its internal state changes. the object will appear to change its class.

Strategy:

Define a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithm vary independently from clients that use it.

Template Method:

Define the Skelton of an operation, deferring some steps to subclasses. Template method subclasses redefine certain steps of an algorithm without changing the algorithms structure.

Visitor:

Represent an operation to be performed on the elements of an object structure. Visitor lets you define a new operation without changing the classes of the elements on which it operates.

Organizing the Catalog

Two criteria

Purpose: what a pattern does

Creational: concern the process of object creation

Structural: the composition of classes or objects

Behavioral: characterize the ways in which classes or objects interact and distribute responsibility

Scope: whether the pattern applies primarily to *classes* or to *objects*

Class patterns deal with relationships between classes and their subclasses. These relationships are established through inheritance, so they are static—fixed at compile-time. Object patterns deal with object relationships, which can be changed at run-time and are more dynamic. Almost all patterns use inheritance to some extent. So the only patterns labeled "class patterns" are those that focus on class relationships. Note that most patterns are in the Object scope.

Creational class patterns defer some part of object creation to subclasses, while

Creational object patterns defer it to another object. The Structural class patterns use inheritance to compose classes, while the Structural object patterns describe ways to assemble objects. The Behavioral class patterns use inheritance to describe algorithms and flow of control, whereas the Behavioral object patterns describe how a group of objects cooperate to perform a task that no single object can carry out alone.

There are other ways to organize the patterns. Some patterns are often used together. For example, Composite is often used with Iterator or Visitor. Some patterns are alternatives: Prototype is often an alternative to Abstract Factory. Some patterns result in similar designs even though the patterns have different intents. For example, the structure diagrams of Composite and Decorator are similar.

Figure 1.1 depicts these relationships graphically. Clearly there are many ways to organize design patterns. Having multiple ways of thinking about patterns will deepen your insight into what they do, how they compare, and when to apply them.

Organizing of catalog

		Purpose		
		Creational	Structural	Behavioral
Scope	Class	Factory Method	Adapter (class)	Interpreter Template Method
	Object	Abstract Factory Builder Prototype Singleton	Adapter (object) Bridge Composite Decorator Facade Flyweight Proxy	Chain of Responsibility Command Iterator Mediator Memento Observer State Strategy Visitor

Case Studies

Problem Statements

1. Library Management System

- It is a support system for a library.
- The library lends books and magazines to borrowers, who are registered in the system, as are the books and magazines.
- The library handles the purchase of new titles for the library. Popular titles are bought in multiple copies. Old books and magazines are removed when they are out of date or in poor condition.
- The librarian is an employee of the library who interacts with the customers (borrowers) and whose work is supported by the system.
- A borrower can reserve a book or magazine that is not currently available in the library, so that when it's returned or purchased by the library, that borrower is notified. The reservation is canceled when the borrower checks out the book or magazine or through an explicit canceling procedure.
- The librarian can easily create, update, and delete information about the titles, borrowers, loans, and reservations in the system.
- The system can run on all popular Web browser platforms (Point-Of-Sale Terminal Internet Explorer 5.1+, Netscape 4.0+, and so on).
- The system is easy to extend with new functionality.

2. Point of Sale System

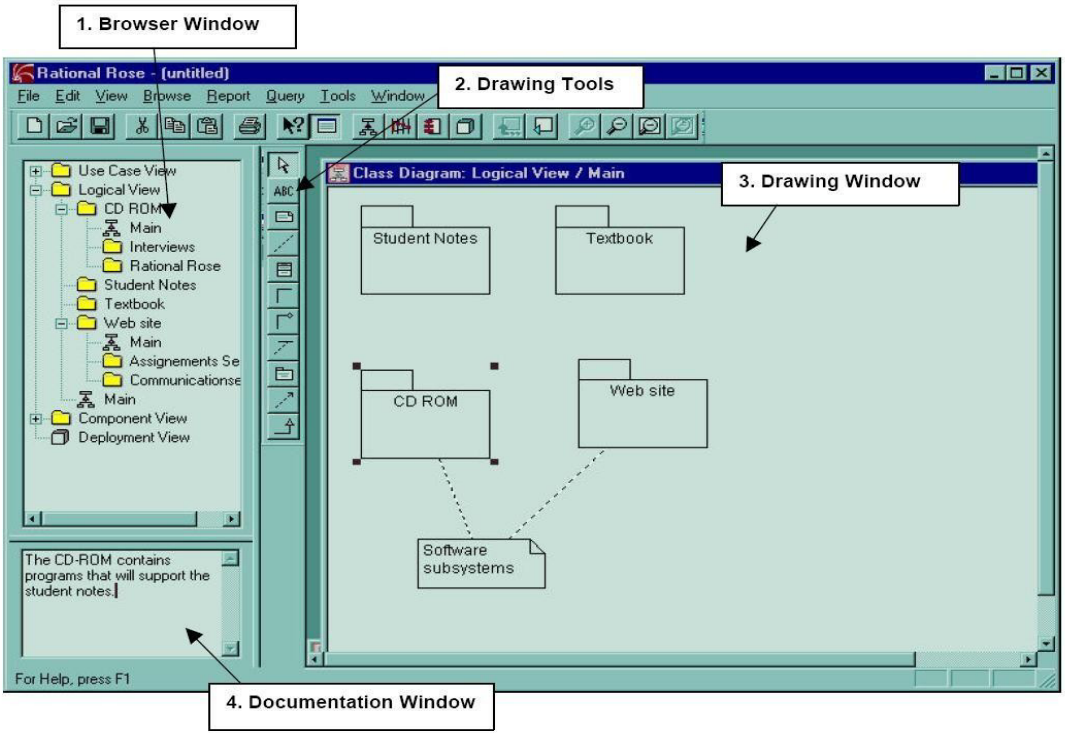
- POS system is a computerized application used to record sales and handle payments; it is typically used in a retail store. It includes hardware components such as a computer and bar code scanner, and software to run the system.
- It interfaces to various service applications, such as a third-party tax calculator and inventory control. These systems must be relatively fault-tolerant; that is, even if remote services are temporarily unavailable, they must still be capable of capturing sales and handling at least cash payments.
- POS system increasingly must support multiple and varied client-side terminals and interfaces. These include a thin-client Web browser terminal, a regular personal computer with something like a Java Swing graphical user interface, touch screen input, wireless PDAs, and so forth.
- Furthermore, we are creating a commercial POS system that we will sell to different clients with disparate needs in terms of business rule

processing.

- Each client will desire a unique set of logic to execute at certain predictable points in scenarios of using the system, such as when a new sale is initiated or when a new line item is added.

Week 1: Familiarization with Rational Rose or Umbrello

Rational Rose is an object-oriented Unified Modeling Language (UML) software design tool intended for visual modeling and component construction of enterprise-level software applications. Two popular features of Rational Rose are its ability to provide iterative development and round-trip engineering.



1. Browser

Browser contains a list of all the modeling elements in the current model. It contains a tree view of all the elements in the current Rose model. This presents a hierarchical view of the analysis and design model, including all the diagrams and all the individual elements that make up a diagram.

2. Documentation Window

Documentation Window may be used to create, review, and modify for any selected modeling element. If nothing is selected, the window displays the string "No selection". The contents of the Documentation Window will change as different modeling elements are selected. It is strongly recommended that each element added to a diagram have documentation to accompany it. To add documentation, right click on the element, select specification, and fill in the documentation field

The documentation will then be shown in the documentation window each time the mouse is clicked on the element.

3. Diagram Windows

Diagram windows allow creating and modifying graphical views of the current model. Each icon on a diagram represents a modeling element. This is the place where the diagram is actually created.

4. Drawing Tools

This tool presents a set of icons that indicate the different elements that can be added to a diagram. The elements that can be used will change, depending on the type of diagram being created. Different diagram types have different sets of icons.

Views in Rational Rose

There are four views for a model created in Rational Rose, each representing the system from a different point of view.

1. Use Case View

The use case view contains the diagrams used in analysis, and all the elements that contain these diagrams. This view looks at actors and use cases along with their interactions. The diagrams in this view are use case diagrams, sequence diagrams and collaboration diagrams. The purpose of the use case view is to visualize what the system must do, without dealing with the specifics of how it will be implemented. More recent versions of Rational Rose also allow for additional documentation in the form of word-processed documents and/or URLs to Web- based materials.

Packages in the use case view can contain actors, use cases, sequence diagrams, and/or collaboration diagrams. To create a package:

- Right-click on the parent modeling element (use case view or another package) to make the shortcut menu visible.
- Select the New: Package menu command. This will add a new package called New Package to the browser.
- While the new package is still selected, enter its name.

Once a package is created, modeling elements may be moved to the package.

To move a modeling element to a package:

- Click to select the modeling element to be relocated.
- Drag the modeling element to the package.

2. Logical View

The logical view contains the diagrams used in object design. The diagrams in this view are class diagrams and state transition diagrams. It offers a detailed view of how the system visualized in the use case view will be implemented. The basic element in this view is the class, which includes an outline of its attributes and operations. This directly corresponds to a class created inour chosen implementation language. From the logical view, skeletal code can be generated for implementation into a computer language. More recent versions of Rational Rose not only can generate skeletal code for Visual C++, Visual Java, or Visual BASIC, but also reverse engineer programs created in these languages into Rational Rose models. This allows existing components to be included in documented models, if there is access to the source code. In addition, changes that need to be made during implementation can be reflected in the documentation of the design model.

3. Component View

The component view is a step up from the logical view and contains diagrams used in system design. This view contains only component diagram. This includes information about the code libraries, executable programs, runtime libraries, and other software components that comprise

the completed systems. Components can be pre-existing; for example, a Windows program in Visual C++ will utilize Microsoft Foundation Class to provide the framework for the Windows interface. Components that do not exist and need to be created by the developers will have to be designed in the logical view.

Rose automatically creates one component diagram called Main. To create an additional component diagram:

- Right-click on the owning package (either the component view itself or a user created package) to make the shortcut menu visible.
- Select the New: Component Diagram menu command. This will place a new component diagram called New Diagram in the browser.
- While the new diagram is still selected, enter its name.

Rose will automatically add the new diagram to the browser.

- To open a component diagram:
  - Double-click on the diagram in the browser.
- To create a component:
  - Click to select the package specification icon from the toolbar.
  - Click on the component diagram to place the component.
  - While the component is still selected, enter its name. Rose will automatically add the new component to the browser.
- To create a dependency relationship:
  - Click to select the dependency icon from the toolbar.
  - Click on the package or component representing the client.
  - Drag dependency arrow to the package or component representing the supplier.

4. Deployment View

The deployment view illustrates how the completed system will be physically deployed. This view contains only deployment diagram.

Week 2, 3 & 4: For each case study

a) Identify and analyze events

Event is an action or occurrence recognized by software that may be handled by the software. Computer events can be generated or triggered by the system, by the user or in other ways. A source of events includes the user, who may interact with the software by way of, for example, keystrokes on the keyboard. Another source is a hardware device such as a timer. Software can also trigger its own set of events into the event loop, e.g. to communicate the completion of a task. Software that changes its behavior in response to events is said to be event- driven, often with the goal of being interactive. The first step in use-case modeling is to define actors who interact with the system and their events.

1.Library Management System:

The actors in the library are identified as follows:

- **Librarian actor**: is user of the system and have management capability to add borrowers, titles, and items.

  ### Events of Librarian

  1. Login to the system
  2. Manage (create, update, delete)Borrowers
  3. Manage (create, update, delete) Titles (Books or Magazines)
  4. Manage (create, update, delete)Items
  5. Search Titles
  6. Browse Titles
  7. Assume Identity of Borrower

- **Borrowers actor**: is also user of the system and Borrower is people who check out and reserve books and magazines. Occasionally a librarian or another library can be a borrower.

  ### Events of Borrower

  1. Register in the system
  2. Login to the system
  3. Make reservation of Titles (Books or Magazines)
  4. Remove reservation
  5. Search Titles
  6. Browse Titles
  7. Checkout Titles
  8. Return Titles

- **Master Librarian actor**: this role is capable of managing the librarians as well. It is possible to add a title to the system before the library has a copy (an item), to enable borrowers to make reservations.

  ### Events of Master Librarian

  1. Login to the system
  2. Manage Librarians

3. Add Titles (Books or Magazines)

2. Point-Of-Sale Terminal:

There are three types of actors can be identified as follows:

- **Primary actor:** has user goals fulfilled through using services. Finduser goals to drive the use cases. Ex: Cashier, System Administrator

  ### Events of Cashier

  1. Process Sale
  2. Handle Returns
  3. Process Rental
  4. Cash In

  ### Events of System Administrator

  1. Manage Security
  2. Manage Users

- **Supporting actor**: provides a service (e.g., information). Often a computer system, but could be an organization or person. The purpose is to clarify external interfaces and protocols. Ex: Sales Activity System

  ### Events of Sales Activity System

  1. Analyze Activity

- **Offstage actor**: has an interest in the behavior of the use case, but is not primary or supporting. Ex: Payment Authorization Service like Tax Calculator, Accounting System, HR System

  ### Events of Tax Calculator

  1. Process Sale
  2. Process Rental

  ### Events of Accounting System

  1. Process Sale
  2. Process Rental

  ### Events of HR System

  1. Cash In

b) Identify Use-cases

Next step in use-case modeling is to define Use-cases of system that describe what the library system provides in terms of functionality to the actors and the functional requirements of the system. While identifying the use cases, we must read and analyze all specifications, as well as discuss the system with potential users of the system and all stakeholders.

1.Library Management System:

The use cases in the library management system are as follows:

- Login
- Search
- Browse
- Make Reservation
- Remove Reservation
- Checkout Item

- Return Item
- Manage Titles
- Manage Items
- Manage Borrowers
- Manage Librarians
- Assume Identity of Borrower

2. Point-Of-Sale Terminal:

The use cases in the POS terminal system are as follows:

- Login
- Process Sale
- Handle Returns
- Process Rental
- Cash In
- Analyze Activity
- Manage Security
- Manage Users

c) Develop event table

An event table is a catalogue of use cases that lists events in rows and key pieces of information about each event in columns. It includes row and columns representing events and their details, respectively. Each column in the event table represents a section as follows:

- **Trigger**: a signal that tells the system that an event has occurred, either the arrival of data needing processing or a point in time.
- **Source**: an external agent or actor that supplies data to the system
- **Response**: an output, produced by the system, that goes to a destination
- **Destination**: an external agent or actor that receives data from the system

1. Library Management System:

Event / Use case	Trigger	Source	Response	Destination
Login to the system	Login	Registered User	Authentication of the user	Registered User
Search Titles	Search	Registered User	Showing searched details about Titles using keywords	Registered User
Browse Titles	Browse	Registered User	Showing browsed details about Titles	Registered User

Manage Borrowers	Create / Update/ Delete Borrower	Librarian	Insertion / Updation / Deletion of Borrower details	Librarian
Manage Titles	Create / Update/ Delete Titles	Librarian	Insertion / Updation / Deletion of Title details	Librarian
Manage Items	Create / Update/ Delete Items	Librarian	Insertion / Updation / Deletion of Item details	Librarian
Assume Identity of Borrower	Identify	Librarian	Getting identify of the borrower	Librarian
Make reservation of Titles	Make Reservation	Borrower	Adding reservation record for particular borrower	Borrower
Remove reservation	Remove Reservation	Borrower	Removing reservation record for particular borrower	Borrower
Checkout Titles	Checkout	Borrower	Showing reservation status of particular borrower	Borrower
Return Titles	Return	Borrower	Updation of loan details	Borrower
Manage Librarians	Create / Update/ Delete Librarians	Master Librarian	Insertion / Updation / Deletion of Librarian details	Master Librarian
Add Titles	Add Title	Master Librarian	Insertion of new Title details	Master Librarian
Enabling borrowers to make reservation	Enable reservatio n	Master Librarian	Enabling of make reservation option for borrower	Master Librarian

2. Point-Of-Sale Terminal:

Event / Use case	Trigger	Source	Resp o nse	Destination
Process Sales	Process Sale	Tax Calculator / Accounting System	Generation of sales / inventory bills	Cashier
Handle Returns	Handle Returns	Cashier	Recording of returned items by the Cashier	Cashier
Process Rental	Process Rental	Tax Calculator / Accounting System	Generation third party service bills	Cashier
Cash In	Cash In	HR System	Receiving ash from HR System	Cashier
Manage Security	Create /Update/ Delete Security	System Administrator	Insertion / Updation /Deletion of Security related information	System Administrator
Manage Users	Create /Update/ Delete Users	System Administrator	Insertion / Updation /Deletion of Users details	System Administrator
Analyze Activity	Analyze Activity	Sales Activity System	Showing of analysis of sales activities	Sales Activity System

d) Identify & analyze domain classes

Objects that represent domain entities are called entities or domain objects. The classes from which domain objects can be instantiated are called Domain classes. An early domain model is useful to establish a core set of classes that represent the things in the problem space of the system to be built.

1.Library Management System:

The use cases in the library management system are as follows:

Borrower
Title
Book Title

Magazine Title
Item Reservation
Loan

2. Point-Of-Sale Terminal:

The domain classes in the POS terminal system are as follows:

Sales Line Item
Item
Sale
Store
Payment
Register

e) Represent use cases and a domain class diagram using Rational Rose

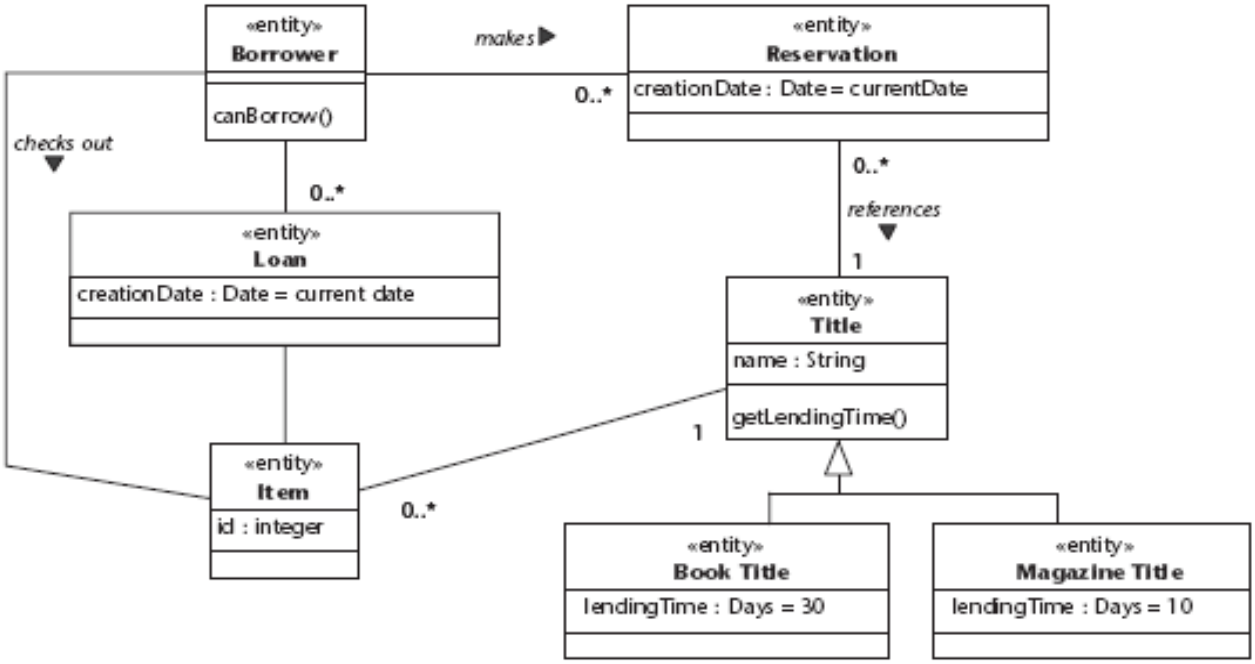
Design class diagrams (DCD) show software class definitions. Classes are shown with their simple attributes and methods listed. Some attributes are depicted using associations (relationships) rather than actually being listed in the class block. These associated attributes refer to complex objects which should also be shown in the diagram. The collaboration diagram indicates methods to be contained in a class with methods posted as relationships.

1. Library Management System:

Representation of domain class

Domain Class	Use cases	Other Operations
Borrower	Register, Login, Search, Browse, MakeReservation,RemoveReservation, Checkout Item, Return Titles	Get borrowed titles, Get borrowed items, Get reservation details
Title	Search Title, Browse Title	Get title quantity
Book Title	Search Title, Browse Title	Get book quantity
Magazine Title	Search Title, Browse Title	Get magazine quantity
Item	Search Item, Browse Item, Checkout Item	Get item quantity
Reservation	Make Reservation, Remove Reservation	Get reservation details
Loan	Checkout Item, Return Titles	Get loan details

Domain class diagram for the library management system:

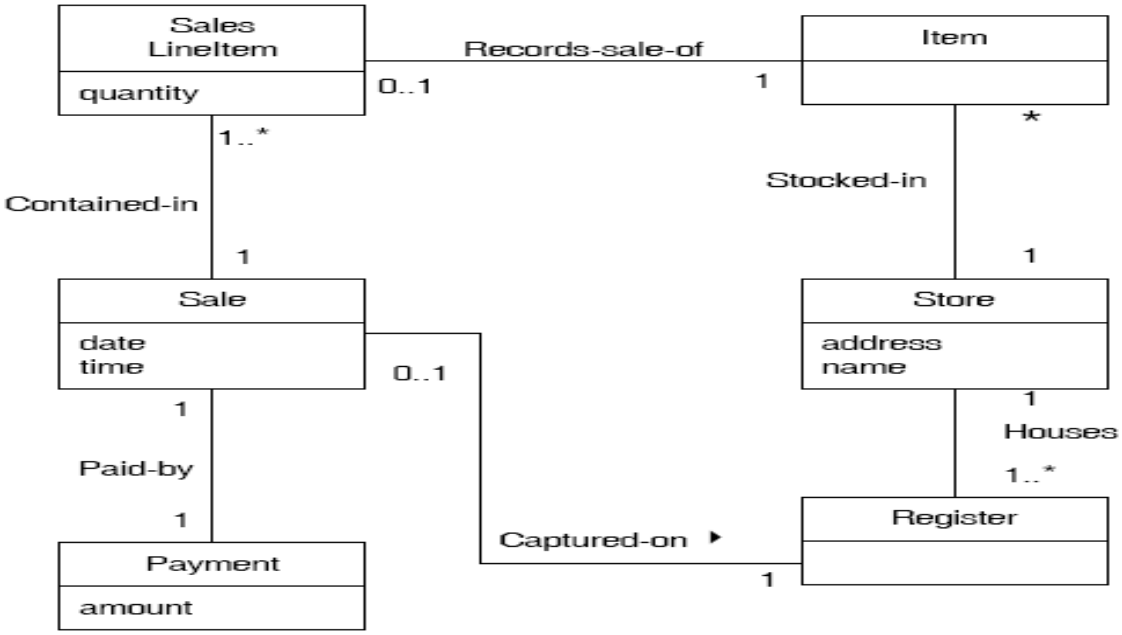


2. Point-Of-Sale Terminal:

Representation of domain class with use cases:

Domain Class	Use cases	Other Operations
Sales Line Item	Process Sale, Handle Returns	Get sales details, Get item details
Item	Handle Returns	Get item quantity, Get item details
Sale	Cash In, Process Sale	Get sales details
Store	Handle Items	Get store details
Payment	Process Rental	Generate bill, Print bill
Register	Process Sale	Get sales details, Get store details

Domain class diagram for POS terminal system



f) Develop CRUD matrix to represent relationships between use cases and problem domain classes

A CRUD matrix is a table showing the functions in an application that affects parts of a database. The CRUD Matrix is an excellent technique to identify the Tables in a Database which are used in any User interaction. CURD means:

- Create - INSERT - to store new data
- Read - SELECT - to retrieve data
- Update - UPDATE - to change or modify data.
- Delete - DELETE - delete or remove data

CRUD Matrix identifies the Tables involved in any CRUD operation. The analysis helps to identify any Tables which are not used, and any

1. Library Management System:

CRUD matrix for library management system:

Event / Use case	Borrower	Title	Item	Reservation	Loan
Login to the system	R				
Search Titles		R			
Browse Titles		R			
Manage Borrowers	CUD				
ManageTitles		CUD			
Manage Items			CUD		
Assume Identity of Borrower	R				
Make reservation of Titles		R		CU	
Remove reservation				D	
Checkout Titles		R			
Return Titles		U		U	U
Manage Librarians					
Add Titles		C			
Enabling borrowers to make reservation	U				

2. Point-Of-Sale Terminal:

CRUD matrix for POS terminal system:

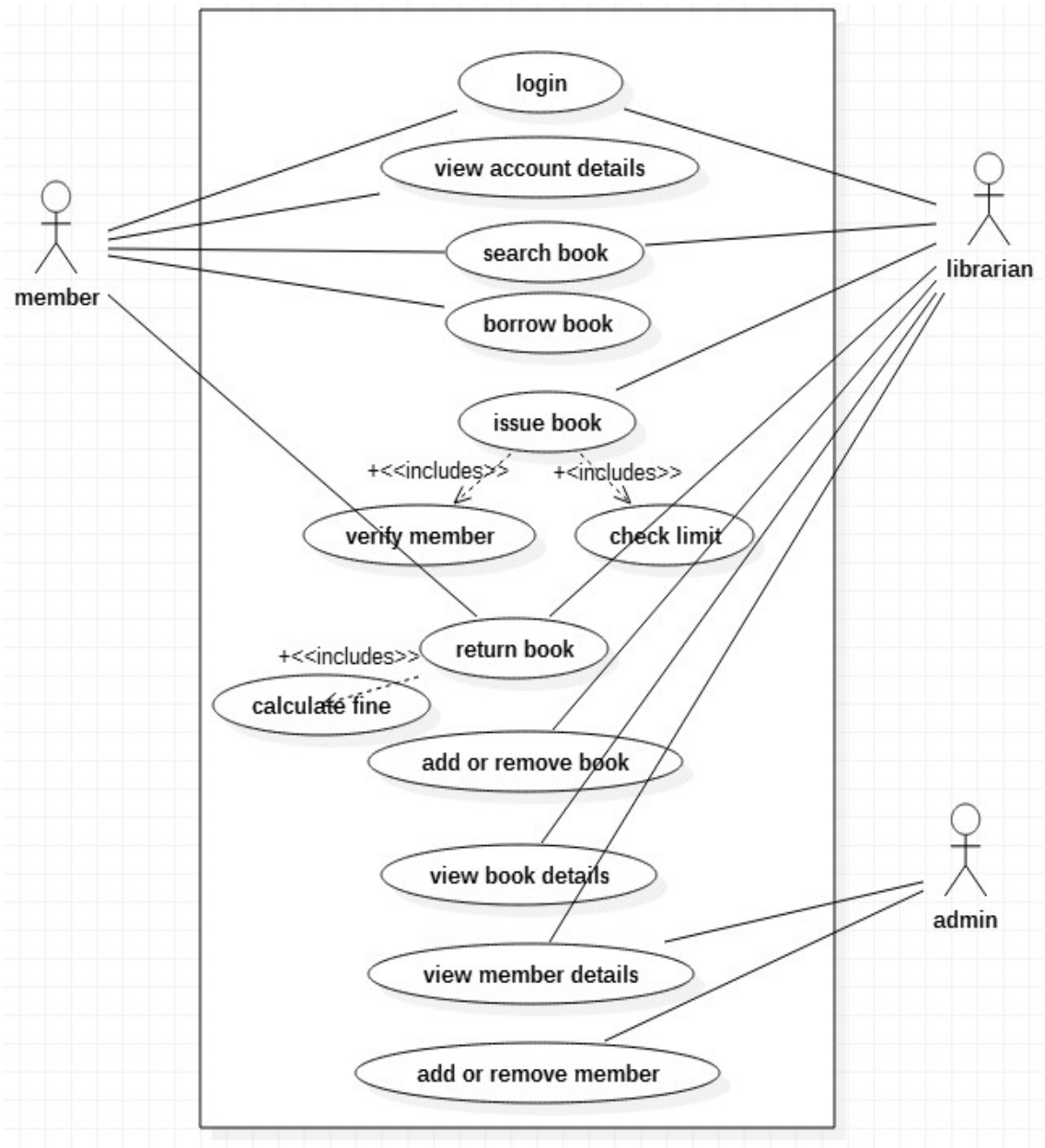
Event / Use case	Sales Line Item	Item	Sale	Payment	Register
Process Sales	U	R	CU		
Handle Returns	U	U			U
Process Rental				CU	
Cash In	U				
Manage Security					
Manage Users					
Analyze Activity	R	R	R	R	R

Week 5 & 6: For each case study

a) Develop Use case diagrams

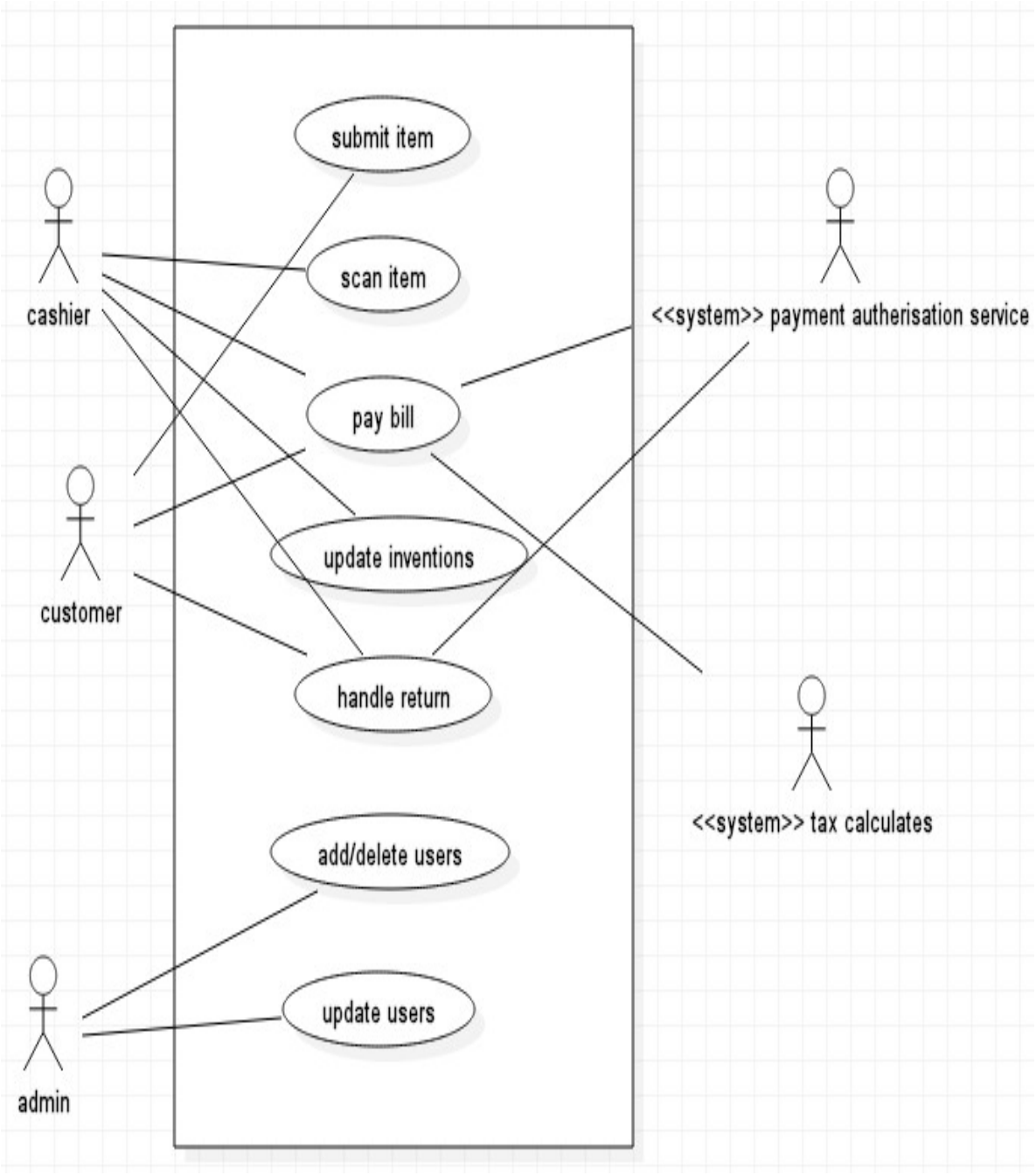
1. Library Management System:

Use case diagram of library management system



2. Point-Of-Sale Terminal:

Use case diagram of point of sale terminal



b) Develop elaborate Use case descriptions &scenarios

Scenario is a specific sequence of actions and interactions between actors and the system under discussion; it is also called a use case instance. **Use case** is a collection of related success and failure scenarios that describe actors using a system to support a goal.

1. Library Management System:

Make Reservation Use case along with its scenarios

Primary Actor:	Borrower
Stakeholders & Interests: 1. Borrower: fast & easy environment for making reservation 2. Librarian: easy management of borrower, reservation & load details 3. Master Librarian: easy management of librarian, borrowers	
Pre Conditions:	Borrower login, Enable for making reservation
Post conditions:	Reservation details for book are saved, Taking necessary actions when book is not available
Main Success Scenarios: 1. Borrower login to the system 2. Borrower search for the required title 3. Selects one of the books form the list of searched results 4. Selects for make reservation option for thebook 5. If book is available book will be issued to the respective borrower 6. Borrower loan details will be updated to indicate issue of book	
Extensions: 1. At any time, System fails: • To support recovery and correct accounting, ensure all transaction sensitive state and events can be recovered from any step of the scenario. 2. Invalid borrower credentials: • System signals error and rejects entry. 3. If selected book is not available: • Borrower need to check until book is returned by other borrower. • Borrower will be notified if new book is add to library.	

2. Point-Of-Sale Terminal:

Process Sale Use case along with its scenarios

Primary Actor:	Cashier
Stakeholders & Interests: 1. Cashier: wants accurate, fast entry, and no payment errors 2. Sales Person: wants sales commissions updated 3. Customer: wants purchase and fast service with minimal effort 4. Company: wants accurately record transactions and satisfy customer interests 5. Manager: wants to be able to quickly perform override operations 6. Government Tax agencies: want to collect tax from every sale 7. Payment Authorization Service: want to receive digital authorization request incorrect format and protocol	
Pre Conditions:	Cashier Identified and Authenticated
Post conditions:	Sale is saved, Tax is correctly calculated, Accounting and Inventory are update, Commissions recorded, Receipt is generated, Payment authentication approvals are recorded
Main Success Scenarios: 1. Customer arrives at POS checkout with items 2. Cashier starts new sale 3. Cashier enters item identifiers 4. System records sales line item & presents item details Cashier repeats steps 3-4 until indicates done. 5. System presents total with taxes and asks for payment 6. Customer pays and system handles payment 7. System logs completed sales and sends payment information to accounting and inventory systems 8. System presents receipt 9. Customer leaves with receipt and items	
Extensions: 1. At any time, System fails: • To support recovery and correct accounting, ensure all transaction sensitive state and events can be recovered from any step of the scenario. 2. Invalid identifier: • System signals error and rejects entry. 3. System detects failure to communicate with external tax calculation system service: • System restarts the service on the POS node, and continues. • Cashier may manually calculate and enter the tax, or cancel the sale. 4. Customer says they intended to pay by cash but don't have enough cash: • Customer uses an alternate payment method. • Customer tells Cashier to cancel sale. Cashier cancels sale on System.	

3. Customer Supporting System:

Add items to cart Use case along with its scenarios

Primary Actor:	Customer
Stakeholders & Interests: 1. Customer: fast & easy environment for placing orders 2. Company: accurate entry of product, payment, shipment details 3. System Administrator: easy maintenance of product details	
Pre Conditions:	Customer login
Post conditions:	Selected items/products are successfully added to cart
Main Success Scenarios: 1. Customer login to the system 2. Borrower search for the required item/product 3. Selects one of the item/product form the list of searched results 4. Add selected item/product to the cart Customer repeats 2,3,4 steps until required items/products added to cart 5. System displays list of items/products added to the cart	

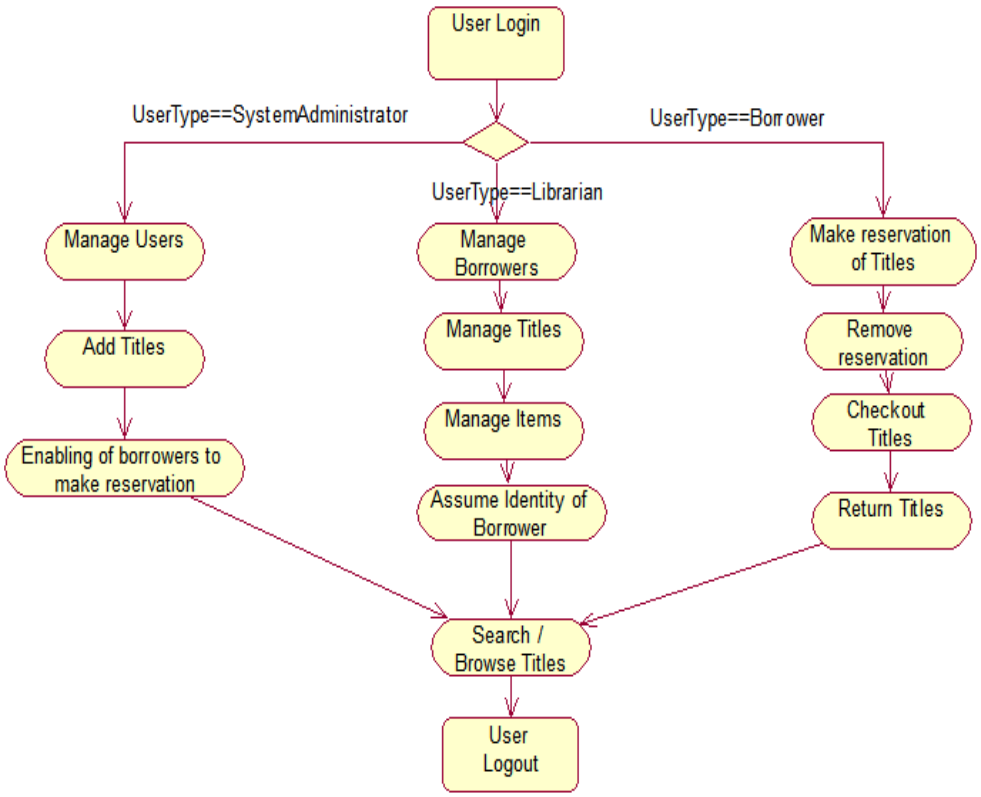
Extensions:

1. At any time, System fails:
   - To support recovery and correct accounting, ensure all transaction sensitive state and events can be recovered from any step of the scenario.
2. Invalid customer credentials:
   - System signals error and rejects entry.
3. If customer is unable to get cart details:
   - Customer need tore-login.
   - Customer need to refresh for getting updated cart details.

c) Develop prototypes (without functionality)

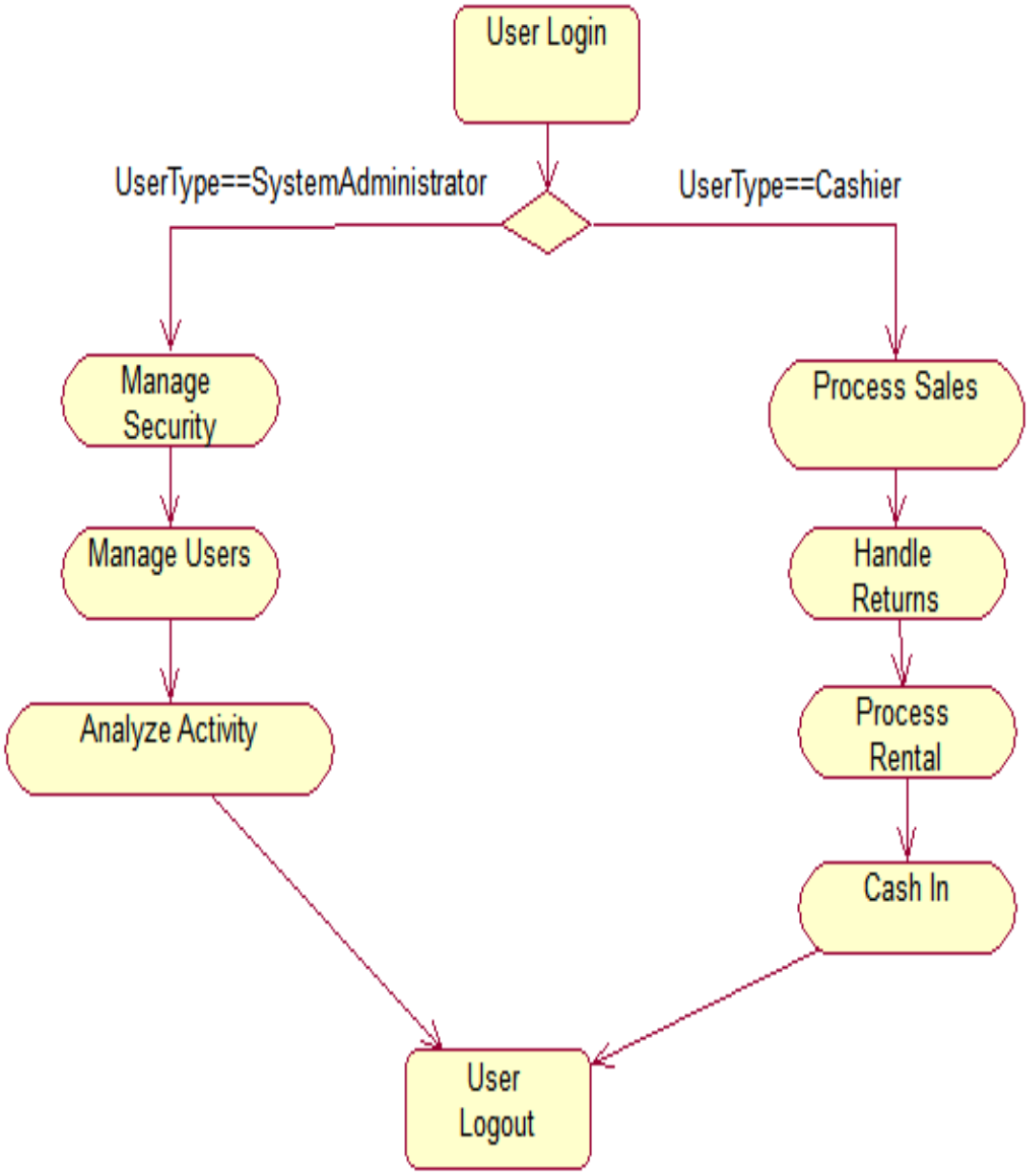
1. Library Management System:

Prototype of LMS



2. Point-Of-Sale Terminal:

Prototype of POS

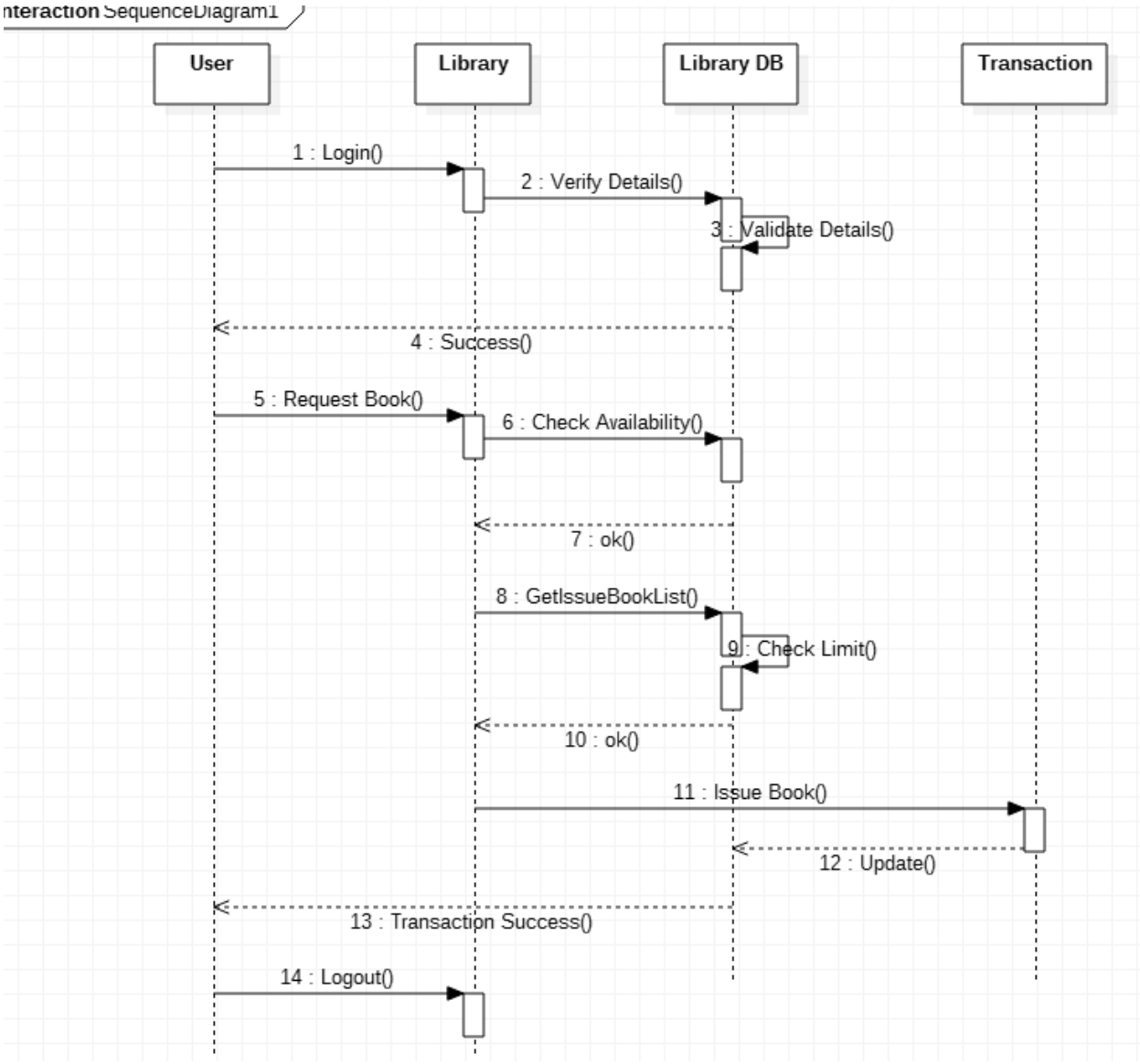


d) Develop system sequence diagrams

A system sequence diagram (SSD) is a picture that shows, for a particular scenario of a use case, the events that external actors generate their order, and inter-system events. All systems are treated as a black box; the emphasis of the diagram is events that cross the system boundary from actors to systems.

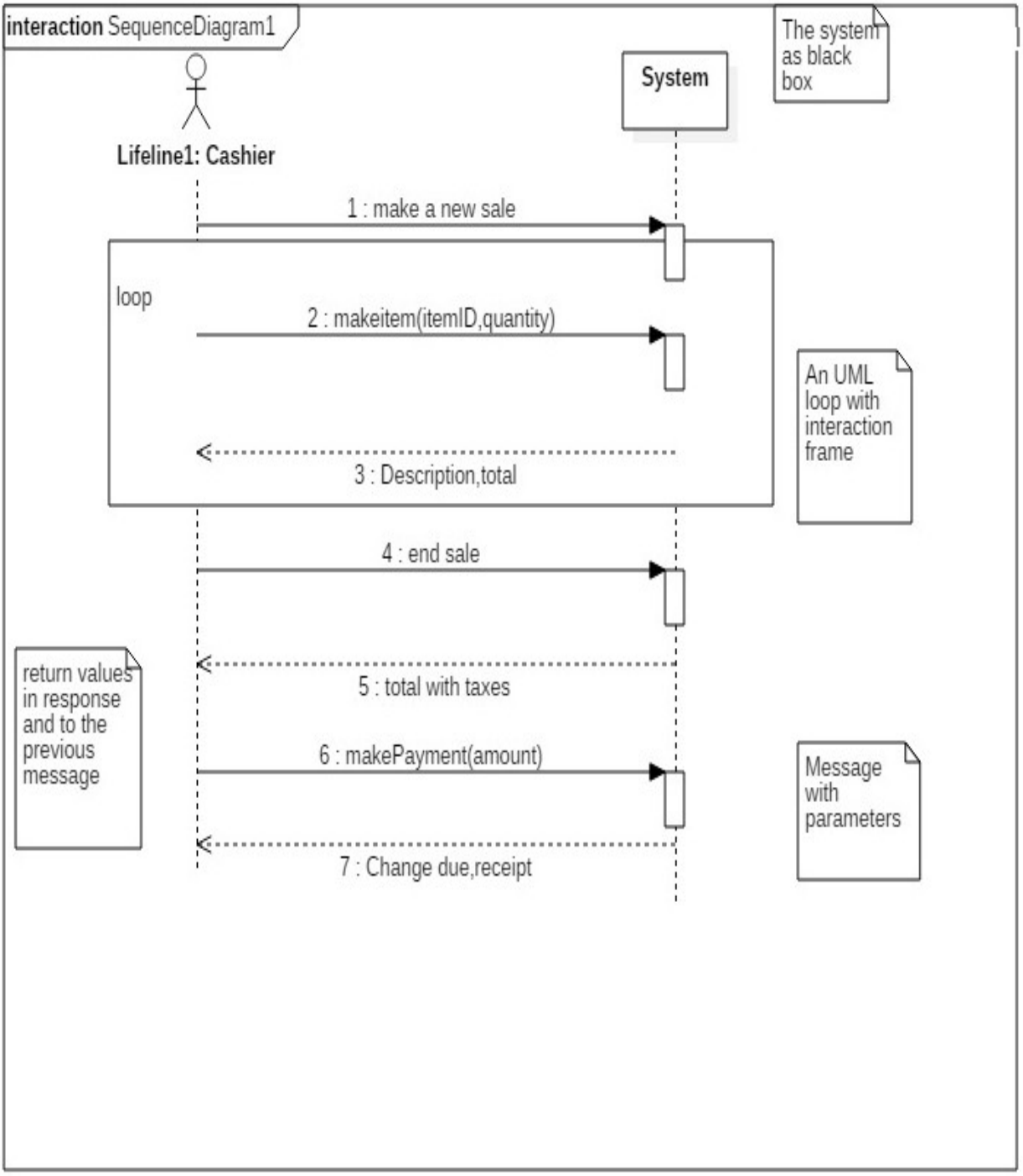
1. Library Management System:

System sequence diagram for library management system



1.Point-Of-Sale Terminal:

System sequence diagram for POS terminal system



Week 7, 8, 9 & 10: For each case study

a) Develop high-level sequence diagrams for each usecase
b) Identify MVC classes / objects for each use

case 1. Library Management System:

MVC classes/objects of LMS

Use case	Model Classes	View Classes	Controller Classes
Make Reservation	Loan	Borrower Frame	Reservation
Remove Reservation			
Checkout Item			Titles Item s
Return Titles/Items			
Manage Titles/Items	Titles Model	Librarian Frame	
Manage Librarians	Users Model		Librarian
Manage Borrowers			Borrower
Assume Identity of Borrower			

2. Point-Of-Sale Terminal:

MVC classes/objects of POS

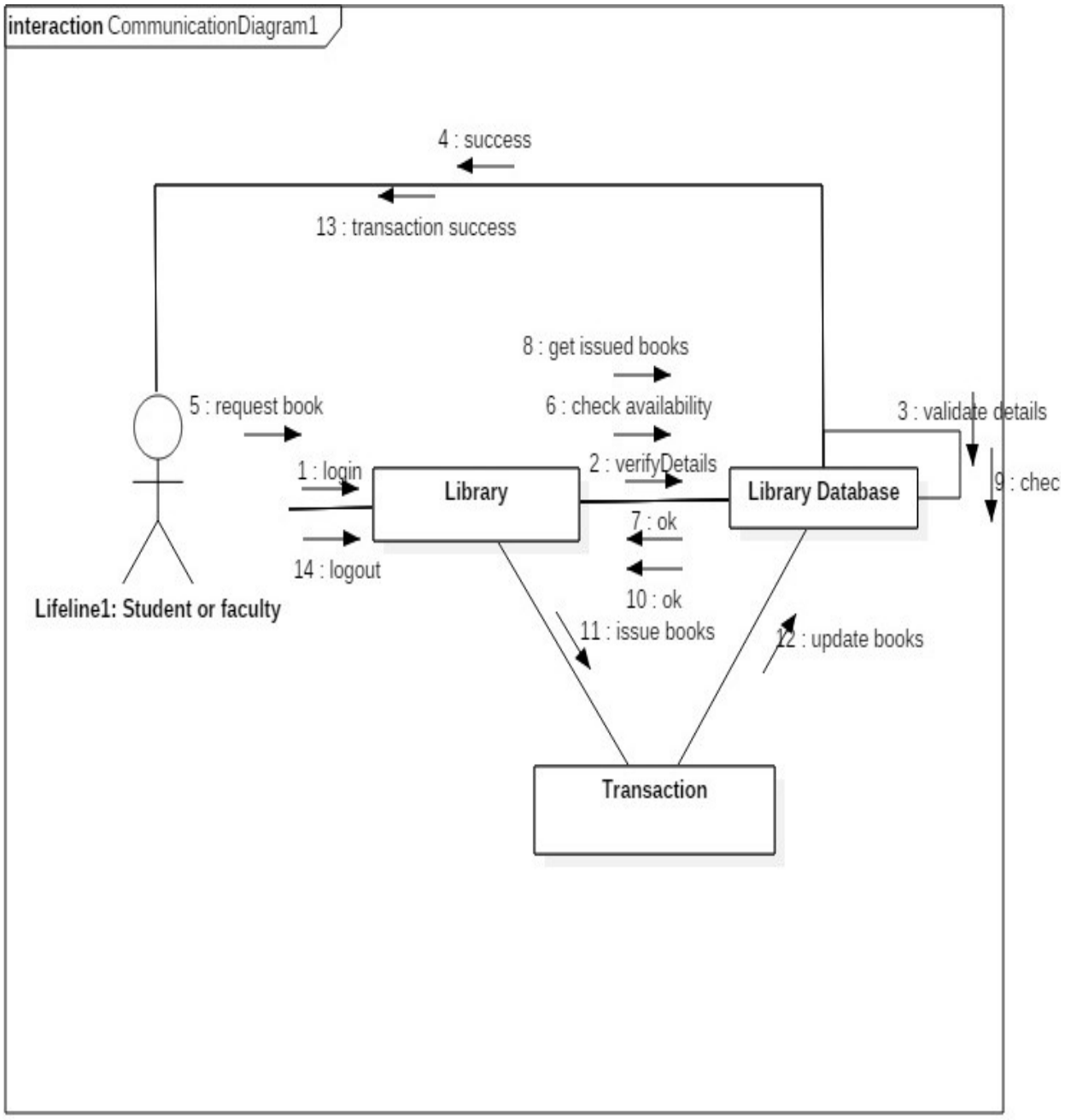
Use case	Model Classes	View Classes	Controller Classes
Process Sale	SalesModel TaxCalculator HRSystem Store	ProcessSaleFrame ProcessSaleConsole	Register Sale Payment SalsLineItem
Handle Returns			
Process Rental			
Cash In			
Analyze Activity	ActivityModel	ActivityFrame	Analyze
Manage Security	UserModel	LoginFrame	Users
Manage Users			

Login			

c) Develop Detailed Sequence Diagrams / Communication diagrams for each use case showing interactions among all the three-layer objects

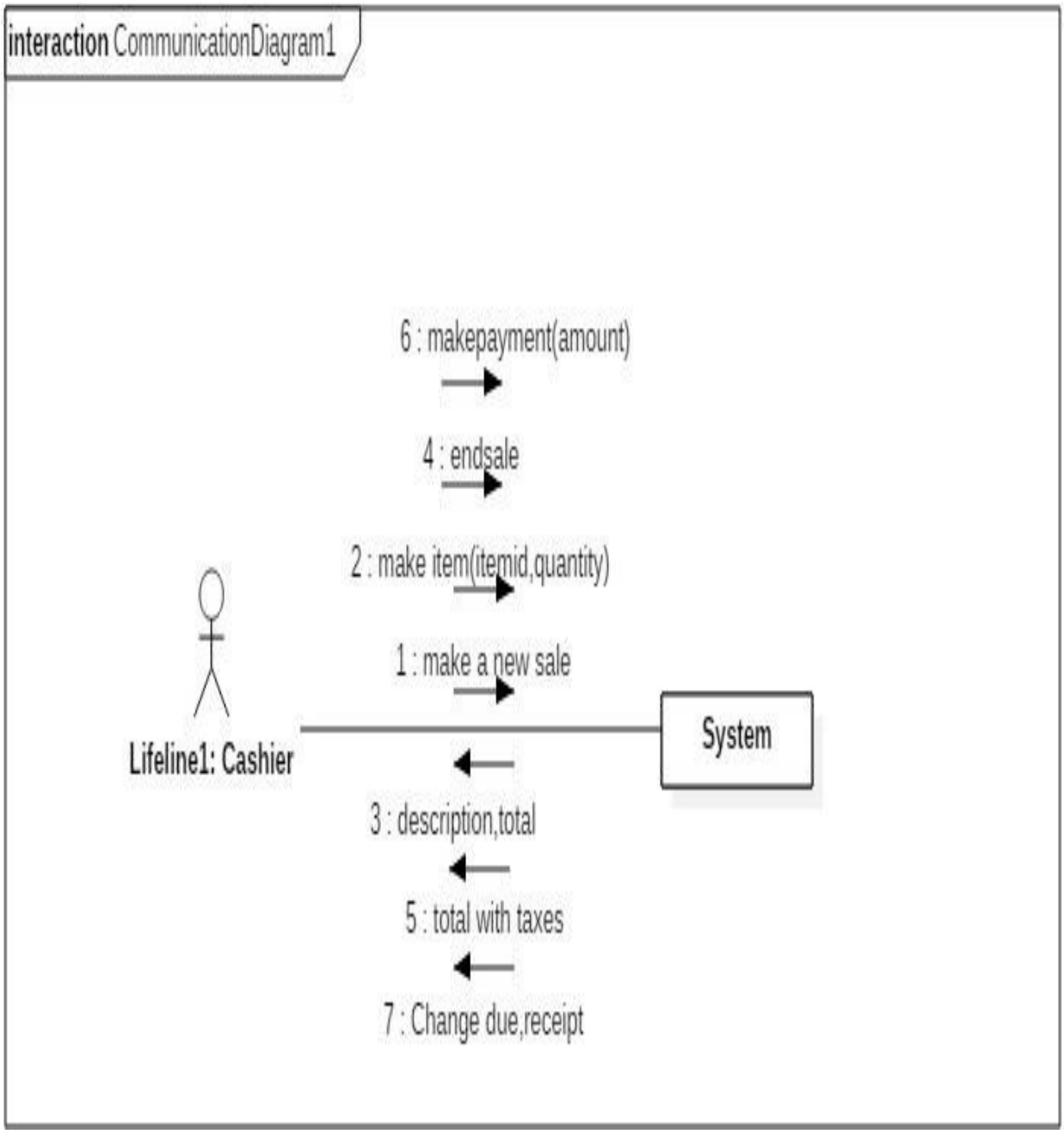
1.Library Management System:

Communication diagrams for LMS



2. Point-Of-Sale Terminal:

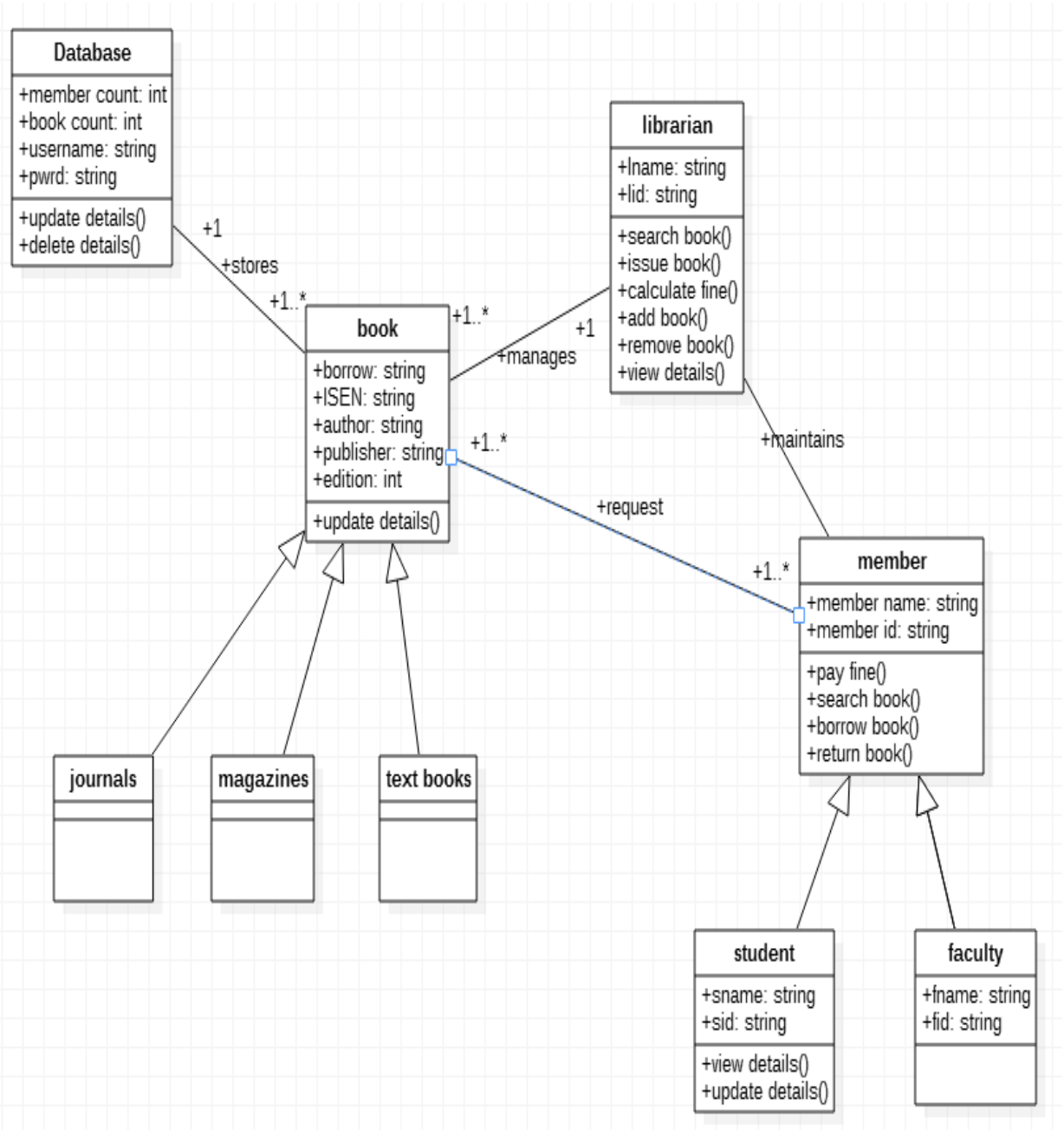
Communication diagrams for POS



d) Develop detailed design class model (use GRASP patterns for responsibility assignment)

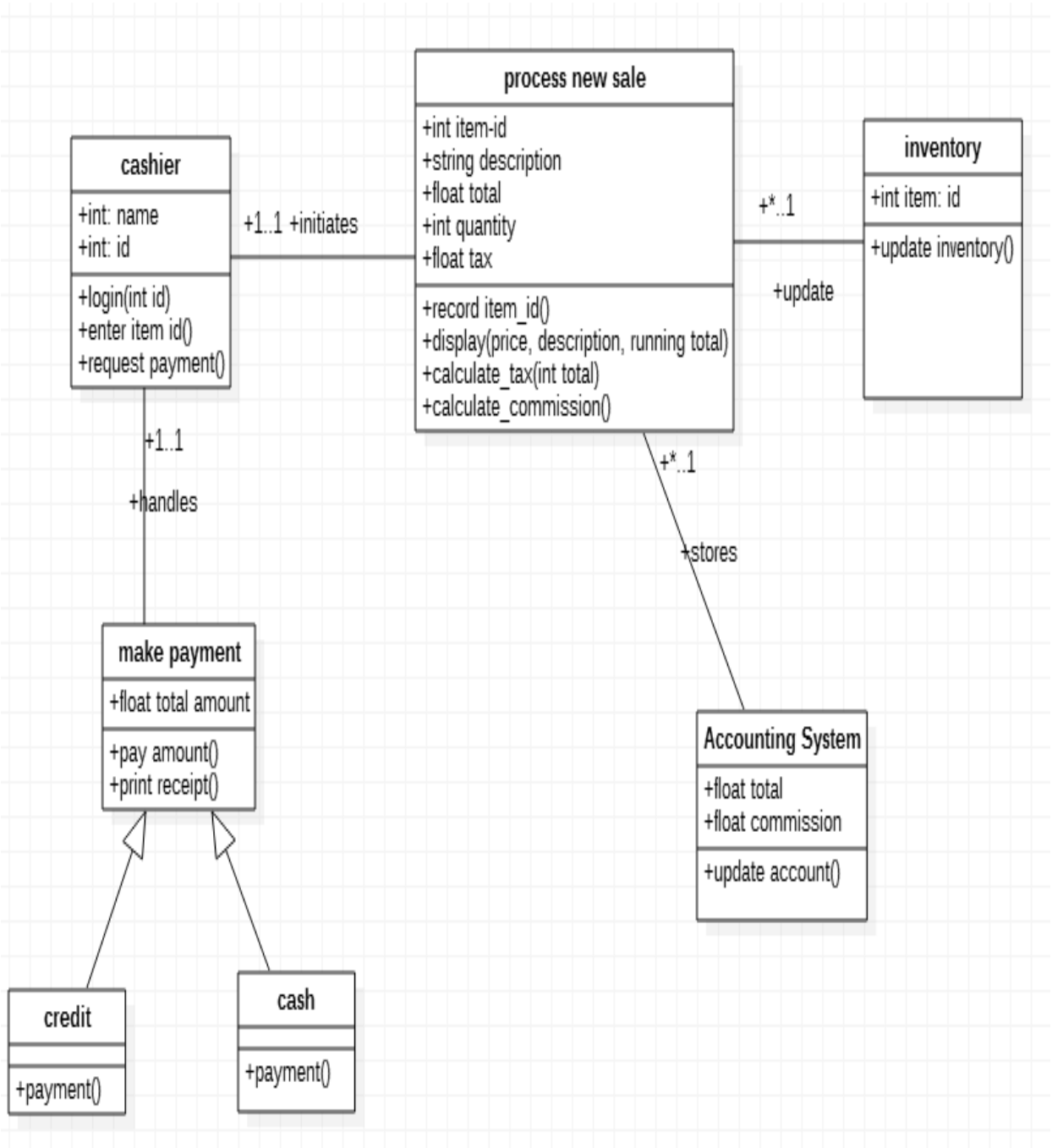
1. Library Management System:

Design class diagram for LMS



2. Point-Of-Sale Terminal:

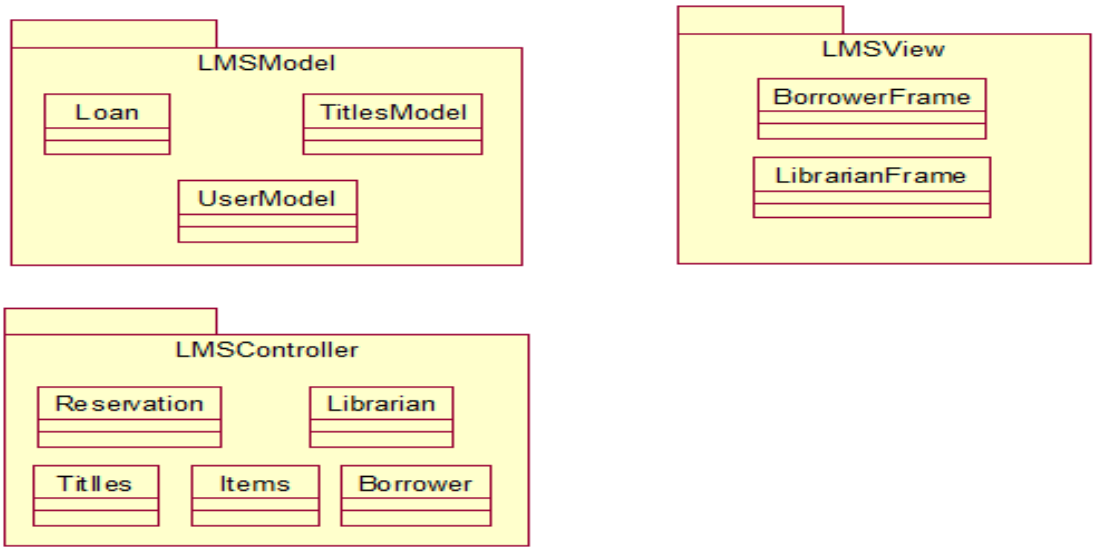
Design class diagram for POS



Develop three-layer package diagrams for each case study

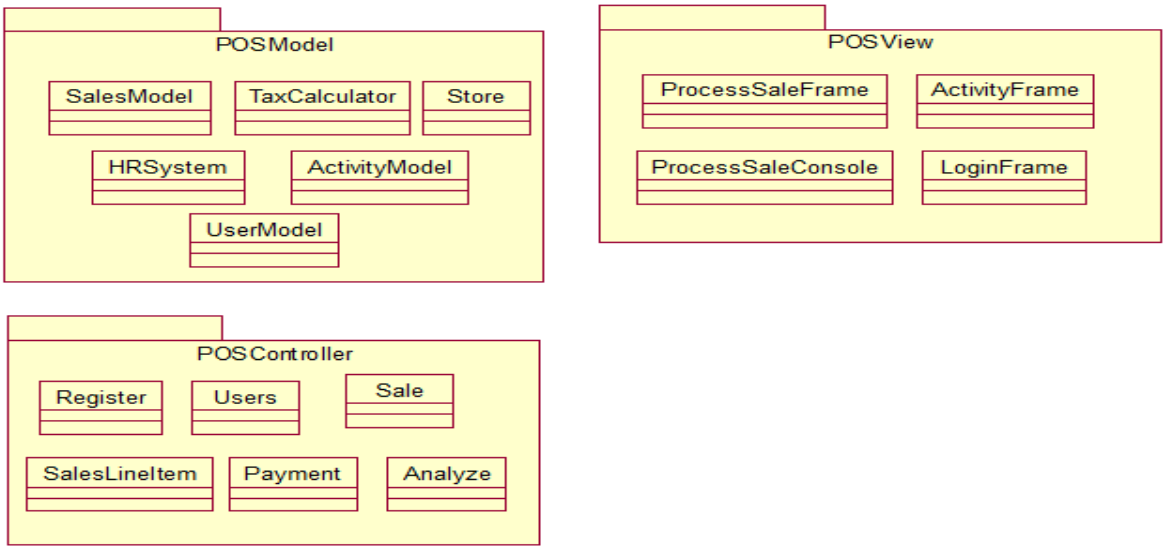
1. Library ManagementSystem:

Package diagram for LMS



2. Point-Of-Sale Terminal:

Package diagram for POS

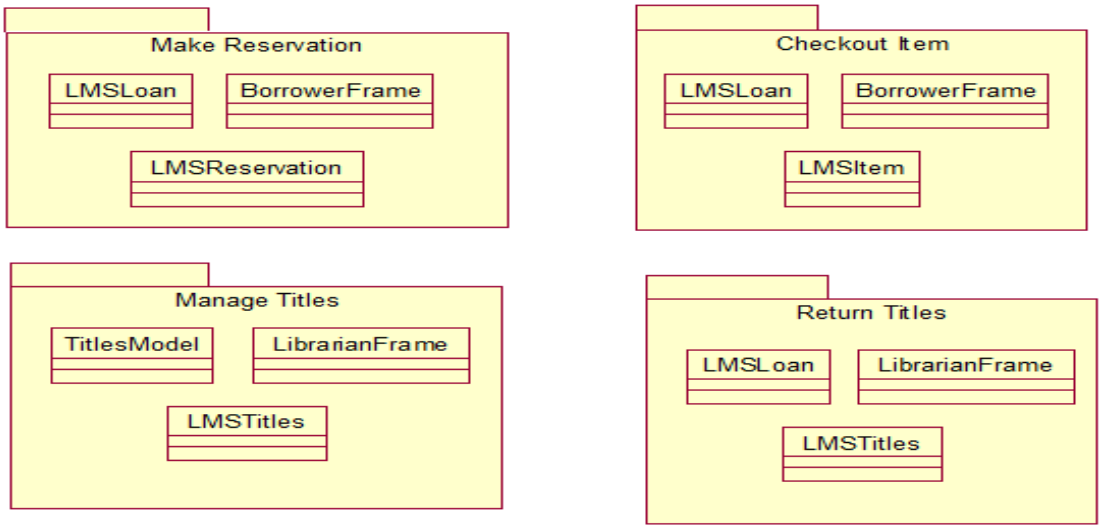


Week 11 & 12: For each case study

a) Develop Use case Packages

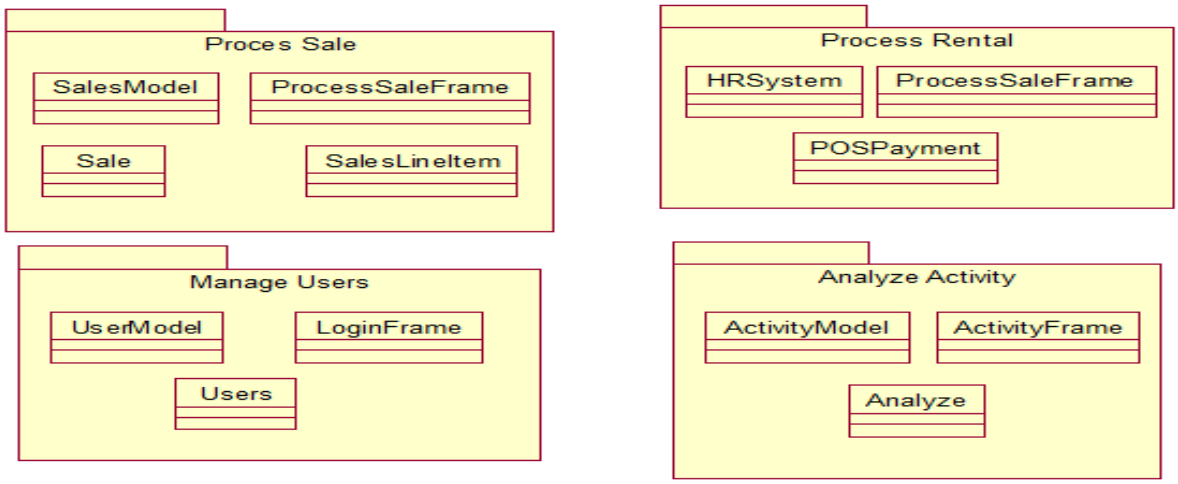
1. Library Management System:

Use case packages for Library Management System



2. Point-Of-Sale Terminal:

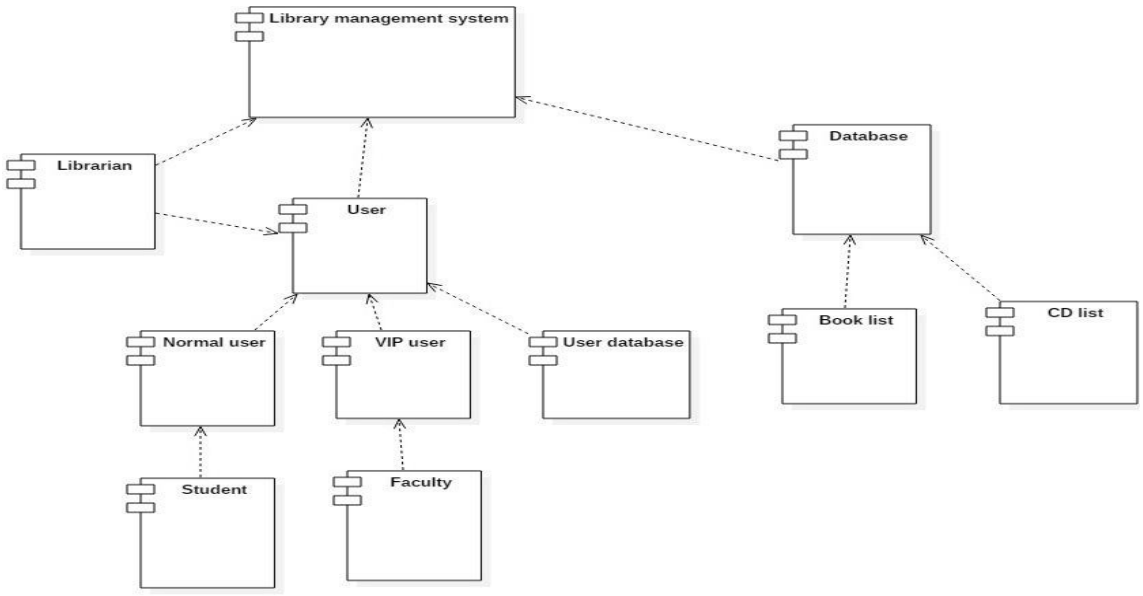
Use case packages for Point-Of-Sale System



b) Develop component diagrams

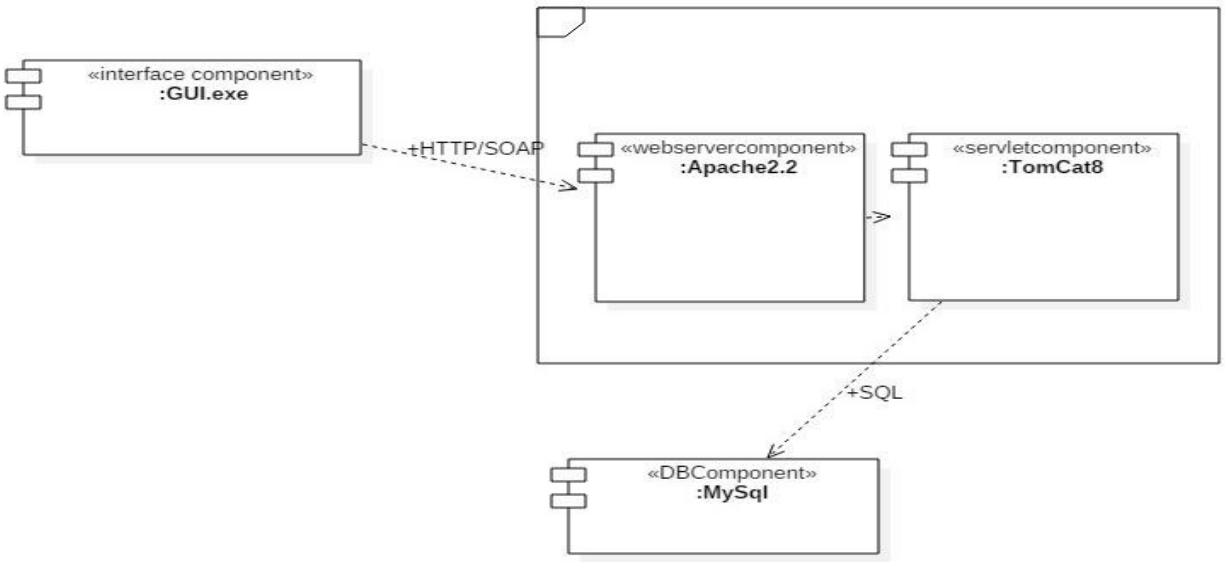
1. Library Management System:

Component diagram for Library Management System



2. Point-Of-Sale Terminal:

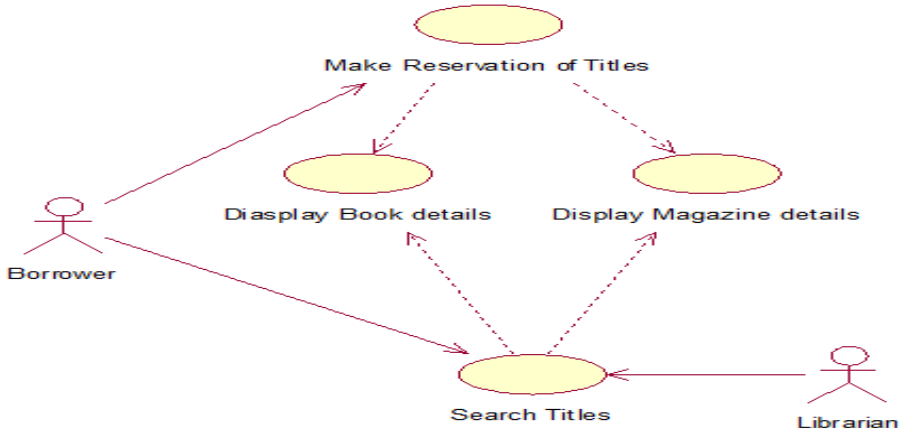
Component diagram for POS



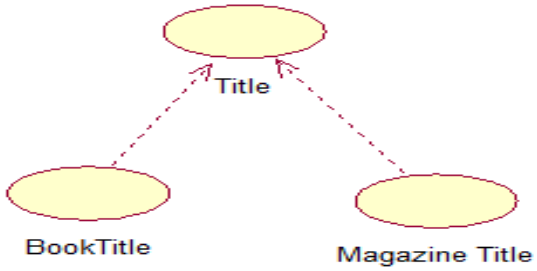
c) Identify relationships between use cases and represent them

1. Library Management System:

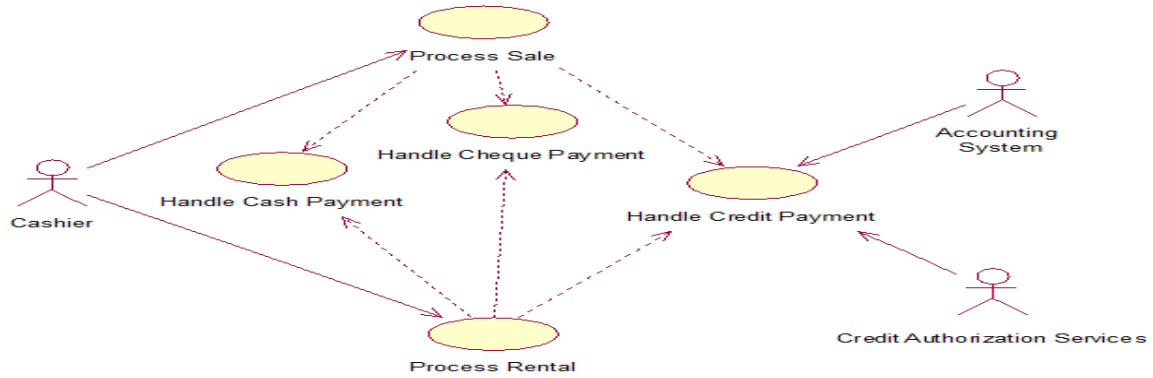
Include relations for Library Management System



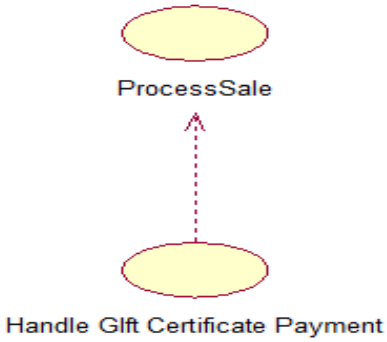
Extends relations for Library Management System



2. Point-Of-Sale Terminal:

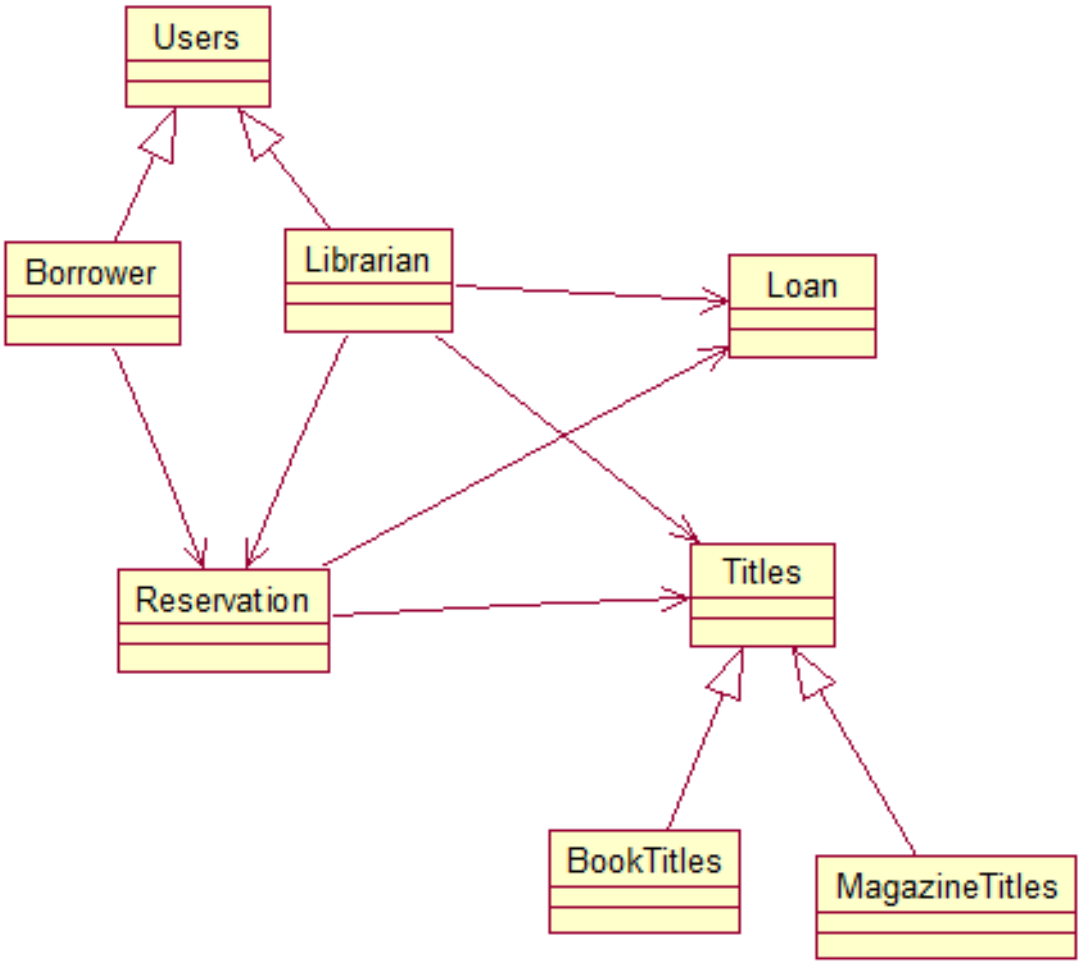


Extends relations for POS

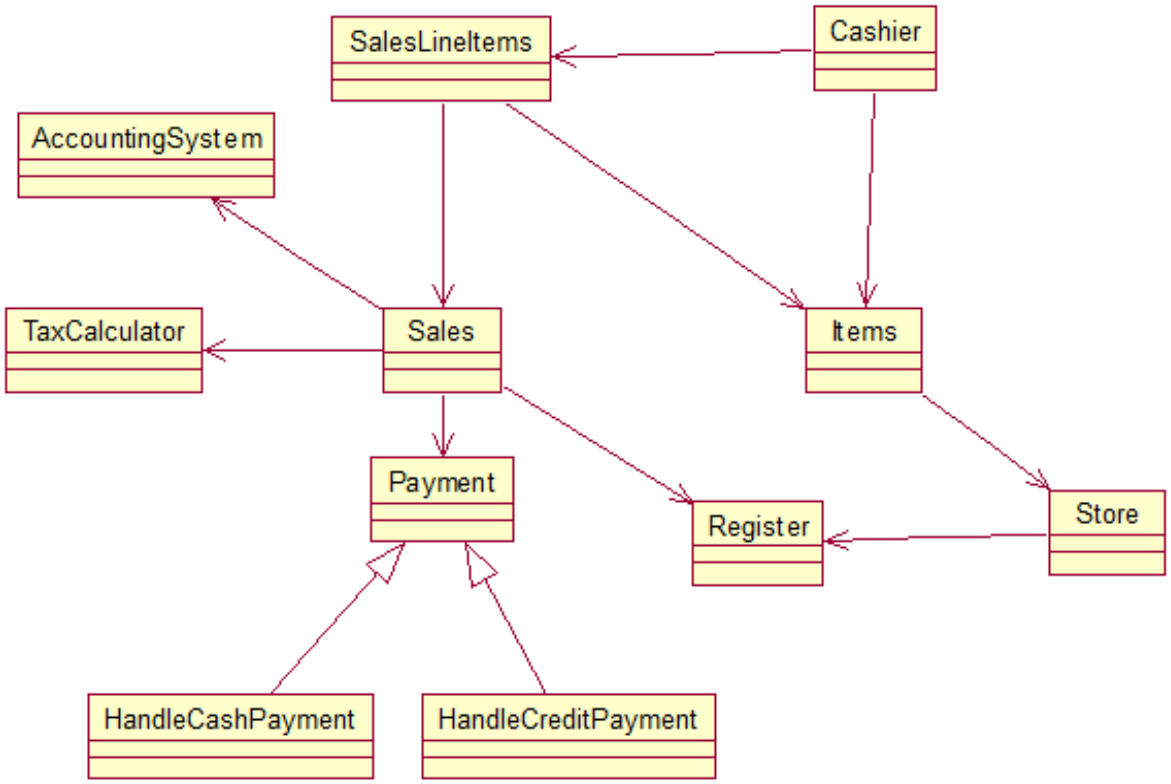


c) Refine domain class model by showing all the associations among classes

1. Library Management System:



2. Point-Of-Sale System

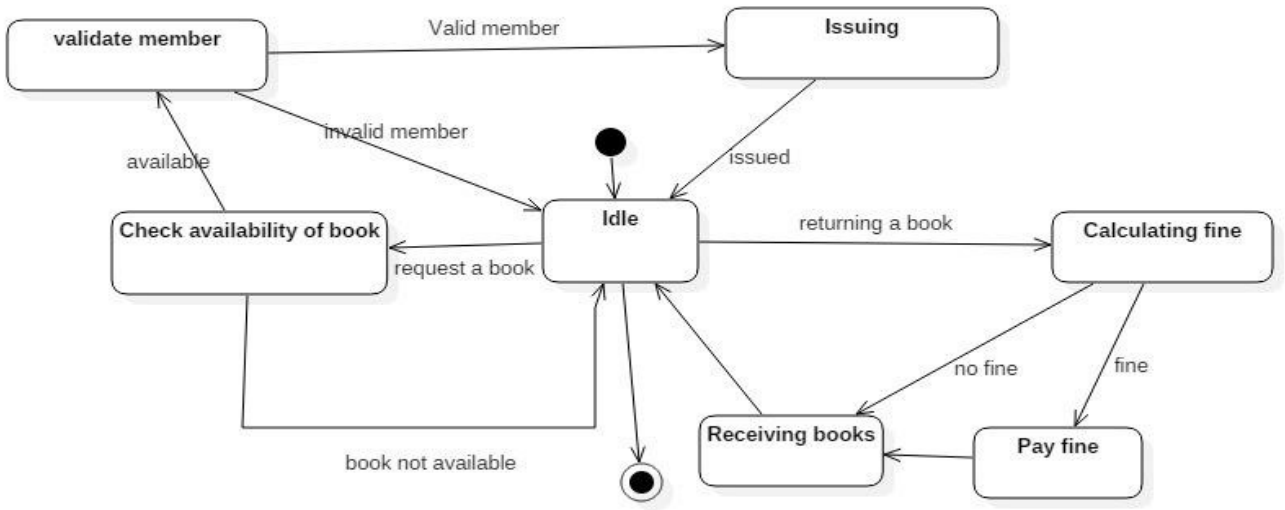


Week 13 onwards: For each case study

a) Develop sample diagrams for other UML diagrams - state chart diagrams, activity diagrams and deployment diagrams

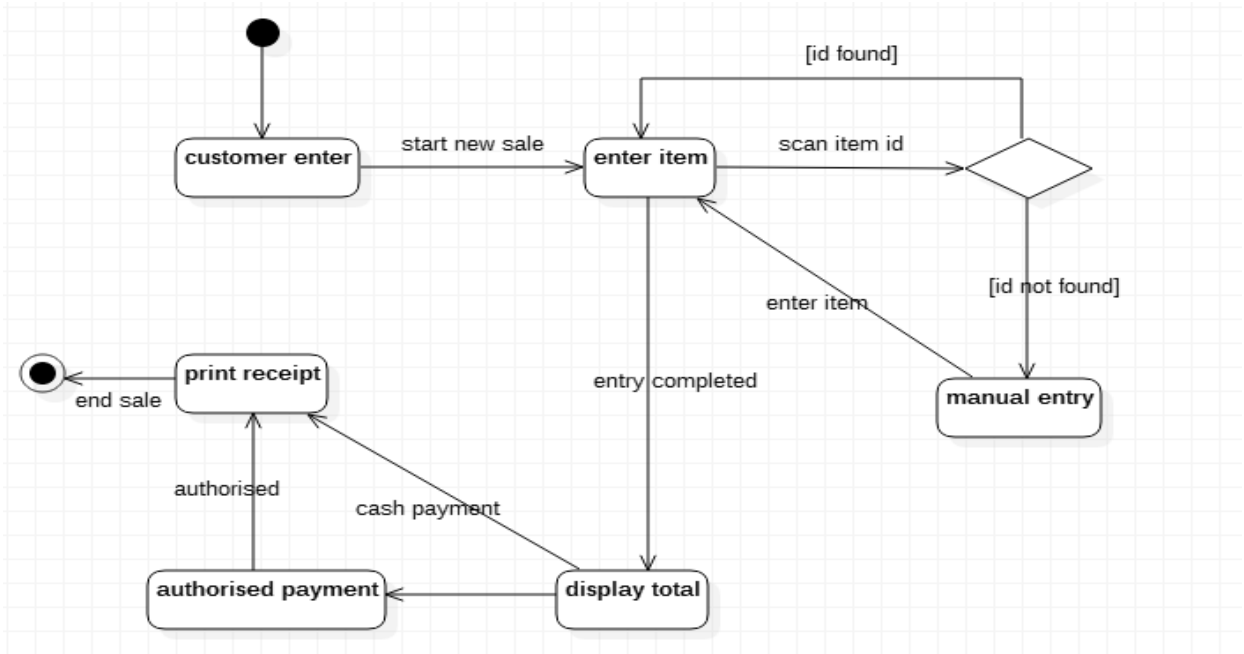
1. Library Management System:

State chart diagram for Library Management System



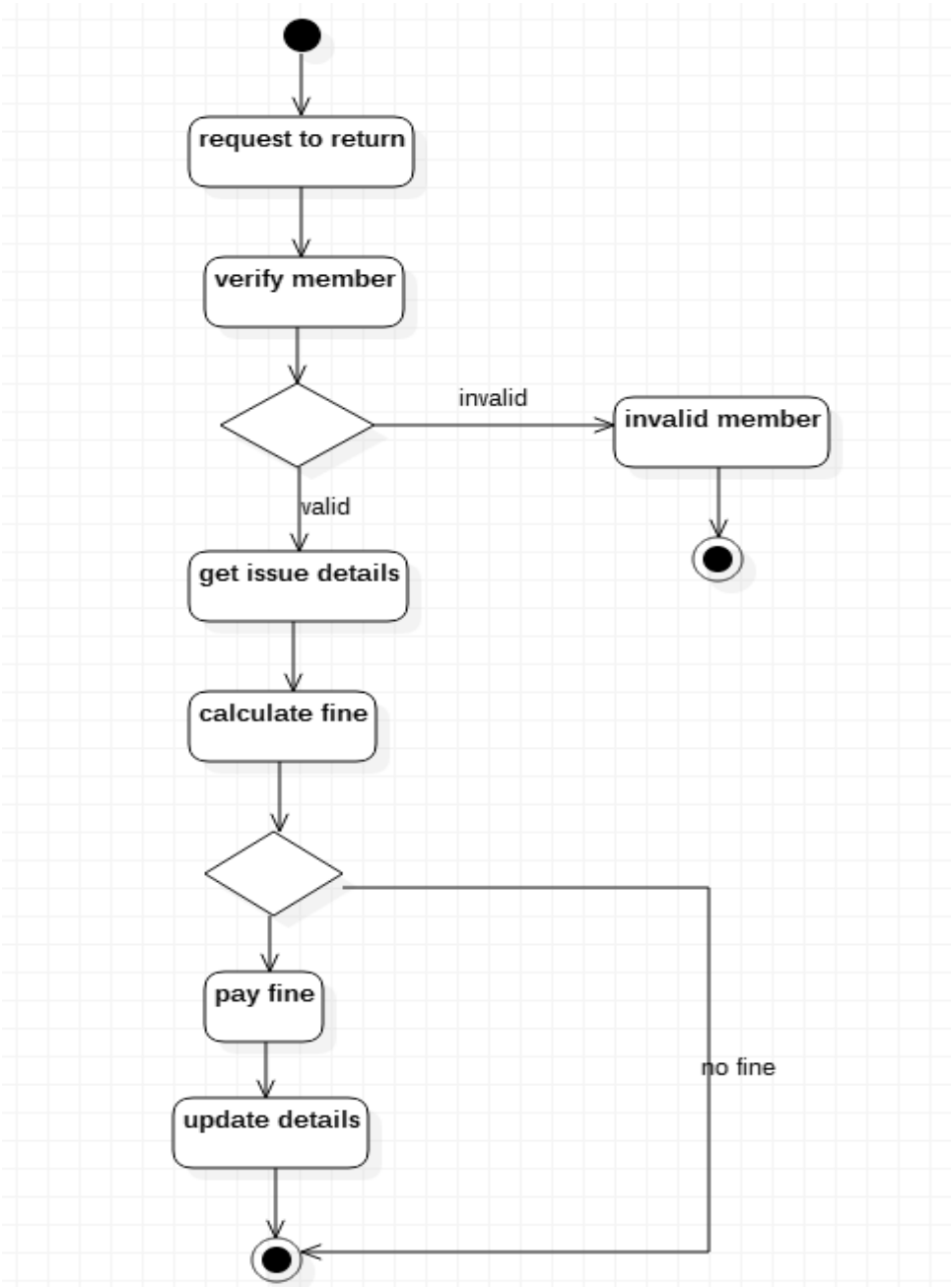
2. Point-Of-Sale Terminal:

State chart diagram for Point - Of - Sale System



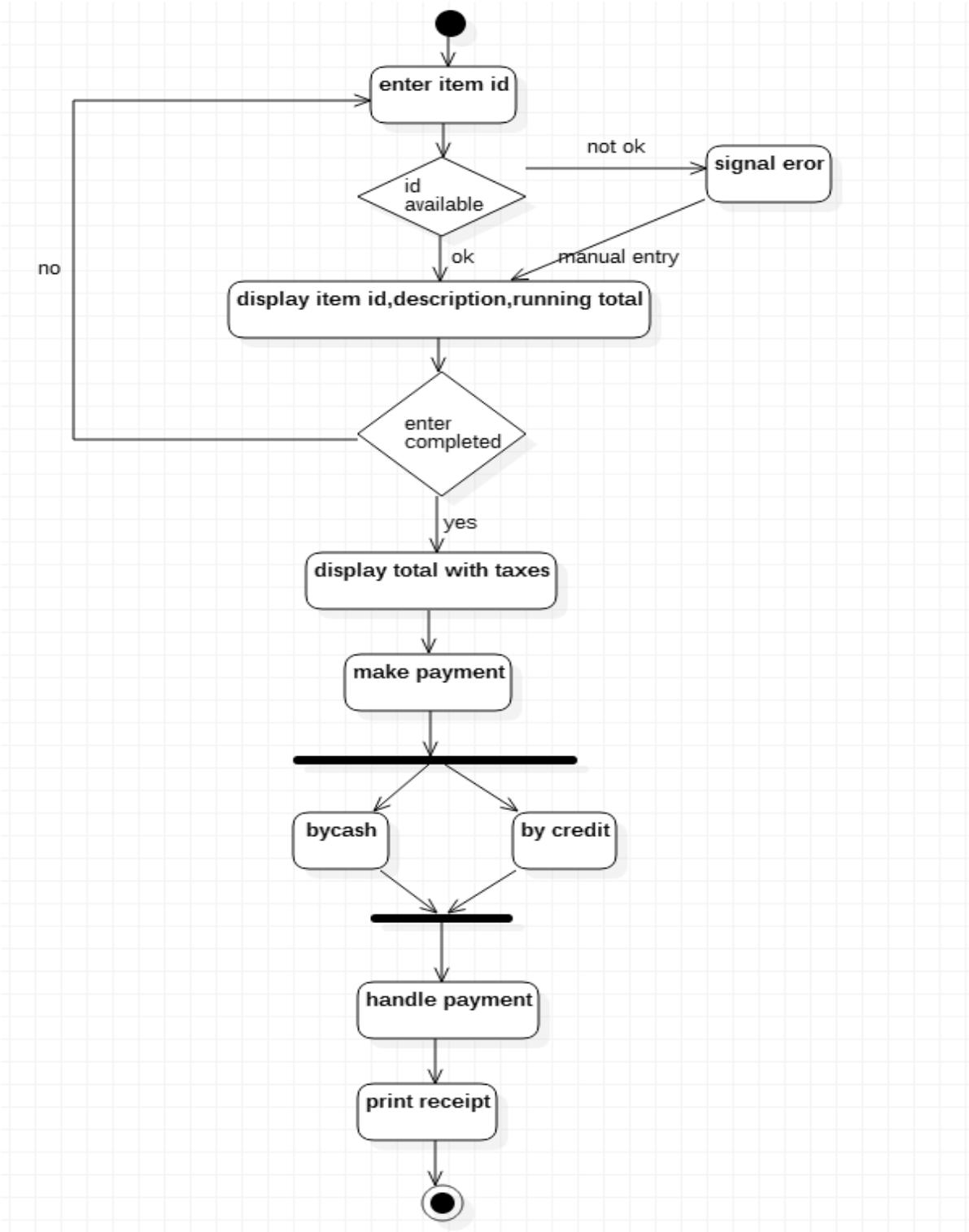
1. Library Management System:

Activity diagram for return a book



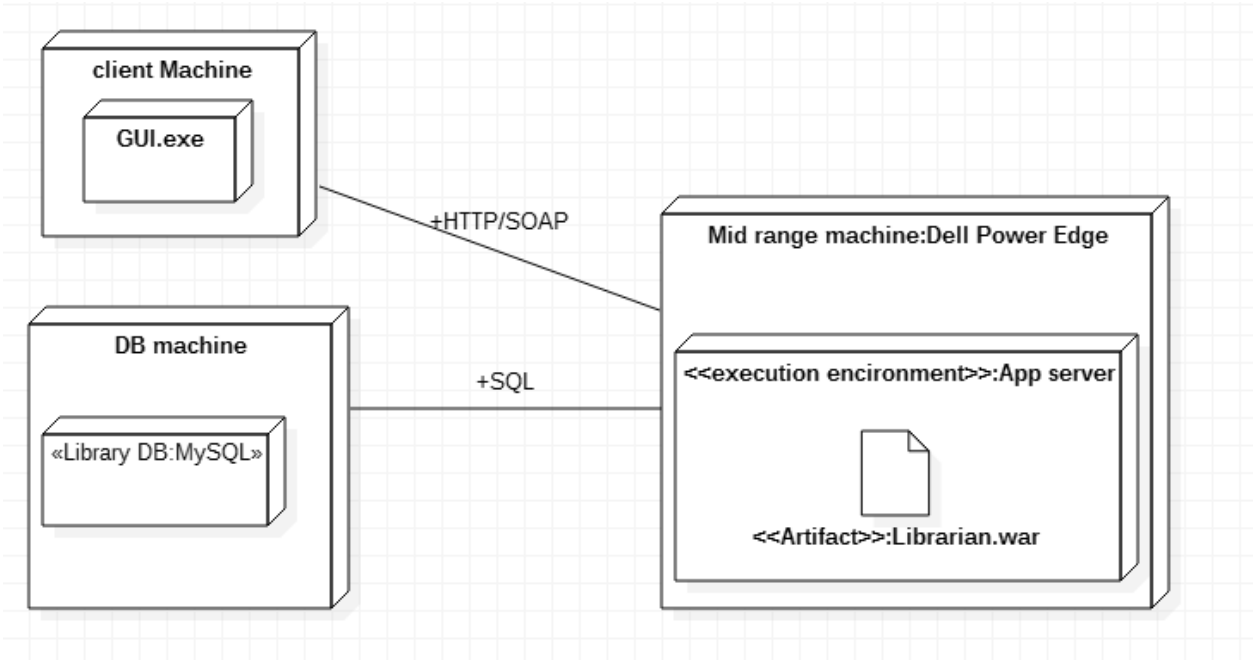
2. Point-Of-Sale Terminal:

Activity diagram for POS



1. Library Management System:

Deployment diagram for Library Management System



2. Point-Of-Sale Terminal:

Deployment diagram
for POS