Sinhgad Institutes

Sinhgad Technical Educational Society's

SINHGAD INSTITUTE OF TECHNOLOGY

LONAVALA

Subject: - 414459: Computer Laboratory VIII

Class:-B.E. Information Technology

Roll No:

Exam Seat No:

Name of The Student:

Subject Incharge:
Prof. R. S. Badodekar
(Information Technology Department)

Sinhgad Technical Educational Society's

SINHGAD INSTITUTE OF TECHNOLOGY LONAVALA

CERTIFICATE

This is to certify that

Mr. / Miss _____,

*Of Class **BE – IT** Roll No. _____ has completed all the practical work in the subject **Computer Laboratory VIII** satisfactorily in the Department of Information Technology as prescribed by Savitribai Phule University of Pune, in the academic year 20__ – 20__.*

Staff In-charge Head of the Department Principal

Sinhgad Institutes

Sinhagad Technical Educational Society's

SINHGAD INSTITUTE OF TECHNOLOGY LONAVALA

Subject:- Computer Laboratory VIII

INDEX

Name: _____ Roll No._____

Class: _____Batch_____ Subject: _____

Sr. No.	Title	Date	Page No.	Remark	Sign of Teacher
1	**Write Problem Statement for System / Project** *Identify Project of enough complexity, which has at least 4-5 major functionalities.* *Identify stakeholders, actors and write detail problem statement for your system.*				
2	**Prepare Use Case Model** *Identify Major Use Cases, Identify actors.* *Write Use Case specification for all major Use Cases. Draw detail Use Case Diagram using UML2.0 notations.*				
3	**Prepare Activity Model** *Identify Activity states and Action states. Draw Activity diagram with Swim lanes using UML2.0 Notations for major Use Cases*				
4	**Prepare Analysis Model-Class Model** *Identify Analysis Classes and assign responsibilities. Prepare Data Dictionary.* *Draw Analysis class Model using UML2.0 Notations. Implement Analysis class Model-class diagram with a suitable object oriented language*				
5	**Prepare a Design Model from Analysis Model** *Study in detail working of system / Project.* *Identify Design classes/ Evolve Analysis Model. Use advanced relationships. Draw Design class Model using OCL and UML2.0 Notations.* *Implement the design model with a suitable object-oriented language.*				

Sr. No.	Title	Date	Page No.	Remark	Sign of Teacher
6	**Prepare Sequence Model.** *Identify at least 5 major scenarios (sequence flow) for your system.* *Draw Sequence Diagram for every scenario by using advanced notations using UML2.0* *Implement these scenarios by taking reference of design model implementation using suitable object-oriented language.*				
7	**Prepare a State Model** *Identify States and events for your system.* *Study state transitions and identify Guard conditions.* *Draw State chart diagram with advanced UML 2 notations.Implement the state model with a suitable object-oriented language*				
8	**Identification and Implementation of GRASP pattern** *Apply any two GRASP pattern to refine the Design Model for a given problem description Using effective UML 2 diagrams and implement them with a suitable object oriented language*				
9	**Identification and Implementation of GOF pattern** *Apply any two GOF pattern to refine Design Model for a given problem description Using effective UML 2 diagrams and implement them with a suitable object oriented language*				

LAB INNOVATION/ ADDITIONAL WORK IF ANY

Sr. No.	Title of Experiment	Date	Page No.	Remark	Sign of Teacher
1)					
2)					

Assignment No. 1

Title: Write Problem Statement for System / Project

Identify Project of enough complexity, which has at least 4-5 major functionalities.
Identify stakeholders, actors and write detail problem statement for your system.

Date of Completion		**Marks for Regularity (2)**	
		Marks for Presentation (3)	
Remark		**Marks for Understanding (5)**	
Sign. of Staff		**Total Marks (10)**	

Prof. R.S. Badodekar **414459 : Computer Laboratory VIII**

Software Requirements Specification

for

<Project>

Version 1.0 approved

Prepared by <author>

<organization>

<date created>

Table of Contents

Revision History

Name	Date	Reason For Changes	Version

1. Introduction

1.1 Purpose

<Identify the product whose software requirements are specified in this document, including the revision or release number. Describe the scope of the product that is covered by this SRS, particularly if this SRS describes only part of the system or a single subsystem.>

1.2 Document Conventions

<Describe any standards or typographical conventions that were followed when writing this SRS, such as fonts or highlighting that have special significance. For example, state whether priorities for higher-level requirements are assumed to be inherited by detailed requirements, or whether every requirement statement is to have its own priority.>

1.3 Intended Audience and Reading Suggestions

<Describe the different types of reader that the document is intended for, such as developers, project managers, marketing staff, users, testers, and documentation writers. Describe what the rest of this SRS contains and how it is organized. Suggest a sequence for reading the document, beginning with the overview sections and proceeding through the sections that are most pertinent to each reader type.>

1.4 Product Scope

<Provide a short description of the software being specified and its purpose, including relevant benefits, objectives, and goals. Relate the software to corporate goals or business strategies. If a separate vision and scope document is available, refer to it rather than duplicating its contents here.>

1.5 References

<List any other documents or Web addresses to which this SRS refers. These may include user interface style guides, contracts, standards, system requirements specifications, use case documents, or a vision and scope document. Provide enough information so that the reader could access a copy of each reference, including title, author, version number, date, and source or location.>

2. Overall Description

2.1 Product Perspective

<Describe the context and origin of the product being specified in this SRS. For example, state whether this product is a follow-on member of a product family, a replacement for certain existing

systems, or a new, self-contained product. If the SRS defines a component of a larger system, relate the requirements of the larger system to the functionality of this software and identify interfaces between the two. A simple diagram that shows the major components of the overall system, subsystem interconnections, and external interfaces can be helpful.>

2.2 Product Functions

<Summarize the major functions the product must perform or must let the user perform. Details will be provided in Section 3, so only a high level summary (such as a bullet list) is needed here. Organize the functions to make them understandable to any reader of the SRS. A picture of the major groups of related requirements and how they relate, such as a top level data flow diagram or object class diagram, is often effective.>

2.3 User Classes and Characteristics

<Identify the various user classes that you anticipate will use this product. User classes may be differentiated based on frequency of use, subset of product functions used, technical expertise, security or privilege levels, educational level, or experience. Describe the pertinent characteristics of each user class. Certain requirements may pertain only to certain user classes. Distinguish the most important user classes for this product from those who are less important to satisfy.>

2.4 Operating Environment

<Describe the environment in which the software will operate, including the hardware platform, operating system and versions, and any other software components or applications with which it must peacefully coexist.>

2.5 Design and Implementation Constraints

<Describe any items or issues that will limit the options available to the developers. These might include: corporate or regulatory policies; hardware limitations (timing requirements, memory requirements); interfaces to other applications; specific technologies, tools, and databases to be used; parallel operations; language requirements; communications protocols; security considerations; design conventions or programming standards (for example, if the customer's organization will be responsible for maintaining the delivered software).>

2.6 User Documentation

<List the user documentation components (such as user manuals, on-line help, and tutorials) that will be delivered along with the software. Identify any known user documentation delivery formats or standards.>

2.7 Assumptions and Dependencies

<List any assumed factors (as opposed to known facts) that could affect the requirements stated in the SRS. These could include third-party or commercial components that you plan to use, issues around the development or operating environment, or constraints. The project could be affected if these assumptions are incorrect, are not shared, or change. Also identify any dependencies the project has on external factors, such as software components that you intend to reuse from another project, unless they are already documented elsewhere (for example, in the vision and scope document or the project plan).>

3. External Interface Requirements

3.1 User Interfaces

<Describe the logical characteristics of each interface between the software product and the users. This may include sample screen images, any GUI standards or product family style guides that are to be followed, screen layout constraints, standard buttons and functions (e.g., help) that will appear on every screen, keyboard shortcuts, error message display standards, and so on. Define the software components for which a user interface is needed. Details of the user interface design should be documented in a separate user interface specification.>

3.2 Hardware Interfaces

<Describe the logical and physical characteristics of each interface between the software product and the hardware components of the system. This may include the supported device types, the nature of the data and control interactions between the software and the hardware, and communication protocols to be used.>

3.3 Software Interfaces

<Describe the connections between this product and other specific software components (name and version), including databases, operating systems, tools, libraries, and integrated commercial components. Identify the data items or messages coming into the system and going out and describe the purpose of each. Describe the services needed and the nature of communications. Refer to documents that describe detailed application programming interface protocols. Identify data that will be shared across software components. If the data sharing mechanism must be implemented in a specific way (for example, use of a global data area in a multitasking operating system), specify this as an implementation constraint.>

3.4 Communications Interfaces

<Describe the requirements associated with any communications functions required by this product, including e-mail, web browser, network server communications protocols, electronic forms, and so on. Define any pertinent message formatting. Identify any communication standards that will be used, such as FTP or HTTP. Specify any communication security or encryption issues, data transfer rates, and synchronization mechanisms.>

4. System Features

<This template illustrates organizing the functional requirements for the product by system features, the major services provided by the product. You may prefer to organize this section by use case, mode of operation, user class, object class, functional hierarchy, or combinations of these, whatever makes the most logical sense for your product.>

4.1 System Feature 1

<Don't really say "System Feature 1." State the feature name in just a few words.>

4.1.1 Description and Priority

<Provide a short description of the feature and indicate whether it is of High, Medium, or Low priority. You could also include specific priority component ratings, such as benefit, penalty, cost, and risk (each rated on a relative scale from a low of 1 to a high of 9).>

4.1.2 Stimulus/Response Sequences

<List the sequences of user actions and system responses that stimulate the behavior defined for this feature. These will correspond to the dialog elements associated with use cases.>

4.1.3 Functional Requirements

<Itemize the detailed functional requirements associated with this feature. These are the software capabilities that must be present in order for the user to carry out the services provided by the feature, or to execute the use case. Include how the product should respond to anticipated error conditions or invalid inputs. Requirements should be concise, complete, unambiguous, verifiable, and necessary. Use "TBD" as a placeholder to indicate when necessary information is not yet available.>

<Each requirement should be uniquely identified with a sequence number or a meaningful tag of some kind.>

REQ-1:
REQ-2:

4.2 System Feature 2 (and so on)

5. Other Nonfunctional Requirements

5.1 Performance Requirements

<If there are performance requirements for the product under various circumstances, state them here and explain their rationale, to help the developers understand the intent and make suitable design choices. Specify the timing relationships for real time systems. Make such requirements as specific as possible. You may need to state performance requirements for individual functional requirements or features.>

5.2 Safety Requirements

<Specify those requirements that are concerned with possible loss, damage, or harm that could result from the use of the product. Define any safeguards or actions that must be taken, as well as actions that must be prevented. Refer to any external policies or regulations that state safety issues that affect the product's design or use. Define any safety certifications that must be satisfied.>

5.3 Security Requirements

<Specify any requirements regarding security or privacy issues surrounding use of the product or protection of the data used or created by the product. Define any user identity authentication requirements. Refer to any external policies or regulations containing security issues that affect the product. Define any security or privacy certifications that must be satisfied.>

5.4 Software Quality Attributes

<Specify any additional quality characteristics for the product that will be important to either the customers or the developers. Some to consider are: adaptability, availability, correctness, flexibility, interoperability, maintainability, portability, reliability, reusability, robustness, testability, and usability. Write these to be specific, quantitative, and verifiable when possible. At the least, clarify the relative preferences for various attributes, such as ease of use over ease of learning.>

5.5 Business Rules

<List any operating principles about the product, such as which individuals or roles can perform which functions under specific circumstances. These are not functional requirements in themselves, but they may imply certain functional requirements to enforce the rules.>

6. Other Requirements

<Define any other requirements not covered elsewhere in the SRS. This might include database requirements, internationalization requirements, legal requirements, reuse objectives for the project, and so on. Add any new sections that are pertinent to the project.>

Appendix A: Glossary

<Define all the terms necessary to properly interpret the SRS, including acronyms and abbreviations. You may wish to build a separate glossary that spans multiple projects or the entire organization, and just include terms specific to a single project in each SRS.>

Appendix B: Analysis Models

<Optionally, include any pertinent analysis models, such as data flow diagrams, class diagrams, state-transition diagrams, or entity-relationship diagrams.>

Appendix C: To Be Determined List

<Collect a numbered list of the TBD (to be determined) references that remain in the SRS so they can be tracked to closure.>

Software Requirements Specification
Version 1.1

Web Accessible Alumni Database

Table of Contents

Table of Figures

1.0. Purpose

1.1. Introduction

This Software Requirements Specification provides a complete description of all the functions and specifications of the Jacksonville State University Computing and Information Sciences (CIS) Web Accessible Alumni Database.
The expected audience of this document is the faculty of CIS, including the faculty who will use this system, Dr. Dennis Martin and studio committee members, and the developer. It will also server as a reference for Studio students.

1.2. Scope

The Jacksonville State University Computing and Information Sciences Web Accessible Alumni Database (CISWAAD) is designed to run on the departmental server and to allow alums to fill out a survey form, create a new database entry, update an existing database entry, or contact another alum. The data will be held in an Access database on the departmental server.

1.3. Glossary

Term	**Definition**
Alum	Graduate of Jacksonville State University undergraduate computer science programs.
BDE	Borland Database Engine
CI	Configuration Item
CIS	Computing and Information Sciences
Entry	Alum stored in the Alum Database
Html	Hyper text markup language
IEEE	Institute of Electrical and Electronic Engineers
QA	Quality assurance
SCMP	Software Configuration Management Plan
SDD	Software Design Document
SEI	Software Engineering Institute, Pittsburgh, Pa
SQAP	Software Quality Assurance Plan
SRS	Software Requirements Specification
Survey	Form filled out and submitted by an Alum using the CISWAAB.
Tbd	To be decided
Tbn	To be named
Web Site	A place on the world wide web

1.4. References

[IEEE] The applicable IEEE standards are published in "IEEE Standards Collection," 2001 edition.

[Bruade] The principal source of textbook material is "Software Engineering: An Object-Oriented Perspective" by Eric J. Bruade (Wiley 2001).

[Reaves SPMP] "Software Project Management Plan Jacksonville State University Computing and Information Sciences Web Accessible Alumni Database." Jacksonville State University, 2003.

1.5. Document overview

The remainder of this document is two chapters, the first providing a full description of the project for the owners of the CIS. It lists all the functions performed by the system. The final chapter concerns details of each of the system functions and actions in full for the software developers' assistance. These two sections are cross-referenced by topic; to increase understanding by both groups involved.

2.0. Overall description

The CISWAAD encompasses numerous files and information from the Alumni Database, as well as files on the department server system. This system will be completely web-based, linking to CISWAAD and the remote web server from a standard web browser. An Internet connection is necessary to access the system.

2.1. System environment

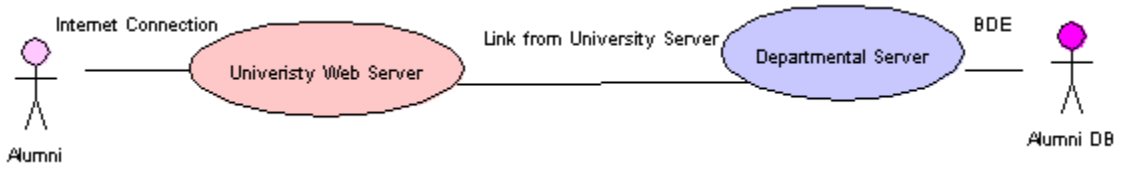


Figure 1 System Design

The CISWAAD web site will be operated from the departmental server. When an Alum connects to the University Web Server, the University Web Server will pass the Alum to the Departmental Server. The Departmental Server will then interact with the Alumni Database through BDE, which allows the Windows type program to transfer data to and from a database.

2.2. Functional requirements definitions

Functional Requirements are those that refer to the functionality of the system, i.e., what services it will provide to the user. Nonfunctional (supplementary) requirements pertain to other information needed to produce the correct system and are detailed separately.

2.3. Use cases

The system will consist of CIS Alumni Home page with five selections.
The first selection is to fill out a survey. The questions on the survey will be created by a designated faculty member. The survey will ask the Alum questions concerning their degree, job experience, how well their education prepared them for their job, and what can the CIS department do to improve itself. This information will be retained on the departmental server and an e-mail will be sent to the designated faculty member.

The second selection is to the Entries section. There are two choices on this page. One choice is to add a new entry. A form is presented to the Alum to be filled in. Certain fields in the form will be required, and list boxes will be used where appropriate. A password typed twice will be required of all new entries.

The second selection of the Entries page is to update an Alum entry. A form will be presented allowing the Alum to enter their year of graduation and then to select themselves from a list. A password will be required before the information will be presented to the Alum to be updated.

The third selection is to search or e-mail an Alum. A form will be presented requiring the requested Alum's year of graduation. The requesting Alum will search a table to see if the requested Alum is in the database, and if so non-sensitive information will be returned. At this

time the Alum can select to e-mail the Alumnus or search for another Alumnus. If the Alum chooses to e-mail the Alumnus a form will be presented for the message to be entered with the sending Alum's name and e-mail. The message, with all necessary information will be forwarded to the requested Alum. The e-mail address of the requested Alum will not be seen by the sending Alum as a privacy measure.
All pages will return the Alum to the CIS Alumni Home Page.

2.3.1. Use Case: Access Alumni Home Page

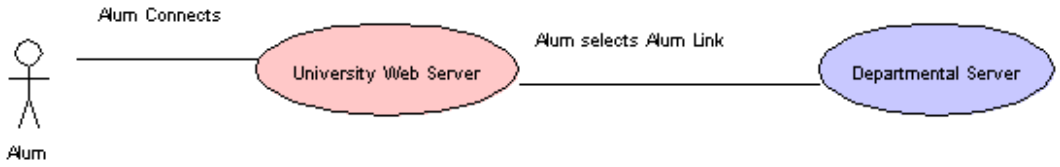


Figure 2 Access Alumni Home Page

Brief Description
The Departmental Web Server is waiting on an Alum to connect.

Initial step-by-step description
For this use case to be initiated, the alum must be connected to the Internet and connected to the University Web Server.
1. The Alum connects to the University Web Server.

2. The Alum selects the Alum link on the CIS home page.

3. The University Web Server passes the Alum to the Alumni Home Page.

Reference SRS 3.2.1

2.3.2. Use Case: Alum Chooses Survey

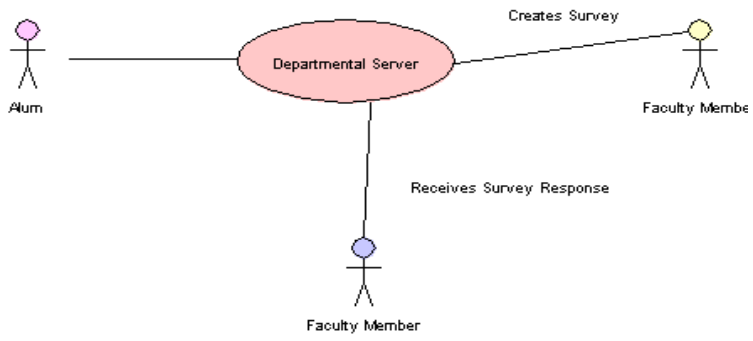


Figure 3 Alum Selects Survey

Brief Description:
The Alum chooses to fill out a survey.

Initial step-by-step description:
For this use case to be initiated the Alum must be connected to the Internet and on the CIS Alumni Home Page.

1. The Alum selects the "Fill out a survey" link.

2. The Departmental Server returns the survey form.

3. The Alum fills in the form.

4. The Alum clicks submit.

5. The Departmental Server retains information in the database designated faculty member will be notified.

6. The Departmental Server returns the Alum to the Alumni Home Page.

Reference SRS 3.2.2

2.3.3. Use Case: Create New Entry

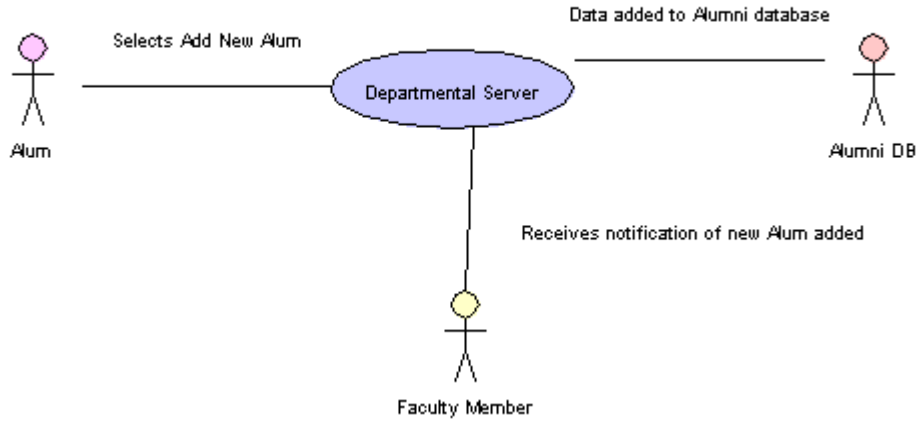


Figure 4 Alum Selects Create a New Entry

Brief Description:
The Alum chooses to create a new entry on the Entries page.

Initial step-by-step description.

For this use case to be initiated the Alum must be connected to the Internet and on the CIS Entries page.

1. The Alum selects the "Add a New Alum" link.

2. The Departmental Server returns the "Add a New Alum Form."

3. The Alum fills in the form.

4. The Alum can choose which fields to make public or private.

5. The Alum clicks submit.

6. The Departmental Server checks to see if all required fields contain data.

7. If all required fields contain data the Departmental Server adds the data to the Alum Database.

8. If a required filed is empty the Departmental Server returns the form to the Alum with a message.

9. The Departmental Server returns the Alum to the Alumni Home Page.

Reference: SRS 3.2.3

2.3.4. Use Case: Update an Entry.

Figure 5 Alum Selects Update an Entry

Brief Description:

The Alum chooses to update an existing entry in the Alumni Database.

Initial step-by-step description:

For this use case to be initiated the Alum must be connected to the Internet and on the CIS Entries page.

1. The Alum chooses the "Update Alumni Information" option.

2. The Department Server presents the Alum with a form.

3. The Alum fills in the year of graduation.

4. The Departmental Server returns a form with all graduates from that year.

5. The Alum checks the correct graduate and enters his/her password

6. The Departmental Server searches the Alumni Database for the Alum name and password.

7. The Departmental Server returns the Alum's data if the password matches.

8. If the password does not match the Departmental Server returns an error message and returns the Alum to the previous page.

9. The Alum changes the appropriate fields and clicks submit.

10. The Departmental Server replaces the old data with the new.

11. The Departmental Server returns the Alum to the CIS Alumni Home Page.

Reference: SRS 3.2.4

2.3.5. Use Case: Search for an Alumni/E-mail and Alumni

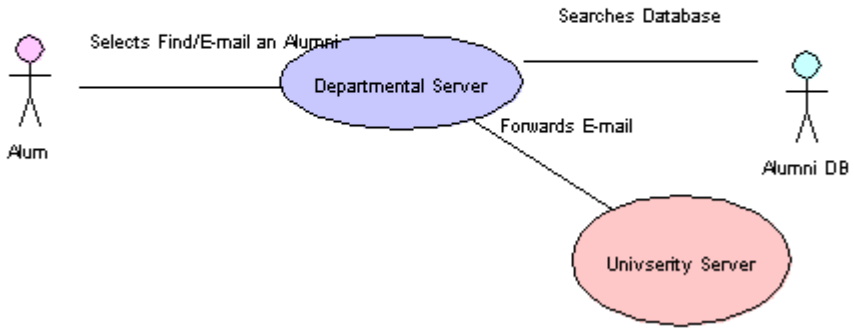


Figure 6 Alum Selects Search/E-mail an Alum

Brief description:

The Alum chooses to search/e-mail Alum.

Initial step-by-step description:

For this use case to be initiated the Alum must be connected to the Internet and on the Alumni CIS Home Page.

1. The Alum chooses "Search for an Alum."

2. The Departmental Server presents a form requesting the year of graduation.

3. The Alum fills in the form and clicks submit.

4. The Departmental Server queries the Alumni Database for the requested information.

5. The Departmental Server returns all Alums that graduated that year.

6. The Alum chooses “E-mail an Alum.”

7. The Departmental Server presents a form.

8. The Alum fills in the form.

9. The Departmental Server checks the to see if the required fields are not empty.

10. The Departmental Server queries the Alumni Database for the particular Alum.

11. If the Alum requested is not in the Alumni Database, if there is no e-mail address for the requested Alum, or if the Alum has requested that no e-mails be forwarded, the Departmental Server will return a message that the requested Alum can not be e-mailed.

12. If the Alum requested is in the Alumni Database and there is an e-mail address the message along with the requested Alum’s e-mail will be forwarded to the requested Alum.

13. The Departmental Server will return a message and return the Alum to the CIS Alumni Home Page.

Reference: SRS 3.2.5

2.4. Non-functional requirements

There are requirements that are not functional in nature. Specifically, these are the constraints the system must work within.
The web site must be compatible with both the Netscape and Internet Explorer web browsers. This system will use the same type of Internet security presently being used by Jacksonville State University.

3.0. Requirement specifications

3.1. External interface specifications

None

3.2. Functional Requirements

3.2.1. Access Alumni Home Page

Use Case Name:	Access Alumni Home Page
Priority	Essential
Trigger	Menu selection
Precondition	Alum is connected to the Internet and on the CIS home page
Basic Path	1. University Web Server sends the Alum to the Departmental Server. 2. The Departmental Server presents the Alum with the Alumni Home Page.
Alternate Path	N/A
Postcondition	The Alum is on the Alumni Home Page
Exception Path	If there is a connection failure the Departmental Server returns to the wait state
Other	
Reference	SRS 2.3.1

3.2.2. Survey

Use Case Name:	Survey
Priority	Essential
Trigger	Selects
Precondition	The Alum is connected to the Internet and on the CIS Alumni Home Page
Basic Path	1. The Departmental Server presents the Alum with a form. 2. The Alum fills in the form and click submit 3. The Departmental Server checks to see if all required fields are not empty. 4. If the required fields are not empty, the Departmental Server creates a new record in then Survey Table of the Alumni Database. 5. If any of the required fields are empty, the Departmental Server returns a message and returns the Alum to the Survey form.

	6. The Departmental Server returns the Alum to the Alumni Home Page
Alternate Path	N/A
Postcondition	The survey record is created in the Survey Table of the Alumni Database.
Exception Path	1. If the connection is terminated before the form is submitted, the fields are all cleared and the Departmental Server is returned to the wait state.
Other	
Reference:	SRS 2.3.2

3.2.3. Create a new entry

Use Case Name:	Create a new entry
Priority	Essential
Trigger	Menu selection
Precondition	The Alum must be connected to the Internet and on the CIS Entries page.
Basic Path	1. The Alum clicks on add a new entry. 2. The Departmental Server returns a form. 3. The Alum fills in the form and clicks submit. 4. The Departmental Server checks to see if any required field is empty. 5. If any required field is empty the Departmental Server will send a message and return the Alum to the new entry form page. 6. If no required field is empty the Departmental Server will create a new record in the Alumni Table in the Alumni Database, and return the Alum to the CIS Alumni Home Page. 7. The Alum may select Cancel. 8. If the Alum selects Cancel, the form is cleared and the Alum is returned to the CIS Alumni Home page.
Alternate Path	N/A
Postcondition	A record is created in the Alumni Table of the Alumni Database.
Exception Path	1. If the connection is terminated before the form is submitted, the fields are cleared and the Departmental Server is returned to the wait state.

	2. If the connection is terminated after the form is submitted, but before the Alum is returned to the CIS Alumni Home Page, the record is created in the Alumni Table of the Alumni Database.
Other	
Reference:	SRS 2.3.3

3.2.4 Update an Entry

Use Case Name:	Update an Entry
Priority	Essential
Trigger	Menu selection
Precondition	The Alum must be connected to the Internet and on the CIS Entries Page.
Basic Path	1. The Alum clicks on update an entry link. 2. The Departmental Server returns a form. 3. The Alum enters his/her year of graduation. 4. The Departmental Server queries the Alumni Database for that particular year and returns a table of all graduates from that year in a form with radio buttons and requesting their password. 5. If the password does not match the Departmental Server returns a message and allows the Alum to try again. 6. If after 3 tries the password does not match, the Departmental Server will return a message telling the Alum to contact the CIS designated faculty member to receive their password. 7. If the password matches go to 8. 8. The Departmental Server returns a form with the data for that Alum in it and a message to update the data they wish and click submit. 9. The Departmental Server with replaces the old data with the new data and returns the Alum to the CIS Alumni Home Page.
Alternate Path	If after three attempts to match the name and password the Departmental Server will return a message and block the Alum from the update section.
Postcondition	The record in the Alumni Table of the

	Alumni Database has been updated and the Alum is returned to the CIS Alumni Home Page.
Exception Path	1. If the connection is terminated before the form is submitted, the fields are cleared and the Departmental Server is returned to the wait state. 2. If the connection is terminated after the form is submitted, but before the Alum is returned to the CIS Alumni Home Page, the record in the Alumni Table of the Alumni Database is updated and the Departmental Server is returned to the wait state
Other	
Reference:	SRS 2.3.4

3.2.5. Search for an Alumni/E-mail an Alumni

Use Case Name:	Search for an Alumni
Priority	If time permits.
Trigger	Menu selection
Precondition	The Alum is connected to the Internet and on the CIS Alumni Home Page.
Basic Path	1. The Alum clicks on e-mail an alumni link. 2. The Departmental Server returns a form. 3. The Alum fills in the form and clicks submit. 4. The Departmental Server checks to see if any required fields are empty. 5. If any required fields are empty the Departmental Server returns a message and the form. 6. If none of the required fields are empty the Departmental Server queries the Alumni Database for the requested Alum's entry. 7. The Departmental Server returns the non-private information on the requested Alum and a message stating if the requested Alum will accept e-mails. 8. If the requested Alum is not in the Alumni Database, the Departmental Server returns a message and the Alum is

	returned to the CIS Home Page. 9. If the requested Alum will accept e-mails, the Alum can select E-mail this Alum. 10. If not the Alum can select Search for another Alum or return to CIS Alumni Home Page. 11. If the Alum chooses to Search for another Alum go to step 2. 12. If the Alum selects return to CIS Alumni Home Page the Departmental Server returns the Alum to the CIS Alumni Home Page. 13. The Departmental Server presents the Alum with a form to fill out and a place for the message. 14. The Alum selects send. 15. The Department Server will forward the e-mail with all necessary information to the requested Alum. 16. The Departmental Server returns a message and returns the Alum to the CIS Alumni Home Page
Alternate Path	N/A
Postcondition	The Alum receives the information on the requested Alum, receives e-mail confirmation message, or is returned to the CIS Alumni Home Page
Exception Path	1. If the connection is terminated before the information is returned, the Departmental Server is returned to the wait state. 2. If the connection is terminated after the information is returned, the Departmental Server is returned to the wait state
Other	
Reference:	SRS 2.3.5

3.3. Detailed non-functional requirements

Attribute Name	Attribute Type	Attribute Size
LastName*#	String	30
FirstName*#	String	30
MaidenName*#	String	30

Address1*#	String	50
Address2#	String	50
City*#	String	30
State*#	String	2
Zip*#	Int	6
Year*#	Int	4
AdditionalDegrees#	String	50
Spouse#	String	30
Children#	String	50
CurrentEmployment#	String	50
EmailAddress#	String	20
ReceiveEmails#^	Boolean	1
Password*#	String	10
EntireRecordVisible*^	Boolean	1

Fields marked with an '*' are required fields. Fields marked with a '#' can be visible or not visible and is determined by the Alum. Fields marked with a '^' are never visible to anyone other than the Alum.

The questions that are used on the survey form will be initially created by a designated faculty member. The questions will be stored in the Question Record of the Survey Table of the Alumni Database. The responses to these questions will be stored in a record in an Answers record in the Survey Table of the Alumni Database.

Hardware:	Departmental Server
Operation System	Window 98 or above
Internet Connection	Existing telephone lines
Code Standard	The web pages will be coded in html by using Front Page.
	The forms will be done in Java Server Pages.
	The connection to the Alumni Database will be done with Windows BDE.
	Each page of the web site will be fully documented.
Performance	The system should generate the records in the appropriate table of the Alumni Database 100% of the time.

3.4. System Evolution

In the future this system will be update to allow students from the Computer Masters Program to join. If time does not permit the search/e-mail section can be done, possibly by another Master Studio student. A report generated by the system of the responses to the survey could be another addition to the CISWAAD in the future.

4.0. Index

Assignment No. 2

Title: Prepare Use Case Model

Identify Major Use Cases, Identify actors.
Write Use Case specification for all major Use Cases.
Draw detail Use Case Diagram using UML2.0 notations.

Date of Completion		**Marks for Regularity (2)**	
		Marks for Presentation (3)	
Remark		**Marks for Understanding (5)**	
Sign. of Staff		**Total Marks (10)**	

Prof. R.S. Badodekar **414459 : Computer Laboratory VIII**

Assignment No - 2. Prepare Use Case Model

Introduction:

- A use case diagram describes how a system interacts with outside actors.
- It is a graphical representation of the interaction among the elements and system.
- Each use case representation a piece of functionality that a system provides to its user.
- Use case identifies the functionality of a system.
- Use case diagram allows for the specification of higher level user goals that the system must carry out.
- These goals are not necessarily to tasks or actions, but can be more general required functionality of the system.
- You can apply use case to capture the intended behavior of the system you are developing, without having to specify how that behavior is implemented.
- A use case diagram at its simplest is a representation of a user's interaction with the system and depicting the specifications of a use case.
- A use case diagram contains four components.
  i. The boundary, which defines the system of interest in relation to the world around it.
  ii. The actors, usually individuals involved with the system defined according to their roles.
  iii. The use cases, which the specific roles are played by the actors within and around the system.
  iv. The relationships between and among the actors and the use cases.

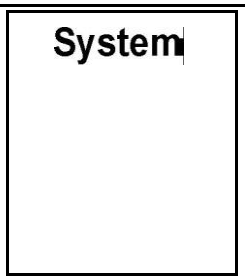
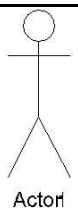
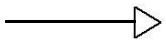
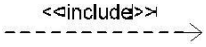
Purpose:

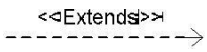
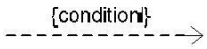
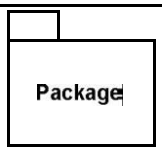
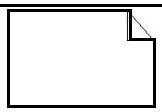
- The main purpose of the use case diagram is to capture the dynamic aspect of a system.
- Use case diagram shows, what software is suppose to do from user point of view.
- It describes the behavior of system from user's point.
- It provides functional description of system and its major processes.
- Use case diagram defines the scope of the system you are building.

When to Use: Use Cases Diagrams

- Use cases are used in almost every project.
- They are helpful in exposing requirements and planning the project.
- During the initial stage of a project most use cases should be defined.

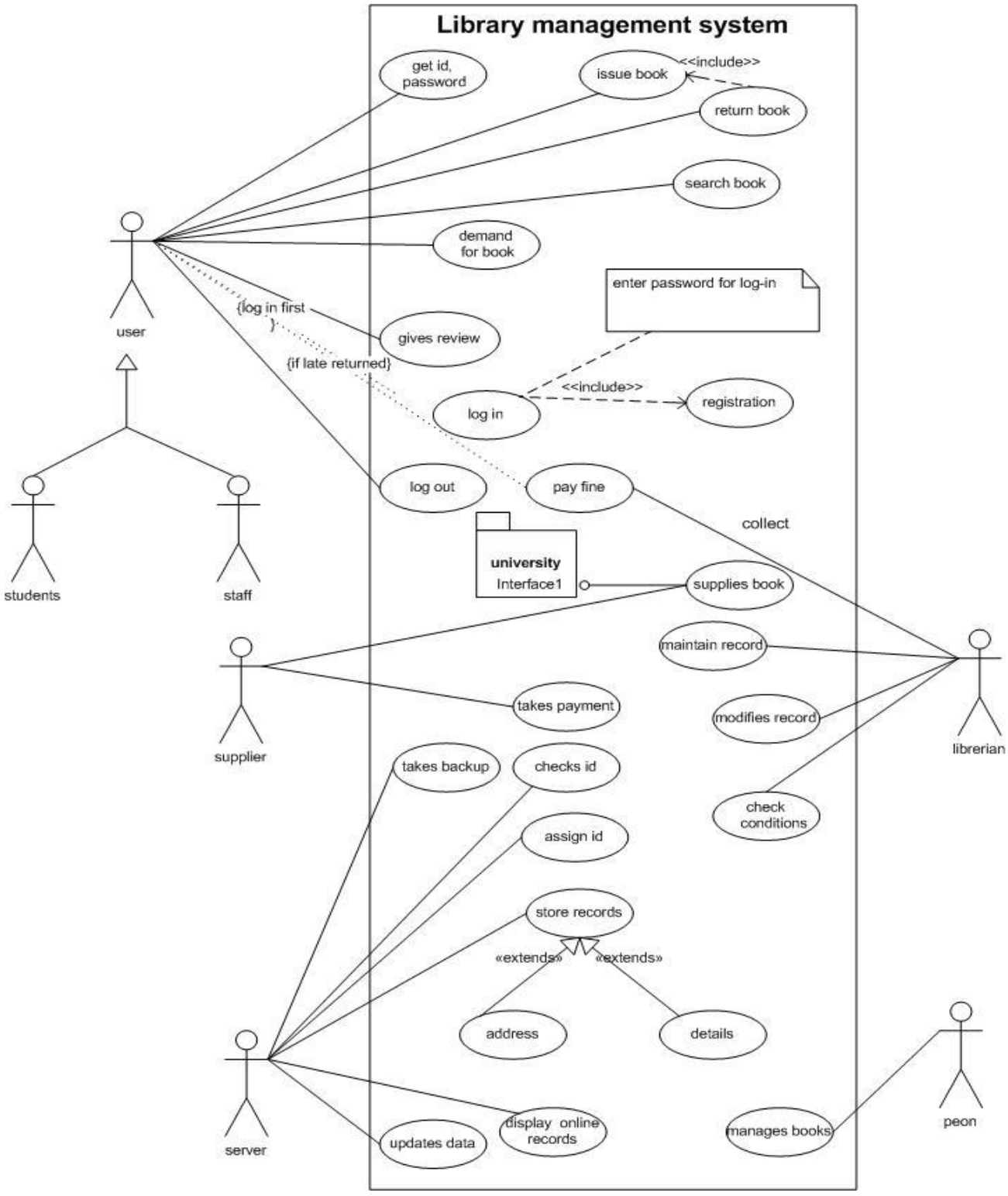
Use Case Notations

No.	Name	Notation	Description
1	System boundary		The scope of a system can be represented by a system boundary. The use cases of the system are placed inside the system boundary, while the actors who interact with the system are put outside the system. The use cases in the system make up the total requirements of the system.
2	Use case		A use case represents a user goal that can be achieved by accessing the system or software application.
3	Actor		Actors are the entities that interact with a system. Although in most cases, actors are used to represent the users of system, actors can actually be anything that needs to exchange information with the system. So an actor may be people, computer hardware, other systems, etc. Note that actor represent a role that a user can play, but not a specific user.
4	Association		Actor and use case can be associated to indicate that the actor participates in that use case. Therefore, an association corresponds to a sequence of actions between the actor and use case in achieving the use case.
5	Generalization		A generalization relationship is used to represent inheritance relationship between model elements of same type.
6	Include		An include relationship specifies how the behavior for the inclusion use case is inserted into the behavior defined for the base use case.

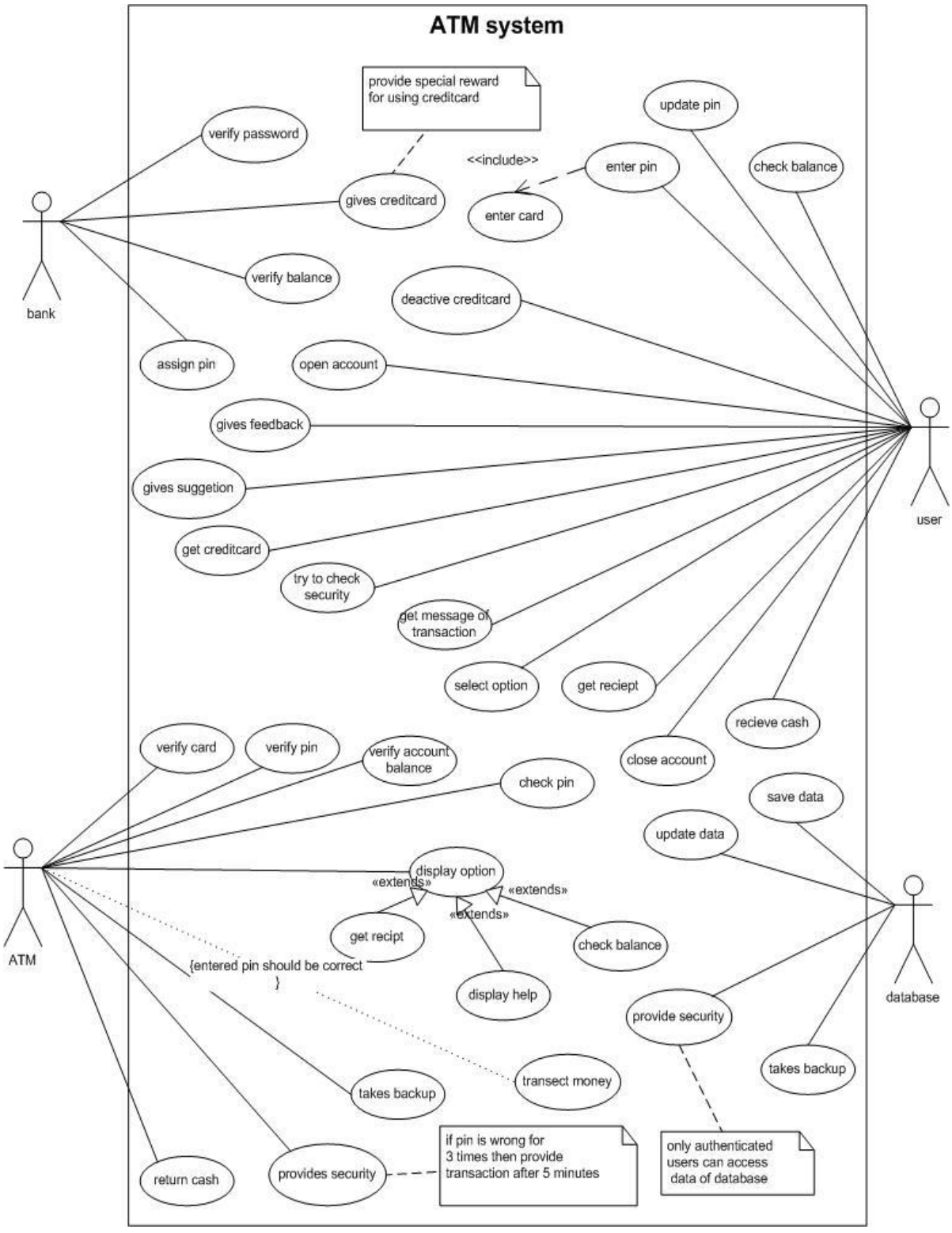
7	Extends		An extend relationship specifies how the behavior of the extension use case can be inserted into the behavior defined for the base use case.
8	Constraint		Show condition exists between actors an activity.
9	Package		Package is defined as collection of classes. Classes are unified together using a package.
10	Interface		Interface is used to connect package and use-case. Head is linked with package and tail linked with use-case.
11	Note		Note is generally used to write comment in use-case diagram.
12	Anchor		Anchor is used to connect a note the use case in use case diagram

Examples:

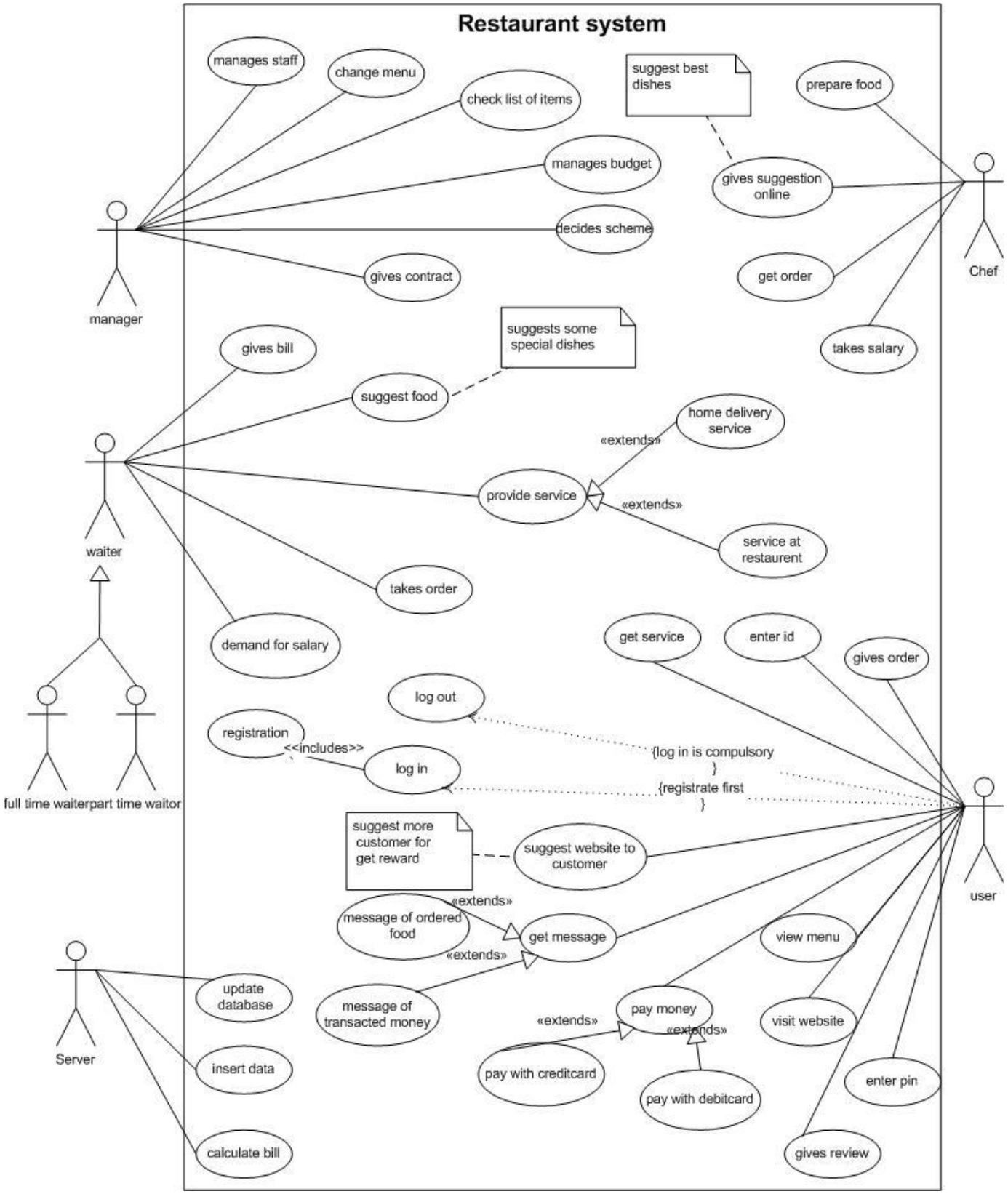
Draw Use case diagram for Library management System



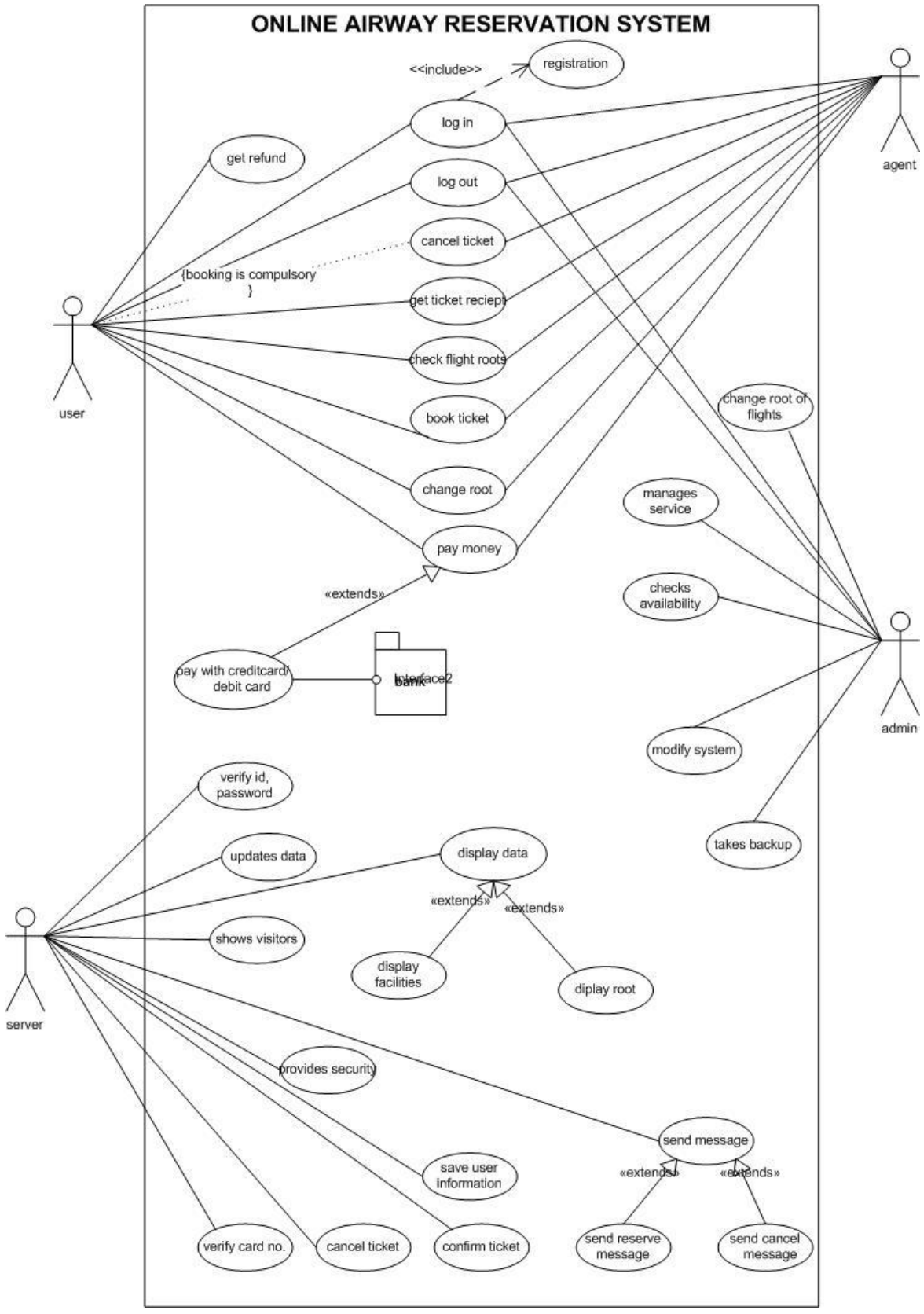
Draw Use-case Diagram For ATM System



Draw Use-case diagram for online restaurant system



Draw Use-case for Online Reservation System



Draw Use-case diagram for online shopping system

Assignment No. 3

Title: Prepare Activity Model

Identify Activity states and Action states.
Draw Activity diagram with Swim lanes using UML2.0 Notations for major Use Cases

Date of Completion		**Marks for Regularity (2)**	
		Marks for Presentation (3)	
Remark		**Marks for Understanding (5)**	
Sign. of Staff		**Total Marks (10)**	

Prof. R.S. Badodekar **414459 : Computer Laboratory VIII**

Assignment No 3 - Activity Diagram

Introduction

- An activity diagram is a type of flow chart with additional support for parallel behavior.
- This diagram explains overall flow of control.
- Activity diagram is another important diagram in UML to describe dynamic aspects of the system.
- Activity diagram is basically a flow chart to represent the flow from one activity to another activity
- The activity can be described as an operation of the system.
- The control flow is drawn from one operation to another. This flow can be sequential, branched or concurrent. This distinction is important for a distributed system.
- Activity diagrams deals with all type of flow control by using different elements like fork, join etc.

Purpose

- Contrary to use case diagrams, in activity diagrams it is obvious whether actors can perform business use cases together or independently from one another.
- Activity diagrams allow you to think functionally.

When to use : Activity Diagrams

- Activity diagrams are most useful when modeling the parallel behavior of a multithreaded system or when documenting the logic of a business process.
- Because it is possible to explicitly describe parallel events, the activity diagram is well suited for the illustration of business processes, since business processes rarely occur in a linear manner and often exhibit parallelisms.
- This diagram is useful to investigate business requirements at a later stage.
- An activity diagram is drawn from a very high level. So it gives high level view of a system. This high level view is mainly for business users or any other person who is not a technical person.
- This diagram is used to model the activities which are nothing but business requirements.
- So the diagram has more impact on business understanding rather *implementation details*.

Activity Diagram Notations

No.	Name	Symbol	Description
1.	Activity	Activity	Represent individual activity of system.
2.	Transition		Represents flow of data from one activity to another.
3.	Decision		Decision node is a control node that accepts tokens on one or more incoming edges and selects outgoing edge from two or more outgoing flows. The notation for a decision node is a diamond-shaped symbol.
4.	Initial activity		Initial node is a control node at which flow starts when the activity is invoked. Activity may have more than one initial node. Initial nodes are shown as a small solid circle.
5.	Final activity		Final node is a control final node that stops all flows in an activity. Activity final nodes are shown as a solid circle with a hollow circle inside. It can be thought of as a goal notated as "bull's eye," or target.
6.	Fork		A fork in the activity diagram has a single incoming transition and multiple outgoing transitions exhibiting parallel behavior.The incoming transition triggers the parallel outgoing transitions.
7.	Join		A join in the activity diagram synchronizes the parallel behavior started at a fork. Join ascertains that all the parallel sets of activities (irrespective of the order) are completed before the next activity starts. It is a synchronization point in the diagram. Each fork in an activity diagram has a corresponding join where the parallel behavior terminates.

Examples

Activity Diagram for Library Management System

Issue and return book

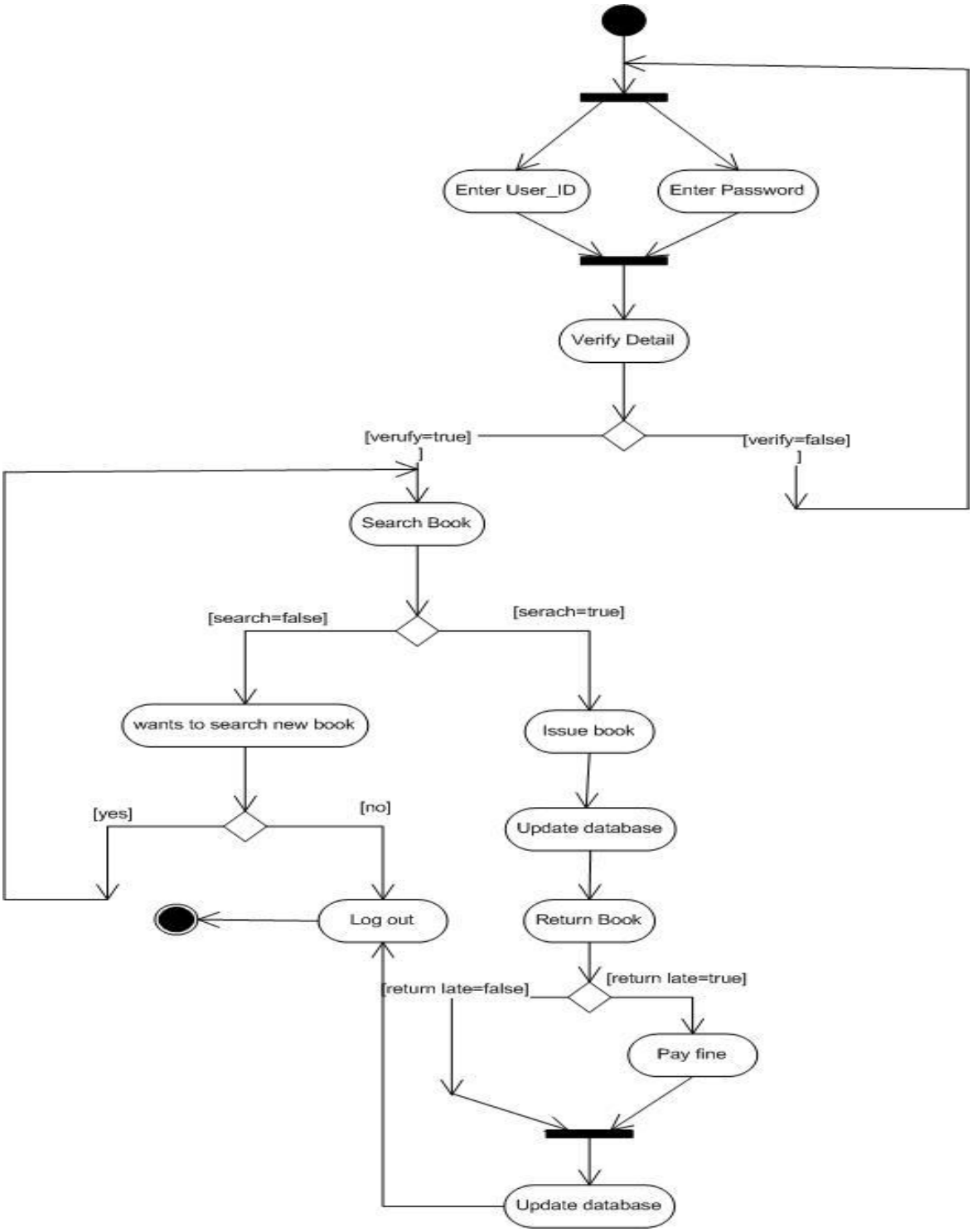
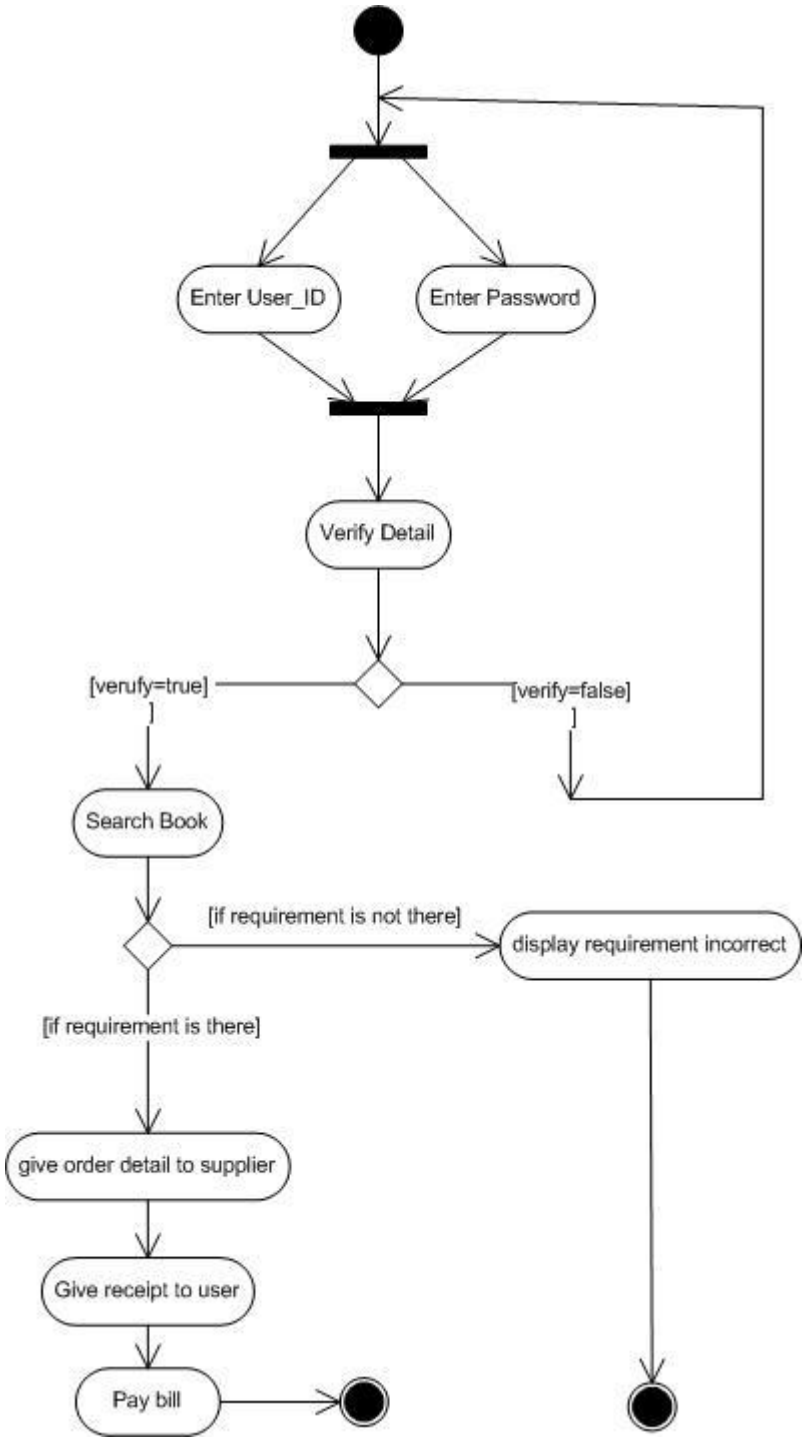
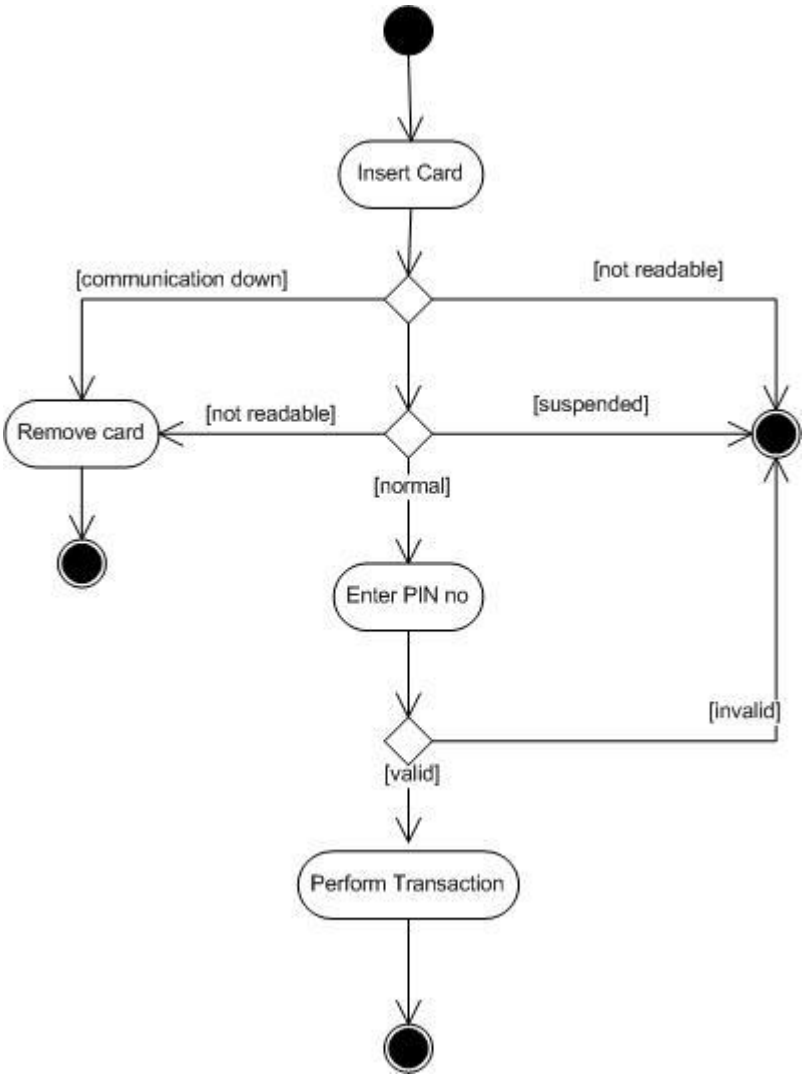


Diagram for ordering new book

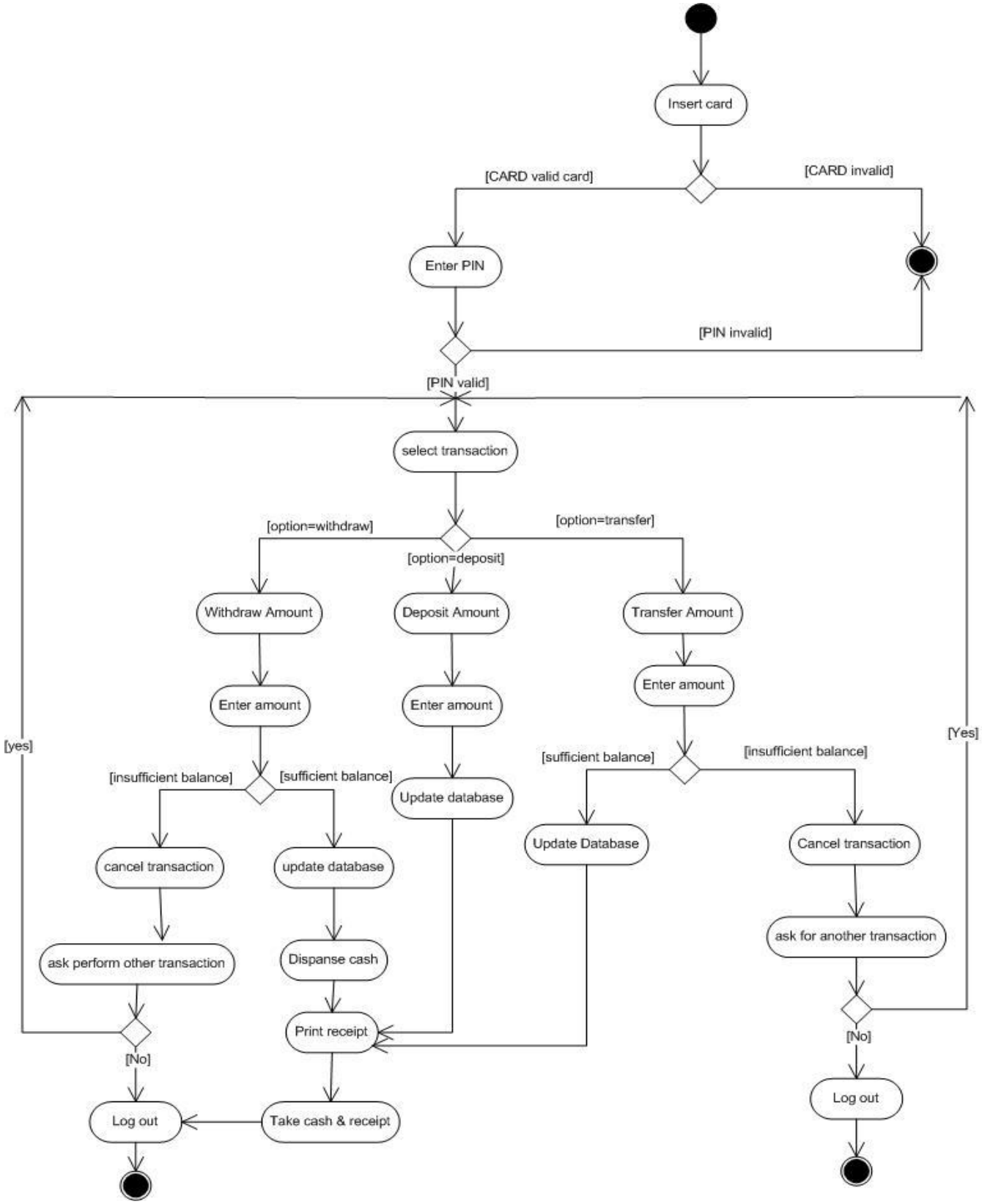


Activity diagram for ATM

Verify PIN number

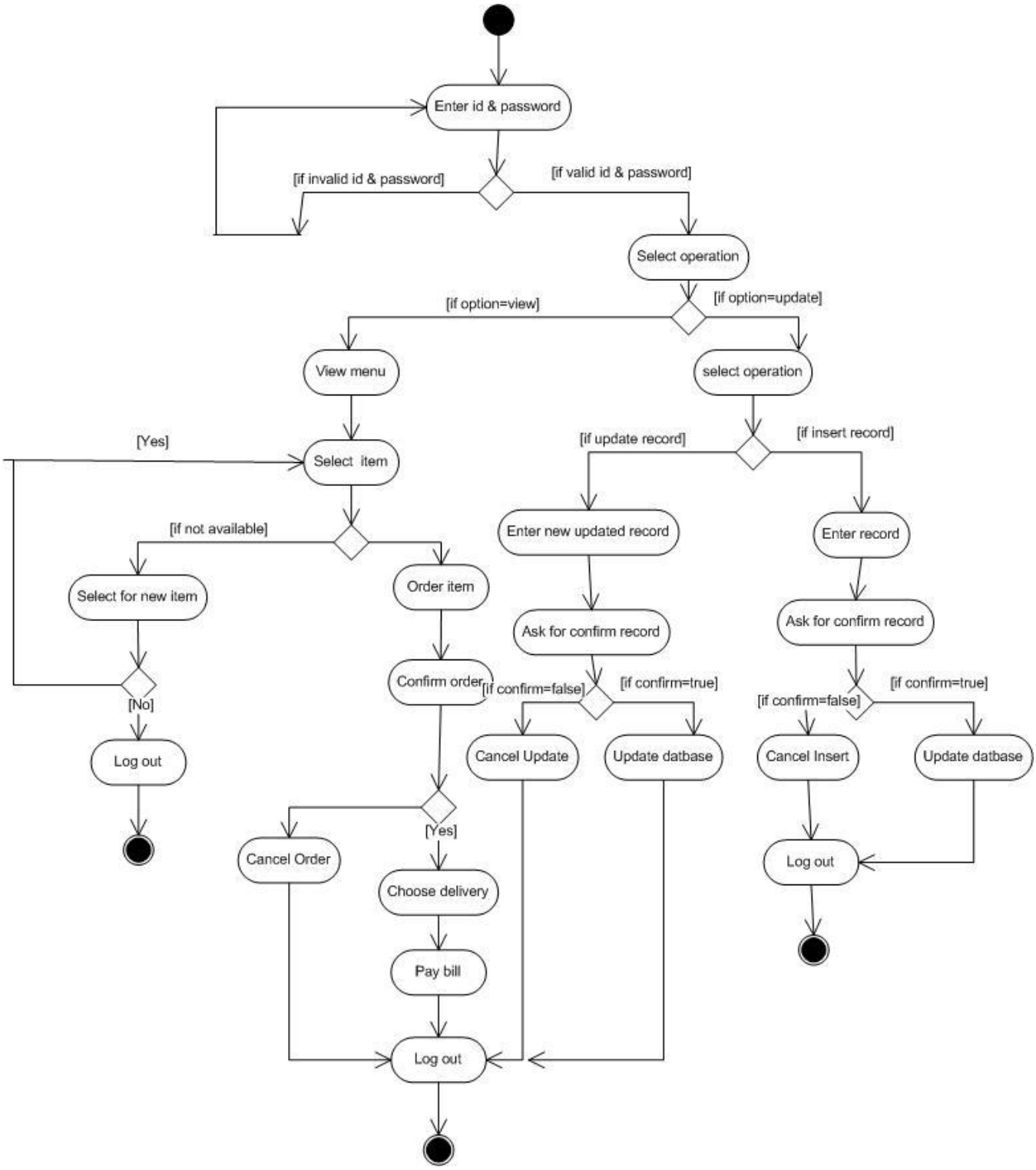


Transaction

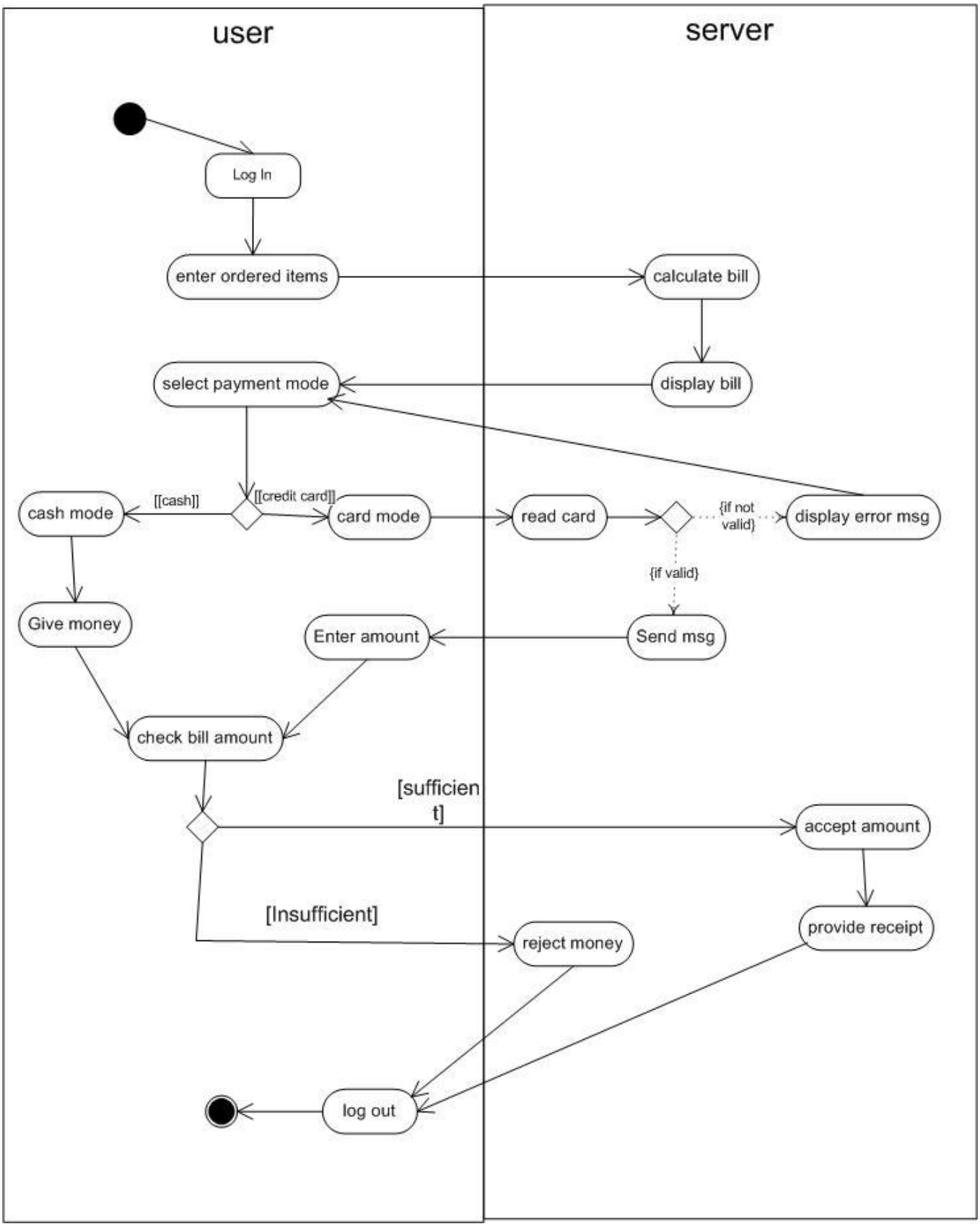


Activity Diagram for Online Restaurant Management System

Place order

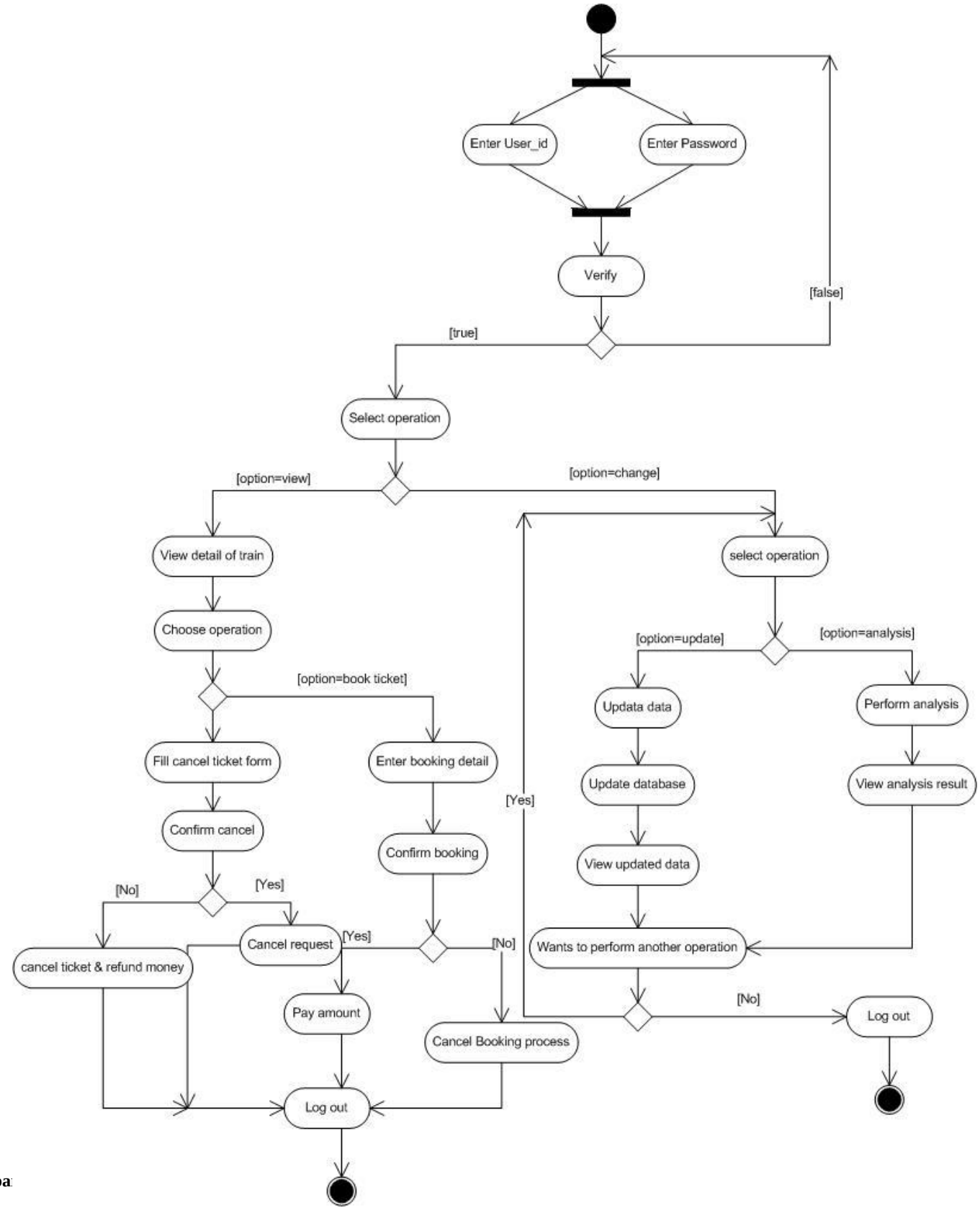


Payment


user
server
Log In
enter ordered items
calculate bill
select payment mode
display bill
cash mode
[[cash]]
[[credit card]]
card mode
read card
{if not valid}
display error msg
{if valid}
Give money
Enter amount
Send msg
check bill amount
[sufficient]
accept amount
[Insufficient]
reject money
provide receipt
log out

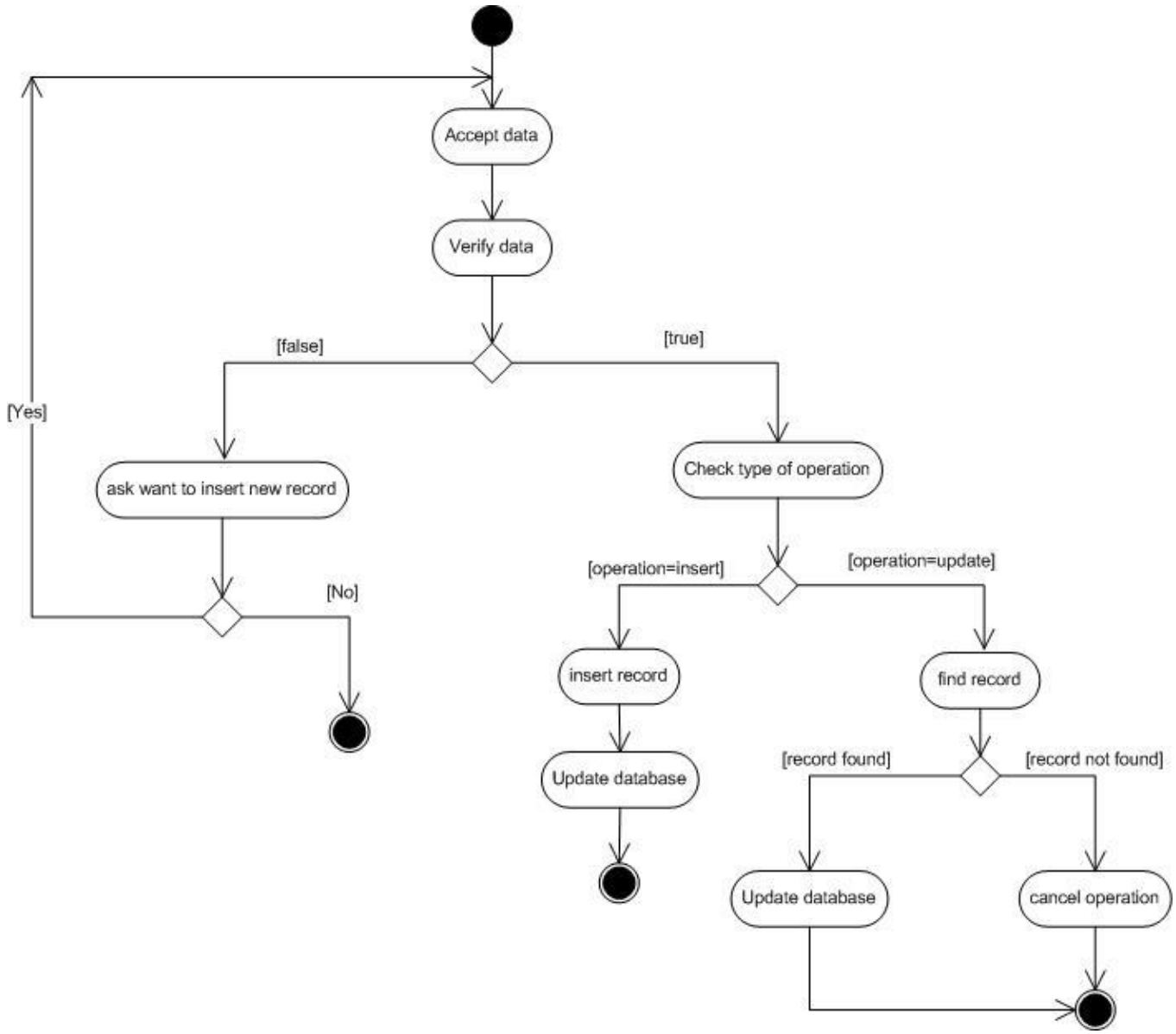
Activity Diagram for Online Reservation System

Booking Process



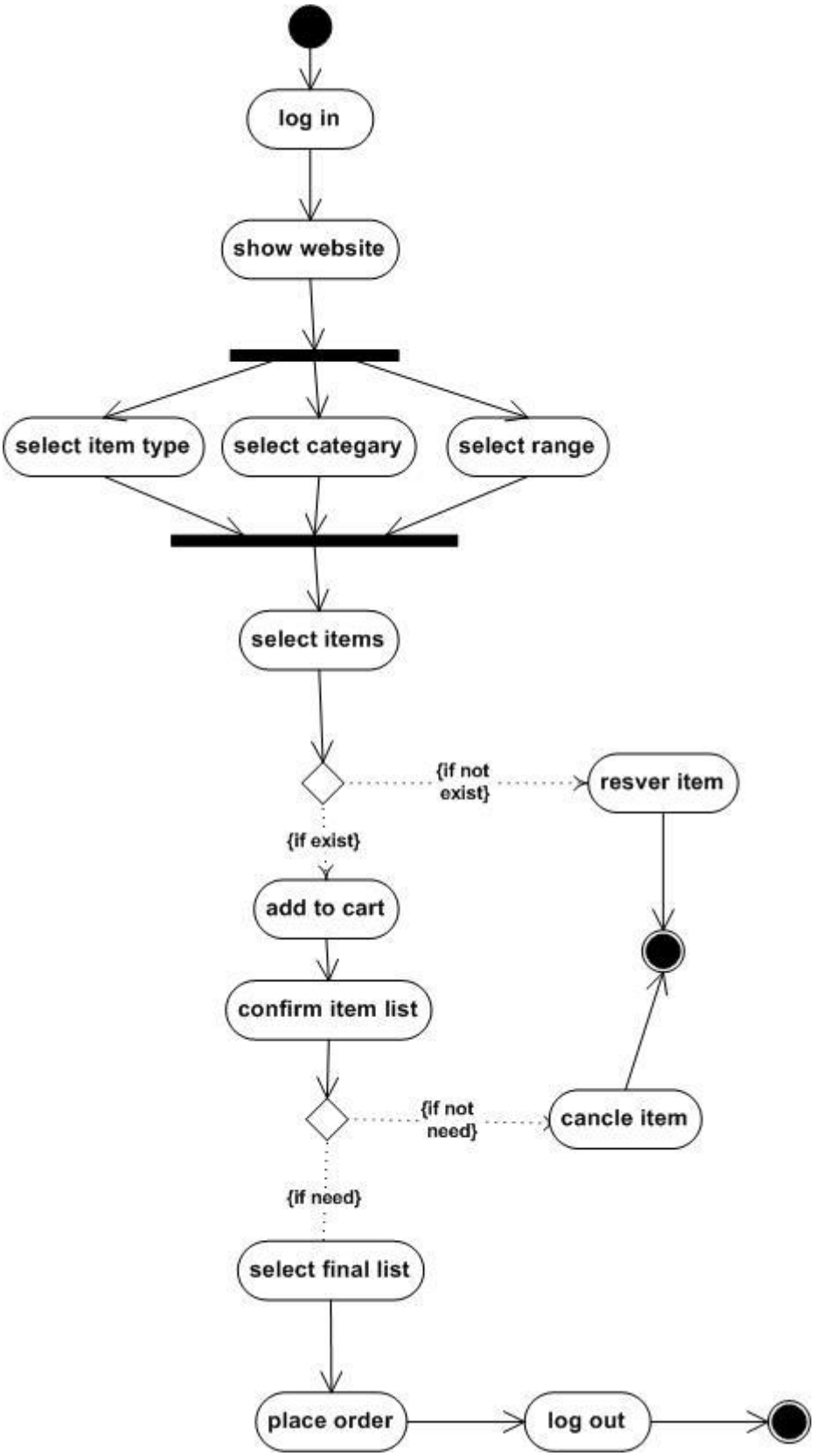
Depa

Server Operation



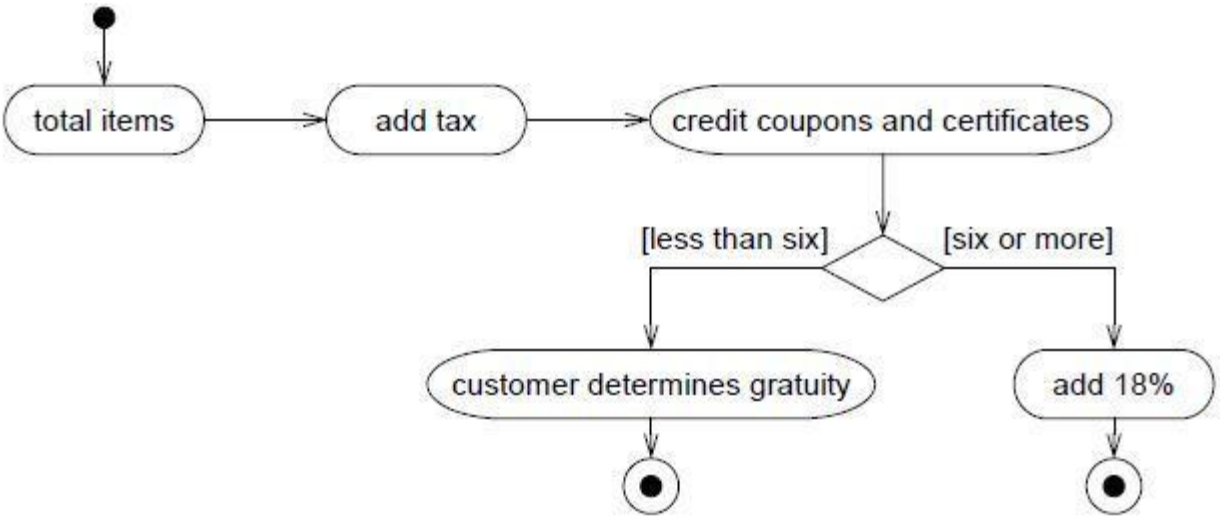
Activity diagram for Online Shopping

Purchase Product



Additional Diagram: Activity Diagram

1. **Prepare an activity diagram for computing a restaurant bill. There should be a charge for each delivered item. The total amount should be subject to tax and a service charge of 18% for groups of six of more. For smaller groups, there should be a blank entry for a gratuity according to the customer's discretion. Any coupons or gift certificates submitted by the customer should be subtracted.**



Assignment No. 4

Title: Prepare Analysis Model-Class Model

Identify Analysis Classes and assign responsibilities. Prepare Data Dictionary. Draw Analysis class Model using UML2.0 Notations. Implement Analysis class Model-class diagram with a suitable object oriented language

Date of Completion		**Marks for Regularity (2)**	
		Marks for Presentation (3)	
Remark		**Marks for Understanding (5)**	
Sign. of Staff		**Total Marks (10)**	

Prof. R.S. Badodekar **414459 : Computer Laboratory VIII**

Assignment No. 4 Prepare Analysis Model-Class Model

Introduction

- The class diagram is a static diagram.
- A class model captures the static structure of a system by characterizing the objects in the system, the relationship between the objects, and the attributes and operations for each class of objects.
- The class diagram can be mapped directly with object oriented languages.
- The class model is the most important amonethe three models.
- Class diagram provide a graphical notation for modeling classes and their relationship.
- They are concise, easy to understand, and work well in practice.
- Class diagrams are the backbone of almost every object-oriented method including UML.
- They describe the static structure of a system.

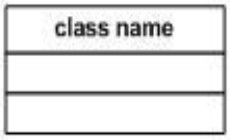
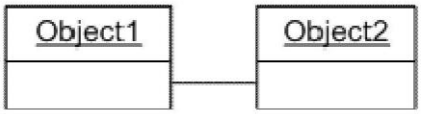
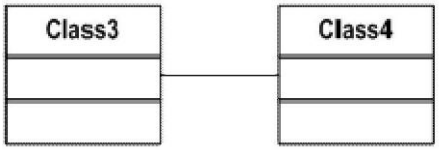
Purpose

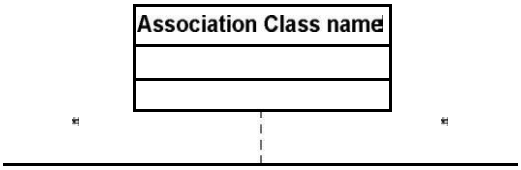
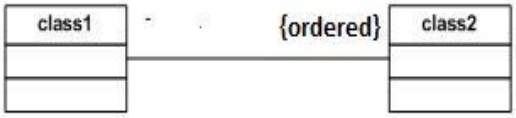
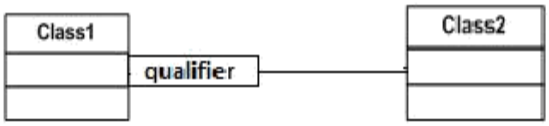
- Analysis and design of the static view of an application.
- Describe responsibilities of a system.
- Base for component and deployment diagrams.

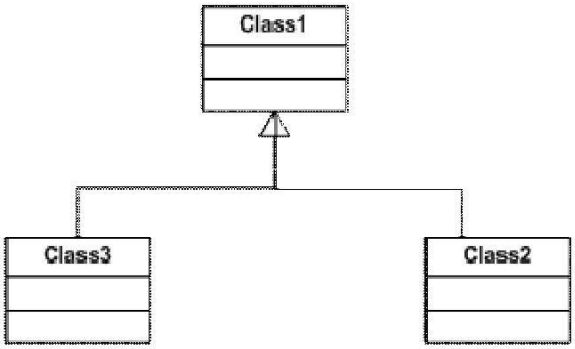
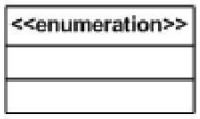
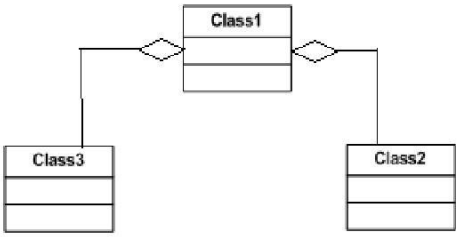
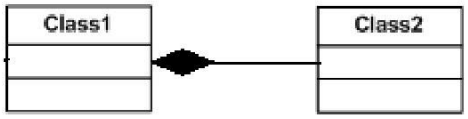
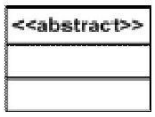
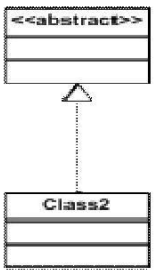
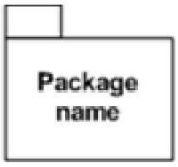
When to use : Class Diagram

- Useful for Forward and Reverse engineering.
- Class diagrams are useful both for abstract modeling and for designing actual programs.
- Developer uses class diagram for implementation decision.
- Business analysts can use class diagrams to model systems from the business perspective.

Class Diagram Notations

Sr. No.	Name	Symbol	Meaning
1.	Class		Class is an entity of the class diagram. It describes a group of objects with same properties& behavior.
2.	Object		An object is an instance or occurrence of a class.
3.	Link		A link is a physical or conceptual connection among objects
4.	Association		An association is a description of a links with common structure & common semantics.
5.	Multiplicity	Ex. 1 to 1 1 to * * to * * to 1 1 to 0….2 	Multiplicity specifies the number of instances of one class that may relate to a single instance of an associated class. It is a constraint on the cardinality of a set.

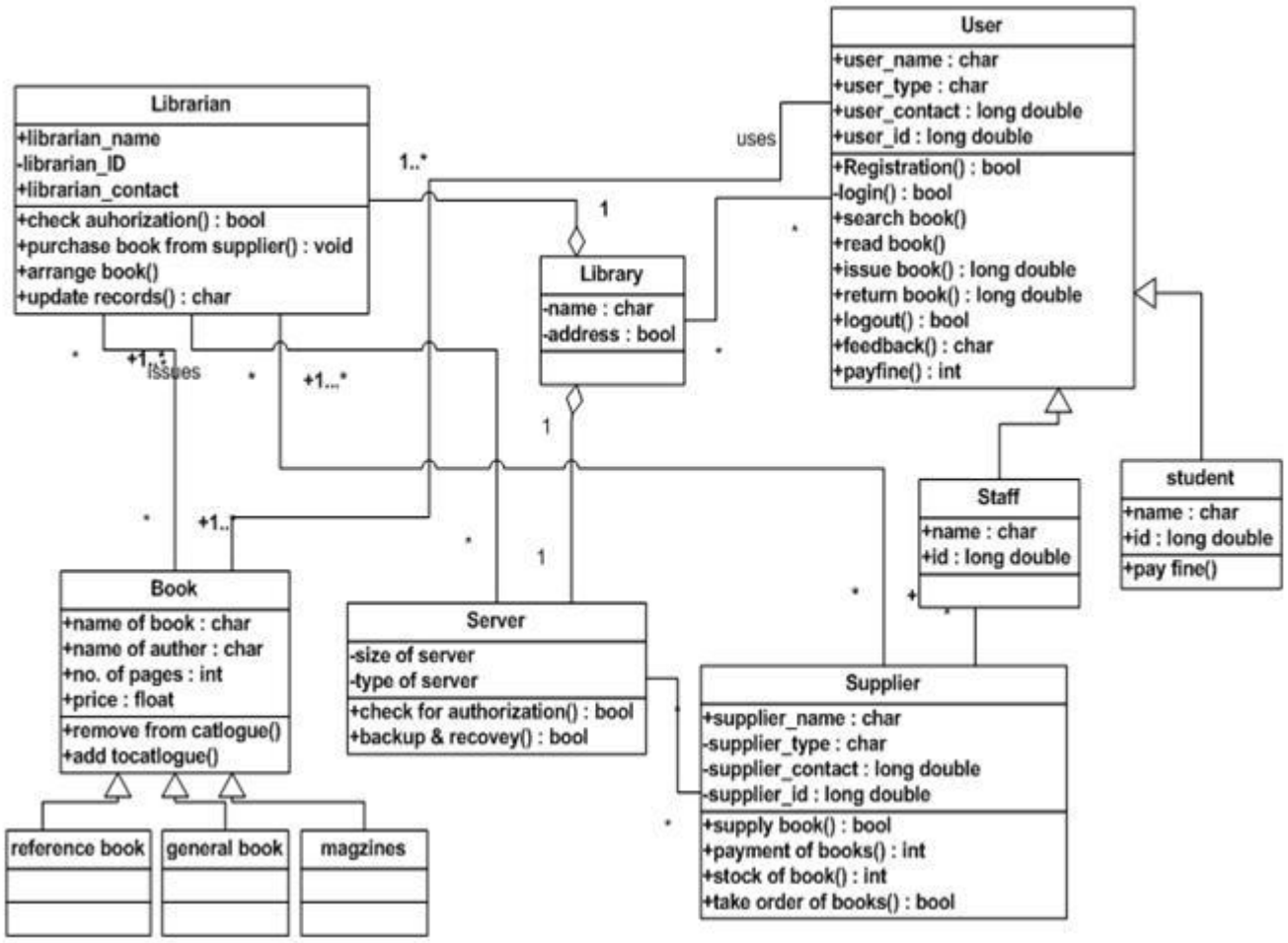
6.	Association class		It is an association that is a class which describes the association with attributes.
7.	cardinality		It describes the count of elements from collection.
8.	ordering		It is used to indicate an ordered set of objects with no duplication allowed.
9.	bag		A bag is a collection of unordered elements with duplicates allowed.
10.	sequence		A sequence is an ordered collection of elements with duplicates allowed.
11.	qualified association		Qualification increases the precision of a model. It is used to avoid many to many multiplicities and it converts into one to one multiplicity.

12.	generalization		Generalization organizes classes by their super-class and sub-class relationship.
13.	enumeration		An enumeration is a data type that has a finite set of values.
14.	aggregation		It is a strong form of association in which an aggregate object is made of constituent parts.
15.	composition		It is a form of aggregation. Composition implies ownership of the parts by the whole.
16.	Abstract class		It is a class that has no direct instances.
17.	Concrete class		It is a class that is intangible; it can have direct instances. Class-2 is example of concrete class
18.	package		A package is a group of elements with common

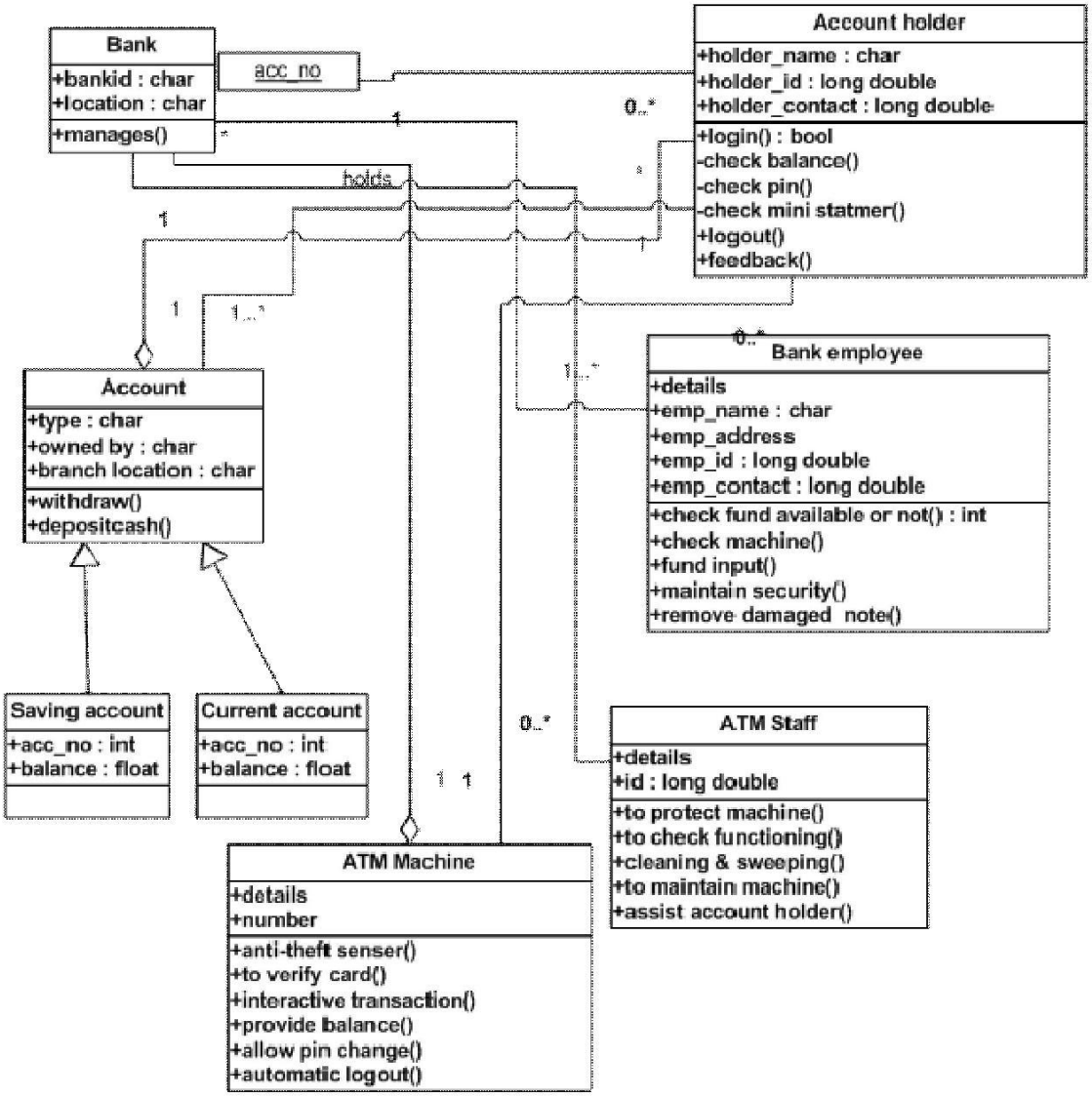
			theme.

Examples:

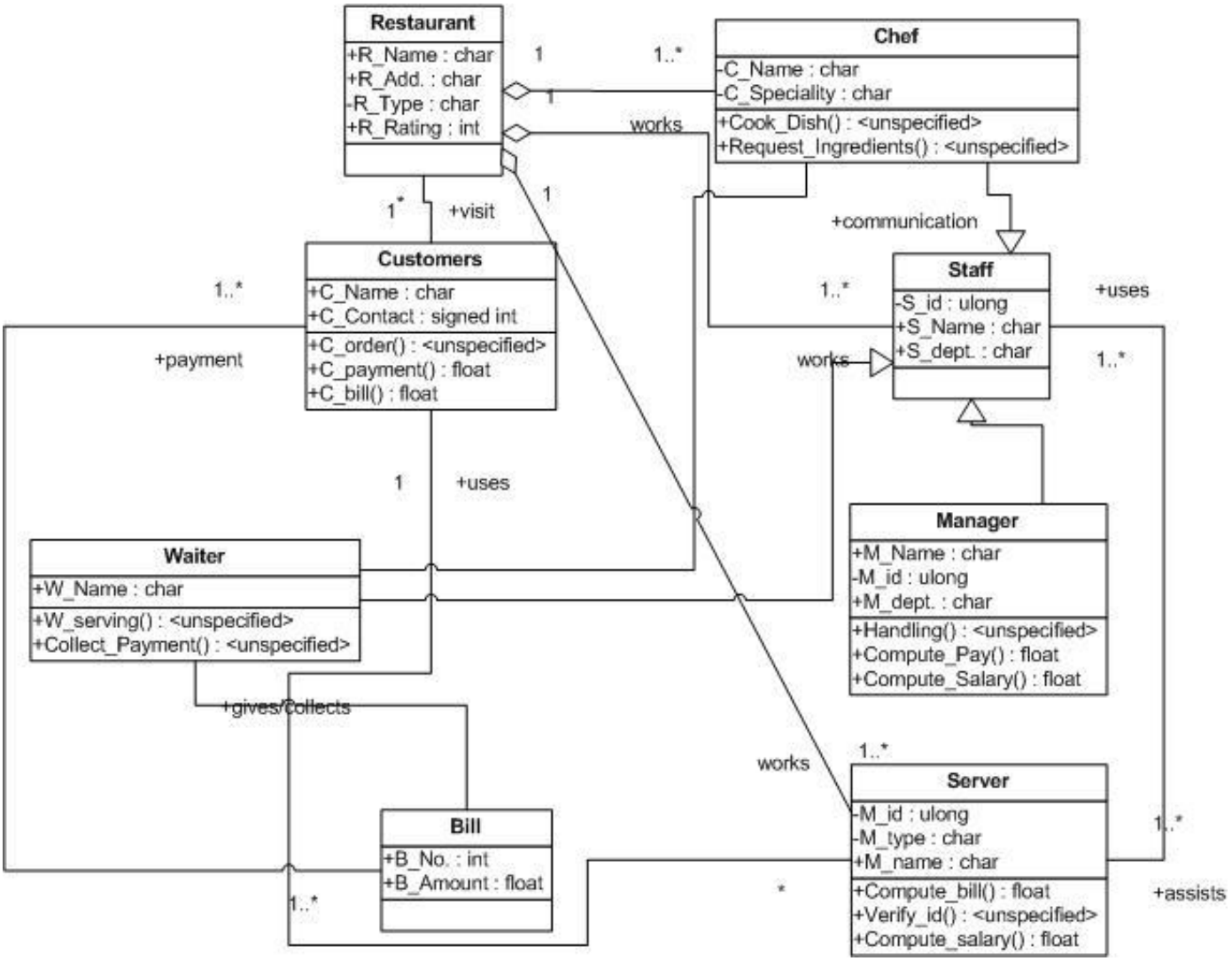
Class Diagram for Library Management System



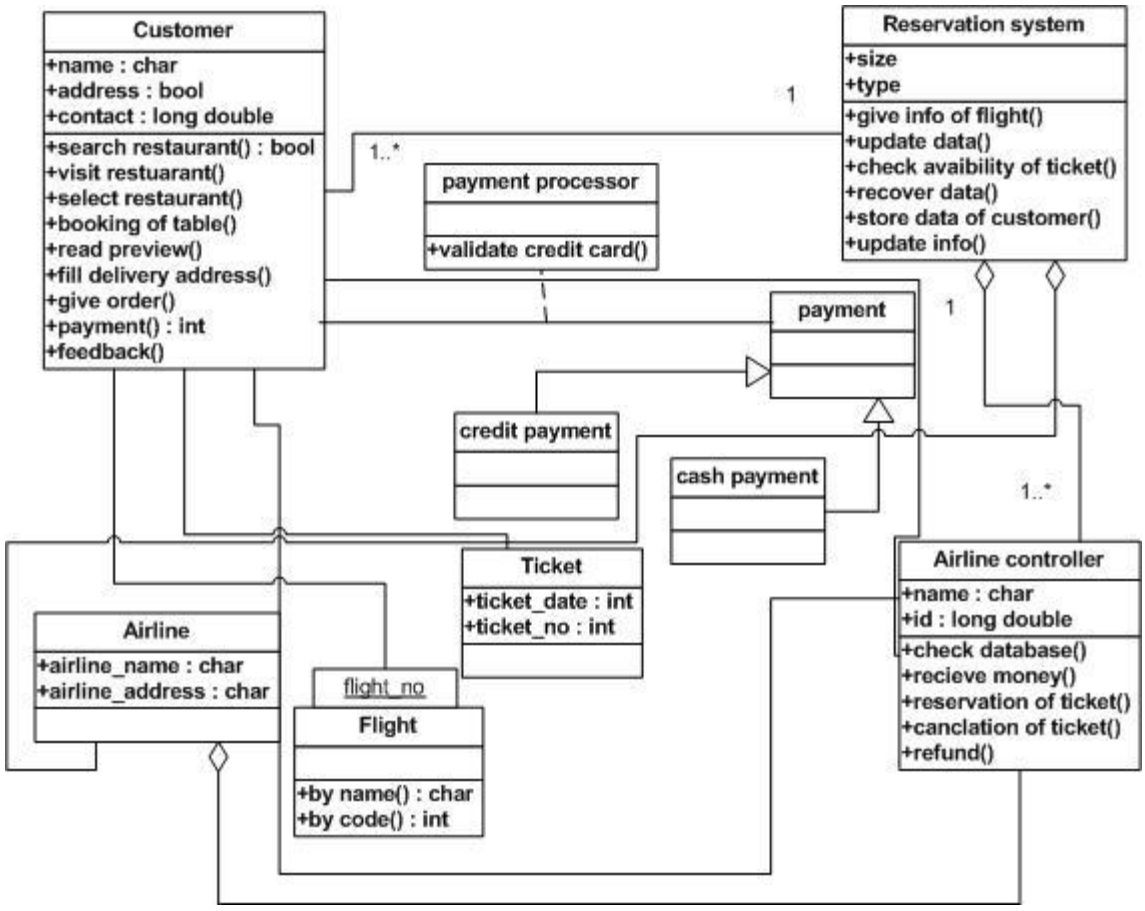
Class Diagram for ATM



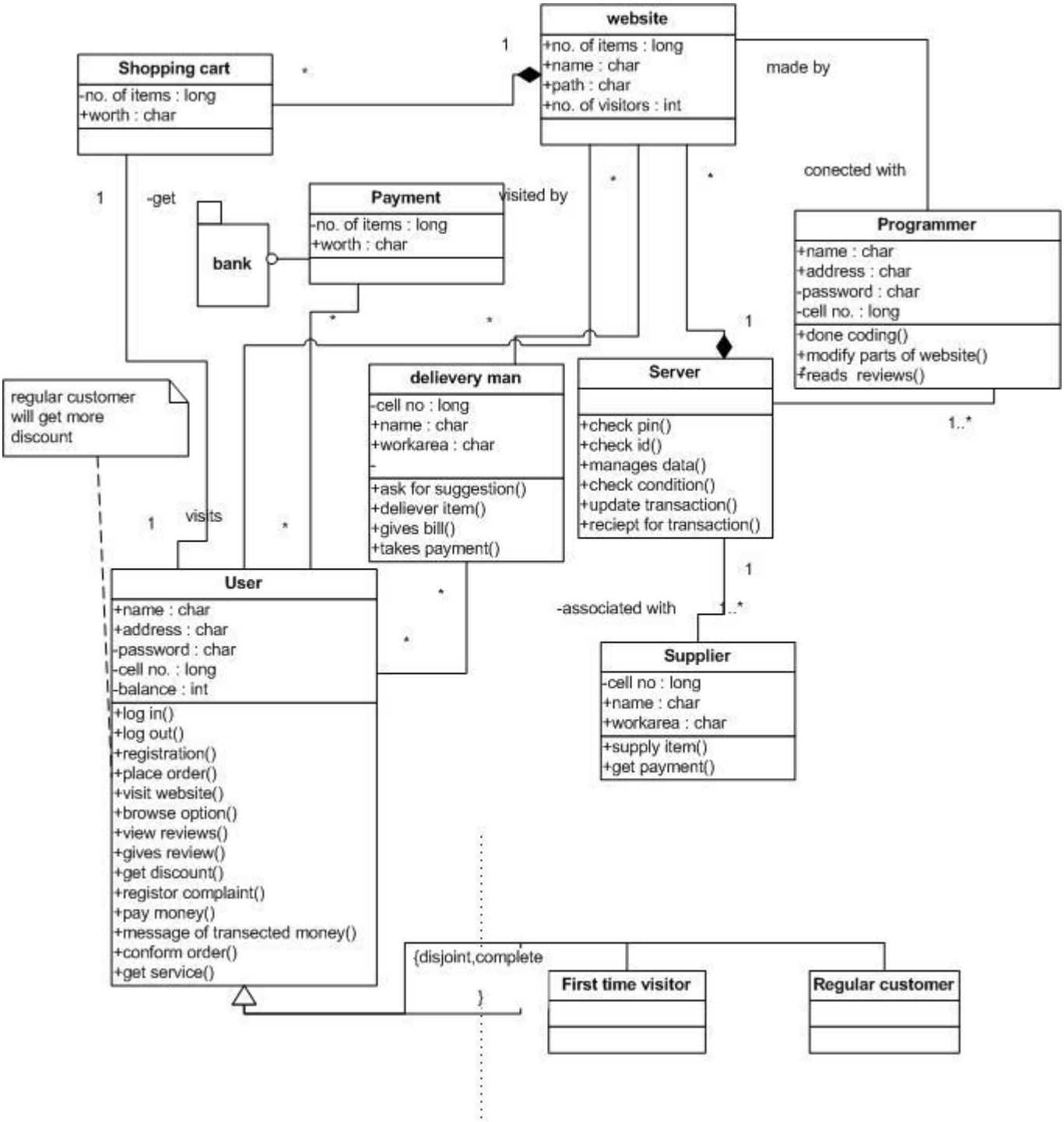
Class Diagram for Online Restaurant System



Class Diagram for Online Reservation System

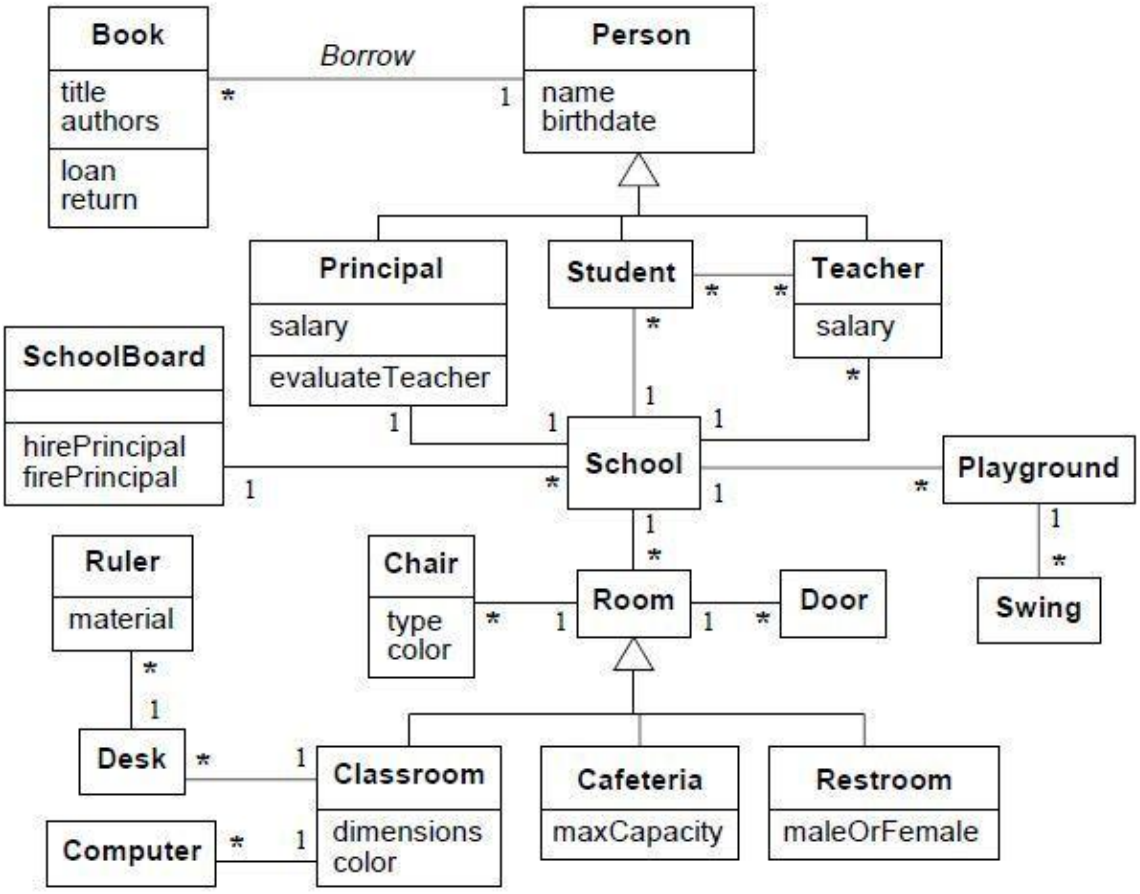


Class Diagram for Online Shopping System



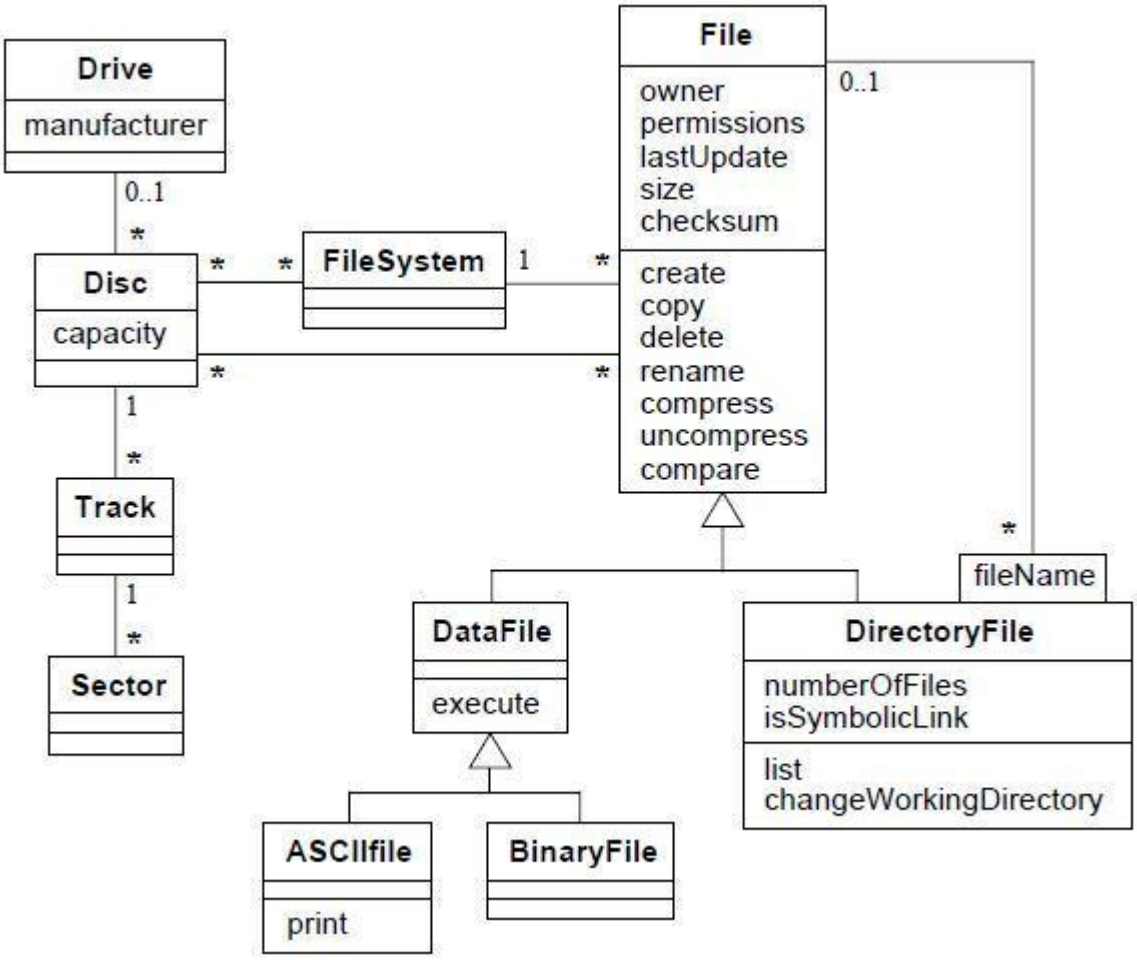
Additional Diagram : Class Diagram

1 Prepare Class diagram showing at least 10 relationships among the following object classes. Include associations and qualified associations, aggregations, generalizations, and multiplicity. You may add additional objects. Also show attributes and operations. School, playground, principal, school board, classroom, book, student, teacher, canteen, restroom, computer, desk, chair.

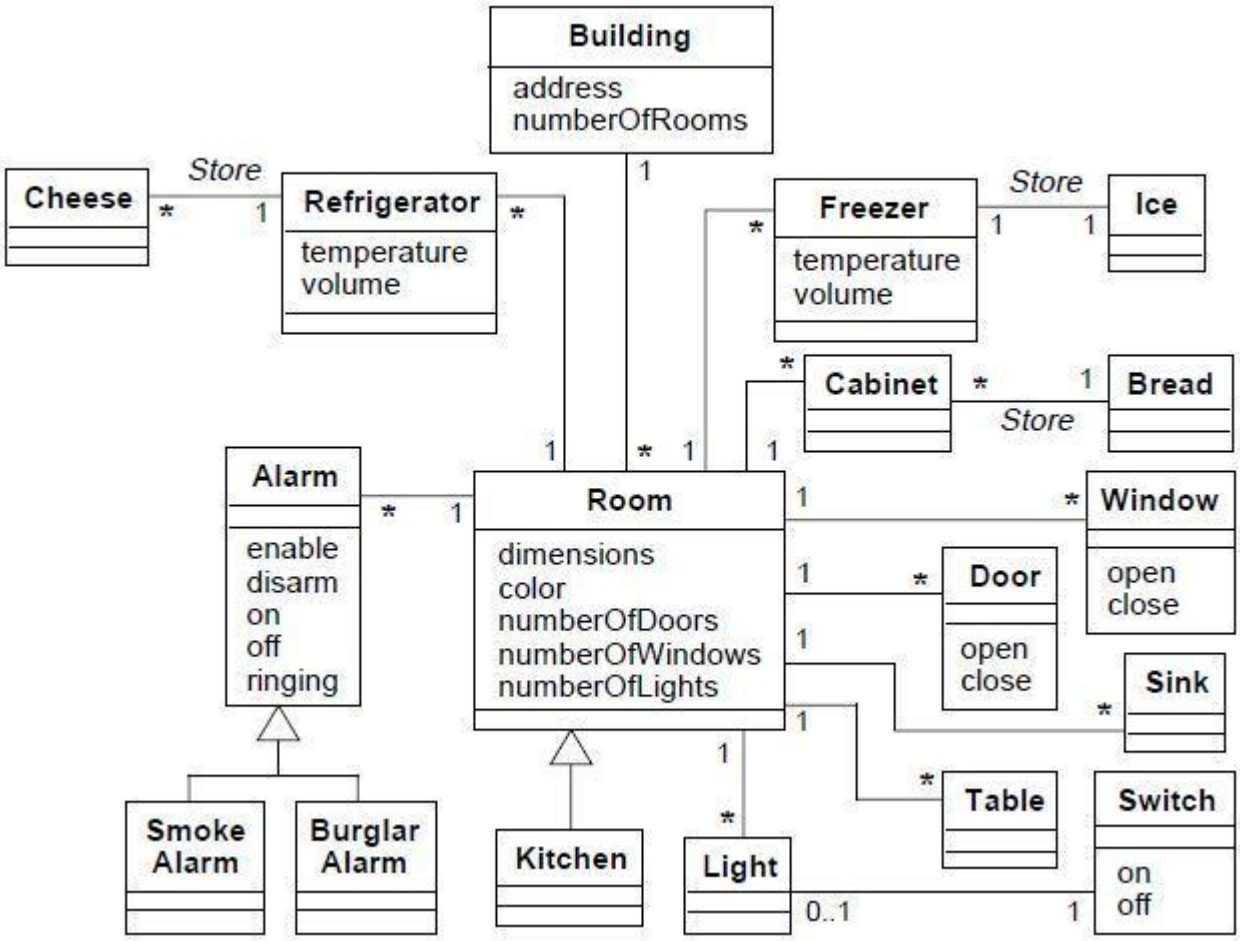


2 Prepare a class diagram for each group of classes. Add at least 10 relationships (associations and

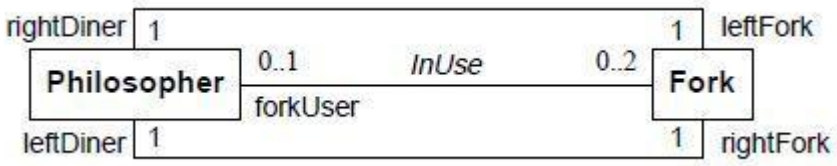
generalizations) to each diagram. File system, file, ASCII file, binary file, directory file, disc, drive, track, and sector.



3 Prepare a class diagram for group of classes. Sink, freezer, refrigerator, table, light, switch, window, smoke alarm, burglar alarm, cabinet, bread, cheese, ice, door, kitchen



4 Prepare a class diagram for the dining philosopher problem. There are 5 philosophers and 5 forks around a circular table. Each philosopher has access to 2 forks, one on either side. Each fork is shared by 2 philosophers. Each fork may be either on the table or in use by one philosopher. A philosopher must have 2 forks to eat.



5 Categorize the following relationships into generalization, aggregation or association. Justify your answer.

i. A country has a capital city.

Association. A capital city and a country are distinct things so generalization certainly does not apply. You could argue that a capital city is a part of a country and thus they are related by aggregation.

ii. A file is an ordinary file or a directory file.

Generalization. The word "or" often is an indicator of generalization. *File* is the super class and *Ordinary File* and *Directory File* are subclasses.

iii. Files contain records.

Aggregation. The word "contain" is a clue that the relationship may be aggregation. A record is a part of a file. Some attributes and operations on files propagate to their constituent records.

iv. A polygon is composed of an ordered set of points.

Aggregation or Composition. The phrase "is composed of" should immediately make you suspicious that there is an aggregation. An ordered set of points is a part of a polygon. Some attributes and operations on a polygon propagate to the corresponding set of points.

v. A drawing object is text, a geometrical object, or a group.

Generalization. Once again, the word "or" should prompt you to think of generalization. *DrawingObject* is the super class. *Text*, *Geometrical Object*, and *Group* are subclasses.

vi. A route connects two cities.

Association. Either *Route* is a class associated with the *City* class, or *Route* is the name of the association from *City* to *City*.

vii. A student takes a course from a professor.

Ternary association.*Student*, *Course*, and *Professor* are distinct classes of equal stature.

viii. A person uses a computer language on a project.

Ternary association.*Person*, *Computer Language*, and *Project* are all classes of equal stature. The association cannot be reduced to binary associations. None of these classes are a-kind-of or a-part-of another class. Thus generalization and aggregation do not apply.

ix. Modems and keyboards are input / output devices.

Generalization. The keyword "are" is the clue. Modem and Keyboard are the subclasses; InputOutputDevice is the super class.

x. Classes may have several attributes.

Association or aggregation. It depends on your perspective and the purpose of the model whether aggregation applies.

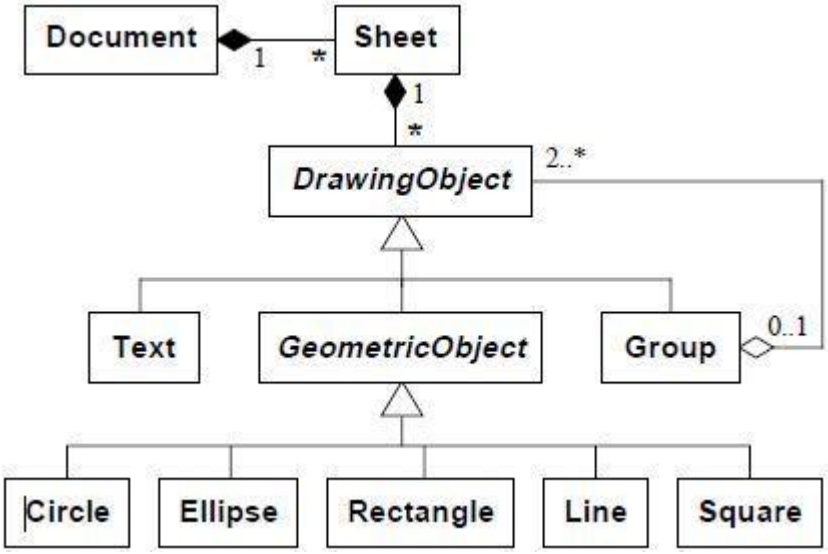
xi. A person plays for a team in a certain year.

Ternary association.*Person*, *Team*, and *Year* are distinct classes of equal stature.

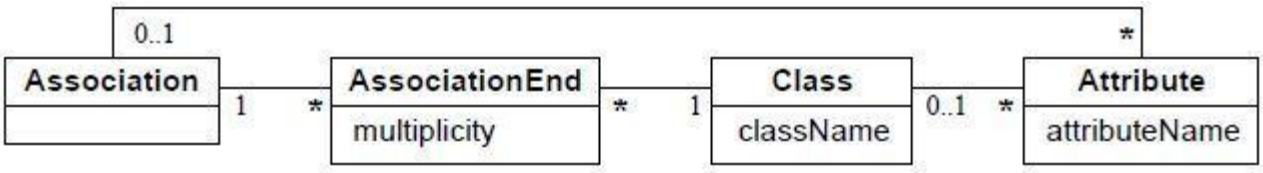
xii. A dining philosopher uses a fork.

Association. Dining philosophers and forks are completely distinct things and are therefore not in a generalization relationship. Similarly, neither object is a part of the other nor the relationship is not aggregation.

5 Prepare a class diagram for a graphical document editor that supports grouping. Assume that a document consists of several sheets. Each sheet contains drawing objects, including text, geometrical objects and groups. A group is simply a set of drawing objects, possibly including other groups. A group must contain at least two drawing objects. A drawing object can be a direct member of at most one group. Geometrical objects include circles, ellipses, rectangles, lines and squares.



6 Prepare a meta-model that supports only the following UML concepts: class, attribute, association, association end, multiplicity, class name, and attribute name. Use only these constructs to build meta-model.



Understanding and implementation of Analysis Level Class Model
Lab pre work: Prepare a class diagram from the given problem description using UML2.0 notations.
Laboratory work: Implement the class diagram with a suitable object oriented language.

❖ **Objective:**
- Identify domain classes and collaboration among them from given requirements
- Prepare analysis model and implement.
- Understanding the implementation details of relationships among classes

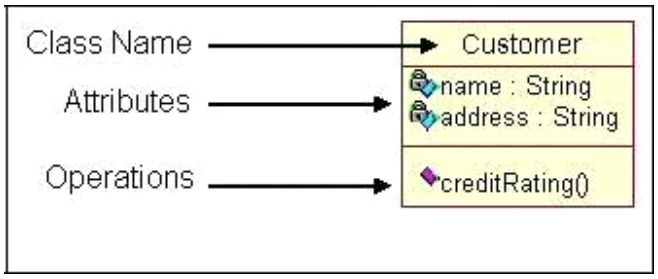
❖ **Relevant Theory**

Class Diagram
The Class diagram shows the building blocks of any object-orientated system. Class diagrams depict the static view of the model or part of the model, describing what attributes and behaviors it has rather that detailing the methods for achieving operations.
Class diagrams are most useful to illustrate relationships between classes and interfaces. Generalizations, aggregations, and associations are all valuable in reflecting inheritance, composition or usage, and connections, respectively.

Classes
A class is an element that defines the attributes and behaviors that an object is able to generate. The behavior is the described by the possible messages the class is able to understand along with operations that are appropriate for each message. Classes may also contain definitions of constraints tagged values and stereotypes.

Class Notation
Classes are represented by rectangles which show the name of the class and optionally the name of the operations and attributes. Compartments are used to divide the class name, attributes and operations. Additionally constraints, initial values and parameters may be assigned to classes.

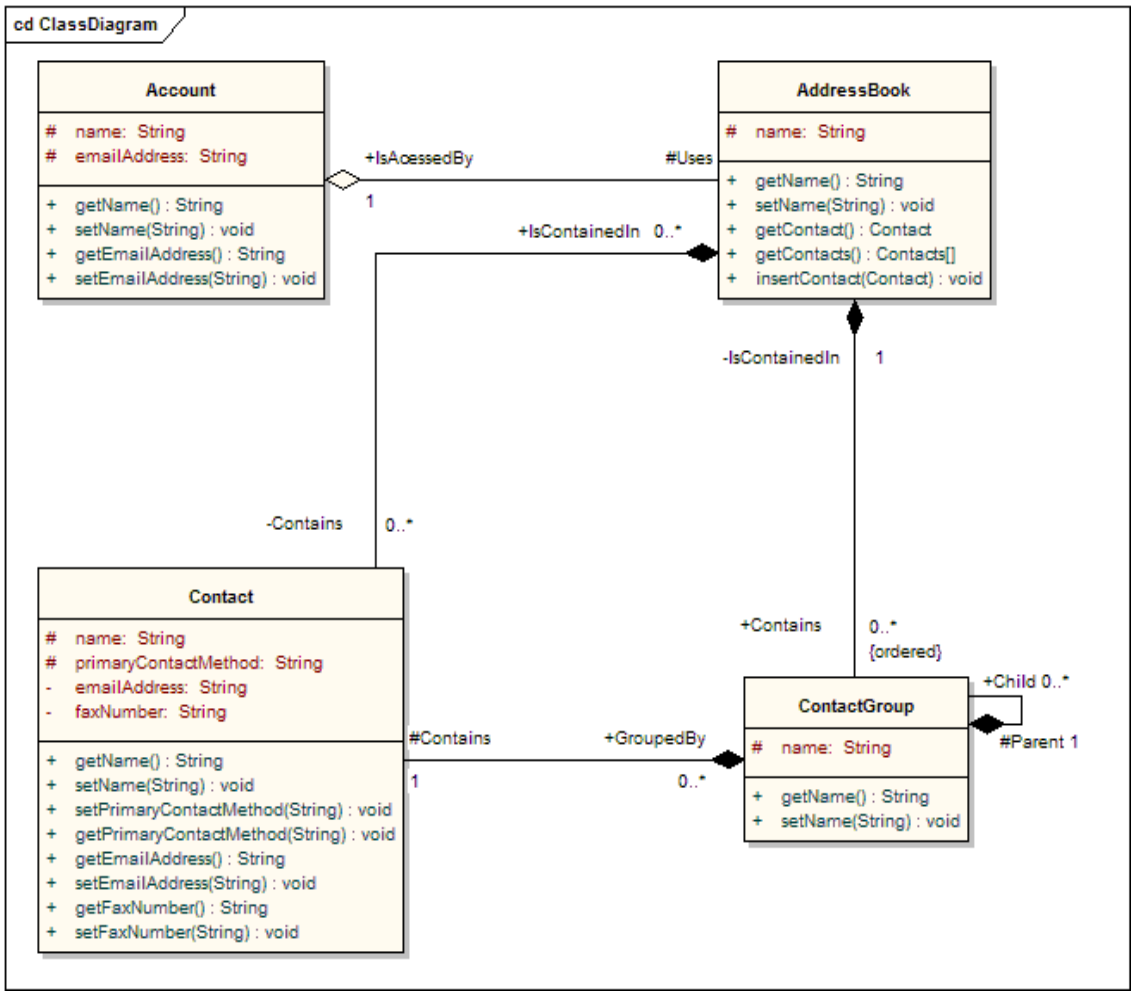
Classes are composed of three things: a name, attributes, and operations. Below is an example of a class.



For Example

The diagram below illustrates aggregation relationships between classes. The lighter aggregation indicates that the class Account uses AddressBook, but does not necessarily contain an instance of it.

The strong, composite aggregations by the other connectors indicate ownership or containment of the source classes by the target classes, for example Contact and ContactGroup values will be contained in AddressBook.

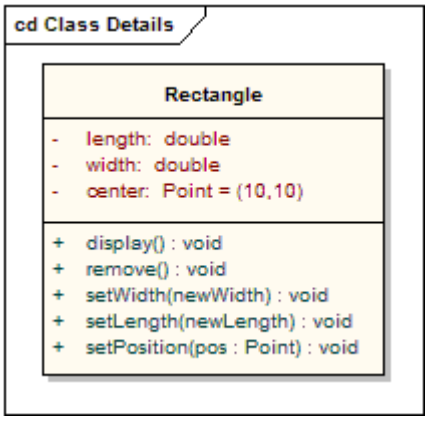


In the diagram the class contains the class name in the topmost compartment, the next compartment details the attributes, with the "center" attribute showing initial values.

The final compartment shows the operations, the setWidth, setLength and setPosition operations showing their parameters.

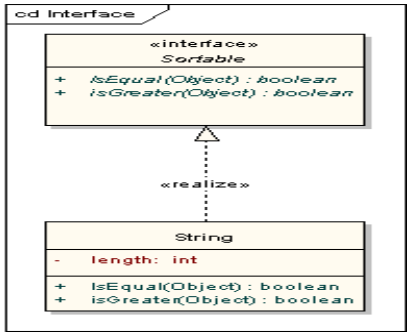
The notation that precedes the attribute or operation name indicates the visibility of the element,

if the + symbol is used the attribute or operation has a public level of visibility, if a - symbol is used the attribute or operation is private. In addition the # symbol allows an operation or attribute to be defined as protected and the ~ symbol indicates package or default visibility.

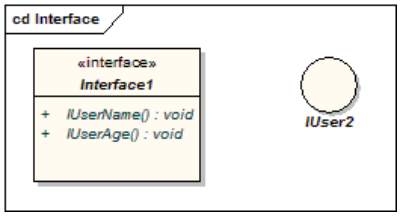


Interfaces

An interface is a specification of behavior that implementers agree to meet. It is a contract. By realizing an interface, classes are guaranteed to support a required behavior, which allows the system to treat non-related elements in the same way – i.e. through the common interface.

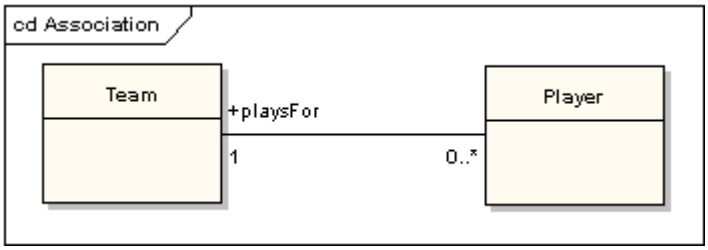


Interfaces may be drawn in a similar style to a class, with operations specified, as shown below. They may also be drawn as a circle with no explicit operations detailed. When drawn as a circle, realization links to the circle form of notation are drawn without target arrows.



Associations

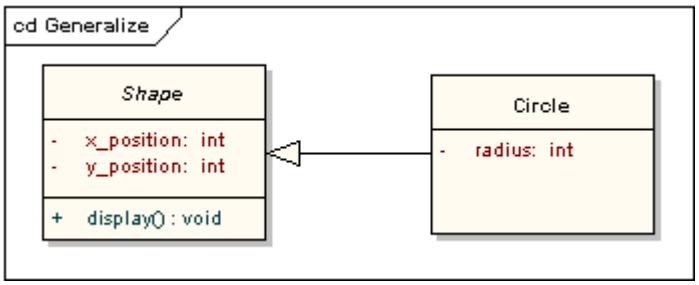
An association implies two model elements have a relationship - usually implemented as an instance variable in one class.
This connector may include named roles at each end, multiplicity or cardinality, direction and constraints.
Association is the general relationship type between elements. For more than two elements, When code is generated for class diagrams, associations become instance variables in the target class.



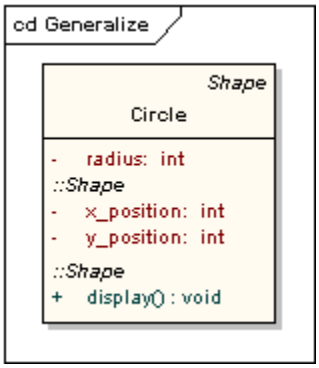
Generalizations

A generalization is used to indicate **inheritance.** Drawn from the specific classifier to a general classifier, the generalize implication is that the source inherits the target's characteristics.

The following diagram shows a parent class generalizing a child class. Implicitly, an instantiated object of the Circle class will have attributes x_position, y_position and radius and a method display().



The following diagram shows an equivalent view of the same information.



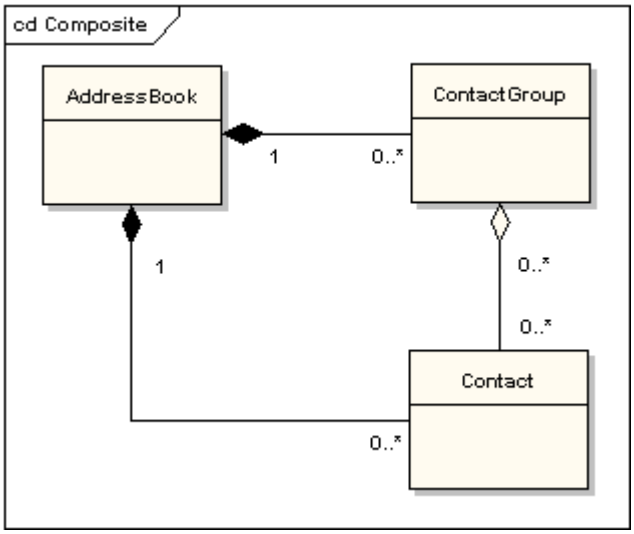
Aggregations

Aggregations are used to depict elements which are made up of smaller components. Aggregation relationships are shown by a white diamond-shaped arrowhead pointing towards the target or parent class.

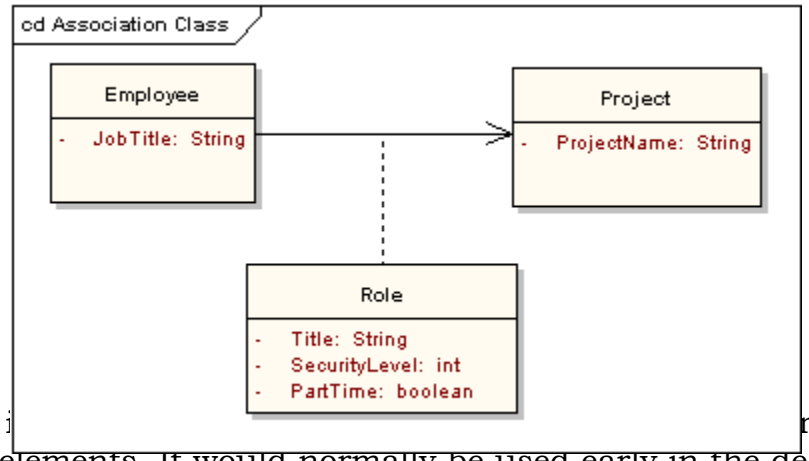
A stronger form of aggregation - **a composite aggregation** - is shown by a black diamond-shaped arrowhead and is used where components can be included in a maximum of one composition at a time.

If the parent of a composite aggregation is deleted, usually all of its parts are deleted with it; however a part can be individually removed from a composition without having to delete the entire composition. Compositions are transitive, asymmetric relationships and can be recursive.

The following diagram illustrates the difference between weak and strong aggregations.
An address book is made up of a multiplicity of contacts and contact groups. A contact group is a virtual grouping of contacts; a contact may be included in more than one contact group.
If you delete an address book, all the contacts and contact groups will be deleted too; if you delete a contact group, no contacts will be deleted.



Association Classes
An association class is a construct that allows an association connection to have operations and attributes. The following example shows that there is more to allocating an employee to a project than making a simple association link between the two classes: the role that the employee takes up on the project is a complex entity in its own right and contains detail that does not belong in the employee or project class.
For example, an employee may be working on several projects at the same time and have different job titles and security levels on each.

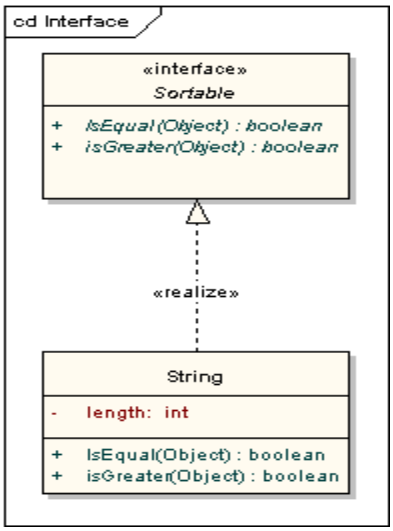


Dependencies
A dependency i relationships between model elements. It would normally be used early in the design process

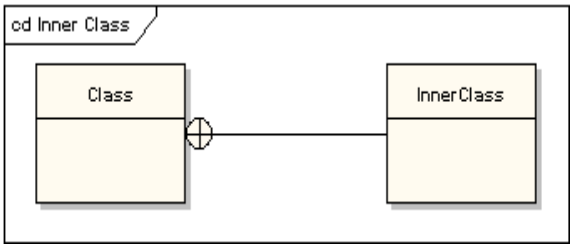
where it is known that there is some kind of link between two elements but it is too early to know exactly what the relationship is.
Later in the design process, dependencies will be stereotyped (stereotypes available include «instantiate», «trace», «import» and others) or replaced with a more specific type of connector.

Traces
The trace relationship is a specialization of a dependency, linking model elements or sets of elements that represent the same idea across models. Traces are often used to track requirements and model changes. As changes can occur in both directions, the order of this dependency is usually ignored. The relationship's properties can specify the trace mapping, but the trace is usually bi-directional, informal and rarely computable.

Realizations
The source objects implements or realizes the destination. Realize is used to express traceability and completeness in the model - a business process or requirement is realized by one or more use cases which are in turn realized by some classes, which in turn are realized by a component, etc. Mapping requirements, classes, etc. across the design of your system, up through the levels of modeling abstraction, ensures the big picture of your system remembers and reflects all the little pictures and details that constrain and define it. A realization is shown as a dashed line with a solid arrowhead and the «realize» stereotype.



Nestings
A nesting is connector that shows that the source element is nested within the target element. The following diagram shows the definition of an inner class although in EA it is more usual to show them by their position in the Project View hierarchy.



❖ Exercises:

Problem Description

A university is having several departments. Each Department should have minimum one faculty member to maximum of 10 faculty member. Each faculty member has residential address within proximity within 2 kms. Attribute of entities are

Department: department name, department type, department start date, currentHOD,

Faculty: Facultyid, Facultyname, subject specialization, Joiningdate, DOB, Salary

Address: building name, Flatno, Streetname, Landmark, Pincode, sate, Country

A. Develop the analysis level class diagram using UML tool

B. Implement the class diagram with a suitable object oriented language.

❖ Conclusion:

Hence the analysis class diagram concepts such as

- Finding classes from given requirements
- Finding the association among them
- Finding simple attributes

have been successfully underrated and also implemented using java.

❖ FREQUENTLY ASKED QUESTIONS for ORAL EXAMINATION:

- Describe Links and Associations with appropriate example.
- Explain following terms:
  - Class
  - Object
  - Aggregation
  - Generalization
- What do you mean by attribute and what are different types of attribute?
- Explain the concept of package with example?
- What do you mean by constraints?
- Explain Mapping Cardinality or multiplicity with example?

Assignment No. 5

Title: Prepare a Design Model from Analysis Model

Study in detail working of system / Project.
Identify Design classes/ Evolve Analysis Model.
Use advanced relationships. Draw Design class Model using OCL and UML2.0 Notations.
Implement the design model with a suitable object-oriented language.

Date of Completion		**Marks for Regularity (2)**	
		Marks for Presentation (3)	
Remark		**Marks for Understanding (5)**	
Sign. of Staff		**Total Marks (10)**	

Prof. R.S. Badodekar **414459 : Computer Laboratory VIII**

Assignment No. 2

❖ **Title of Assignment:**

Purpose: Implementation of a design model

Lab pre work: Prepare a design model from analysis model in the form of UML 2 class diagram.

Laboratory work: Implement the design model with a suitable object oriented language

❖ **Objective:**

- Identify more classes and collaboration from knowledge of problem domain and given requirements
- Identify Attributes with datatypes, methods with their parameter, visibility of member of classes, Multiplicity, constraints, role of classes in relationships
- Prepare Design model from analysis model and implement.
- Understanding the implementation details of attributes, methods, and relationships among classes

❖ **Relevant Theory**

The analysis model is refined and formalized to get a design model.

Design modeling, means refinement to analysis level models to adapt to the actual implementation environment.

In design space, yet another new dimension has been added to the analysis space to include the implementation environment.

This means that we want to adopt our analysis model to fit in the implementation model at the same time as we refine it.

Creating Design level Class Diagrams

Class diagrams model the static structure of a package or of a complete system.

As the blueprints of system, class diagrams model the objects that make up the system, allowing to display the relationships among those objects and to describe what the objects can do and the services they can provide.

Class diagrams

In UML, class diagrams are one of six types of structural diagram. Class diagrams are fundamental to the object modeling process and model the static structure of a system. Depending on the complexity of a system, you can use a single class diagram to model an entire system, or you can use several class diagrams to model the components of a system.

Class diagrams are the blueprints of your system or subsystem. You can use class diagrams to model the objects that make up the system, to display the relationships between the objects, and to describe what those objects do and the services that they provide.

Class diagrams are useful in many stages of system design. In the analysis stage, a class diagram can help you to understand the requirements of your problem domain and to identify its components. In an object-oriented software project, the class diagrams that you create during the early stages of the project contain classes that often translate into actual software classes and objects when you write code. Later, you can refine your earlier analysis and conceptual models into class diagrams that show the specific parts of your system, user interfaces, logical implementations, and so on. Your class diagrams then become a snapshot that describes exactly how your system works, the relationships between system components at many levels, and how you plan to implement those components.

You can use class diagrams to visualize, specify, and document structural features in your models. For example, during the analysis and design phases of the development cycle, you can create class diagrams to perform the following functions:

- Capture and define the structure of classes and other classifiers
- Define relationships between classes and classifiers
- Illustrate the structure of a model by using attributes, operations, and signals
- Show the common classifier roles and responsibilities that define the behavior of the system
- Show the implementation classes in a package
- Show the structure and behavior of one or more classes
- Show an inheritance hierarchy among classes and classifiers
- Show the workers and entities as business object models

During the implementation phase of a software development cycle, you can use class diagrams to convert your models into code

Relationships in class diagrams

Abstraction relationships

An abstraction relationship is a dependency between model elements that represents the same concept at different levels of abstraction or from different viewpoints.
In an abstraction relationship, one model element, the client, is more refined or detailed than the other, the supplier. T
he different types of abstraction relationships include derivation, realization, refinement, and trace relationships.
All abstraction relationships can connect model elements that are in the same model or in different models.

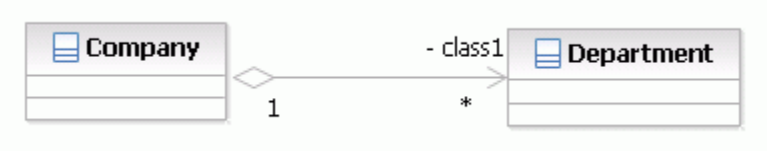
Abstraction relationships do not usually have names and appear as a dashed line with an open arrow pointing from the detailed model element to the general model element.
As the following figure illustrates, when you create an abstraction relationship, the «abstraction» keyword appears beside the connector.

You can assign the following stereotypes to an abstraction relationship to identify the type of abstraction in a model:

- «derive»
- «realize»
- «refine»
- «trace»

Aggregation relationships

In UML models, an aggregation relationship shows a class as a part of or subordinate to another class.
An aggregation is a special type of association in which objects are assembled or configured together to create a more complex object.
Data flows from the whole classifier, or aggregate, to the part. A part classifier can belong to more than one aggregate classifier and it can exist independently of the aggregate.
For example, a Department class can have an aggregation relationship with a Company class, which indicates that the department is part of the company. Aggregations are closely related to compositions.
You can name an association to describe the nature of the relationship between two classifiers; however, names are unnecessary if you use association end names.
As the following figure illustrates, an aggregation association appears as a solid line with an unfilled diamond at the association end, which is connected to the classifier that represents the aggregate. Aggregation relationships do not have to be unidirectional.



Association relationships

In UML models, an association is a relationship between two classifiers, such as classes, that describes the reasons for the relationship and the rules that govern the relationship.
An association represents a structural relationship that connects two classifiers. Like attributes, associations record the properties of classifiers.

For example, in relationships between classes, you can use associations to show the design decisions that you made about classes in your application that contain data, and to show which of those classes need to share data. You can use an association's navigability feature to show how an object of one class gains access to an object of another class or, in a reflexive association, to an object of the same class.
The name of an association describes the nature of the relationship between two classifiers and should be a verb or phrase.
In the diagram editor, an association appears as a solid line between two classifiers.

Association ends

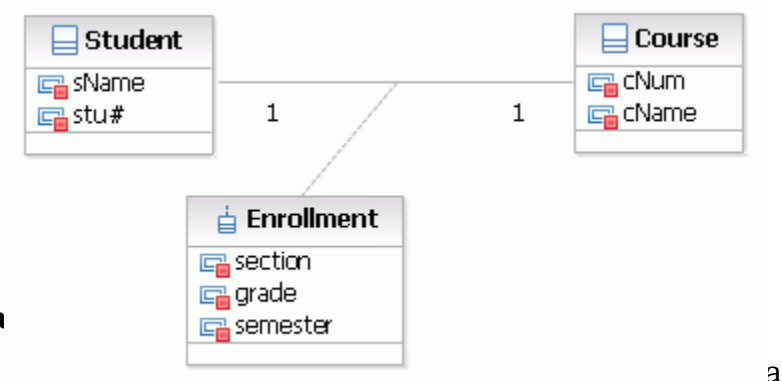
An association end specifies the role that the object at one end of a relationship performs. Each end of a relationship has properties that specify the role of the association end, its multiplicity, visibility, navigability, and constraints.

Example

In an e-commerce application, a customer class has a single association with an account class. The association shows that a customer instance owns one or more instances of the account class. If you have an account, you can locate the customer that owns the account.

Association classes

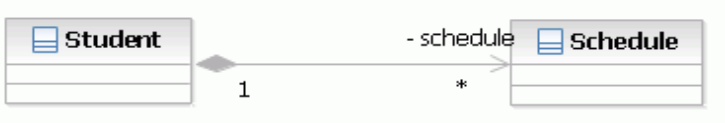
In UML diagrams, an association class is a class that is part of an association relationship between two other classes.
You can attach an association class to an association relationship to provide additional information about the relationship. An association class is identical to other classes and can contain operations, attributes, as well as other associations.
For example, a class called Student represents a student and has an association with a class called Course, which represents an educational course. The Student class can enroll in a course. An association class called Enrollment further defines the relationship between the Student and Course classes by providing section, grade, and semester information related to the association relationship.
As the following figure illustrates, an association class is connected to an association by a dotted line.



Composition a

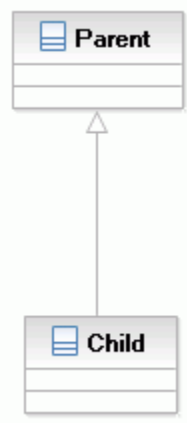
A composition art relationship and is a form of aggregation. A composition association relationship specifies

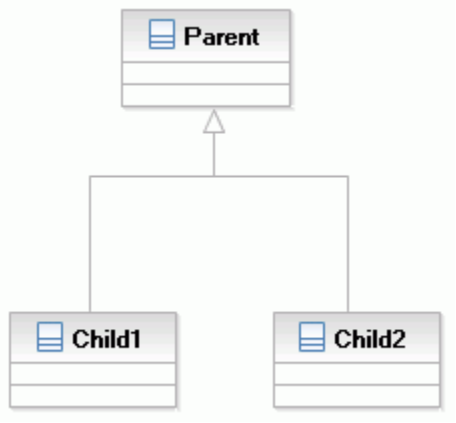
that the lifetime of the part classifier is dependent on the lifetime of the whole classifier.
In a composition association relationship, data usually flows in only one direction (that is, from the whole classifier to the part classifier). For example, a composition association relationship connects a Student class with a Schedule class, which means that if you remove the student, the schedule is also removed.
As the following figure illustrates, a composition association relationship appears as a solid line with a filled diamond at the association end, which is connected to the whole, or composite, classifier.



Generalization relationships

In UML modeling, a generalization relationship is a relationship in which one model element (the child) is based on another model element (the parent).
You can add generalization relationships to capture attributes, operations, and relationships in a parent model element and then reuse them in one or more child model elements. Because the child model elements in generalizations inherit the attributes, operations, and relationships of the parent, you must only define for the child the attributes, operations, or relationships that are distinct from the parent.
The parent model element can have one or more children, and any child model element can have one or more parents. It is more common to have a single parent model element and multiple child model elements.
Generalization relationships do not have names.
As the following figures illustrate, a generalization relationship as a solid line with a hollow arrowhead that points from the child model element to the parent model element.





Interface re: ationships

In UML diagrams, an interface realization relationship is a specialized type of implementation relationship between a classifier and a provided interface. The

interface realization relationship specifies that the realizing classifier must conform to the contract that the provided interface specifies.
Typically, interface realization relationships do not have names. If you name an interface realization, the name is displayed beside the connector in the diagram.
As the following figure illustrates, an interface realization relationship is displayed in the diagram editor as a dashed line with a hollow arrowhead. The interface realization points from the classifier to the provided interface.

Realization relationships

In UML modeling, a realization relationship is a relationship between two model elements, in which one model element (the client) realizes the behavior that the other model element (the supplier) specifies. Several clients can realize the behavior of a single supplier.
As the following figure illustrates, a realization as a dashed line with an unfilled arrowhead that points from the client (realizes the behavior) to the supplier (specifies the behavior).

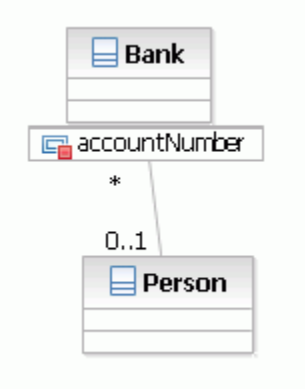
Qualifiers on association ends

In UML, qualifiers are properties of binary associations and are an optional part of association ends.
A qualifier holds a list of association attributes, each with a name and a type. Association attributes model the keys that are used to index a subset of relationship instances.
A qualifier is visually represented as a rectangle attached to the qualified end of the association relationship. The list of association attributes is displayed in the qualifier box.
Qualifiers identify subsets of related instances in association navigations; they provide a model of indices or keys for association ends. It is rare to see qualifiers on both ends of an association, because only the unqualified element typically varies in multiplicity.
In a navigation context, qualifiers are used to select a specific object pair from the set of all related objects in that association. In an implementation context, each qualifier value points to a unique target object. Generally, if an application requires the retrieval of data based on search keys, the model should use qualified associations.

Example

In a banking application, a class called Bank represents a banking institution and has an association with a class called Person, which represents an individual. Each individual is associated with the bank through several bank accounts. The account number qualifies the association, and it enables the indexing of many associations between the Person and Bank classes.

As the following figure illustrates, the qualifier is attached to the association end that corresponds to the Bank class.



Visibility in Class diagram

Use visibility indicates who can access the information contained within a class. There are four types of visibilities :

1. **Private visibility** hides information from anything outside the class partition. UML notation for private is +
2. **Public visibility** allows all other classes to view the marked information. UML notation for public is -.
3. **Protected visibility** allows child classes to access information they inherited from a parent class. UML notation for protected is #.
4. **Package or default visibility** allows classes to access information within same package. UML notation for package visibility is ~ symbol

Attributes in Class

In UML models, attributes represent the information, data, or properties that belong to instances of a classifier.

A classifier can have any number of attributes or none at all. Attributes describe a value or a range of values that instances of the classifier can hold. You can specify an attribute's type, such as an integer or Boolean, and its initial value. You can also attach a constraint to an attribute to define the range of values it holds.

Attribute names are short nouns or noun phrases that describe the attribute. The UML syntax for an attribute name incorporates information in addition to its name, such as the attribute's visibility, type, and initial value as shown in the following example.

```
visibility «stereotype» name : type-expression = initial-value
```

Example
In an e-commerce application, a Customer class has an attribute that holds the amount of money in the customer's balance as shown in the following example.

```
- balance : MoneyType = 0.00
```

- - is the attribute's visibility, in this case, private
- balance is the attribute's name, which describes a particular property of the Customer class
- MoneyType is the attribute's type, in this case an instance of the MoneyType class
- 0.00 is the attribute's initial value, zero

Operations in Class

In UML models, operations represent the services or actions that instances of a classifier might be requested to perform.
A classifier can have any number of operations or none at all. Operations define the behavior of an instance of a classifier
You can add operations to identify the behavior of many types of classifier in your model. In classes, operations are implementations of functions that an object might be required to perform. Well-defined operations perform a single task.
Operations can have exceptions, elements that are created when the operation encounters an error.
Every operation in a classifier must have a unique signature. A signature comprises the operation's name and its ordered list of parameter types. The UML syntax for an operation name is as follows:

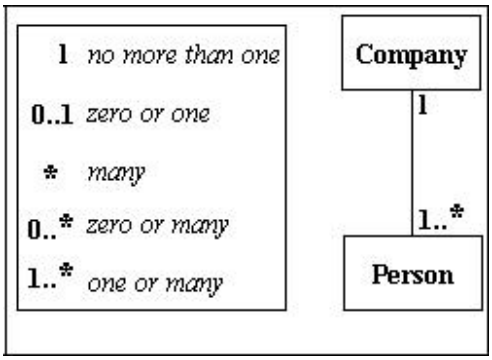
```
Visibility «stereotype» name(parameter list) : return-type
```

For example, in an e-commerce application a Customer class has the following operation: - getBalance([in] day: Date) : MoneyType.

❖ **Multiplicity (Cardinality)**

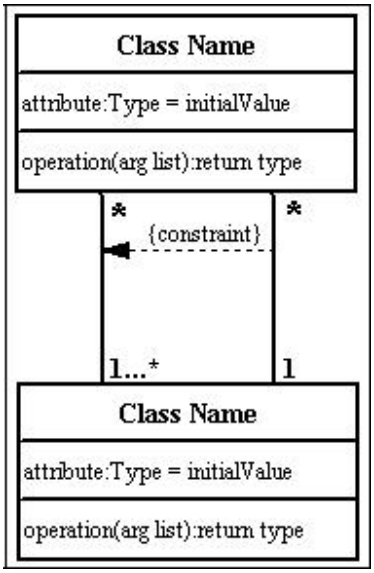
The multiplicity of the association denotes how many the number of objects from one class can associated with how many no of objects from another

class. . For example, one company will have one or more employees, but each employee works for one company only.



❖ **Constraint:**

Constraints are Boolean expressions that must get evaluated true. Constraints can be applied on objects, Links and Generalization. Place constraints inside curly braces {}.Example for the same is given below.



❖ **Exercises:**

Problem Description

A university is having several departments. Each Department should have minimum one faculty member to maximum of 10 faculty member. Each faculty member has residential address within proximity within 2 kms. Attribute of entities are

Department: department name, department type, department start date, currentHOD,

Faculty: Facultyid, Facultyname, subject specialization, Joiningdate, DOB, Salary

Address: building name, Flatno, Streetname, Landmark, Pincode, sate, Country

A. Develop the design level class diagram using UML tool

B. Implement the class diagram with a suitable object oriented language.

❖ **Conclusion:**

Hence prepared and implemented a design model from analysis model with another new dimension to include the implementation environment.
Design model consist of :

- More classes then analysis model which are identified from knowledge of problem domain and given requirements
- Classes with visibility to their members
- Attributes and methods with datatypes
- Probable quires and transaction among classes
- Classes with added behavior
- Method with their parameters and implementation

FREQUENTLY ASKED QUESTIONS FOR ORAL EXAMINATION:

- Explain Qualified Association? Draw its Notation in using an example
- How will you use Association as a class?
- Compare Aggregation, Association and composition. Give an example of each.
- Explain concept of Generalization and inheritance with suitable Example.
- What is N-ary association? Show with example.

Assignment No. 6

Title: Prepare Sequence Model.

Identify at least 5 major scenarios (sequence flow) for your system.
Draw Sequence Diagram for every scenario by using advanced notations using UML2.0
Implement these scenarios by taking reference of design model implementation using suitable object-oriented language.

Date of Completion		**Marks for Regularity (2)**	
		Marks for Presentation (3)	
Remark		**Marks for Understanding (5)**	
Sign. of Staff		**Total Marks (10)**	

Prof. R.S. Badodekar **414459 : Computer Laboratory VIII**

Assignment No. 6 - Prepare Sequence Model.

Introduction

- Sequence diagrams model the dynamic aspects of a software system.
- The emphasis is on the "sequence" of messages rather than relationship between objects.
- A sequence diagram maps the flow of logic or flow of control within a usage scenario into a visual diagram enabling the software architect to both document and validate the logic during the analysis and design stages.
- Sequence diagrams provide more detail and show the message exchanged among a set of objects over time.
- Sequence diagrams are good for showing the behavior sequences seen by users of a diagram shows only the sequence of messages not their exact timing.
- Sequence diagrams can show concurrent signals.

Purpose

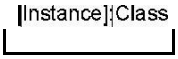
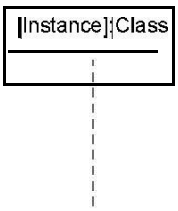
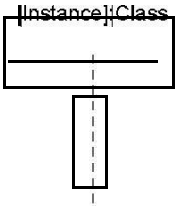
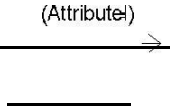
- The main purpose of this diagram is to represent how different business objects interact.
- A sequence diagram shows object interactions arranged in time sequence.
- It depicts the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario.

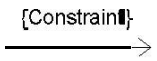
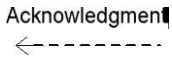
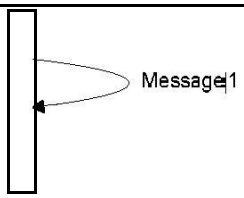
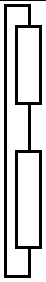
When to use : Sequence Diagram

- Sequence diagram can be a helpful modeling tool when the dynamic behavior of objects needs to be observed in a particular use case or when there is a need for visualizing the "big picture of message flow".
- A company's technical staff could utilize sequence diagrams in order to document the behavior of a future system.
- It is during the design period that developers and architects utilize the diagram to showcase the system's object interactions, thereby putting out a more fleshed out overall system design.

Sequence Diagram Notations

Sr. No.	Name	Notation	Description
1	Object		It represents the existence of an object of a particular time.

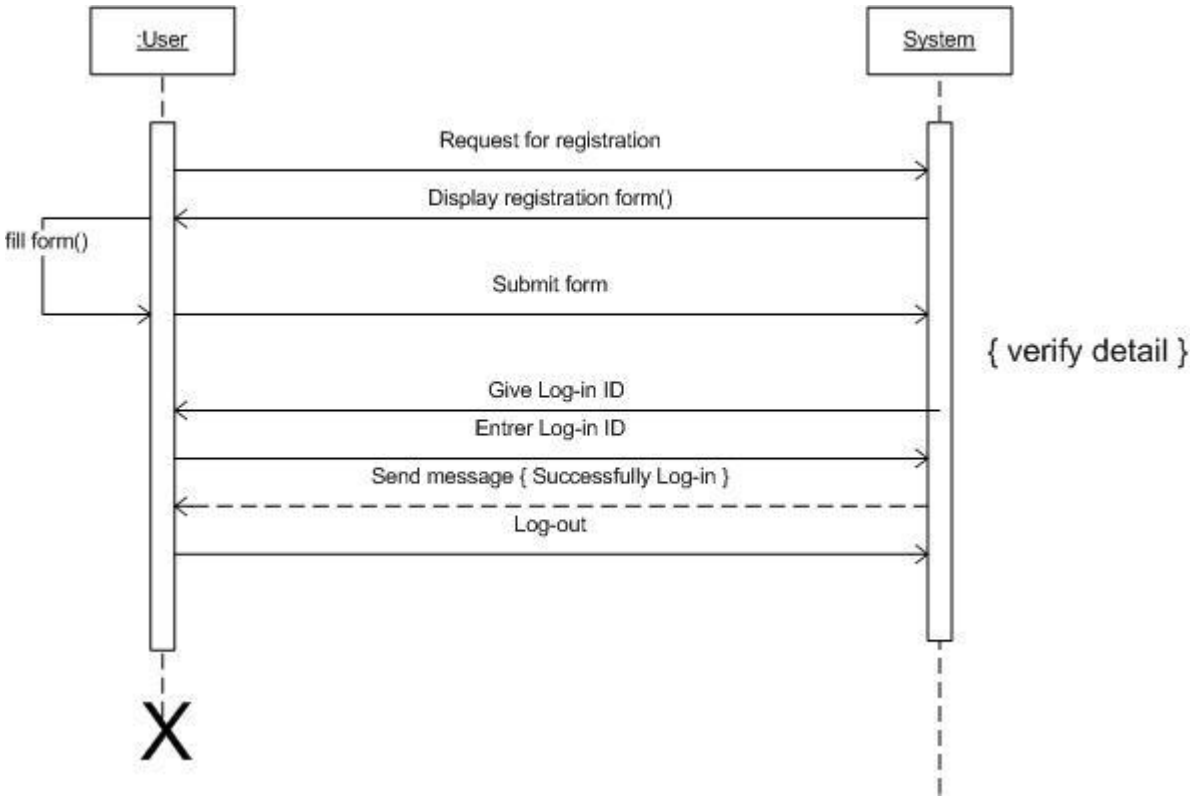
			
2	Life line		Lifeline represents the duration during which an object is alive and interacting with other objects in the system. It is represented by dashed lines.
3	Scope		It shows the time period during which an object or actor is performing an action.
4	Message transition		To send message from one object to another.
5	Message with attribute		To send message with some particular attribute

6	Message with constraint		To send message from one object to othervy some constraint.
7	Acknowledgement		It represent communication between objects conveys acknowledgement.
8	Self message		Self message occurs when an object sends a message to itself.
9	Recursive message		Self message occurs when an object sends a message to itself within recursive scope.

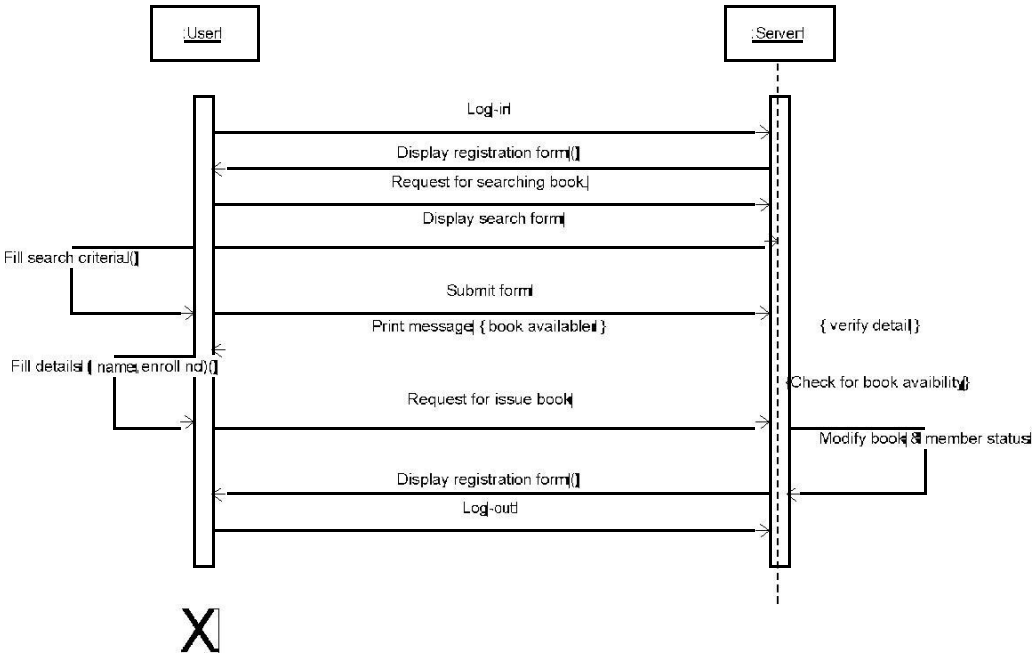
Examples:

Sequence Diagram for library management system:-

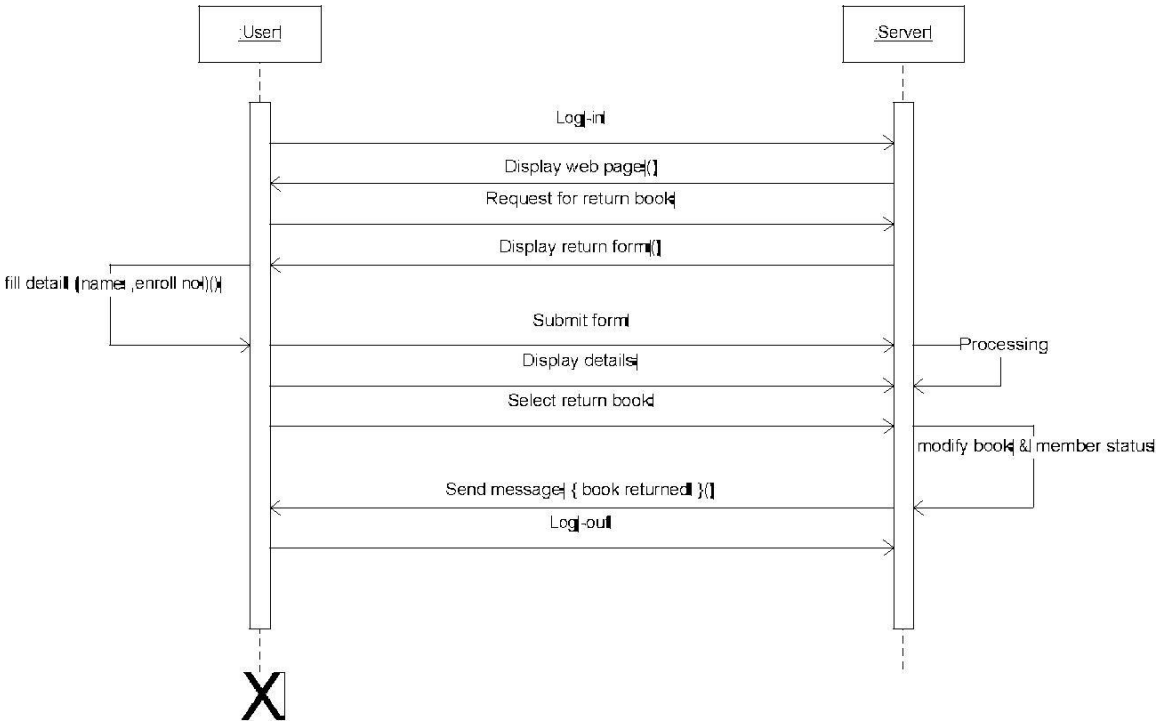
Registration



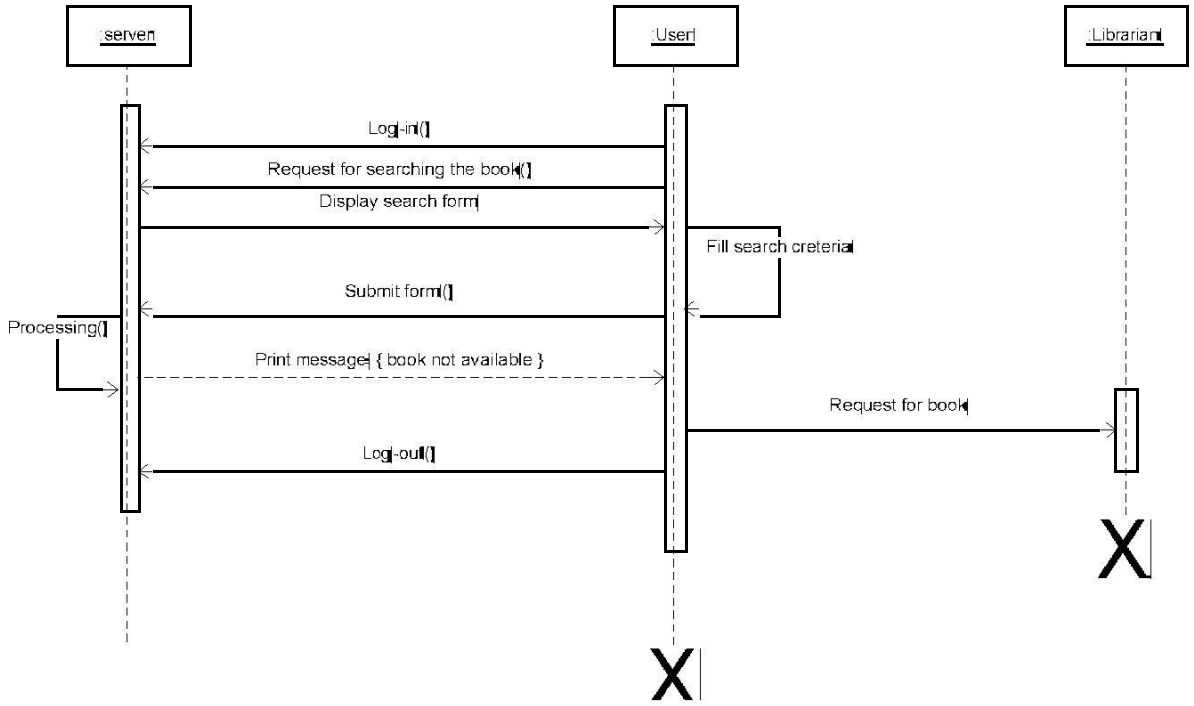
Issue book


:User
:Server
Log-in
Display registration form()
Request for searching book
Display search form
Fill search criteria()
Submit form
Print message({book available})
{ verify detail }
Fill details(name, enroll no)()
Check for book availibility()
Request for issue book
Modify book & member status
Display registration form()
Log-out
X

Return book

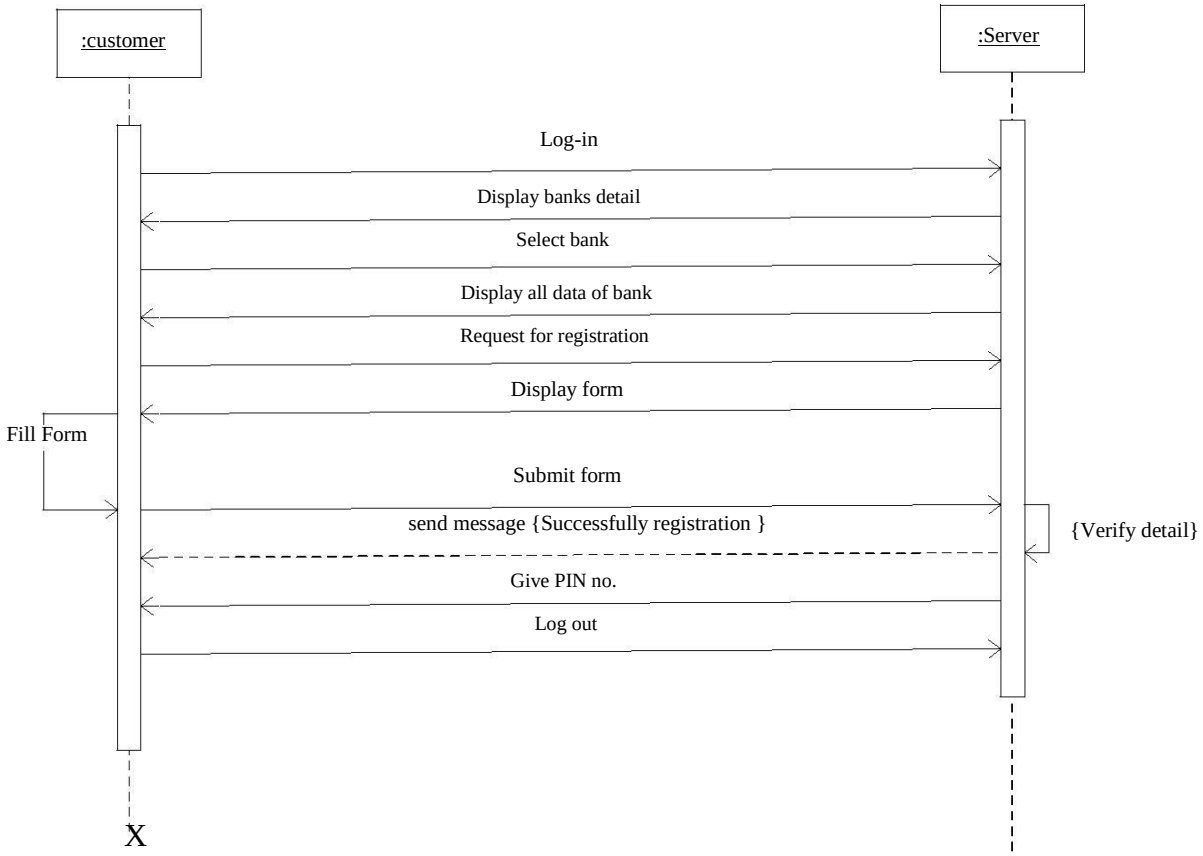

:User
:Server
Log-in
Display web page()
Request for return book
Display return form()
fill detail (name ,enroll no)()
Submit form
Processing
Display details
Select return book
modify book & member status
Send message (book returned)()
Log-out

Book not available

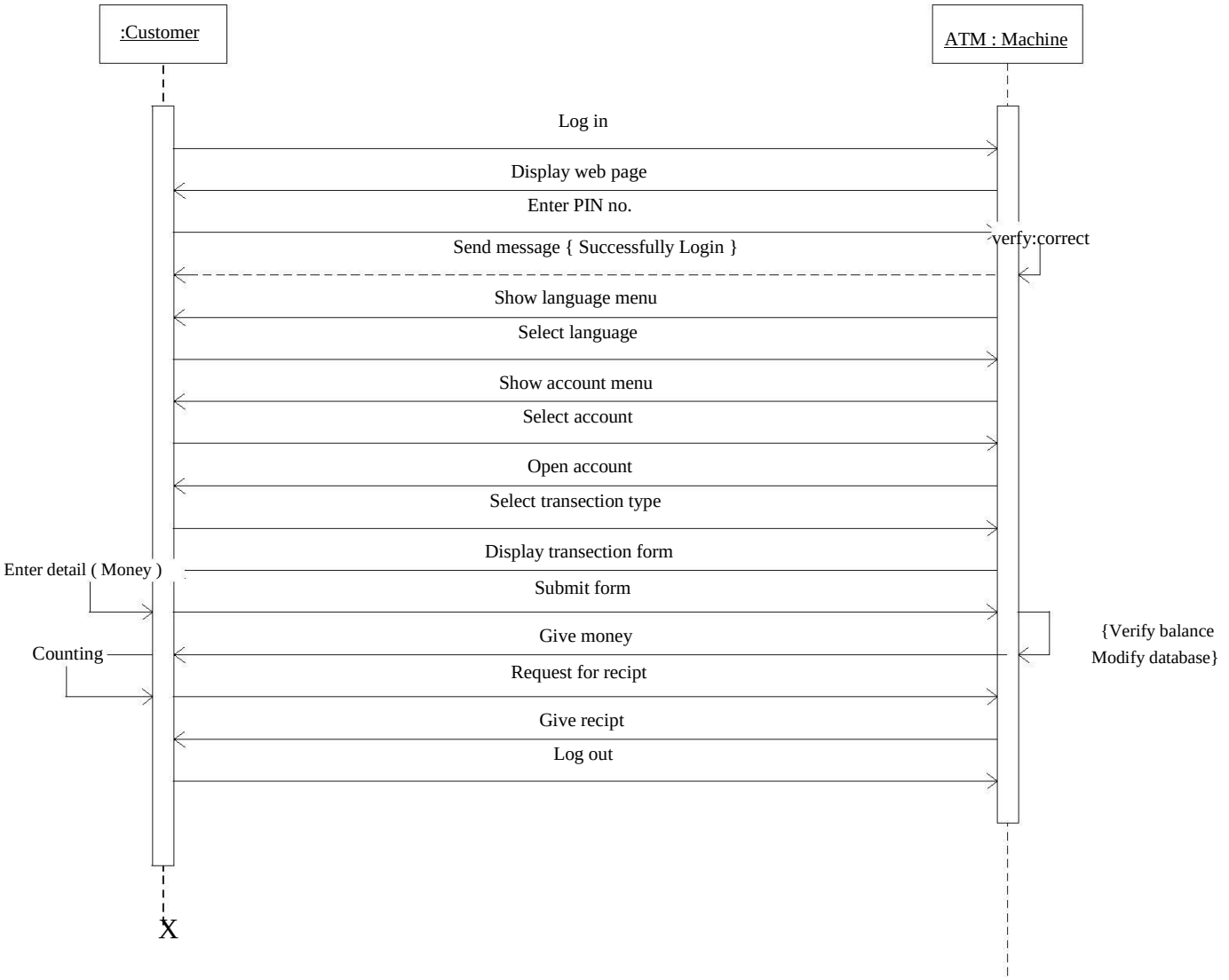


Sequence Diagram For ATM Management System:-

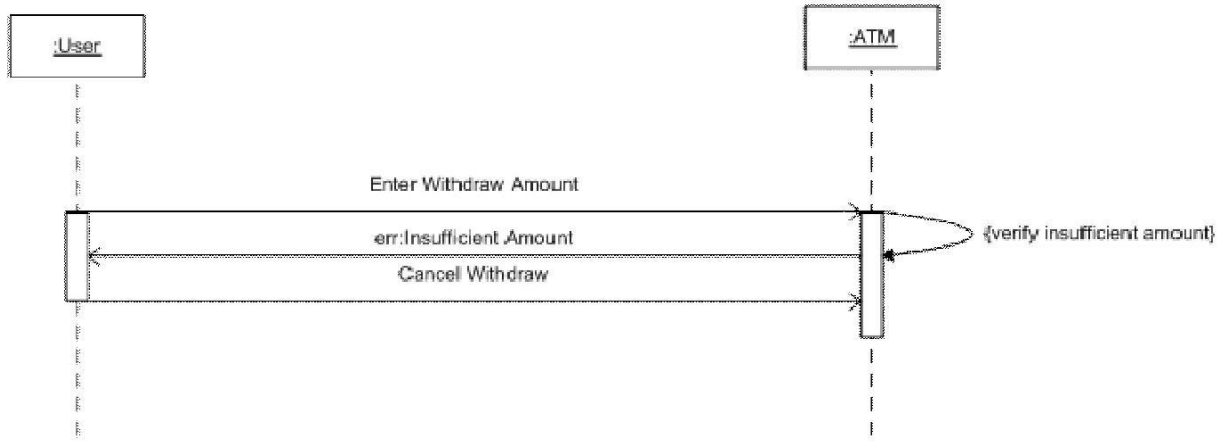
Create account



Transaction

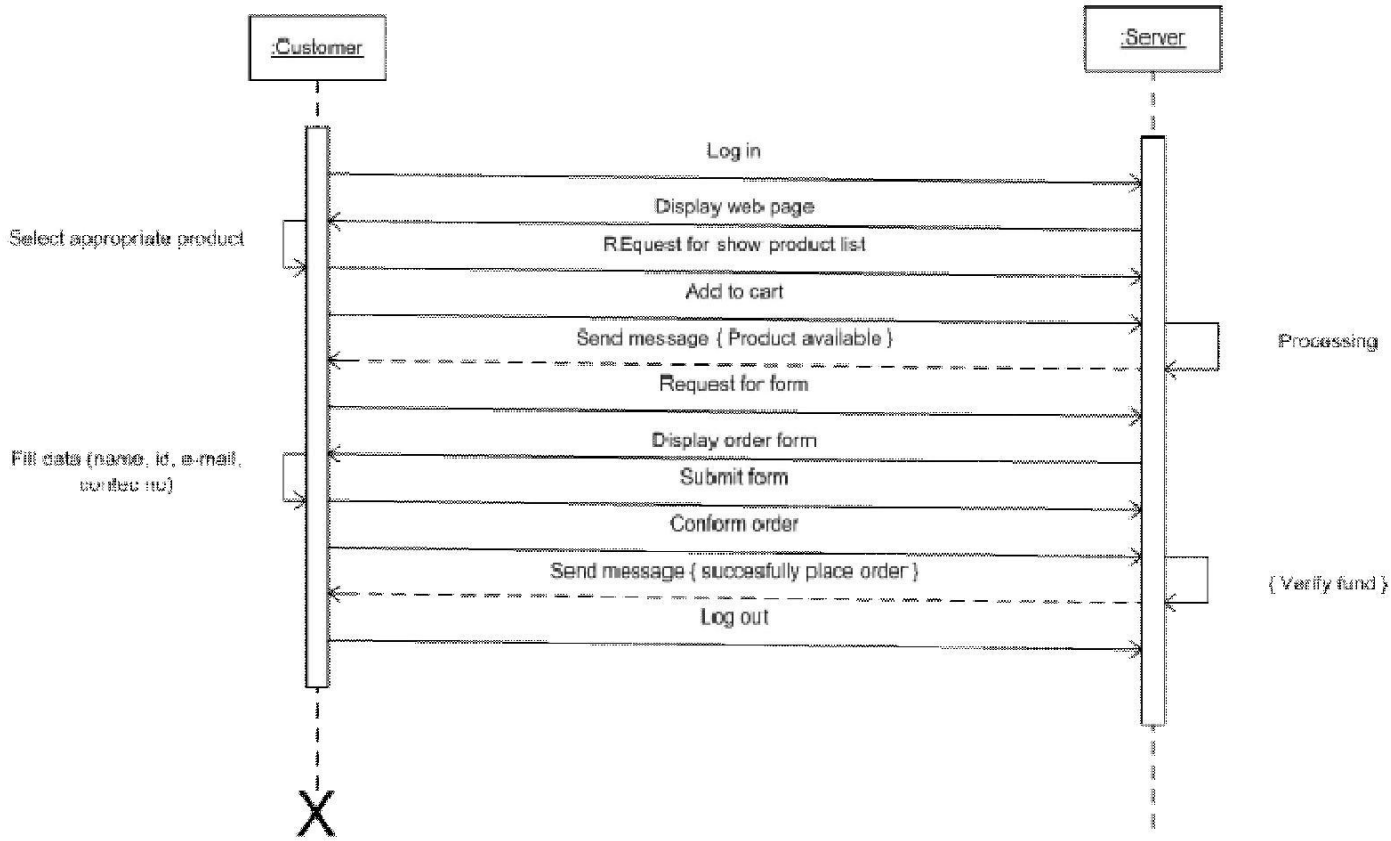


Exceptional case

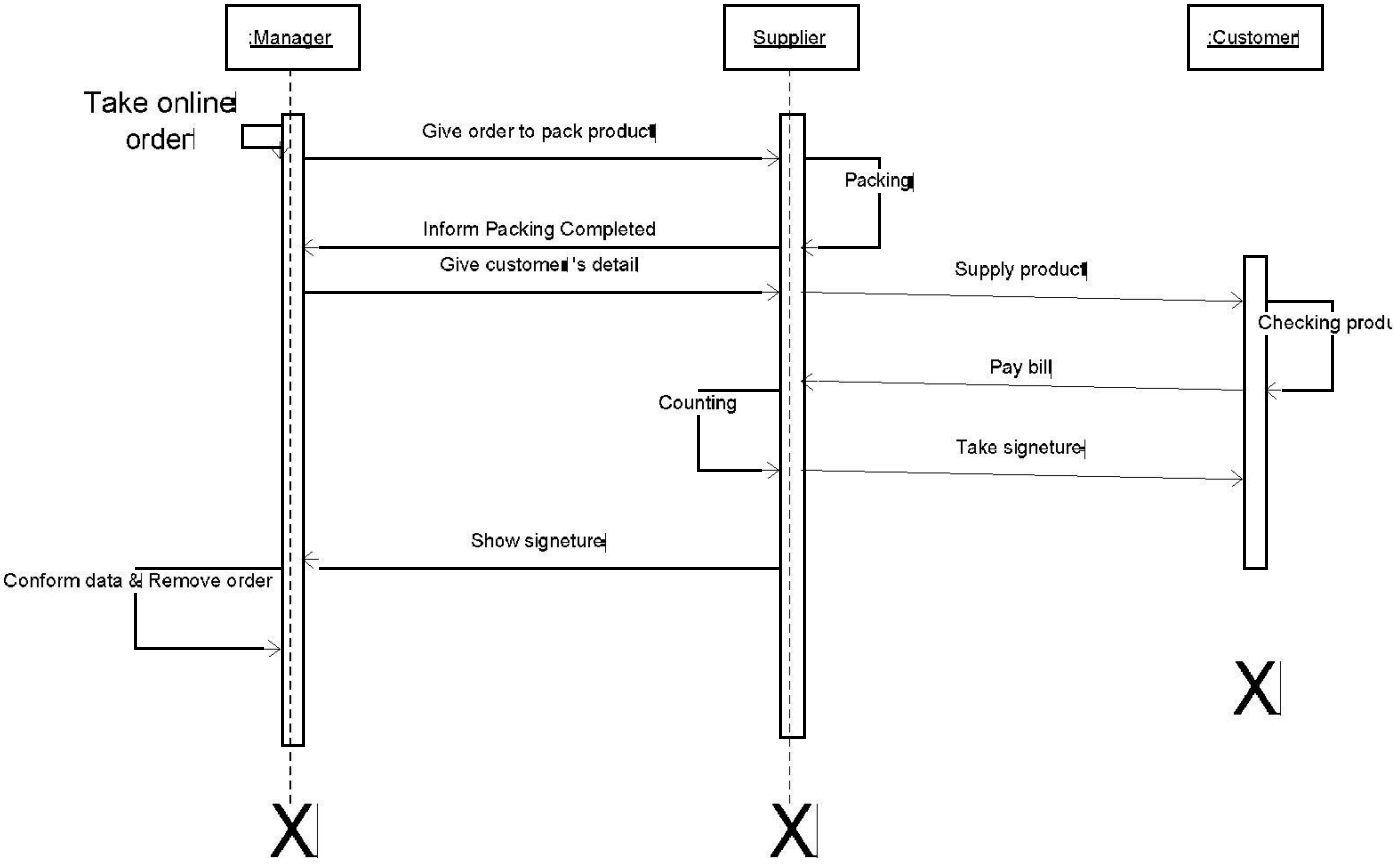


Sequence diagram for Online shopping system:-

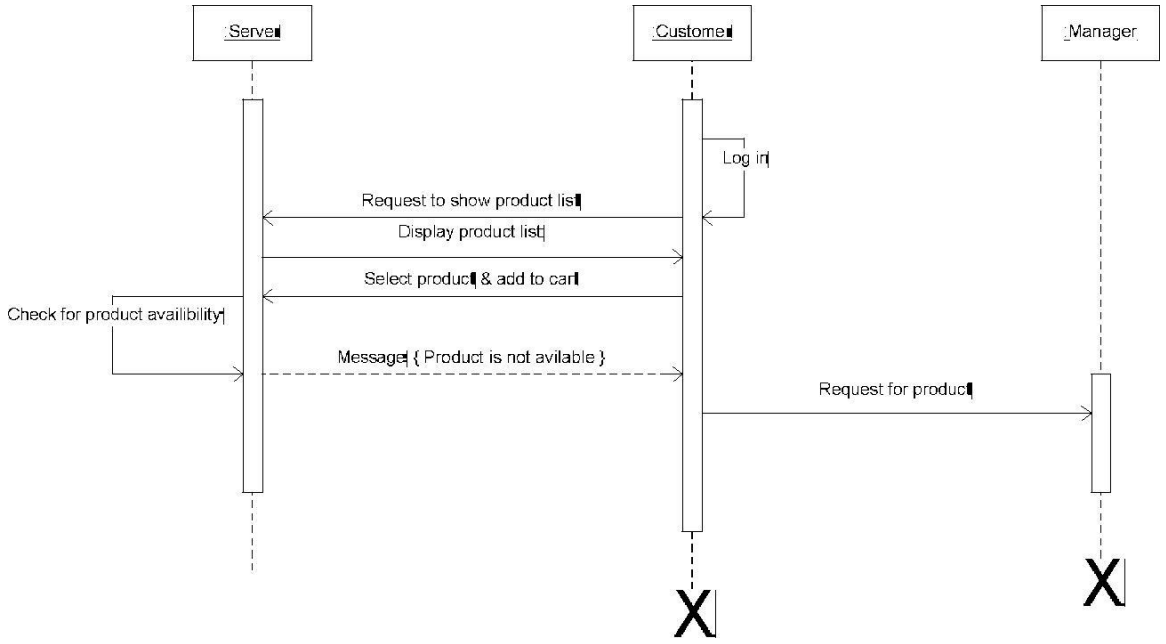
Place order



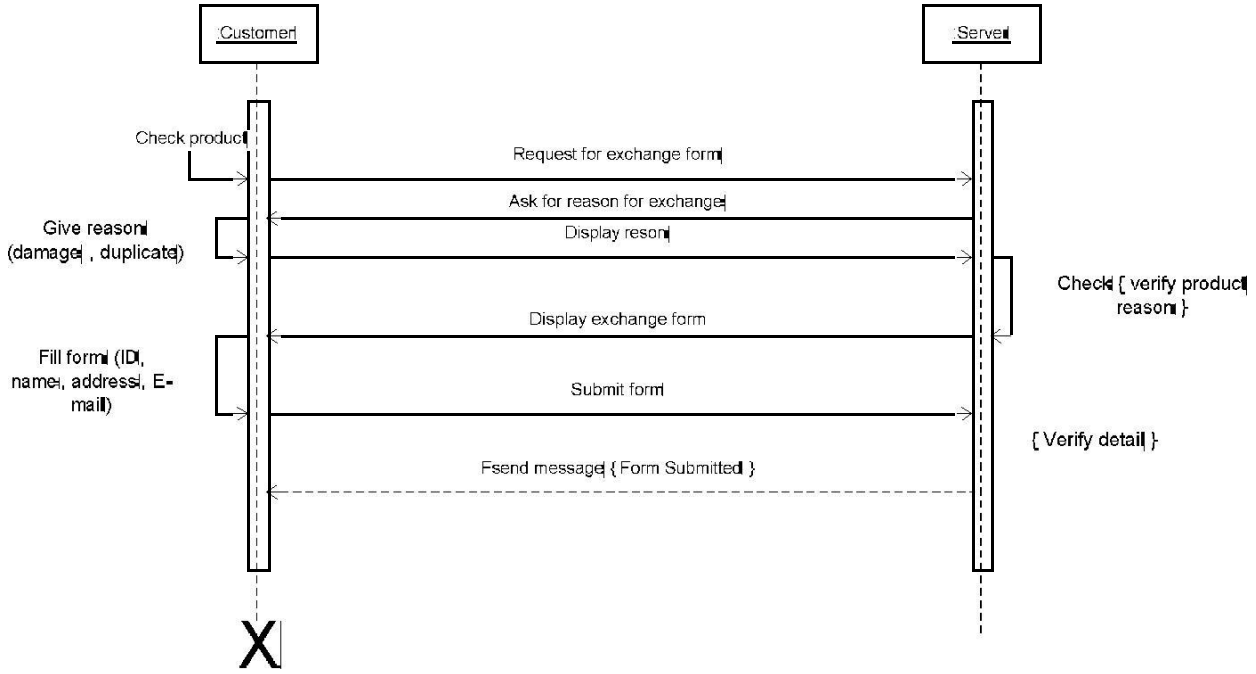
Supply order



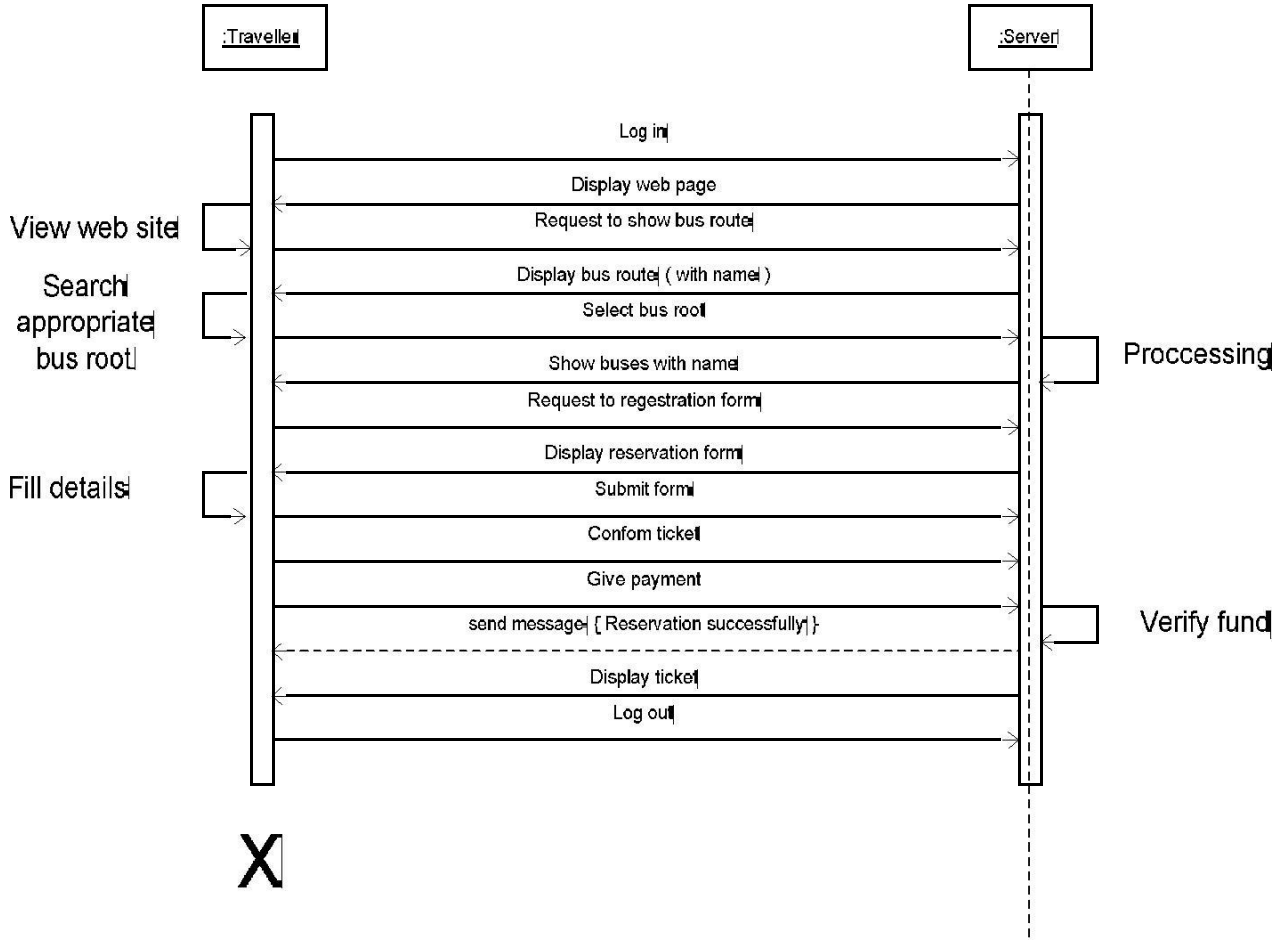
Product not available



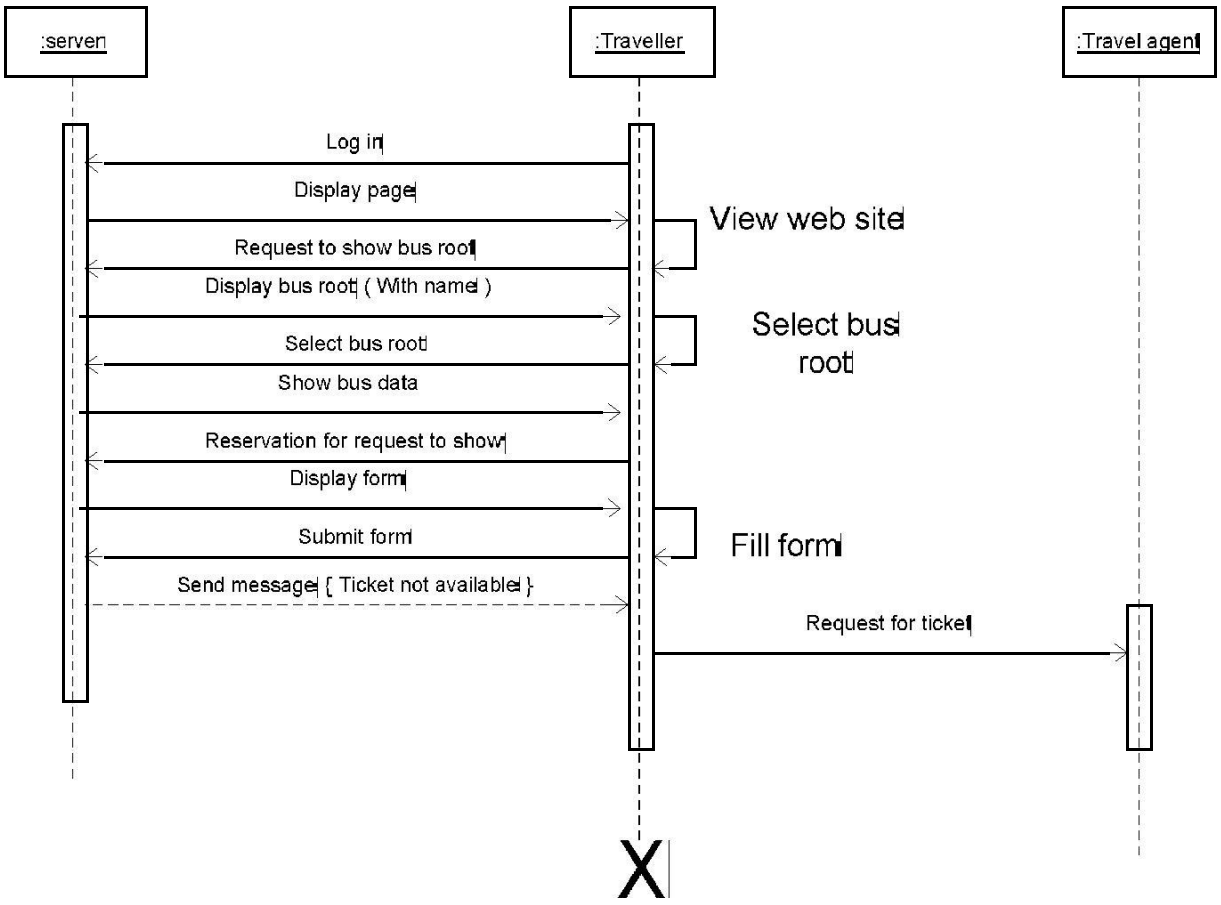
Product Exchange



Sequence diagram for Bus reservation system:-Reservation

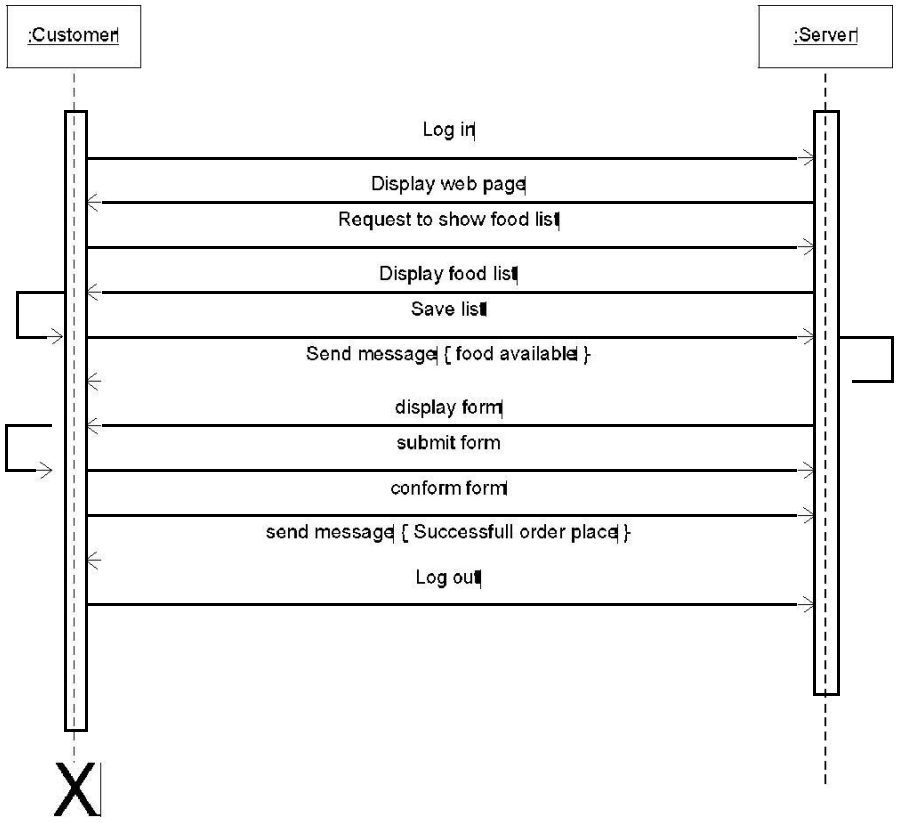


Ticket not available

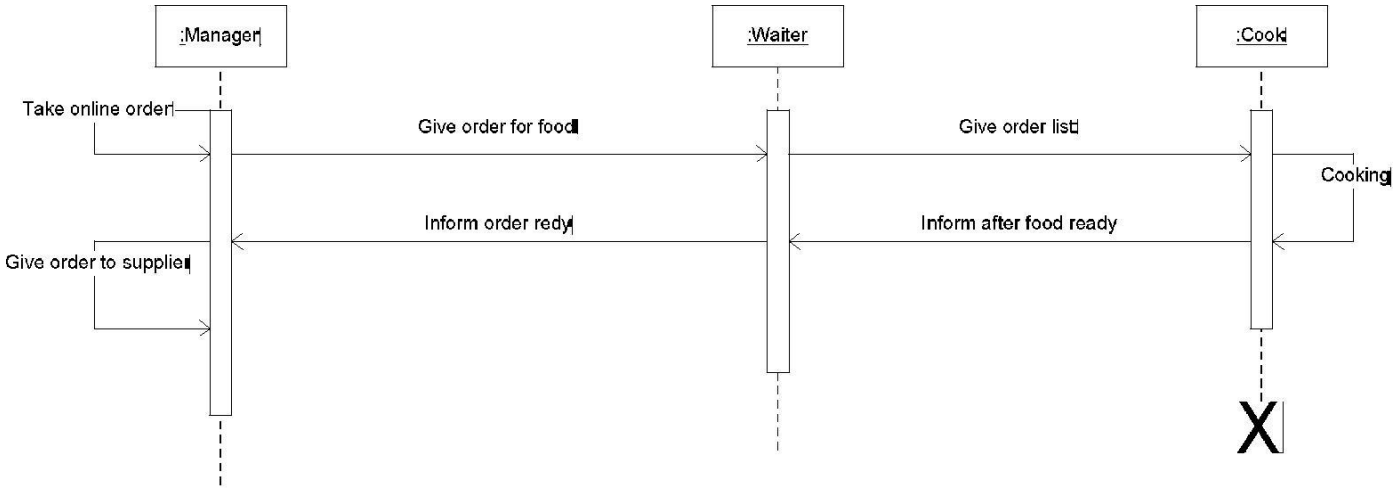


Sequence diagram for Online Restaurant Management System:-

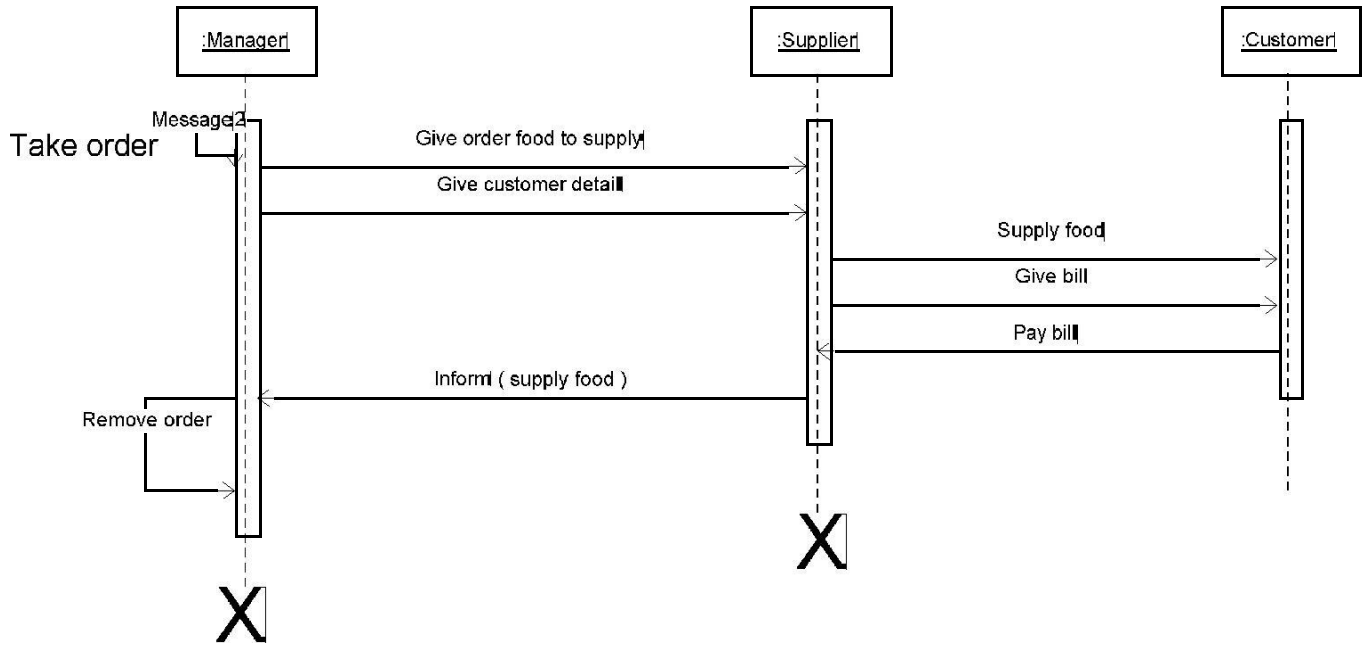
Place online order



Prepare food



Supply food


:Manager
:Supplier
:Customer
Message 2
Take order
Give order food to supply
Give customer detail
Supply food
Give bill
Pay bill
Inform (supply food)
Remove order

A Washing Machine system is used as case study. Following the proposed modeling approach, a UML model, consisting of one class diagram and eight sequence diagrams, was built to represent static and dynamic aspects of the Washing Machine system

The static view is represented by the class diagram illustrated in Fig 1. Basically, this model is composed of seven concrete classes (WashingMachine, Timer, WashOption, Engine, DoorSensor, WasterSensor, and TempSensor), the abstract class Sensor, and the interface Machine. The classes Engine and WashingMachine implement the interface Machine, while the concrete classes DoorSensor, TempSensor, and WaterSensor extend Sensor. The WashingMachine is the main class, which has the method main, beside of other methods representing the washing machine operations. WashingMachine is associated to the classes Engine, WaterSensor, WashOption, and Timer. In the class Timer, attributes, named value and duration, represent current value of the timer and duration for a given operation, respectively. This class has also somemethods such as *setDuration*, *getValue*, and *getDuration*, used to access attributes and whose usages are demonstrated in the sequence diagrams depicted in Fig. 1

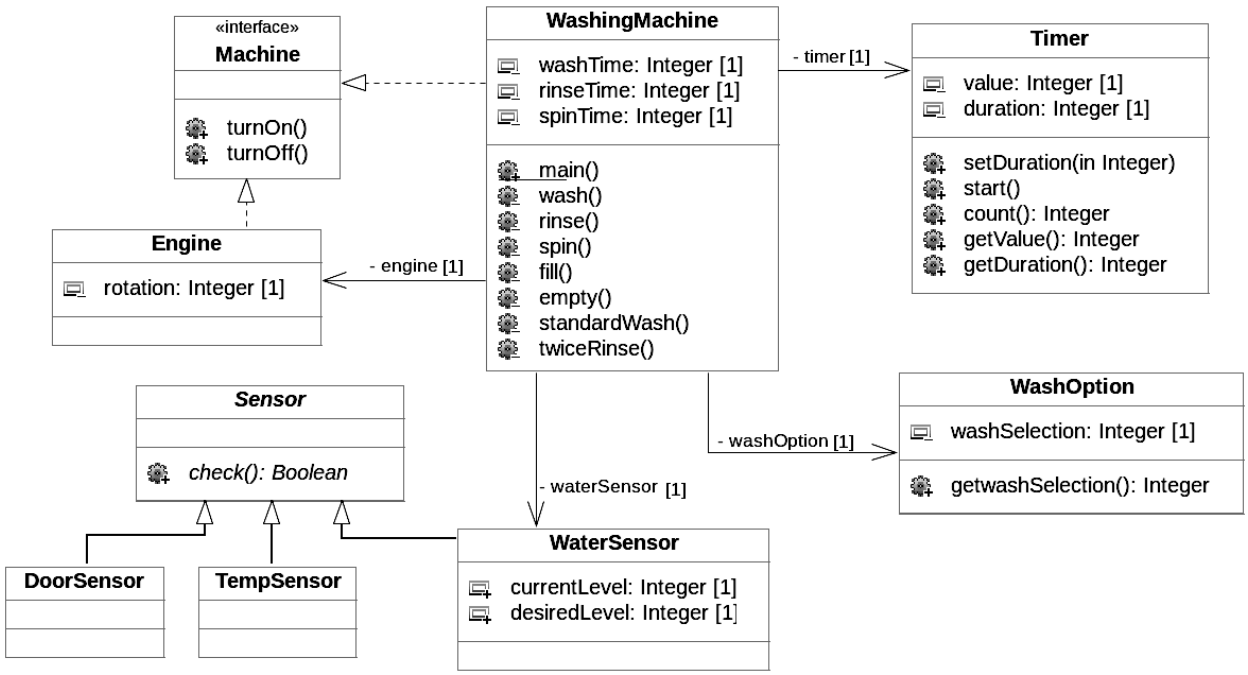


Fig. 1

Following our approach, a sequence diagram (sd) is used to represent the behavior of the method *main* of the class *WashingMachine*, which is depicted in Fig. 2. In the sd*Main*, two lifelines represent the objects *washMachine*and *washOption*, which interact on this scenario. Firstly, the washing machine must identify the desired operation. To check it, the object *washMachine*invokes the method *getWashSelection*from *washOption*. This method returns an integer, whose value is verified by the *alt* operator to determine the operation mode ("1" for *standardWash, "2"* for *twiceRinse*, and "3" for *spin*) selecting the method to be invoked in such case.

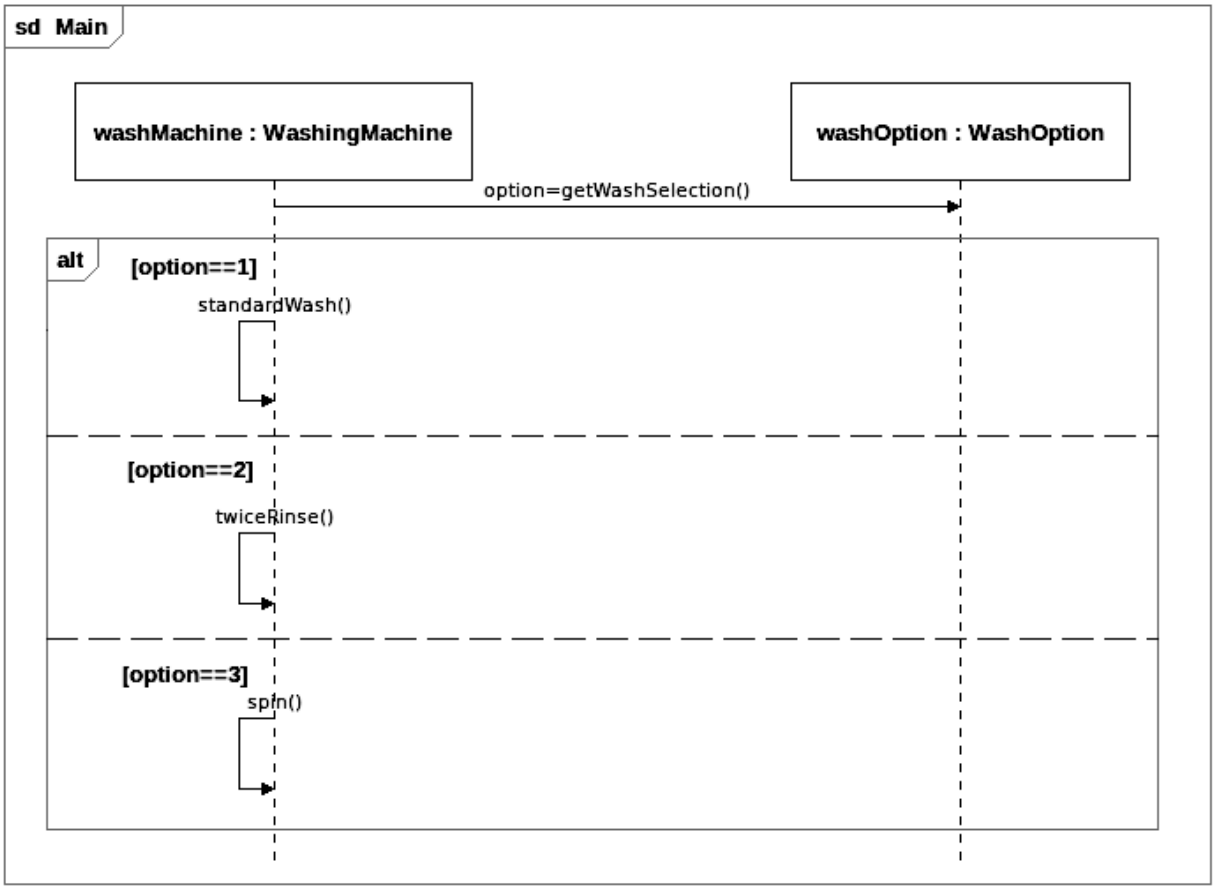


Fig. 2

Fig. 3 (a) illustrates a sequence diagram representing the spin operation, in which three lifelines, *washMachine*, *engine*, and *timer*, represent the objects evolved on this scenario. Firstly, the *washMachine*invokes the method *turnOn*from *engine*, requesting the starting of engine movement. After that, the time for the operation is sent to the *timer* using *setDuration(spinTime)*. The *timer* is the object responsible to control the period of time for each operation and it

executes the *Period* interaction (detailed on Fig. 3 (b)). The last operation is *turn off* the engine, represented by the message *turnOff*.

Fig 3 (b) shows the *Period* sequence diagram, which represents the duration of each machine operation, detailing the *Period* interaction referenced in the sd*Spin* (Fig.3 (a)). To initialize the count by the *timer*, *washMachine*invokes the method *start*. The

method*count* should be invoked several times in order to increment the current value of the object *timer*, which is modeled using a loop operator.

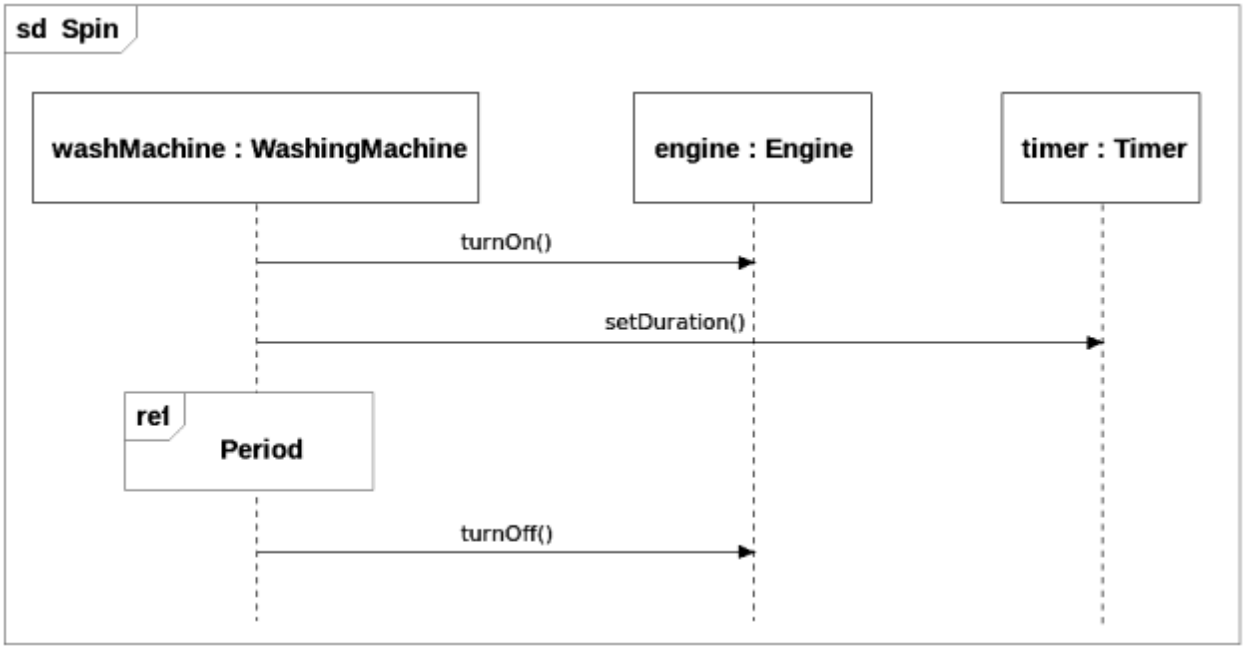


Fig. 3 (a)

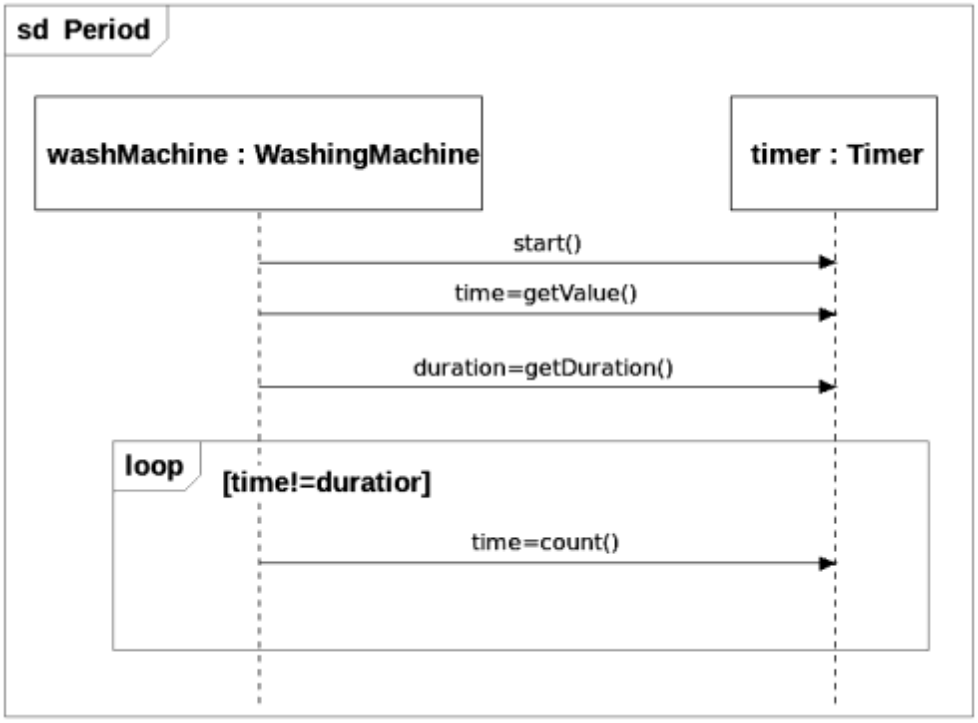


Fig. 3 (b)

Figures 4 (a) and 4(b) illustrate the code generated for the abstract class *Sensor* and for the interface *Machine*, declaring entities, its attributes and methods.

Fig. 5(a) illustrates the generated code of the class *WaterSensor*, subclass of *Sensor*, as represented by the statement *extends* in line 2. Fig. 5(b) depicts the code for *WashingMachine*, which implements the interface *Machine*. Both fragments of code has declarations of attributes, including attributes representing associations between classes, as, for example, the attributes *Engine*, *WaterSensor*, and *WashOption*in *WashingMachine*code. The generated code also include the constructors of both classes, and the class attributes are initialized from the arguments indicated by the constructor signature. The *WaterSensor*code includes an invocation for the constructor of the super class (in line 13) and the declaration of the inherited method *check* (in line 20).

Our approach is able to generate get and set methods including definition of parameters and return. The fragments of code depicted in Fig 6(a) and 6(b) describes get and set methods generated for some attributes of the class *WashingMachine*. To demonstrate the behavioral code generation, code fragments of the method main of the *WashingMachine*, were generated from the corresponding sequence diagram (Fig 2) and are depicted in Fig. 7. In the first line of the method

body (line 88), the method *getWashSelection*is invoked, according the message sent from *WashingMachine*to the *washOption*, and returning a value that is signed to the variable option. Most interactions in the sd*Main* are represented inside an alt fragment, responsible to define the operation mode according to the option value. The code correspondent to this fragment starts in line 89, and finishes in line 104. The generated code includes invocations of methods from other classes (line 141) and from the owner class (line 91), and also includes the arguments passed as parameters for the methods (see line 142). In

line 148, the loop fragment represented in the sd from Fig. 3(b) is represented by the statement *while*. To finalize the code, the tool generates the methods defined in the interface *Machine* (from 155 to 158), which must be implemented in this class, since a realization relationship was defined between these entities.

```
1
2public abstract class Sensor{
3
4    /**Attribute of Return Method check */
5    public boolean full;
6
7    /** Methods */
8    public abstract boolean check();
9}
```

Fig. 4 (a)

```
1
2public interface Machine{
3
4    /**Method */
5    void turnOn();
6    void turnOff();
7}
```

Fig. 4 (b)

```
1
2 public class WaterSensor extends Sensor{
3
4     /**Attributes */
5     public int currentLevel;
6     public int desiredLevel;
7
8     /**Attribute of Return Method levelTest */
9     public boolean value;
10
11    /** Constructor */
12    public WaterSensor( int currentLevel , int desiredLevel ){
13        super();
14        this.currentLevel = currentLevel;
15        this.desiredLevel = desiredLevel;
16    }
17
18    /**Abstract Method of Super */
19    public boolean check(){
20        return full;
21    }
22
```

Fig. 5 (a)

```
 1
 2 public class WashingMachine implements Machine{
 3
 4     /**Attributes */
 5     private Engine engine;
 6     private WaterSensor waterSensor;
 7     private WashOption washOption;
 8     private Timer timer;
 9     private int washTime;
10     private int rinseTime;
11     private int spinTime;
12
13     /** Constructor */
14     public WashingMachine( Engine engine , WaterSensor waterSensor ,
15             WashOption washOption , Timer timer , int washTime ,
16             int rinseTime , int spinTime ){
17         this.engine = engine;
18         this.waterSensor = waterSensor;
19         this.washOption = washOption;
20         this.timer = timer;
21         this.washTime = washTime;
22         this.rinseTime = rinseTime;
23         this.spinTime = spinTime;
24     }
```

Fig. 5(b)

```
57      /** Set */
58      public void setEngine( Engine engine ){
59          this.engine = engine;
60      }
61
62      public void setWaterSensor( WaterSensor waterSensor ){
63          this.waterSensor = waterSensor;
64      }
65
66      public void setWashTime( int washTime ){
67          this.washTime = washTime;
68      }
```

```
27      /** Get */
28      public Engine getEngine(){
29          return this.engine;
30      }
31
32      public WaterSensor getWaterSensor(){
33          return this.waterSensor;
34      }
35
36      public int getWashTime(){
37          return this.washTime;
38      }
```

Fig. 6 (a)

```
 85      /** Methods */
 86      public static void main( String args[] ){
 87          /** Specified from Sequence Diagram Main */
 88          option = washOption.getwashSelection();
 89          switch(option){
 90              case 1:
 91                  standardWash();
 92                  break;
 93
 94              case 2:
 95                  twiceRinse();
 96                  break;
 97
 98              case 3:
 99                  spin();
100                  break;
101
102              default:
103                  break;
104              }//endSwitch
105      }
```

Fig. 6 (b)

```
139     private void spin(){
140         /** Specified from Sequence Diagram Spin */
141         engine.turnOn();
142         timer.setDuration(spinTime);
143 
144         /** Specified from Sequence Diagram Period */
145         timer.start();
146         time = timer.getValue();
147         duration = timer.getDuration();
148         while(time != duration){
149             time = timer.count();
150         }//endWhile
151         engine.turnOff();
152     }
153 
154     /** Methods From Realization of Machine */
155     public void turnOn(){
156     }
157 
158     public void turnOff(){
159     }
```

Fig. 7

Assignment No. 7

Title: Prepare a State Model

Identify States and events for your system.
Study state transitions and identify Guard conditions.
Draw State chart diagram with advanced UML 2 notations.
Implement the state model with a suitable object-oriented language

Date of Completion		**Marks for Regularity (2)**	
		Marks for Presentation (3)	
Remark		**Marks for Understanding (5)**	
Sign. of Staff		**Total Marks (10)**	

Prof. R.S. Badodekar **414459 : Computer Laboratory VIII**

Assignment No. 7 - Prepare a State Model

Introduction

- *A **state diagram** is a graph in which nodes correspond to states and directed arcs correspond to transitions labeled with event names.*
- A state diagram combines states and events in the form of a network to model all possible object states during its life cycle, helping to visualize how an object responds to different stimuli.
- A state diagram is a graph whose nodes are states and whose directed arcs are transitions between states.
- A state diagram specifies the state sequence caused by event sequence.
- State names must be unique within the scope of a state diagram.
- All objects in a class execute the state diagram for that class, which models their common behavior.
- We can implement state diagrams by direct interpretation or by converting the semantics into equivalent programming code.

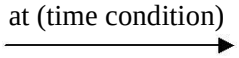
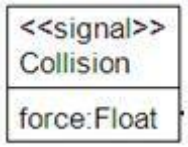
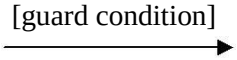
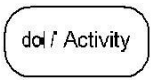
Purpose

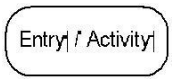
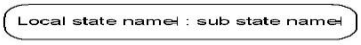
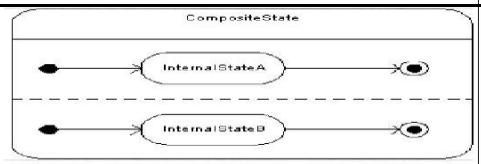
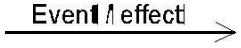
- The state model describes those aspects of objects concerned with time and the sequencing of operations events that mark changes, states that define the context for events, and the organization of events and states.
- They are used to give an abstract description of the behavior of a system.
- It provides direction and guidance to the individual counties within the states.
- It specifies the possible states, what transitions are allowed between states.
- It describes the common behavior for the objects in a class and each object changes its behavior from one state to another.
- It is used to describe the dependence of the functionality on the state of the system that is how the functionality of an object depends on its state and how its state changes as a result of the events that it receives.
- It describes dynamic behavior of the objects of the system.

When to use: State Diagram

- They are perfectly useful to model behavior in real time system.
- Each state represents a named condition during the life of an object during which it satisfies some condition or waits for some event.
- It determines how objects of that class react to events.
- For each object state, it determines what actions the object will perform when it receives an event.

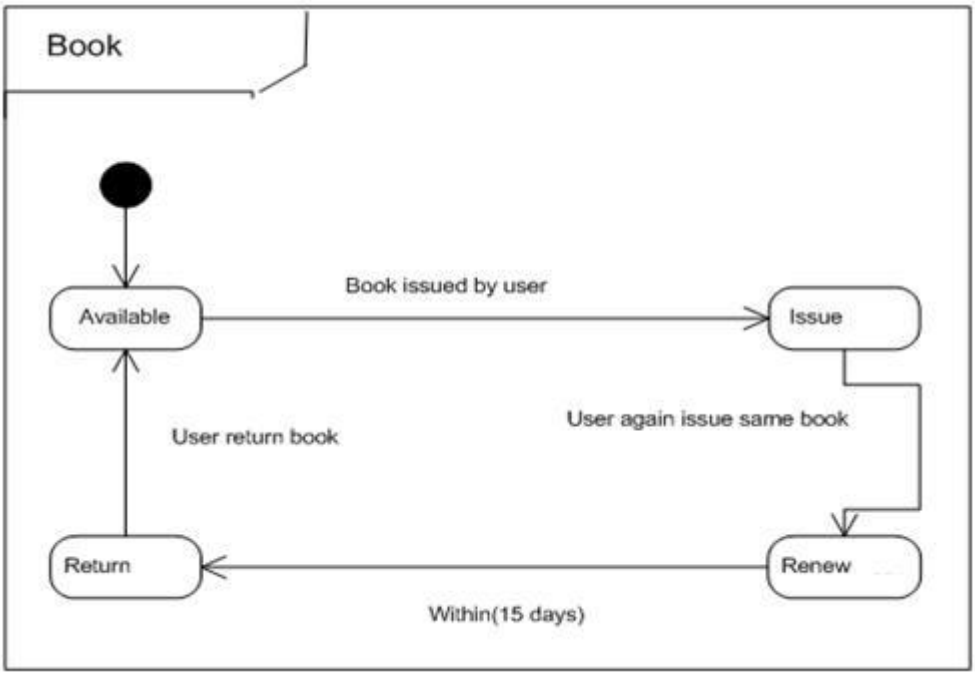
State Diagram Notations

No.	Name	Notation	Description
1	State		A state is an abstraction of the values and links of an object. State models a situation during which some (usually implicit) invariant condition holds.
2	Transition		A transition is a directed relationship between a source state and a target state. It may be part of a compound transition, which takes the state machine from one state configuration to another
3	Event		A transition is an instantaneous change from one to another state
4	Change Event		A change in value of a Boolean expression
5	Time Event		The arrival of an absolute time or the passage of a relative amount of time
6	Signal Event		Receipt of an explicit, named, asynchronous communication among objects.
7	Guarded transition		A guard condition is a Boolean expression that must be true in order for a transition to occur.
8	Do activity		A do activity an activity that continuous for extended time within state.

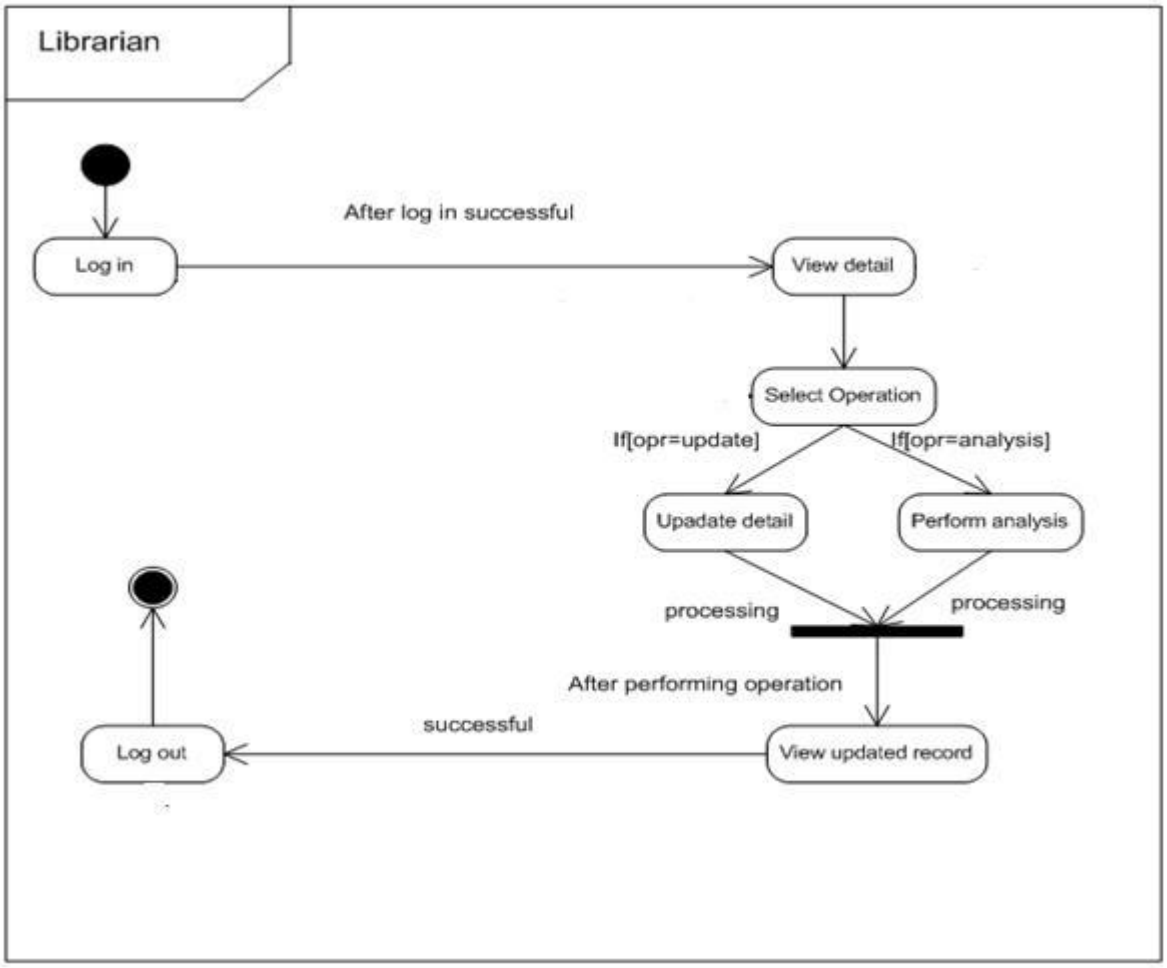
9	Entry activity		An state is entered by any incoming transition the entry activity is performed
10	Exit activity		When the state is exited by any outgoing transition the exit activity is performed
11	Nested State Diagram Sub machine Diagram		A submachine state specifies the insertion of the specification of a submachine. The state machine that contains the submachine state is called the containing state machine.
12	Composite State		A state can be refined hierarchically by composite states.
13	Activity effect		An activity is actual behavior that can be invoked by any number of effects
14	Initial state point		It shows the starting state of object.
15	Final state point		It shows the terminating state of object.

Examples:

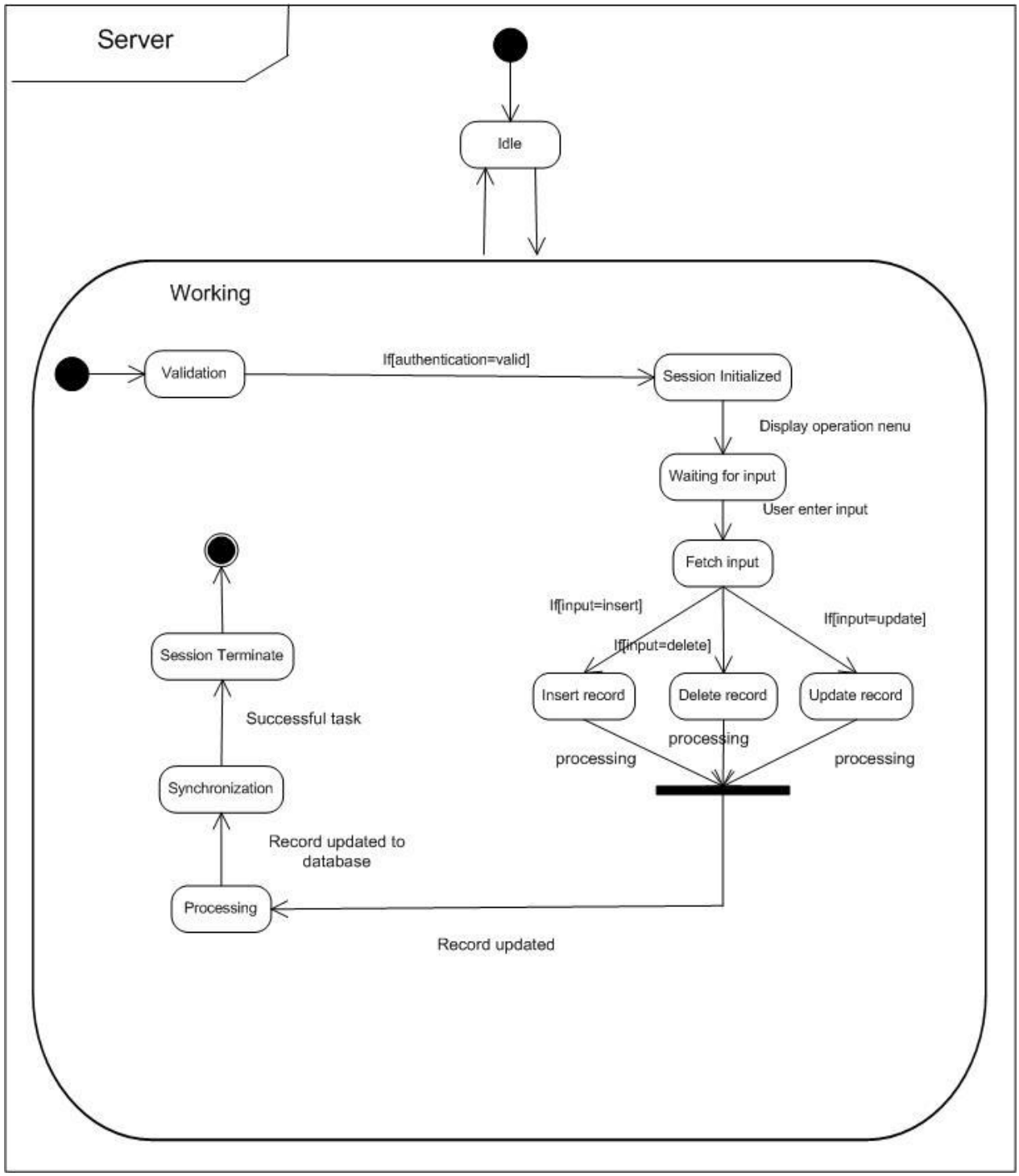
State diagram for library management system



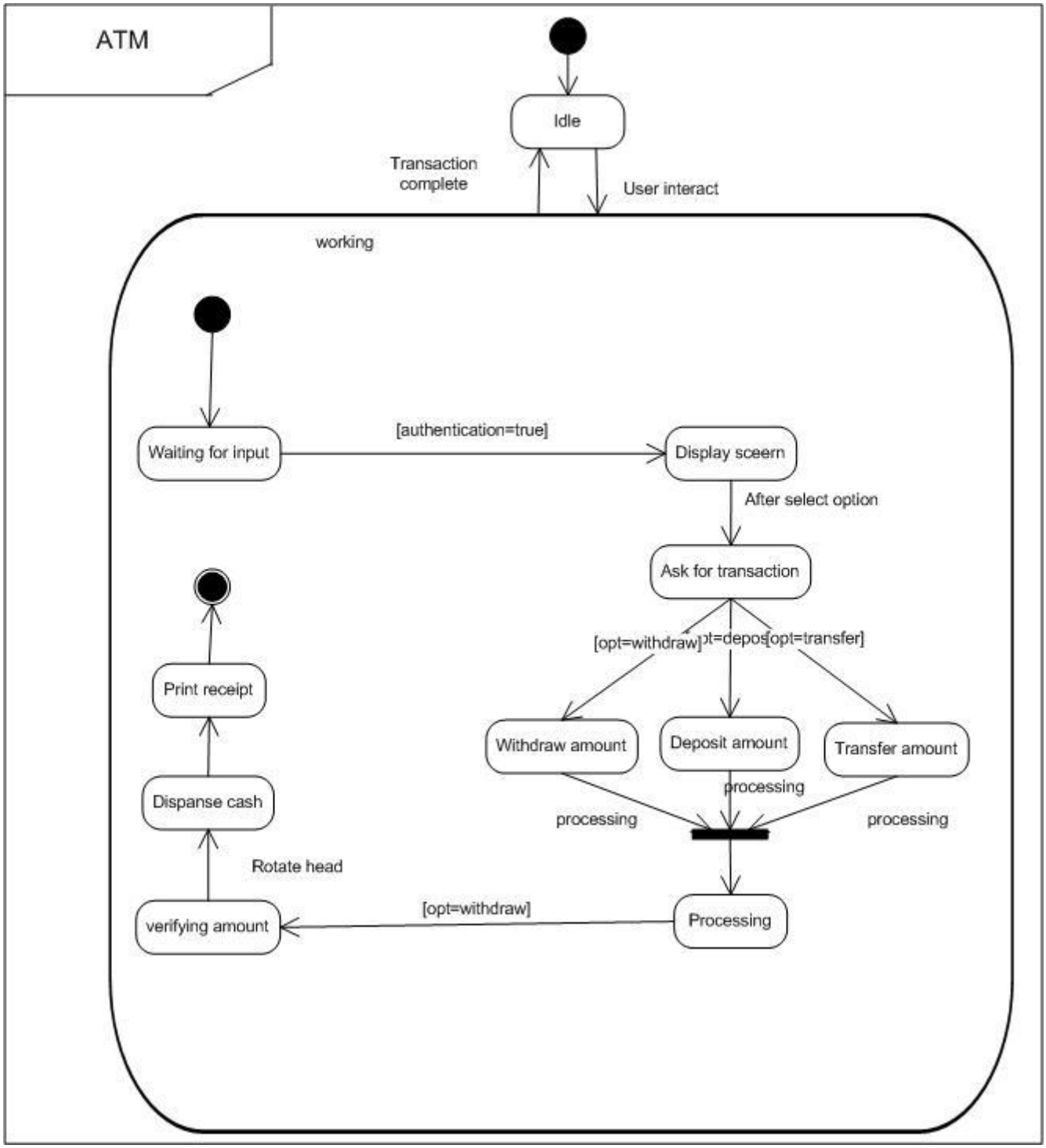
State diagram for library management system



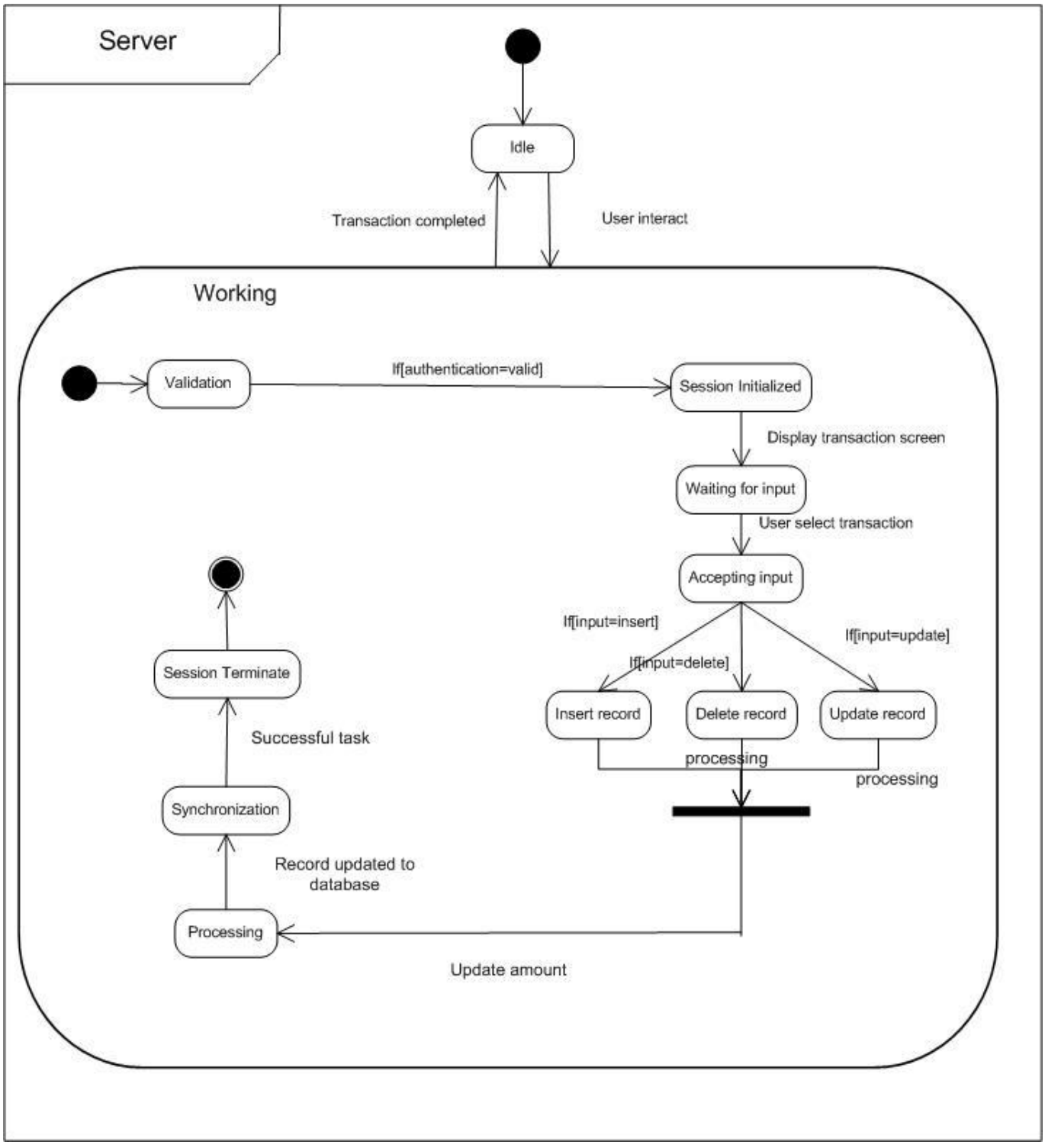
State diagram for library management system



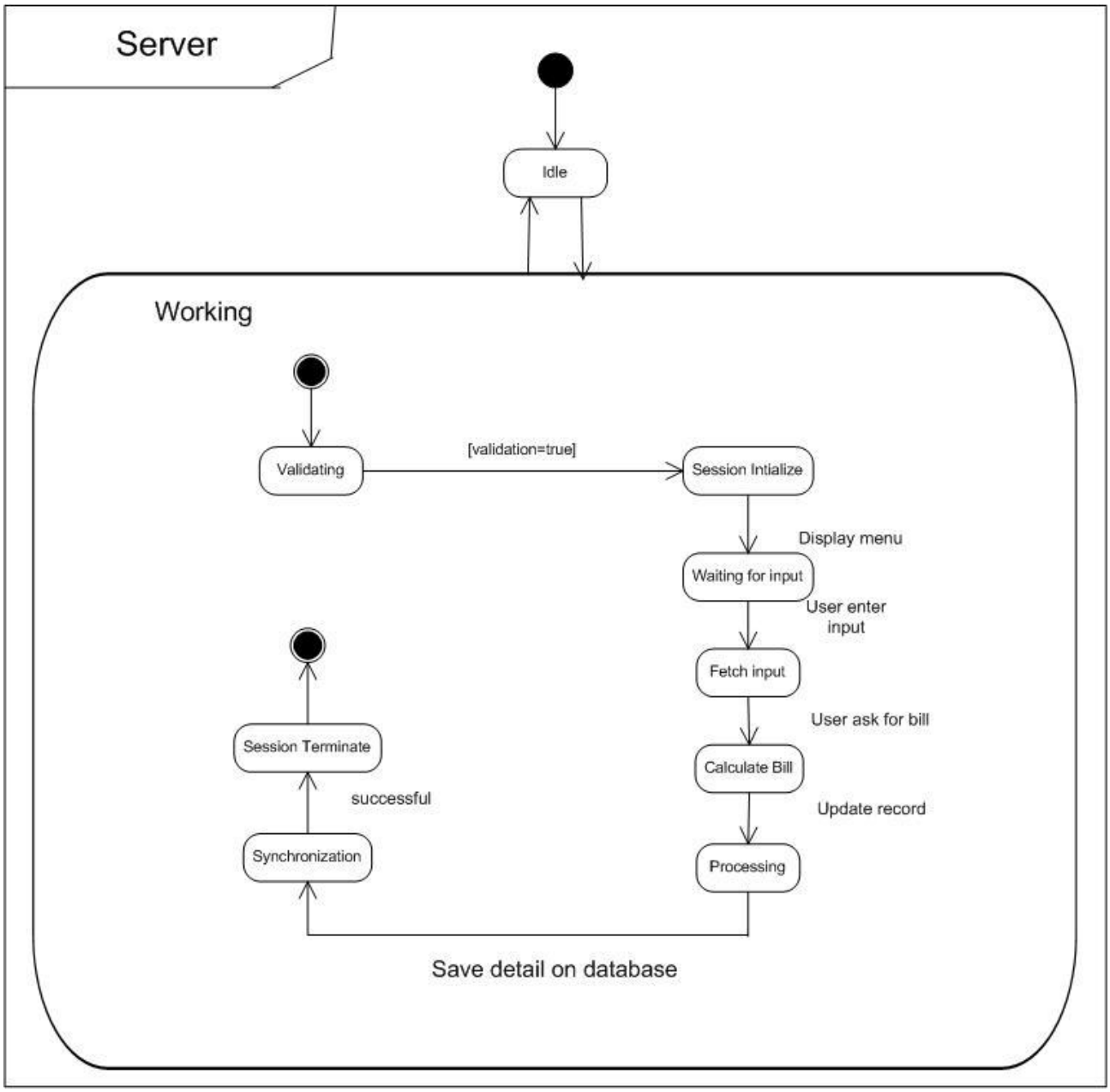
State diagram for ATM Management System



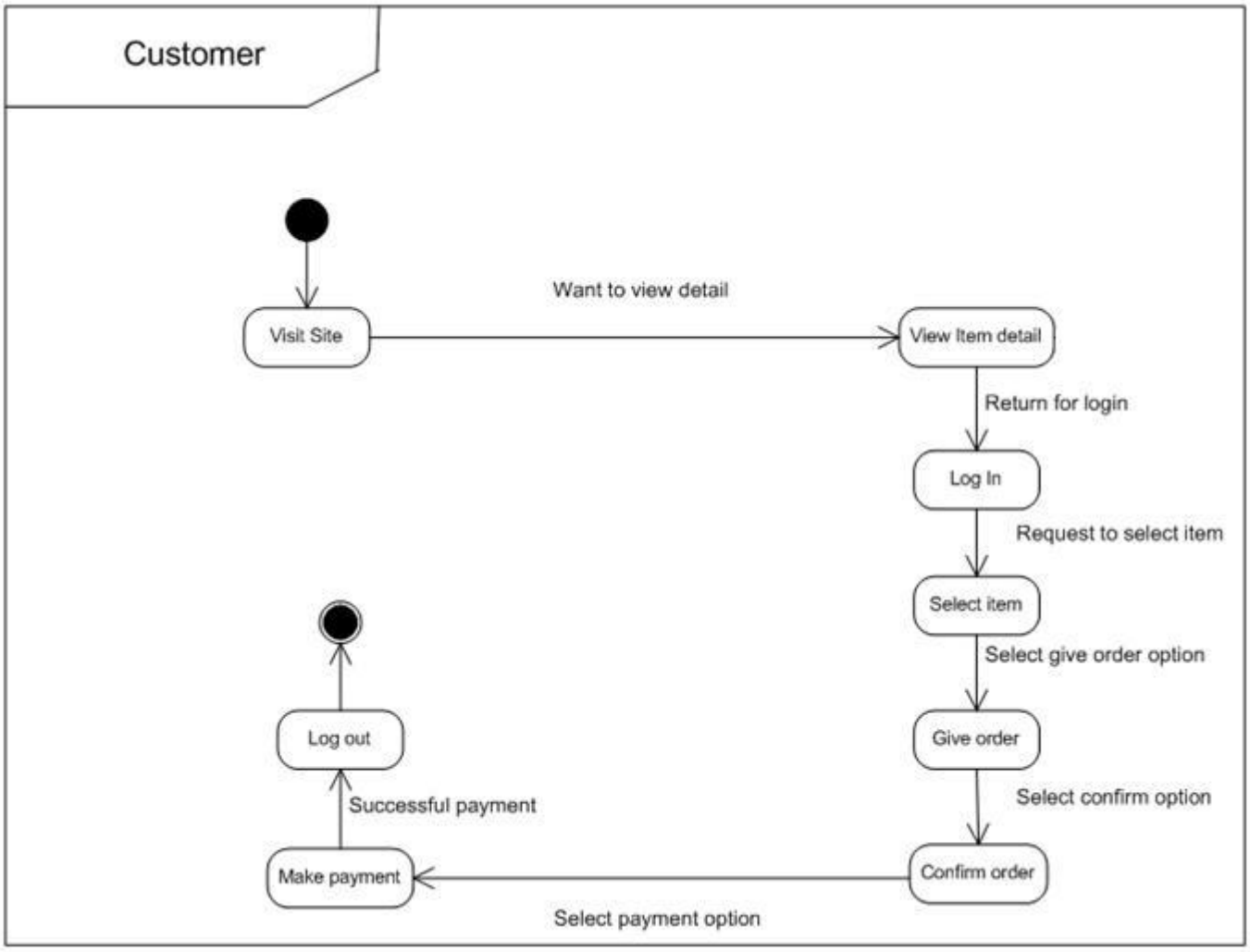
State diagram for ATM Management System



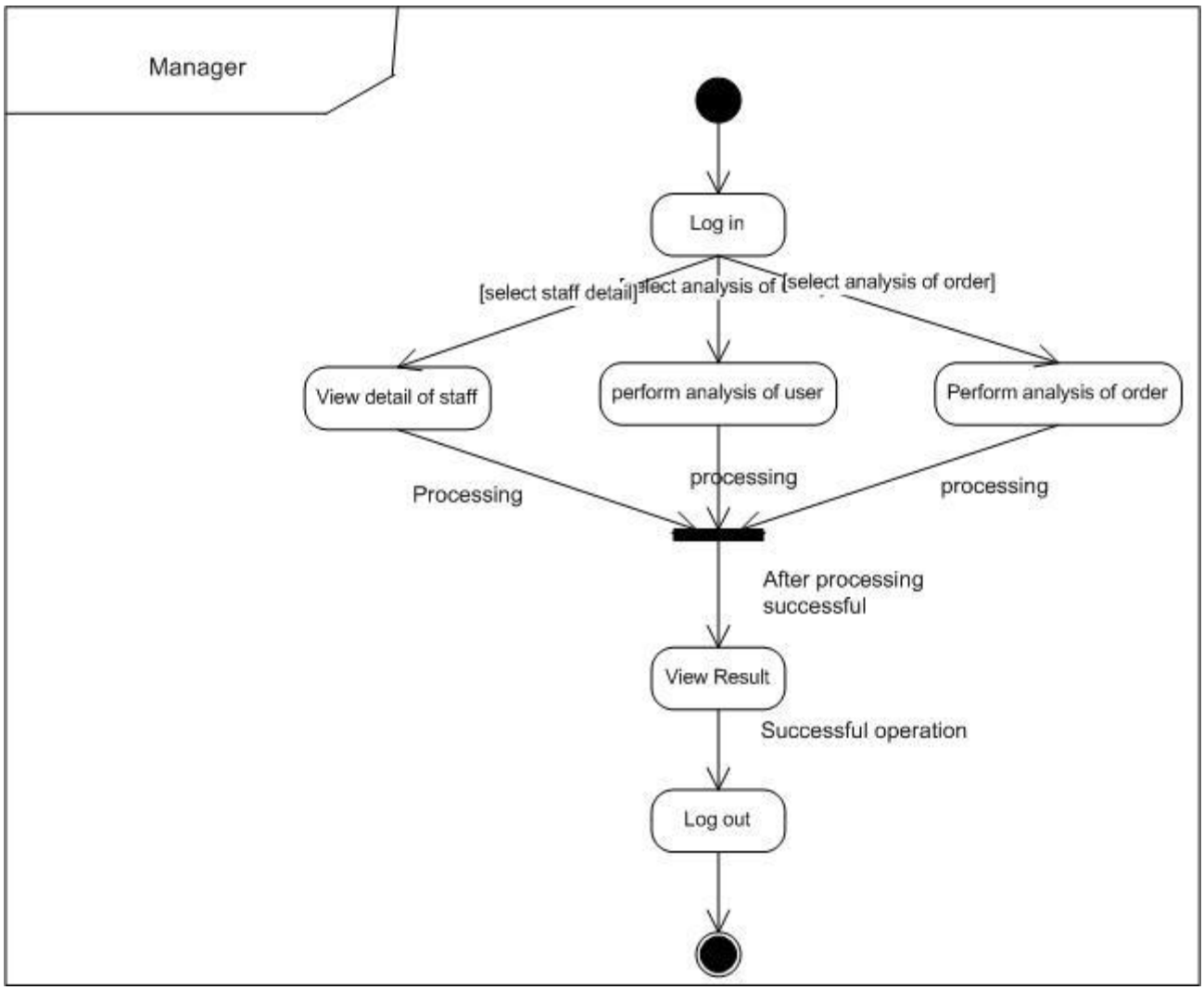
State diagram for Online Restaurant management



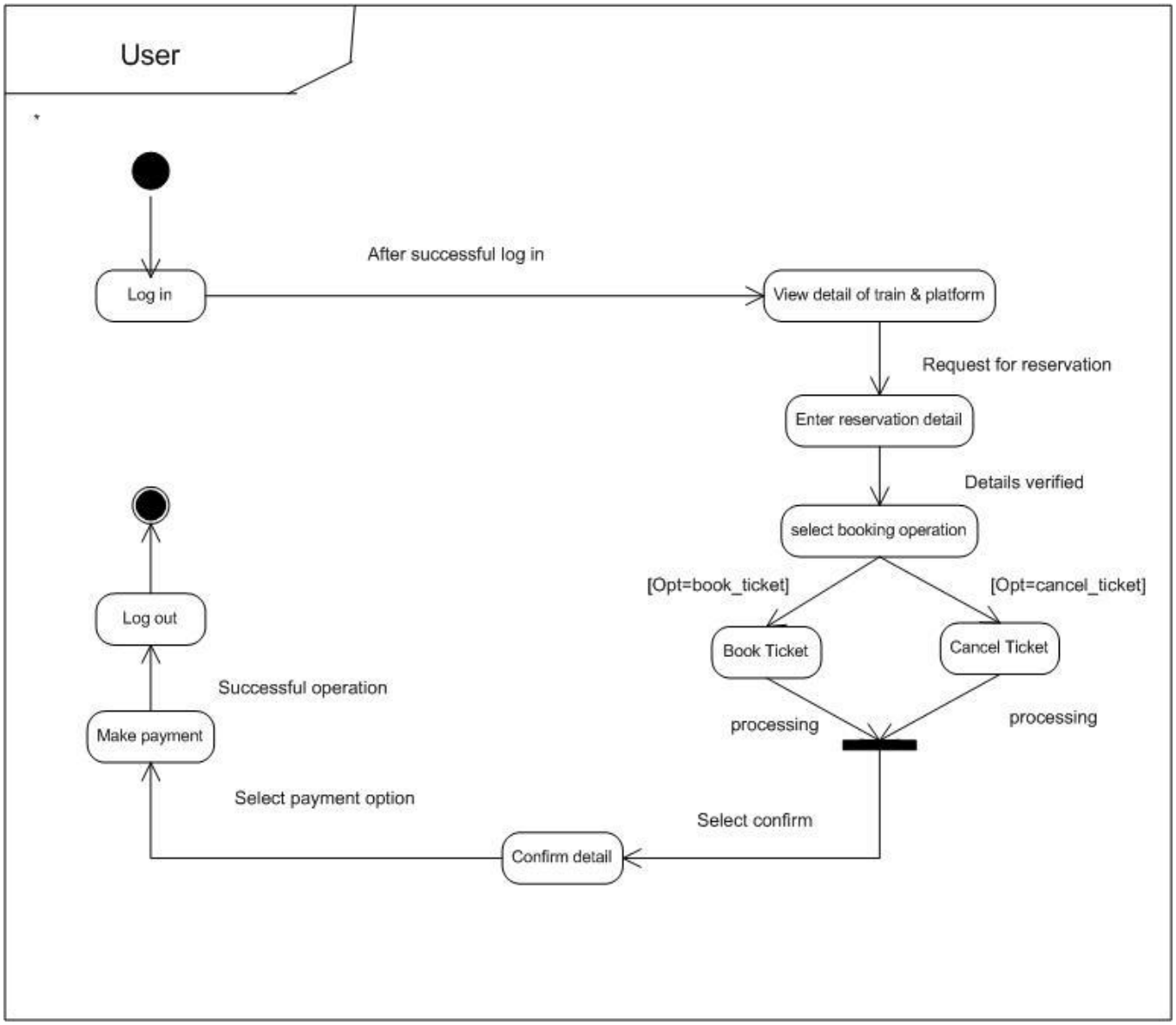
State diagram for Online Restaurant Management System



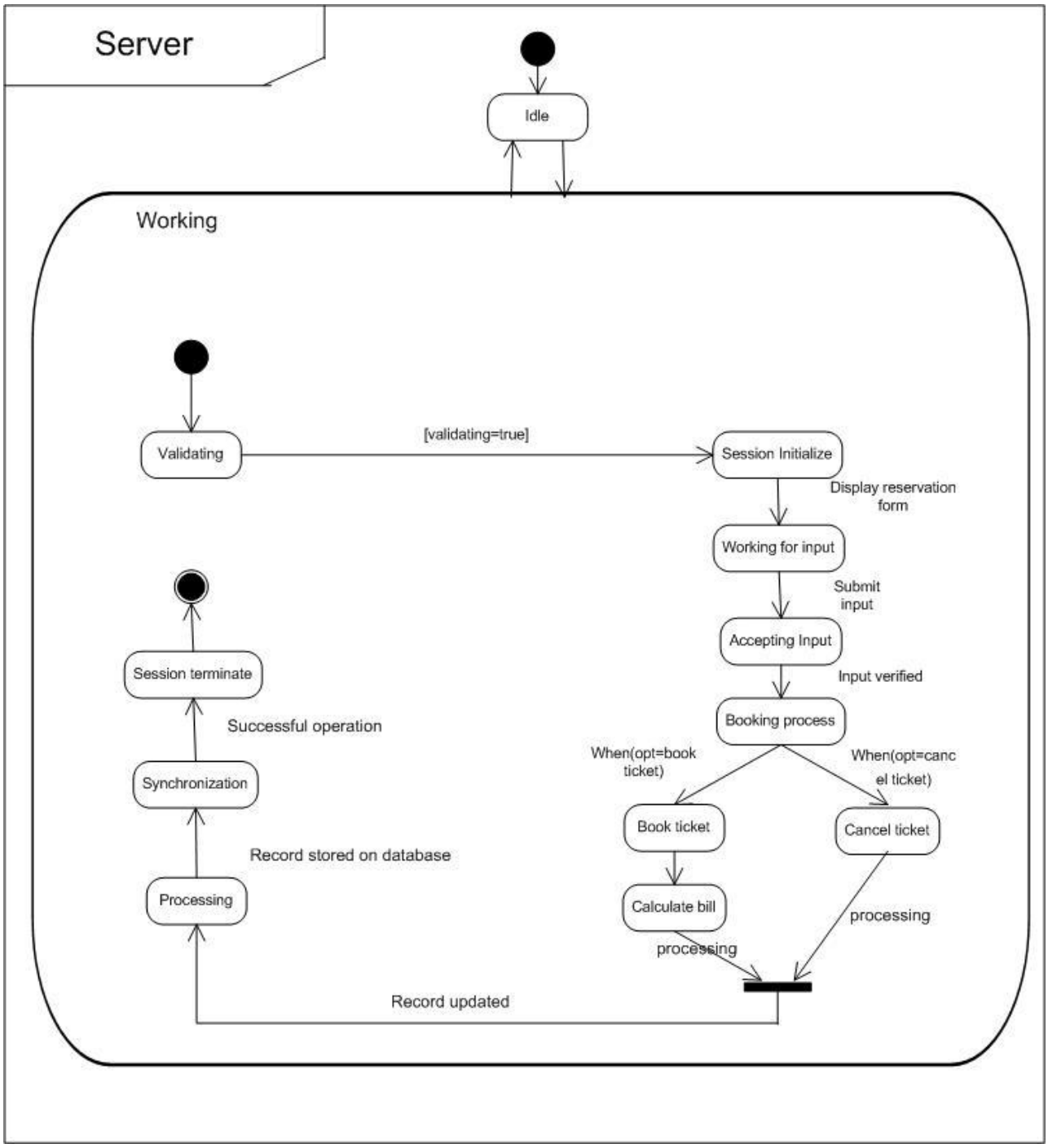
State diagram for Online Restaurant management



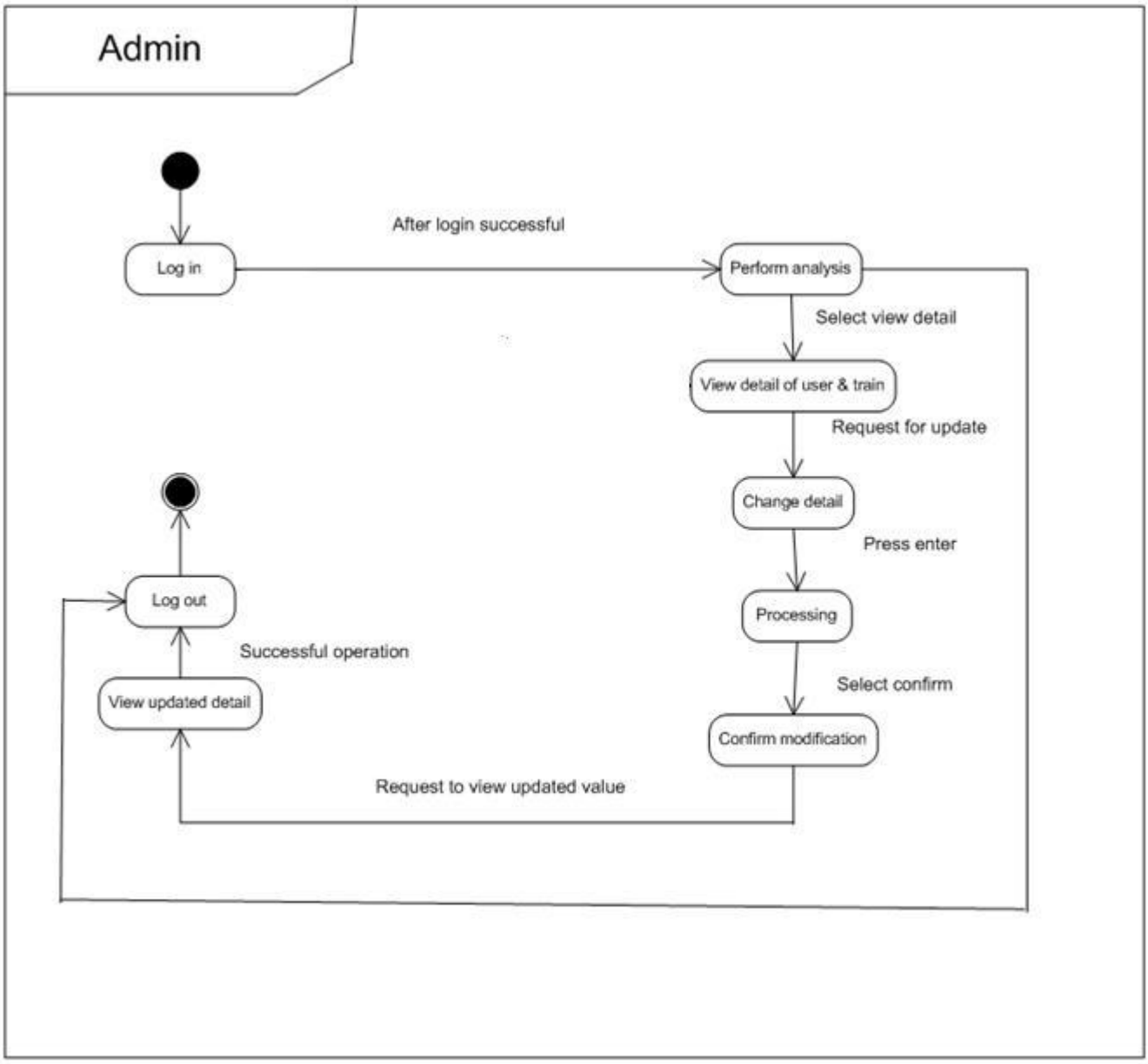
State diagram for online reservation system



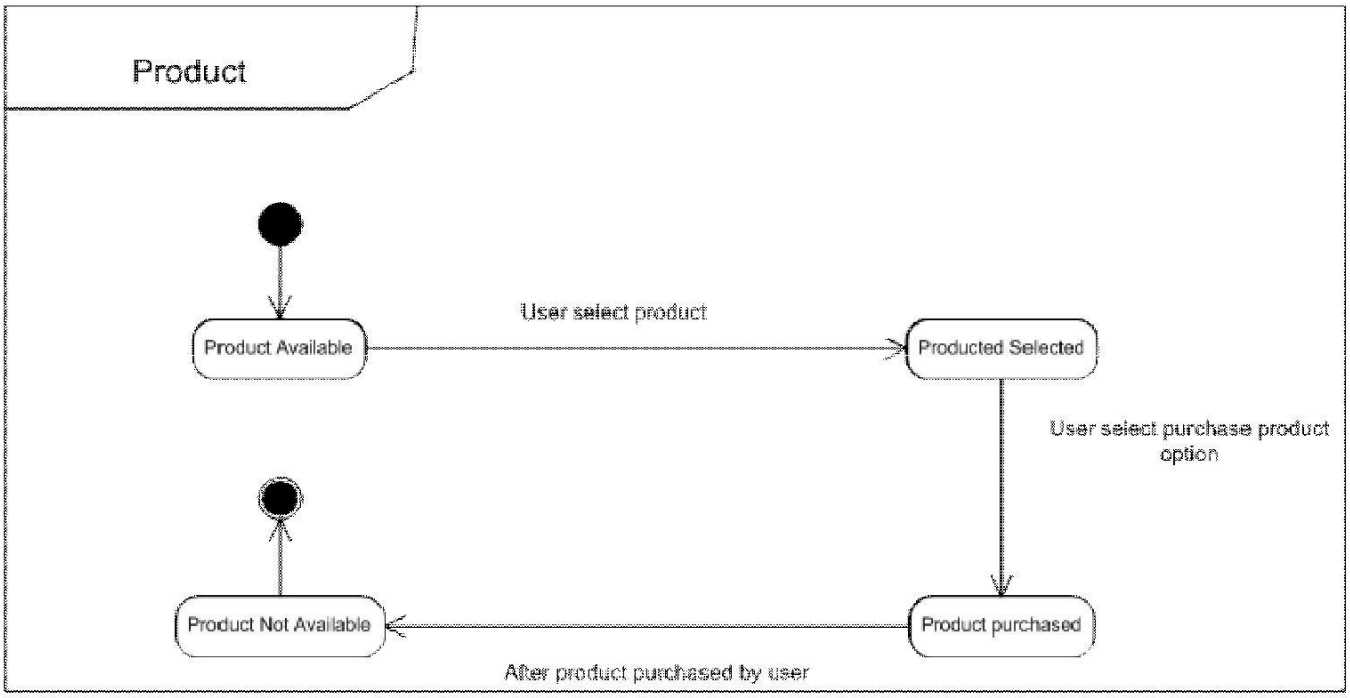
State diagram for online reservation system



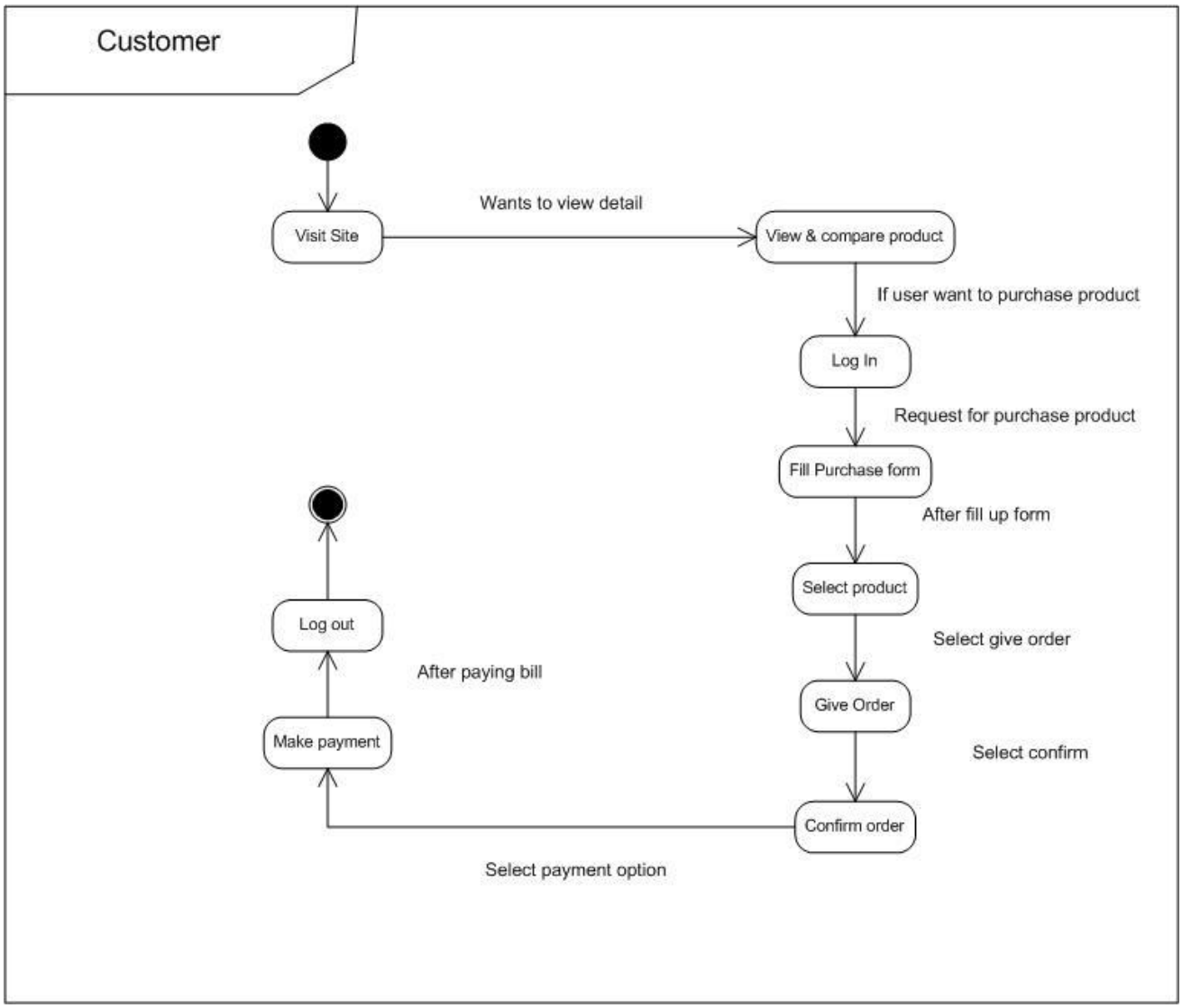
State diagram for online reservation system



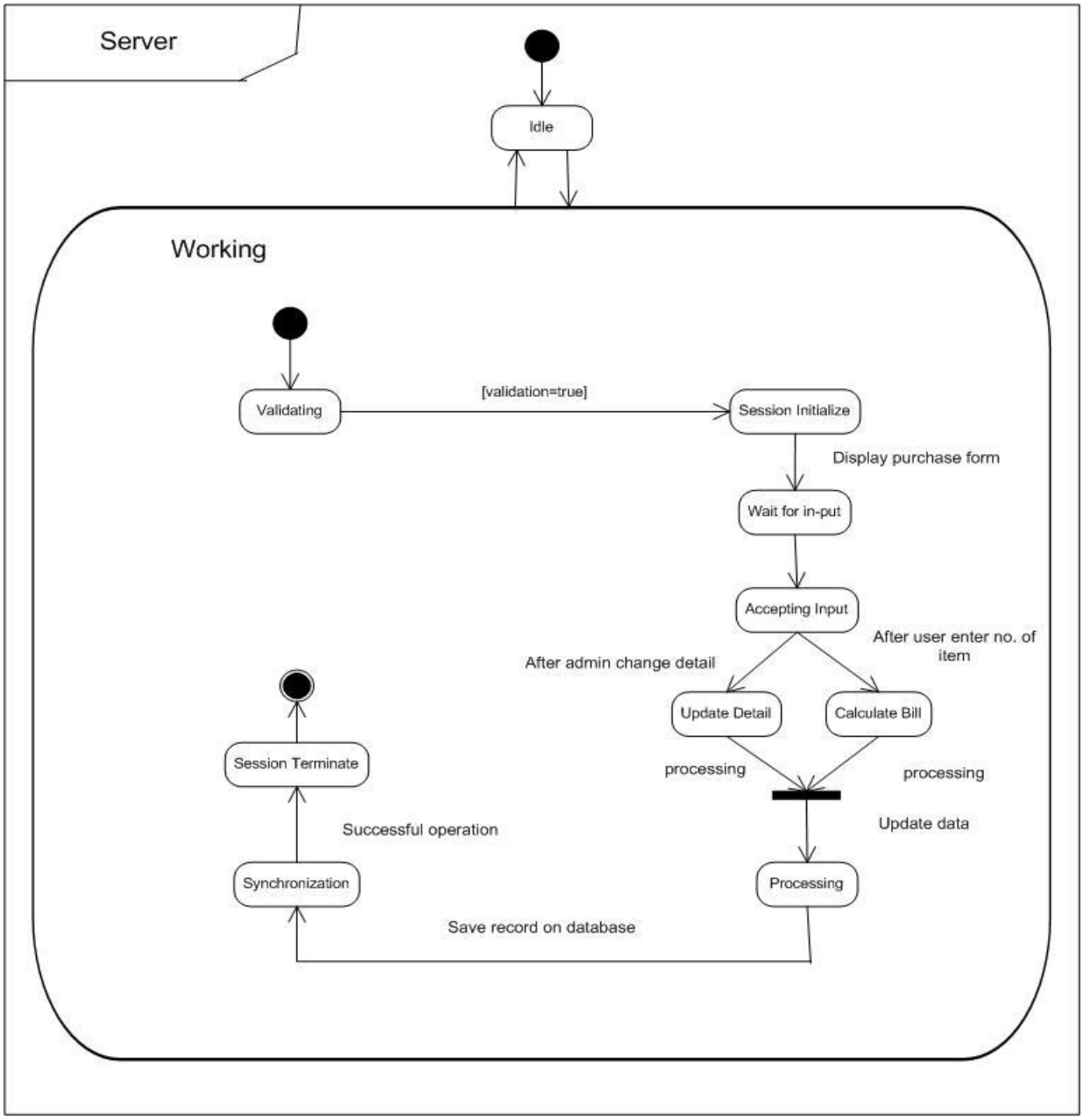
State diagram for online shopping system



State diagram for online shopping system

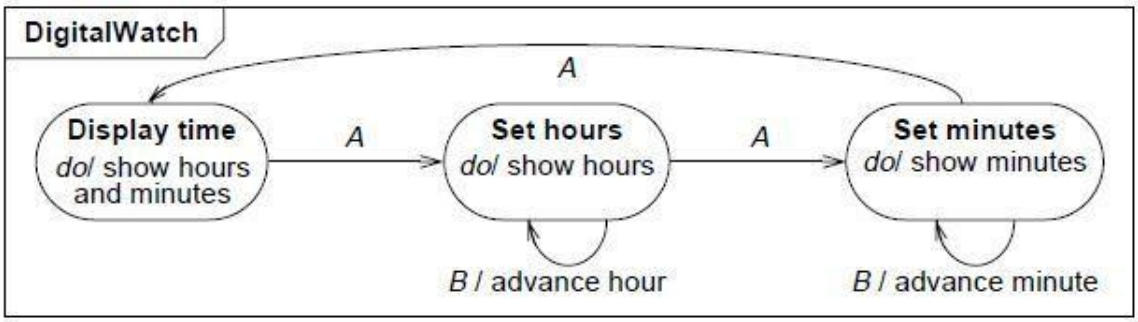


State diagram for online shopping system

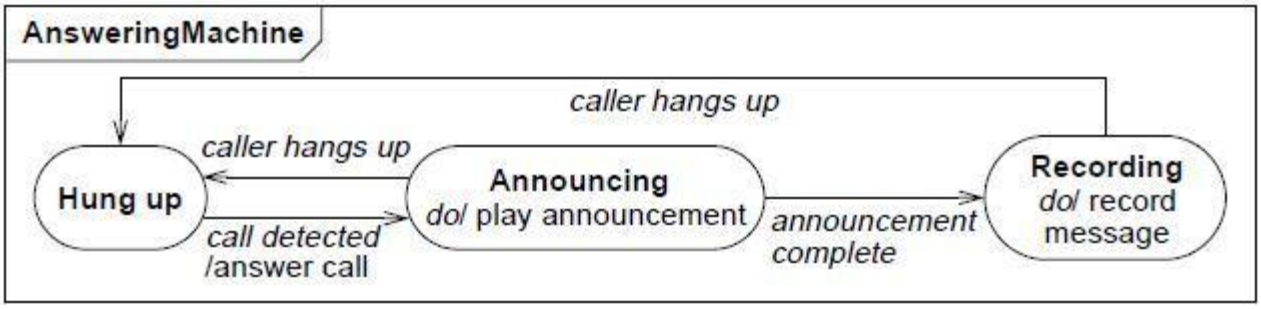


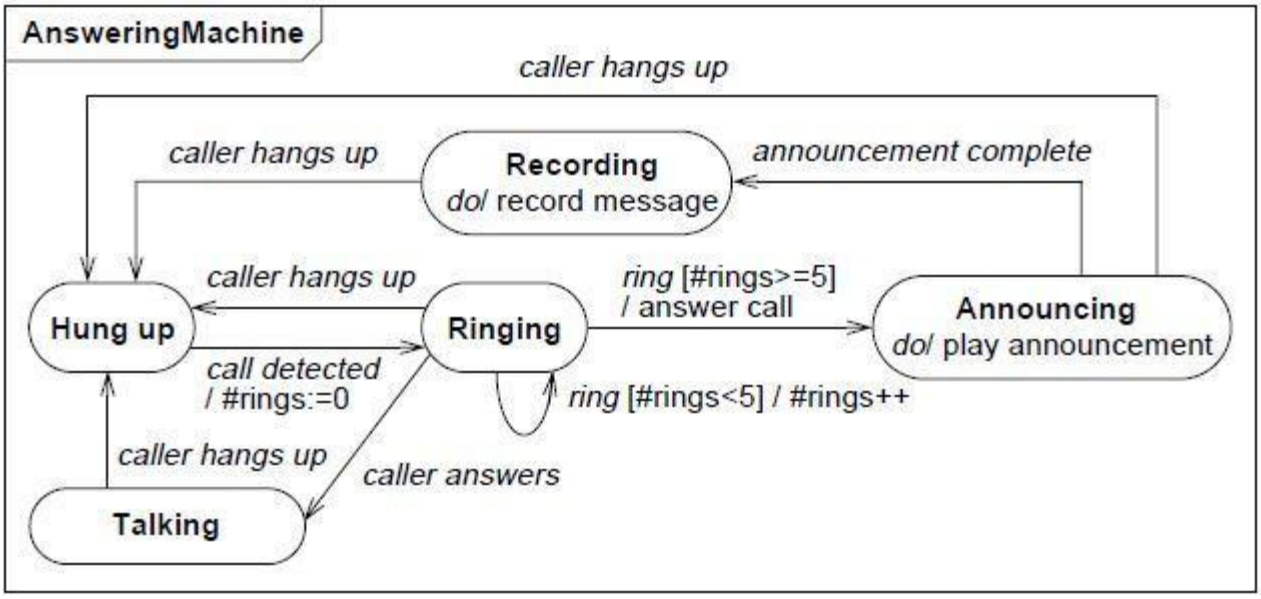
Additional Diagram: State Diagram

1. **A simple digital watch has a display and two buttons to set it, the A button and the B button. The watch has two modes of operation, display time and set time. In the display time mode, the watch displays hours and minutes, separated by a flashing colon. The set time mode has two sub modes, set hours and set minutes. The A button selects modes. Each time it is pressed, the mode advances in the sequence: display, set hour, set minutes, display, etc. Within the sub modes, the B button advances the hours or minutes once each time it is pressed. Buttons must be released before they can generate another event. Prepare a State diagram of the watch.**



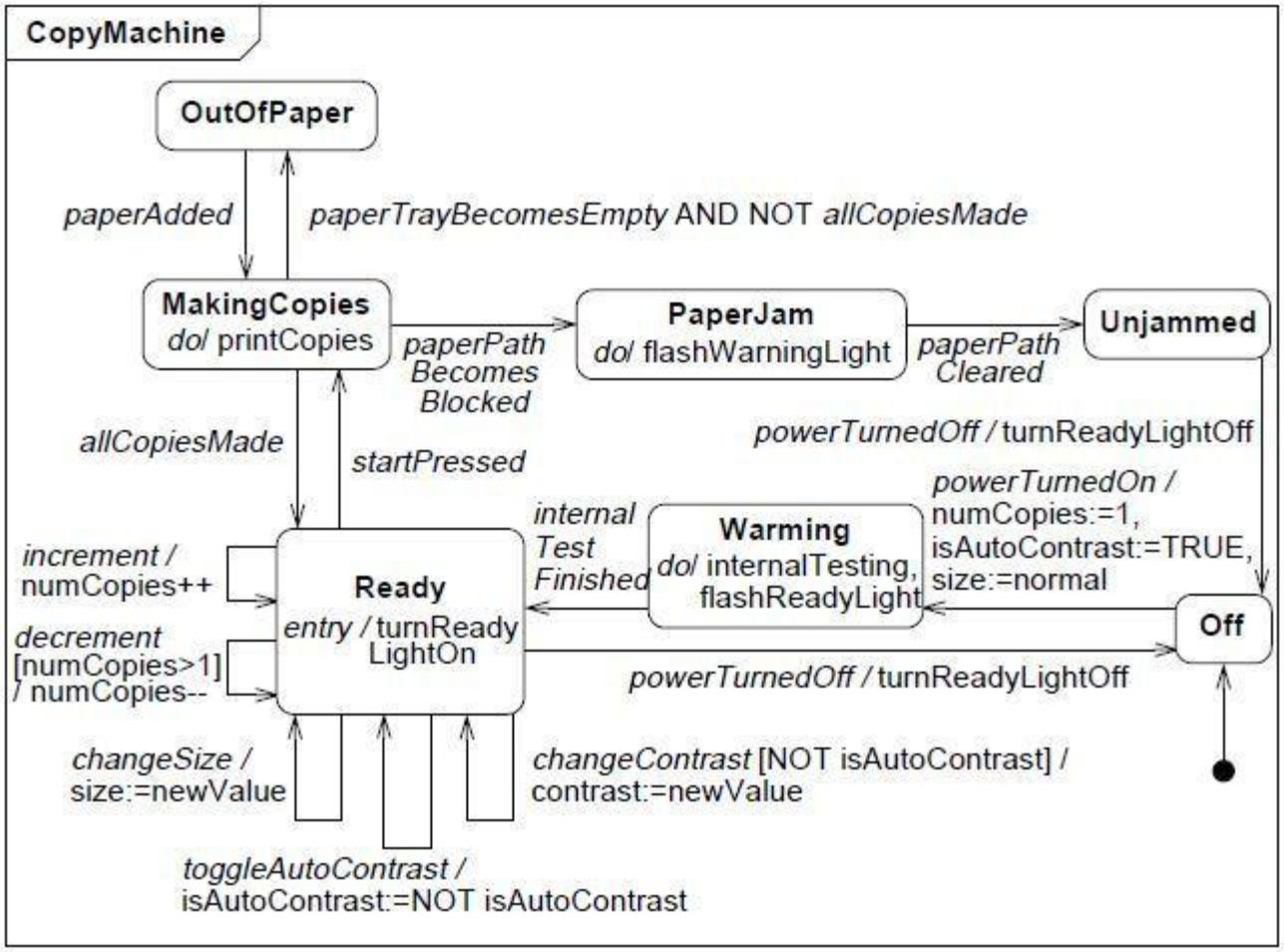
2. Draw state diagram for the control of a telephone answering machine. The machine detects an incoming call on the first ring and answers the call with a prerecorded announcement. When the announcement is complete, the machine records the caller's message. When the caller hands up, the machine hangs up and shuts off. Place the following in the diagram: call detected, answer call, play announcement, record message, caller hangs up, announcement complete.





4. Differentiate state and event. List different types of events. Identify states and events for a Photocopier (Zerox) machine from the description given below and draw the state diagram for the same. Initially the machine is off. When the operator switches on the machine, it first warms up during which it performs some internal tests.

Once the tests are over, machine is ready for making copies. When operator loads a page to be photocopied and press 'start' button, machine starts making copies according to the number of copies selected. While machine is making copies, machine may go out of paper. Once operator loads sufficient pages, it can start making copies again. During the photocopy process, if paper jam occurs in the machine, operator may need to clean the path by removing the jammed paper to make the machine ready.



❖ **Purpose:** Implementation of a state model from the given description.
Lab pre work: Prepare a state model from the given problem description and draw a state diagram using UML2 notations
Laboratory work: Implement the state model with a suitable object oriented language

❖ **Objective:**
- Interpret state chart diagrams.
- Develop state chart diagrams to illustrate the internal workings of individual classes.
- Implement the developed state chart diagram using java programming
-

❖ **Relevant Theory**

State Machine Diagram:

A state machine diagram models the behavior of a single object, specifying the sequence of events that an object goes through during its lifetime in response to events.

The name of the diagram itself clarifies the purpose of the diagram and other details. It describes different states of a component in a system. The states are specific to a component/object of a system.

A State chart diagram describes a state machine. Now to clarify it state machine can be defined as a machine which defines different states of an object and these states are controlled by external or internal events.

As State chart diagram defines states it is used to model lifetime of an object.

Purpose:

State chart diagram is one of the five UML diagrams used to model dynamic nature of a system. They define different states of an object during its lifetime. And these states are changed by events.

So State chart diagrams are useful to model reactive systems. Reactive systems can be defined as a system that responds to external or internal events.

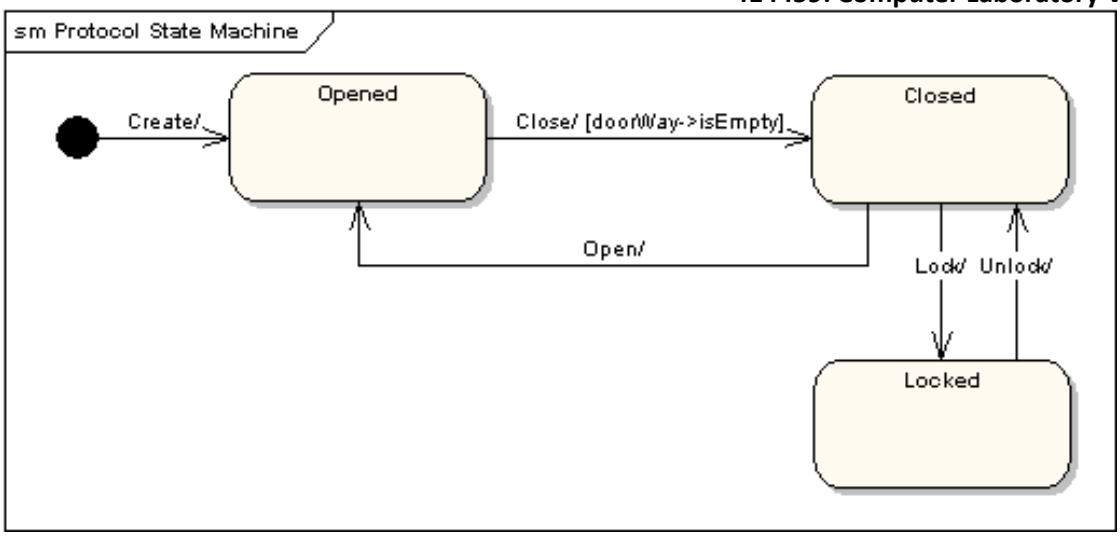
State chart diagram describes the flow of control from one state to another state. States are defined as a condition in which an object exists and it changes when some event is triggered. So the most important purpose of State chart diagram is to model life time of an object from creation to termination.

State chart diagrams are also used for forward and reverse engineering of a system.

But the main purpose is to model reactive system.

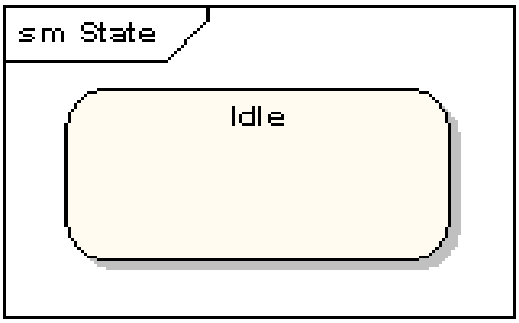
Following are the main purposes of using State chart diagrams:
- To model dynamic aspect of a system.
- To model life time of a reactive system.
- To describe different states of an object during its life time.
- Define a state machine to model states of an object.

As an example, the following state machine diagram shows the states that a door goes through during its lifetime.

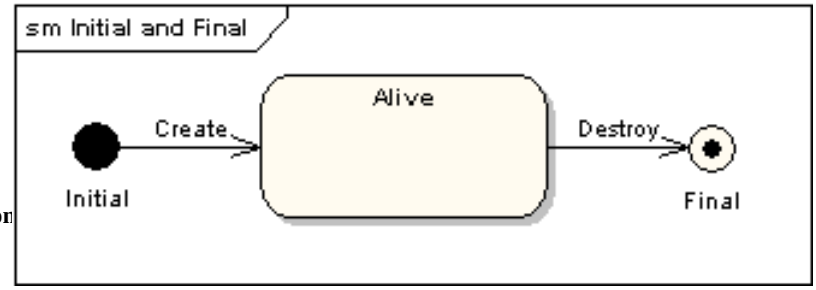


The door can be in one of three states: "Opened", "Closed" or "Locked". It can respond to the events Open, Close, Lock and Unlock. Notice that not all events are valid in all states;
For example, if a door is opened, you cannot lock it until you close it. Also notice that a state transition can have a guard condition attached: if the door is Opened, it can only respond to the Close event if the condition doorWay->isEmpty is fulfilled.
The syntax and conventions used in state machine diagrams will be discussed in full in the following sections.

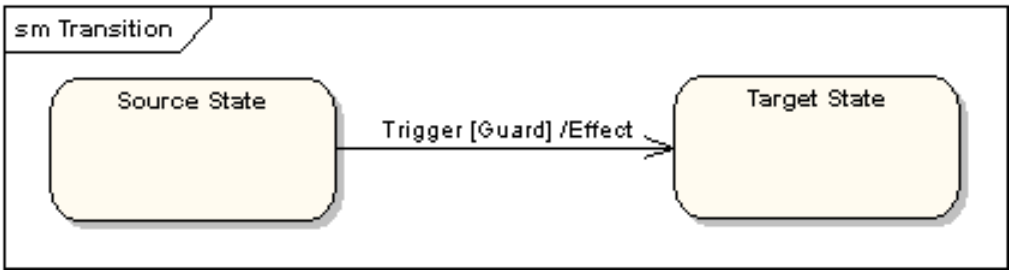
States
A state is denoted by a round-cornered rectangle with the name of the state written inside it.



Initial and Final States
The initial state is denoted by a filled black circle and may be labeled with a name. The final state is denoted by a circle with a dot inside and may also be labeled with a name.

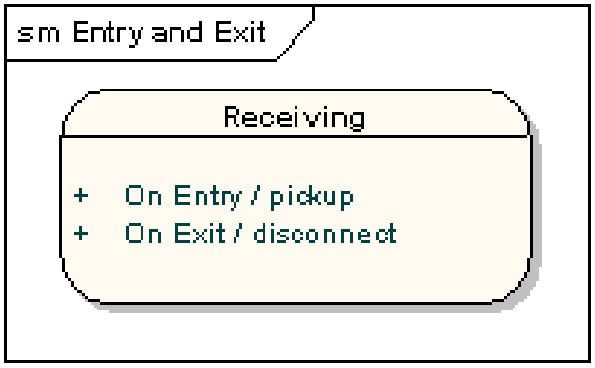


Transitions
Transitions from one state to the next are denoted by lines with arrowheads. A transition may have a trigger, a guard and an effect, as below.



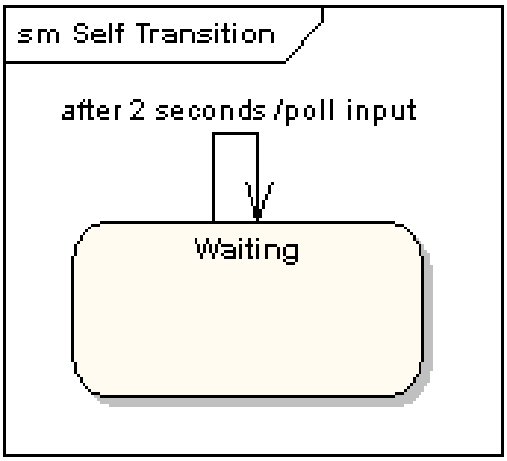
"**Trigger**" is the cause of the transition, which could be a signal, an event, a change in some condition, or the passage of time.
"**Guard**" is a condition which must be true in order for the trigger to cause the transition. "Effect" is an action which will be invoked directly on the object that owns the state machine as a result of the transition.

State Actions
In the transition example above, an effect was associated with the transition. If the target state had many transitions arriving at it, and each transition had the same effect associated with it, it would be better to associate the effect with the target state rather than the transitions.
This can be done by defining an entry action for the state. The diagram below shows a state with an entry action and an exit action.



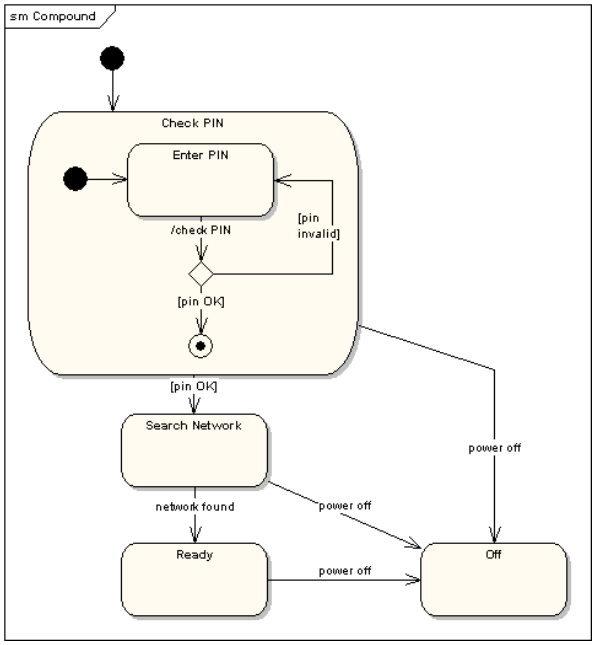
It is also possible to define actions that occur on events, or actions that always occur. It is possible to define any number of actions of each type.

Self-Transitions
A state can have a transition that returns to itself, as in the following diagram. This is most useful when an effect is associated with the transition.

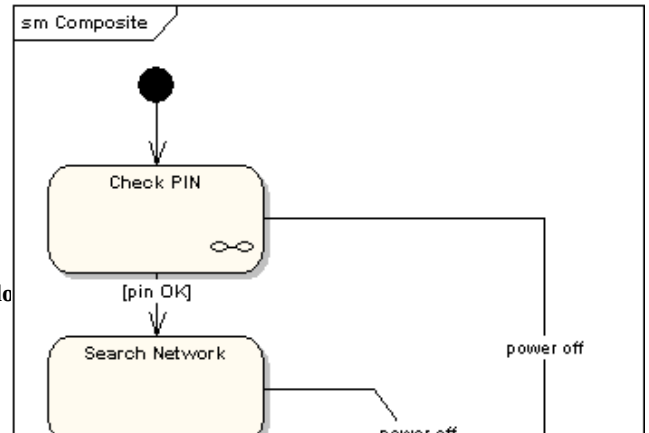


Compound States

A state machine diagram may include sub-machine diagrams, as in the example below.



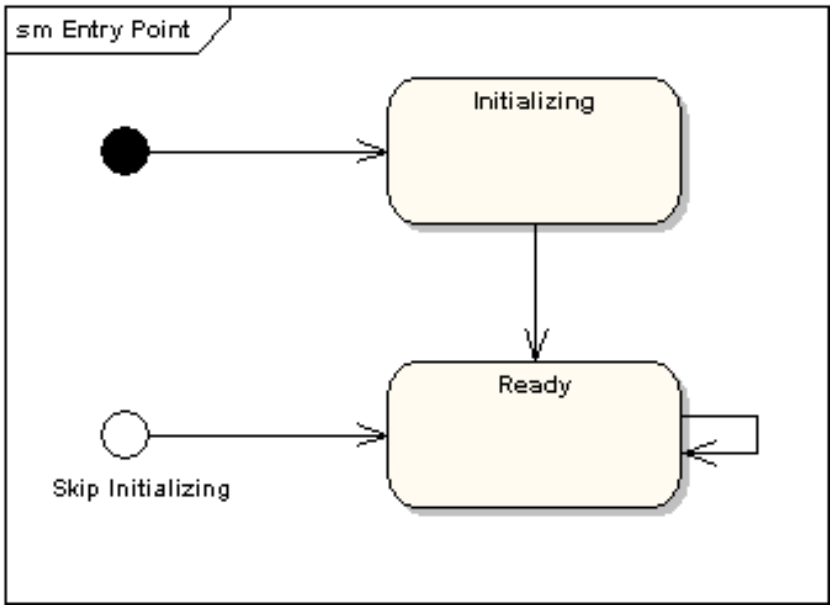
The alternative way to show the same information is as follows.



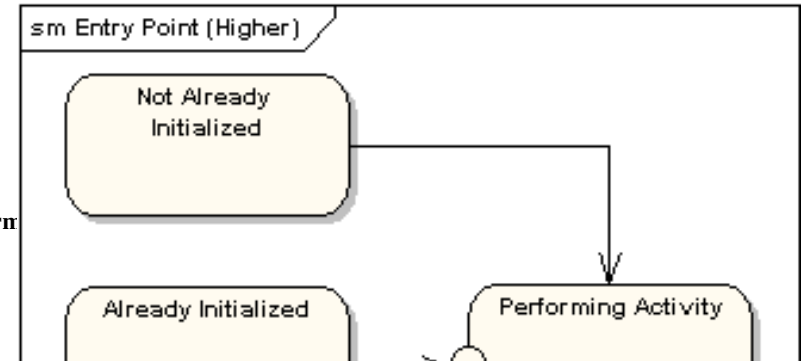
The notation in the above version indicates that the details of the Check PIN sub-machine are shown in a separate diagram.

Entry Point

Sometimes you won't want to enter a sub-machine at the normal initial state. For example, in the following sub-machine it would be normal to begin in the "Initializing" state, but if for some reason it wasn't necessary to perform the initialization, it would be possible to begin in the "Ready" state by transitioning to the named entry point.

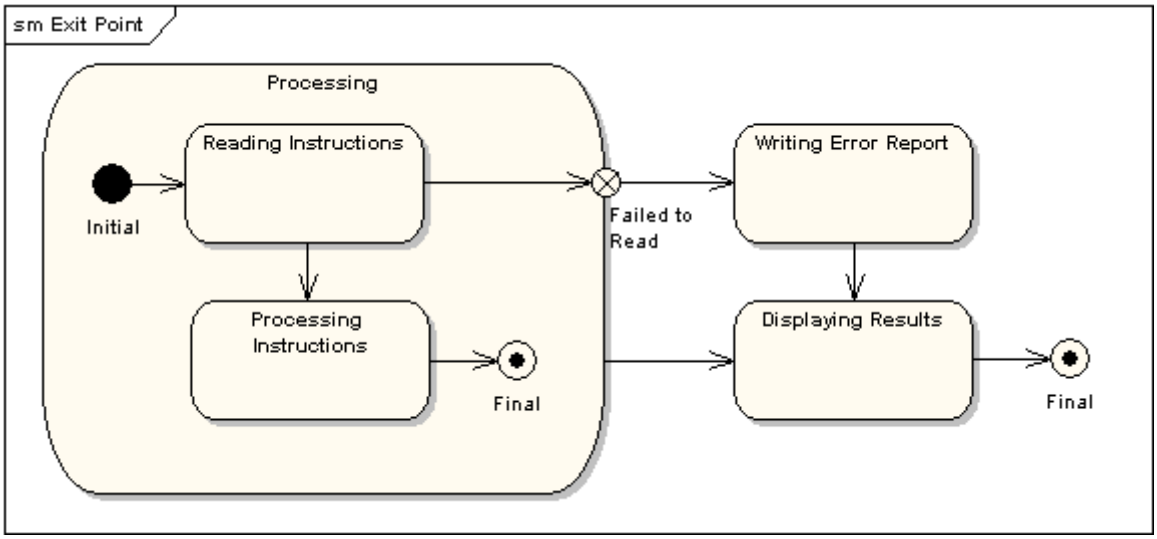


The following diagram shows the state machine one level up.



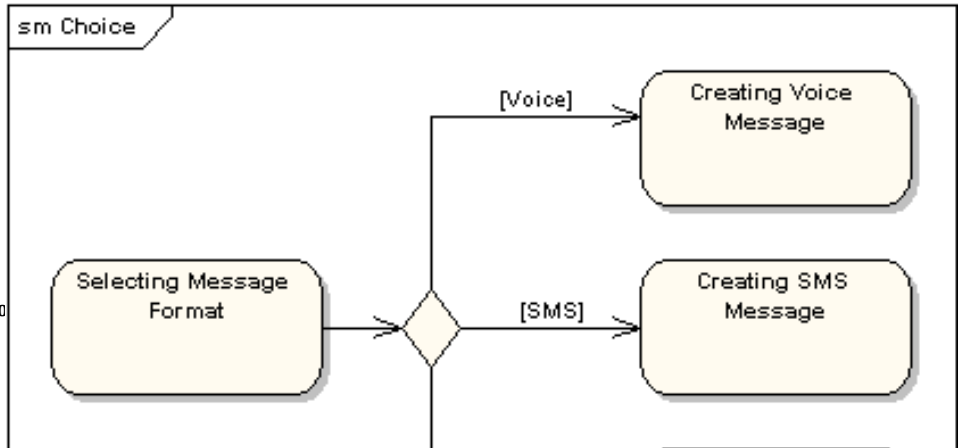
Exit Point

In a similar manner to entry points, it is possible to have named alternative exit points. The following diagram gives an example where the state executed after the main processing state depends on which route is used to transition out of the state.



Choice Pseudo-State

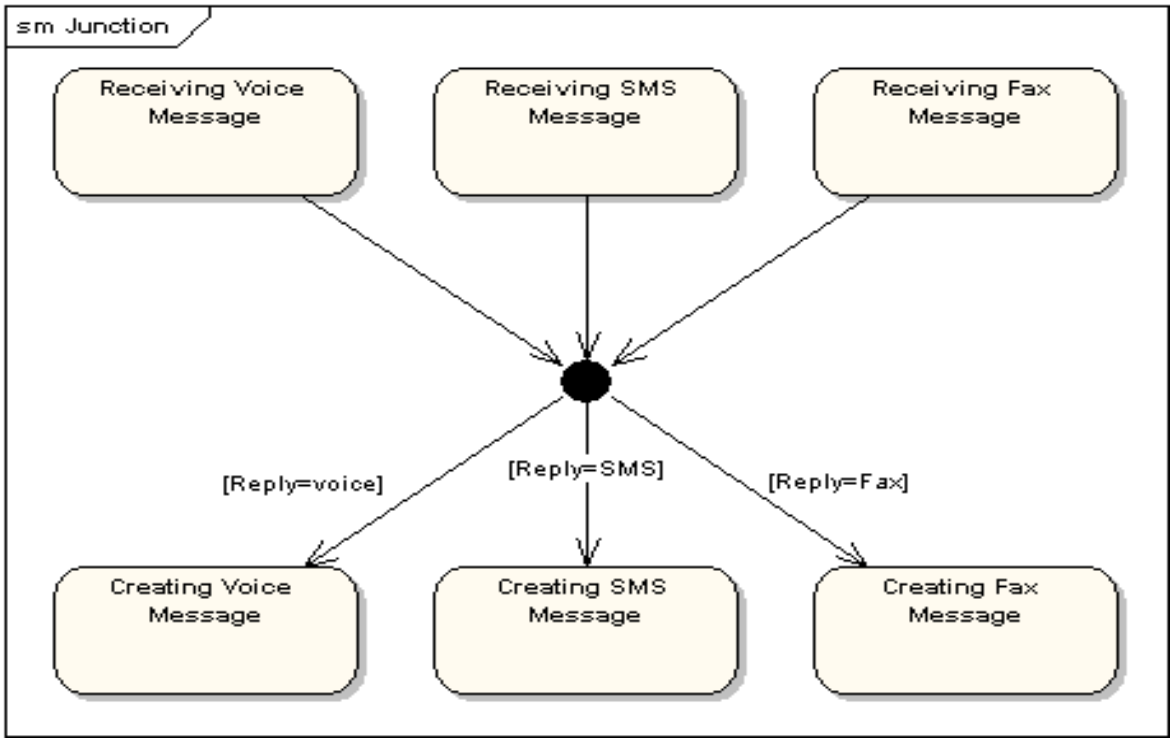
A choice pseudo-state is shown as a diamond with one transition arriving and two or more transitions leaving. The following diagram shows that whichever state is arrived at, after the choice pseudo-state, is dependent on the message format selected during execution of the previous state.



Junction Pseudo-State

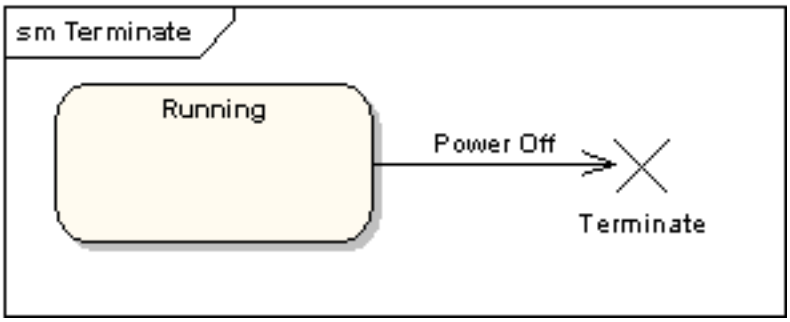
Junction pseudo-states are used to chain together multiple transitions. A single junction can have one or more incoming, and one or more outgoing, transitions; a guard can be applied to each transition. Junctions are semantic-free.

A junction which splits an incoming transition into multiple outgoing transitions realizes a static conditional branch, as opposed to a choice pseudo-state which realizes a dynamic conditional branch.



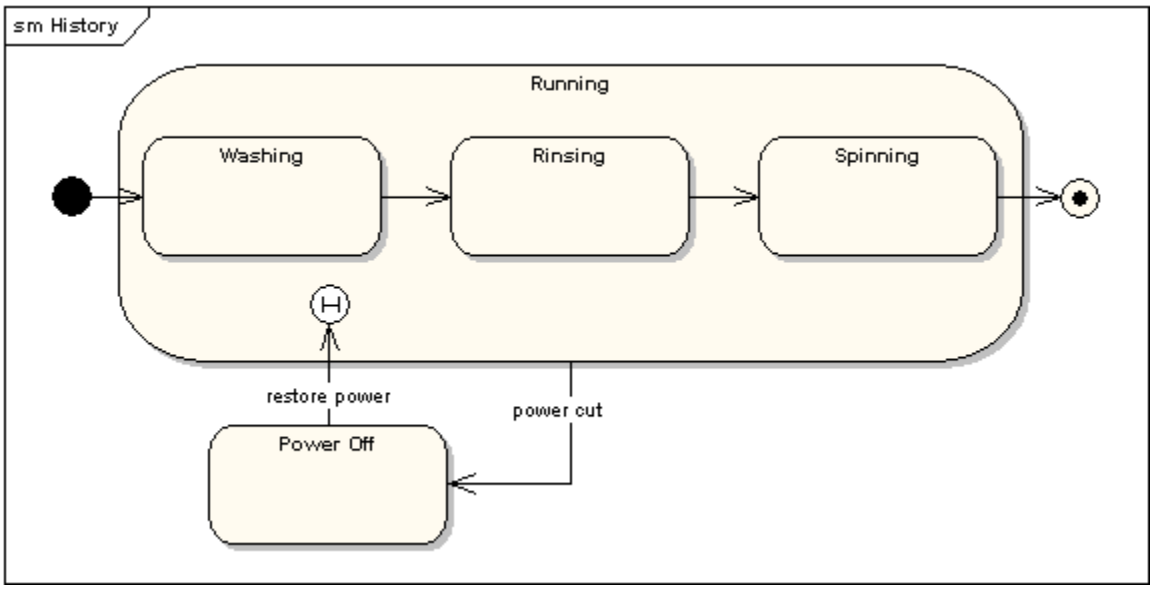
Terminate Pseudo-State

Entering a terminate pseudo-state indicates that the lifeline of the state machine has ended. A terminate pseudo-state is notated as a cross.



History States

A history state is used to remember the previous state of a state machine when it was interrupted. The following diagram illustrates the use of history states. The example is a state machine belonging to a washing machine.

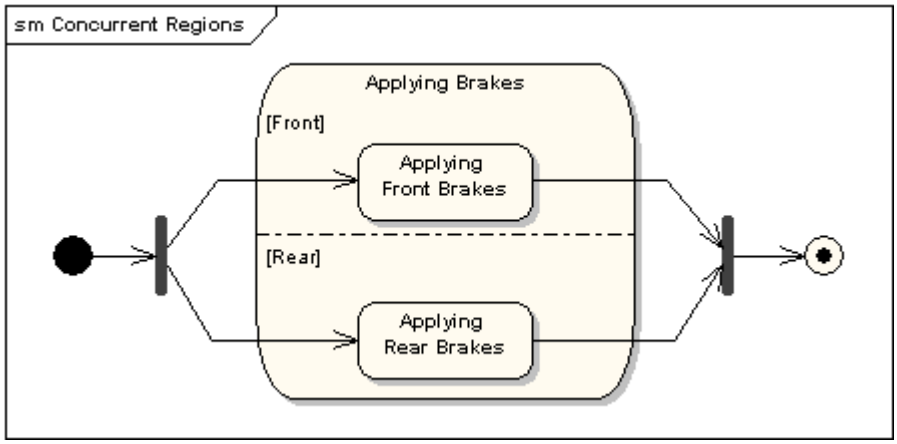


In this state machine, when a washing machine is running, it will progress from "Washing" through "Rinsing" to "Spinning".

If there is a power cut, the washing machine will stop running and will go to the "Power Off" state. Then when the power is restored, the Running state is entered at the "History State" symbol meaning that it should resume where it last left-off.

Concurrent Regions

A state may be divided into regions containing sub-states that exist and execute concurrently. The example below shows that within the state "Applying Brakes", the front and rear brakes will be operating simultaneously and independently. Notice the use of fork and join pseudo-states, rather than choice and merge pseudo-states. These symbols are used to synchronize the concurrent threads.



How to draw State chart Diagram?

State chart diagram is used to describe the states of different objects in its life cycle. So the emphasis is given on the state changes upon some internal or external events. These states of objects are important to analyze and implement them accurately.

State chart diagrams are very important for describing the states. States can be identified as the condition of objects when a particular event occurs.

Before drawing a State chart diagram we must have clarified the following points:

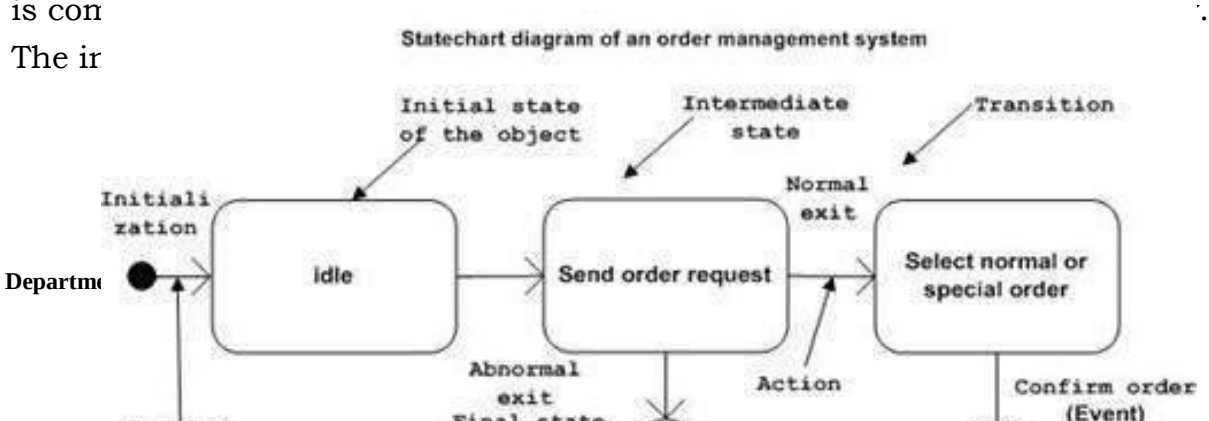
- Identify important objects to be analyzed.
- Identify the states.
- Identify the events.

The following is an **example** of a State chart diagram where the state of Order object is analyzed.

The first state is an idle state from where the process starts. The next states are arrived for events like send request, confirm request, and dispatch order. These events are responsible for state changes of order object.

During the life cycle of an object (here order object) it goes through the following states and there may be some abnormal exists also. This abnormal exit may occur due to some problem in the system. When the entire life cycle is con .

The ir



Where to use State chart Diagrams?

From the above discussion we can define the practical applications of a State chart diagram. State chart diagrams are used to model dynamic aspect of a system like other four diagrams discussed in this tutorial. But it has some distinguishing characteristics for modeling dynamic nature.

State chart diagram defines the states of a component and these state changes are dynamic in nature.

So its specific purpose is to define state changes triggered by events. Events are internal or external factors influencing the system.

State chart diagrams are used to model states and also events operating on the system.

When implementing a system it is very important to clarify different states of an object during its life time and state chart diagrams are used for this purpose. When these states and events are identified they are used to model it and these models are used during implementation of the system.

If we look into the practical implementation of State chart diagram then it is mainly used to analyze the object states influenced by events.

This analysis is helpful to understand the system behavior during its execution.

So the main usages can be described as:

- To model object states of a system.
- To model reactive system. Reactive system consists of reactive objects.
- To identify events responsible for state changes.
- Forward and reverse engineering.

❖ **Exercises:**

Problem Description

Draw State chart diagram for saving account object having following functionality such as

- Open account

- Withdraw account
- Deposit accounts
- Close account
- Suspend account

Following conditions should be taken care while developing the state machine

A. **Open account:** minimum balance Rs, 500. If balance is below rs, 500 do not create account object
B. **Withdraw money:** balance should be greater than or equal to Rs. 500
C. **Close Account:** account should be in inactive state and balance should be zero
D. **Deposit money:** minimum deposit should be 100.

❖ **Conclusion:**

Thus the concept of behavioral modeling was studied and developed the state chart machine for class under consideration

Also same is implemented using JAVA programming language

FREQUENTLY ASKED QUESTIONS FOR ORAL EXAMINATION:

1. With context of state diagram define state, do activity, entry and exit?
2. What are different types of events used in state diagram?
3. Explain concept of event in state model? What are different types of events?
4. Explain the concept of nested state with example?
5. Explain concept of concurrent sub-states?
6. What is relation between class model and state model?

Assignment No. 8

Title: Identification and Implementation of GRASP pattern

Apply any two GRASP pattern to refine the Design Model for a given problem description Using effective UML 2 diagrams and implement them with a suitable object oriented language

Date of Completion		**Marks for Regularity (2)**	
		Marks for Presentation (3)	
Remark		**Marks for Understanding (5)**	
Sign. of Staff		**Total Marks (10)**	

Prof. R.S. Badodekar **414459 : Computer Laboratory VIII**

GRASP Patterns

What are GRASP Patterns?

They describe fundamental principles of object design and responsibility assignment, expressed as patterns.

After identifying your requirements and creating a domain model, then add methods to the software classes, and define the messaging between the objects to fulfill the requirements. The GRASP patterns are a learning aid to help one understand essential object design, and apply design reasoning in a methodical, rational, explainable way. This approach to understanding and using design principles are based on patterns of assigning responsibilities.

Responsibilities and Methods

The UML defines a responsibility as "a contract or obligation of a classifier". Responsibilities are related to the obligations of an object in terms of its behavior. Basically, these responsibilities are of the following two types:

- knowing
- doing

Doing responsibilities of an object include:

- doing something itself, such as creating an object or doing a calculation
- initiating action in other objects
- controlling and coordinating activities in other objects

Knowing responsibilities of an object include:

- knowing about private encapsulated data
- knowing about related objects
- knowing about things it can derive or calculate

Responsibilities are assigned to classes of objects during object design. For example, I may declare that "a Sale is responsible for creating SalesLineItems" (a doing), or "a Sale is responsible for knowing its total" (a knowing). Relevant responsibilities related to "knowing" are often inferable from the domain model because of the attributes and associations it illustrates.

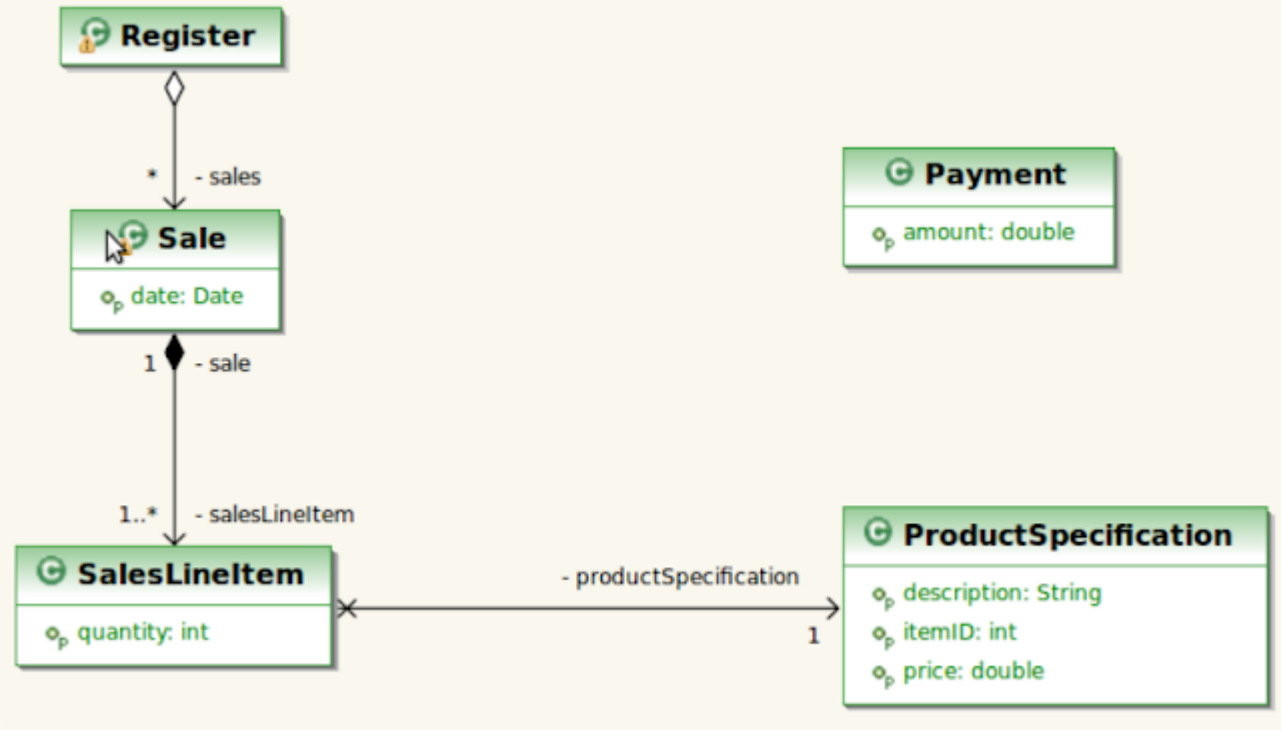
POS (Point Of Sale) application is used to explain all the **GRASP Patterns**.

Let's understand the POS(Point Of Sale) application briefly and then apply **GRASP Patterns** to POS Application.

Point of Sale (POS) application

Let's consider Point of Sale (POS) application: a Brief overview of POS application.

1. Application for a shop, restaurant, etc. that registers sales.
2. Each sale is of one or more items of one or more product types and happens at a certain date.
3. A product has a specification including a description, unitary price, and identifier.
4. The application also registers payments (say, in cash) associated with sales.
5. A payment is for a certain amount, equal or greater than the total of the sale.



How to Apply the GRASP Patterns (General Responsibility Assignment Software Patterns)
Let's discuss five GRASP patterns. There is a separate Post for each **GRASP Pattern**

- Information Expert
- Low Coupling
- High Cohesion
- Controller
- Creator

Information Expert GRASP Pattern

Problem

What is a general principle of assigning responsibilities to objects?
A Design Model may define hundreds or thousands of software classes, and an application may require hundreds or thousands of responsibilities to be fulfilled. During object design, when the interactions between objects are defined, we make choices about the assignment of responsibilities to software classes. Done well, systems tend to be easier to understand, maintain, and extend, and there is more opportunity to reuse components in future applications.

Solution

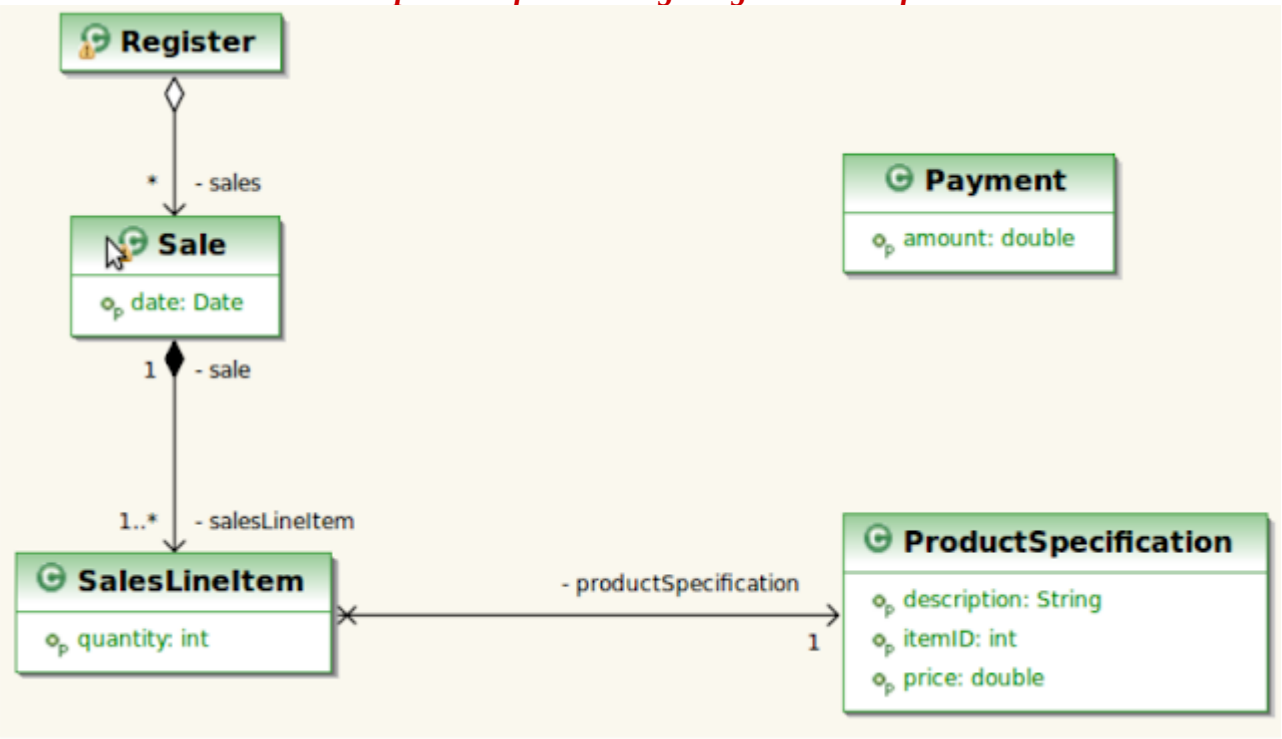
Assign a responsibility to the information expert - the class that has the information necessary to fulfill the responsibility.

Example

Let's consider Point of Sale (POS) application: a Brief overview of POS application.

- Application for a shop, restaurant, etc. that registers sales.
- Each sale is of one or more items of one or more product types and happens at a certain date.
- A product has a specification including a description, unitary price, and identifier.
- The application also registers payments (say, in cash) associated with sales.
- A payment is for a certain amount, equal or greater than the total of the sale.

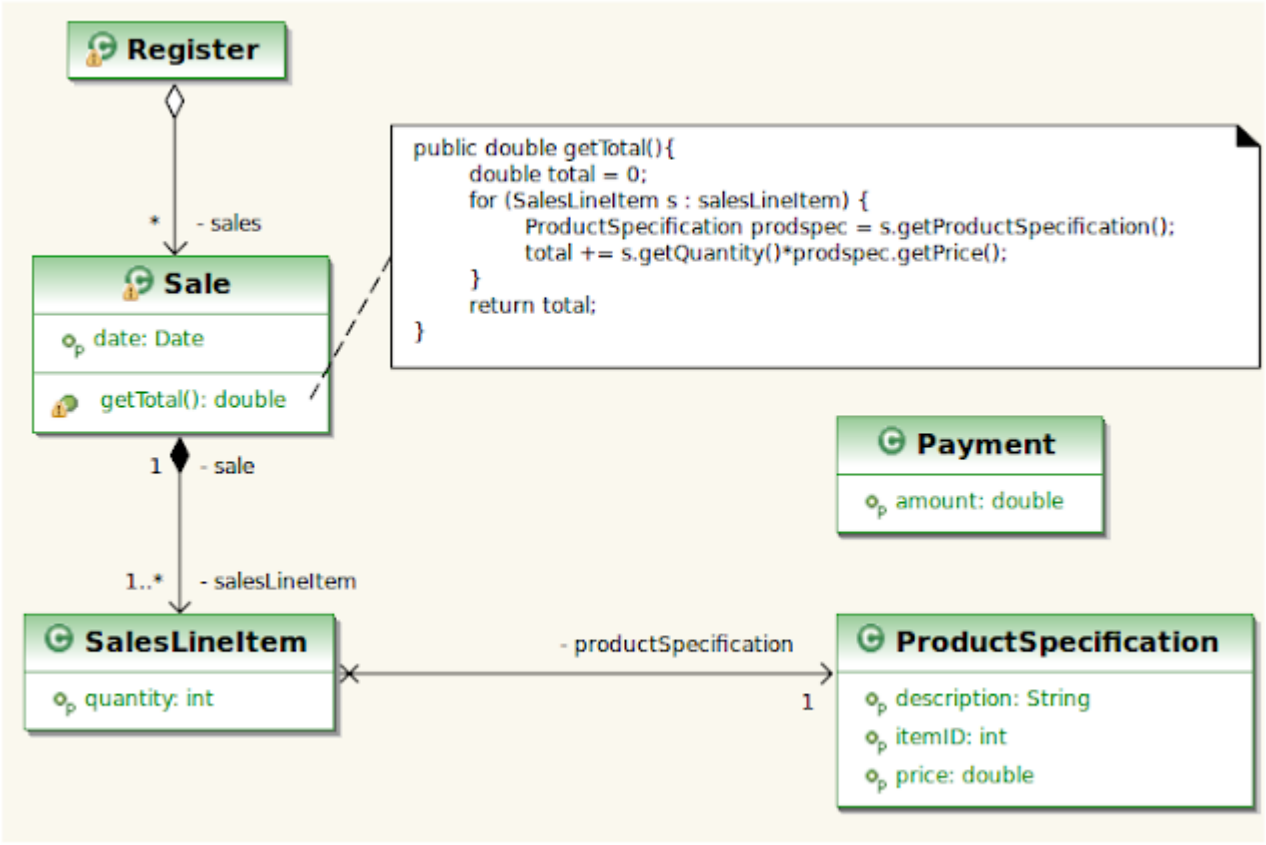
Problem statement: Who's responsible for knowing the grand total of a Sale?



In the POS application, some class needs to know the grand total of a sale.

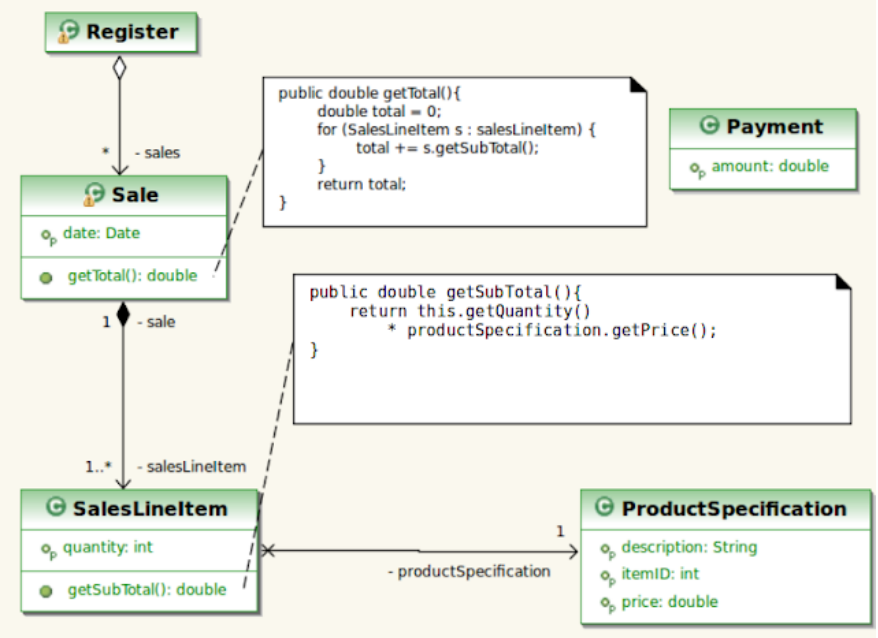
Start assigning responsibilities by clearly stating the responsibility.
By *Information Expert*, we should look for that class of objects that has the information needed to determine the total.
It is necessary to know about all the **SalesLineItem** instances of a sale and the sum of their subtotals.

A **Sale** instance contains these; therefore, by the guideline of Information Expert, **Sale** is a suitable class of object for this responsibility; it is an information expert for the work.
We are not done yet. What information is needed to determine the line item subtotal? **SalesLineItem.quantity** and **ProductSpecification.price** are needed.
The **SalesLineItem** knows its quantity and its associated ProductSpecification; therefore, by Expert, **SalesLineItem** should determine the subtotal; it is the information expert.
In terms of an interaction diagram, this means that the **Sale** needs to send get-Subtotal messages to each of the **SalesLineItems** and sum the results; this design is shown in Figure :



To fulfill the responsibility of knowing and answering its subtotal, a Sales- LineItem needs to know the product price. The **ProductSpecification** is an information expert on answering its **price**; therefore, a message must be sent to it asking for its price.

Complete Class Diagram



In conclusion, to fulfill the responsibility of knowing and answering the sale's total, three responsibilities were assigned to three design classes of objects as follows

Design Class	Responsibility
Sale	knows sale total
SalesLineItem	knows line item subtotal
ProductSpecification	knows product price

Sample Code

Let's write sample method in **Sale** Domain model class.

```
public class Sale {
 //...
 public double getTotal(){
  double total = 0;
  for (SalesLineItem s : salesLineItem) {
   ProductSpecification prodspec
   = s.getProductSpecification();
   total += s.getQuantity()*prodspec.getPrice();
  }
  return total;
 }
}
```

This is above code is enough? It is not enough right because here we need **getSubTotal** from **salesLineItem**.
Let's create the method in **SaleLineItem** domain model class.

```
public class SalesLineItem {
  //...
  public double getSubTotal(){
```

```
 return this.getQuantity()
  productSpecification.getPrice();
 }
}
```

Observe **ProductSpecification** Domain model provides getPrice method gives a price.
Fulfillment of a responsibility may require information spread across different classes, each expert on its own data.

Benefits

- Information encapsulation is maintained since objects use their own information to fulfill tasks. This usually supports low coupling, which leads to more robust and maintainable systems. (Low Coupling is also a GRASP pattern that is discussed in the following section).
- Behavior is distributed across the classes that have the required information, thus encouraging more cohesive "lightweight" class definitions that are easier to understand and maintain. High cohesion is usually supported (another pattern discussed later).

Low Coupling GRASP Pattern

Problem

How to support low dependency, low change impact, and increase reuse?

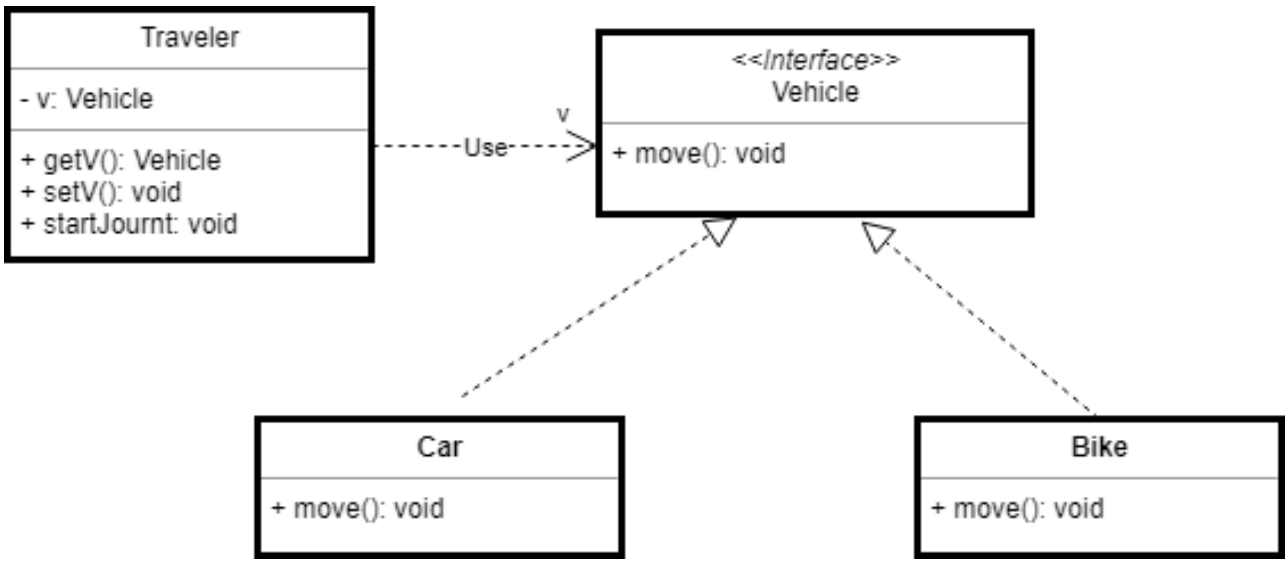
Solution

Assign responsibilities so that coupling remains low. Try to avoid one class to have to know about many others.

Key points about low coupling

- Low dependencies between “artifacts” (classes, modules, components).
- There shouldn’t be too much of dependency between the modules, even if there is a dependency it should be via the interfaces and should be minimal.
- Avoid tight-coupling for collaboration between two classes (if one class wants to call the logic of a second class, then first class needs an object of second class it means the first class creates an object of the second class).
- Strive for loosely coupled design between objects that interact.
- Inversion Of Control (IoC) / Dependency Injection (DI) - With DI objects are given their dependencies at creation time by some third party (i.e. Java EE CDI, Spring DI…) that coordinates each object in the system. Objects aren’t expected to create or obtain their dependencies—dependencies are injected into the objects that need them. The key benefit of DI—loose coupling.

Implementation



This is an example of loose coupling. In this class, **Traveler** class is not tightly coupled with **Car** or **Bike** implementation. Instead by applying dependency injection mechanism, the loose coupling implementation is achieved to allow start journey with any class which has implemented **Vehicle** interface.

Step 1: **Vehicle** interface to allow loose coupling implementation.

```
interface Vehicle {
 public void move();
}
```

Step 2: **Car** class implements Vehicle interface.

```
class Car implements Vehicle {
 @Override
 public void move() {
  System.out.println("Car is moving");
 }
}
```

Step 3: **Bike** class implements Vehicle interface.

```
class Bike implements Vehicle {
 @Override
 public void move() {
  System.out.println("Bike is moving");
 }
}
```

Step 4: Now create **Traveller** class which holds the reference to Vehicle interface.

```
class Traveler {
 private Vehicle v;
 public Vehicle getV() {
  return v;
 }
```

```
public void setV(Vehicle v) {
    this.v = v;
}

public void startJourney() {
 v.move();
 }
}
```

Step 5: Test class for loose coupling example - Traveler is an example of loose coupling.

```
public static void main(String[] args) {
 Traveler traveler = new Traveler();
 traveler.setV(new Car()); // Inject Car dependency
 traveler.startJourney(); // start journey by Car
 traveler.setV(new Bike()); // Inject Bike dependency
 traveler.startJourney(); // Start journey by Bike
}
```

High Cohesion GRASP Pattern

Problem

How to keep classes focused, understandable and manageable?

Solution

Assign responsibilities so that cohesion remains high. Try to avoid classes to do too much or too different things.

The term cohesion is used to indicate the degree to which a class has a single, well-focused responsibility. Cohesion is a measure of how the methods of a class or a module are meaningfully and strongly related and how focused they are in providing a well-defined purpose to the system.

A class is identified as a low cohesive class when it contains many unrelated functions within it. And that what we need to avoid because big classes with unrelated functions hamper their maintaining. Always make your class small and with precise purpose and highly related functions.

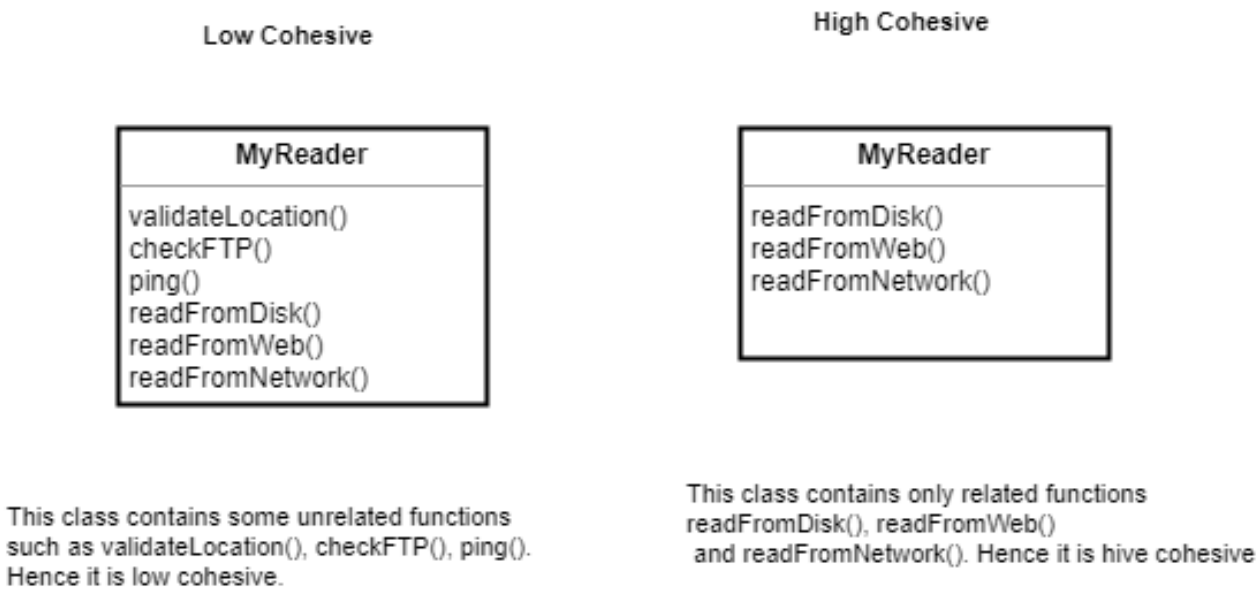
Key points about high cohesion

- The code has to be very specific in its operations.
- The responsibilities/methods are highly related to class/module.

- The term cohesion is used to indicate the degree to which a class has a single, well-focused responsibility. Cohesion is a measure of how the methods of a class or a module are meaningfully and strongly related and how focused they are in providing a well-defined purpose to the system. The more focused a class is the higher its cohesiveness - a good thing.
- A class is identified as a low cohesive class when it contains many unrelated functions within it. And that what we need to avoid because big classes with unrelated functions hamper their maintaining. Always make your class small and with precise purpose and highly related functions.

Example

In this example, the purpose of **MyReader** class is to read the resource and it does that only. It does not implement other unrelated things. Hence it is highly cohesive.




```
class HighCohesive {
 // ---------- functions related to read resource
 // read the resource from disk
 public String readFromDisk(String fileName) {
  return "reading data of " + fileName;
 }

 // read the resource from web
 public String readFromWeb(String url) {
  return "reading data of " + url;
 }

 // read the resource from network
 public String readFromNetwork(String networkAddress) {
  return "reading data of " + networkAddress;
 }
}
```

Controller GRASP Pattern

Problem

Who should be responsible for handling an input system event?

An input system event is an event generated by an external actor. They are associated with system operations of the system in response to system events, just as messages and methods are related.

For example, when a writer using a word processor presses the "spell check" button, he is generating a system event indicating "perform a spell check."

A Controller is a non-user interface object responsible for receiving or handling a system event. A Controller defines the method for the system operation.

Who should be responsible for handling an input event, which object/s beyond the UI layer receives interaction?

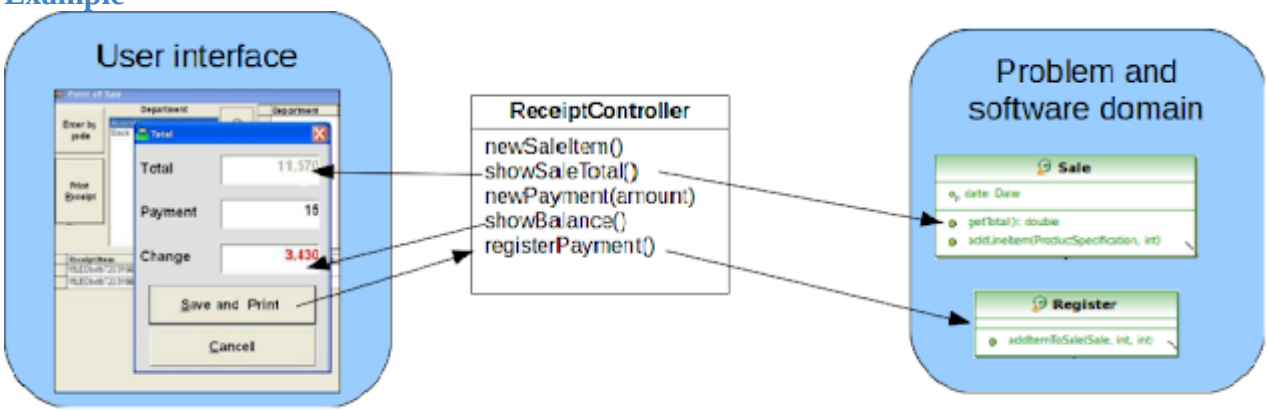
Solution

Assign the responsibility for receiving or handling a system event message to a class representing one of the following choices:

- Represents the overall system, device, or subsystem (facade controller).
- Represents a use case scenario within which the system event occurs, often named <UseCaseName>Handler, <UseCaseName>Coordinator, or <Use-CaseName>Session (use-case or session controller).
- Use the same controller class for all system events in the same use case scenario.
- Informally, a session is an instance of a conversation with an actor. Sessions can be of any length but are often organized in terms of use cases (use case sessions).

Corollary: Note that "window," "applet," "widget," "view," and "document" classes are not on this list. Such classes should not fulfill the tasks associated with system events, they typically receive these events and delegate them to a controller.

Example



Benefits

- Either the UI classes or the problem/software domain classes can change without affecting the other side.
- The controller is a simple class that mediates between the UI and problem domain classes, just forwards
- event handling requests

- output requests

Creator GRASP Pattern

Problem

Who should be responsible for creating a new instance of some class? The creation of objects is one of the most common activities in an object-oriented system. Consequently, it is useful to have a general principle for the assignment of creation responsibilities. Assigned well, the design can support low coupling, increased clarity, encapsulation, and reusability.

Solution

Assign class B the responsibility to create an instance of class A if one or more of the following is true:

• B aggregates A objects.
• B contains A objects.
• B records instances of A objects.
• B closely uses A objects.
• B has the initializing data that will be passed to A when it is created (thus B is an Expert with respect to creating A).

B is a creator of A objects.

If more than one option applies, prefer a class B which aggregates or contains class A.

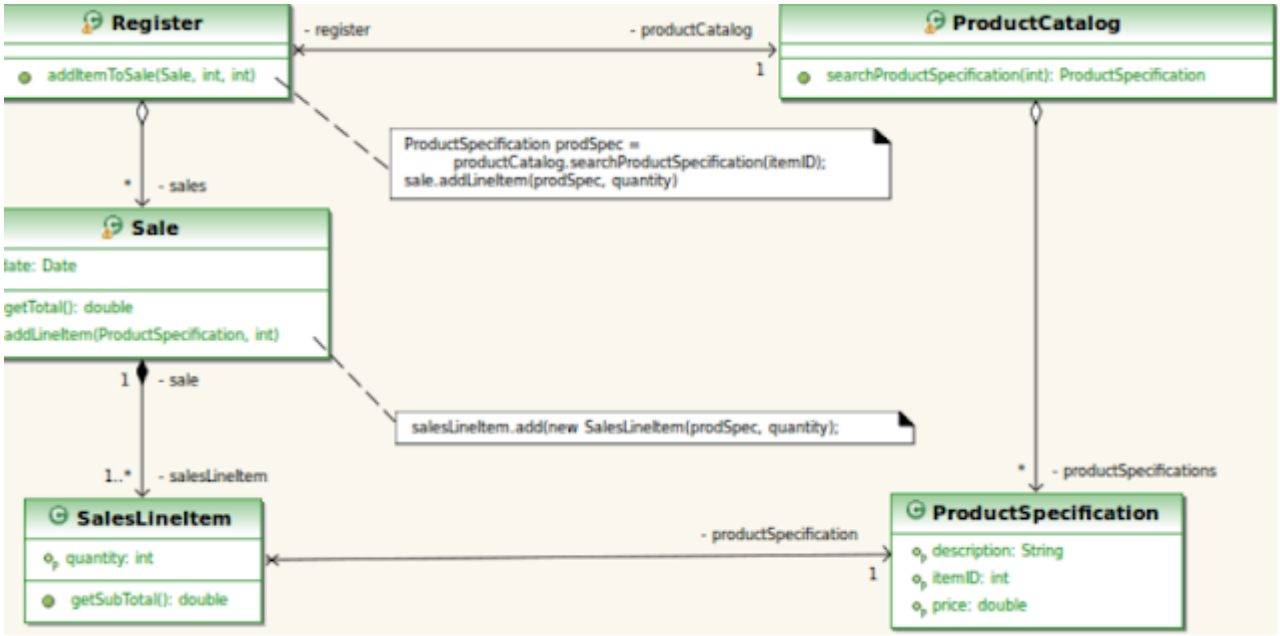
Example

Let's consider Point of Sale (POS) application: a Brief overview of POS application.

- Application for a shop, restaurant, etc. that registers sales.
- Each sale is of one or more items of one or more product types and happens at a certain date.
- A product has a speci cation including a description, unitary price, and identifier.
- The application also registers payments (say, in cash) associated with sales.
- A payment is for a certain amount, equal or greater than the total of the sale.

Problem statement: who should be responsible for creating a SalesLineItem instance?

In the POS application, who should be responsible for creating a **SalesLineItem** instance? By **Creator**, we should look for a class that aggregates, contains, and so on, **SalesLineItem** instances.



Since a Sale contains (in fact, aggregates) many **SalesLineItem** objects, the Creator pattern suggests that **Sale** is a good candidate to have the responsibility of creating **SalesLineItem** instances.
This leads to a design of object interactions: Creating a SalesLineItem

Sample Code

```
class Sale {
   List<SalesLineItem> salesLineItem
   = new ArrayList<SalesLineItem>();
   //...
   public void addLineItem(ProductSpecification prodSpec,int quantity) {
      salesLineItem.add(new SalesLineItem(prodSpec, quantity);
 return salesLineItem;
   }
}
```

Benefits

- Low coupling (described next) is supported, which implies lower maintenance dependencies and higher opportunities for reuse. Coupling is probably not increased because the created class is likely already visible to the creator class, due to the existing associations that motivated its choice as creator.

Assignment No. 9

Title: Identification and Implementation of GOF pattern

Apply any two GOF pattern to refine Design Model for a given problem description Using effective UML 2 diagrams and implement them with a suitable object oriented language

Date of Completion		**Marks for Regularity (2)**	
		Marks for Presentation (3)	
Remark		**Marks for Understanding (5)**	
Sign. of Staff		**Total Marks (10)**	

Prof. R.S. Badodekar **414459 : Computer Laboratory VIII**