Maharaja Surajmal Institute of Technology
C-4, Janakpuri, New Delhi-110058

SOFTWARE ENGINEERING

Practical Lab File

Subject Code: ETCS-353

2020 - 2024

Submitted By:
Anshita Chaudhary
B.Tech CSE 1 (Morning)
Sr. No.: 23
Enrl. No.: 02415002720

Submitted To:
Dr. Savita Ahlawat
Associate Professor
CSE Department

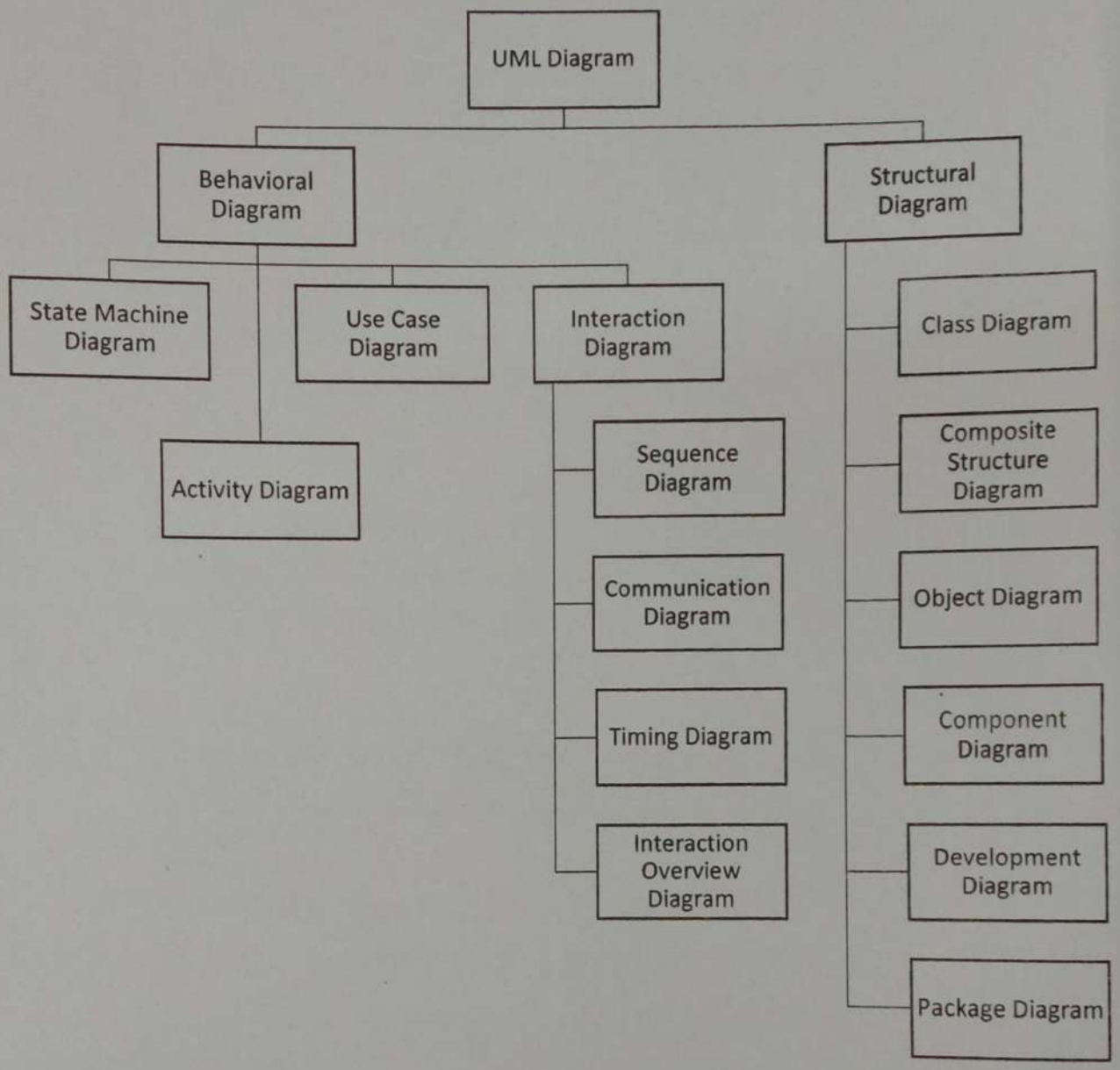
SEMESTER – V

INDEX

S.No.	Name Of Experiment	Date	Signature
01.	Introduction to UML.	27-09-22	18/10
02.	To study the properties of Use case Diagram. Draw and explain Use case diagram of: (a) ATM Machine (b) Library Management system	4-10-22	18/10
03.	To study the structural view diagram: class Diagram.	11-10-22	01/11
04.	To study behavioral view diagram: Sequence Diagram.	18-10-22	
05.	To study behavioral view diagram: collaboration Diagram.	01-11-22	6/12/22
06.	To generate code for a given class diagram and write the steps involved in code generation.	15-11-22	13/12/22
07.	To draw the behavioral view diagram: state-chart diagram.	13-12-22	20/12

INDEX

S.No.	Name Of Experiment	Date	Signature
08.	To draw the implementation view diagram: Component diagram.	20-12-22	27/12
09.	To draw environmental view diagram: Deployment diagram for the system.	27-12-22	3/1/23


UML Diagram
Behavioral Diagram
Structural Diagram
State Machine Diagram
Use Case Diagram
Interaction Diagram
Activity Diagram
Sequence Diagram
Communication Diagram
Timing Diagram
Interaction Overview Diagram
Class Diagram
Composite Structure Diagram
Object Diagram
Component Diagram
Development Diagram
Package Diagram

EXPERIMENT-01

Aim: Introduction to UML.

Theory:

The UML stands for Unified Modelling language, is a standard general-purpose visual modeling language in the field of Software Engineering. It is used to specify, visualize, construct and document the primary artifacts of the software system. It helps in designing and characterizing, especially these software systems that incorporate the concept of Object Orientation. It describes the working of both software and hardware systems.

The UML was developed in 1994-95 by Grady Booch, Ivar Jacobson and James Rumbagh at the Rational Software. In 1997, it got adopted as a standard by the Object Management Group.

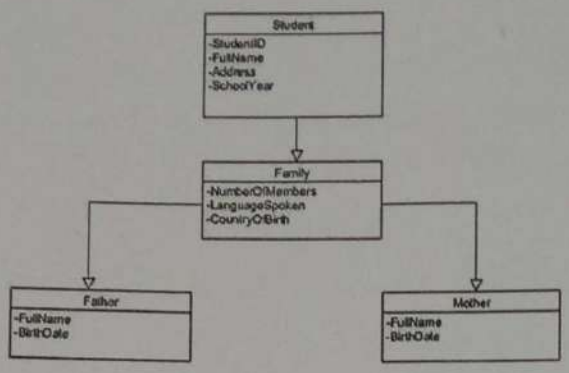
Characteristics of UML:

1. It is a generalized modeling language.
2. It is distinct from other programming languages like C++, python, etc..
3. It is interrelated to object-oriented analysis and design.
4. It is used to visualize the workflow of a system.
5. It is pictorial language, used to generate powerful modeling artifacts.

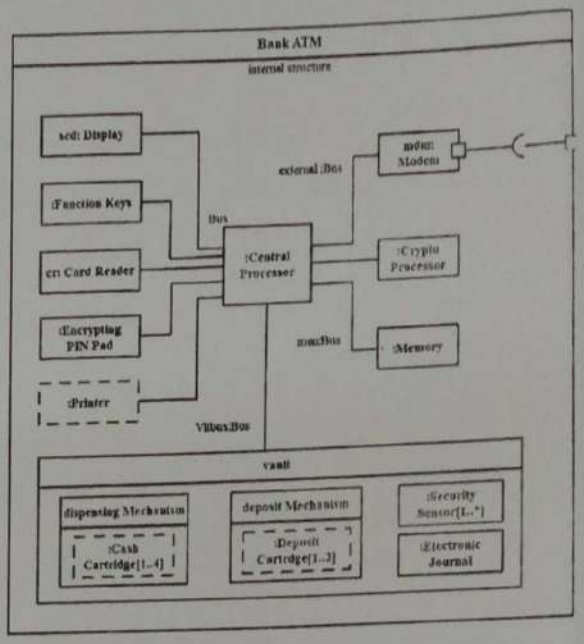
Design:

UML offers a way to visualize a system's architectural blueprints in a diagram, including elements such as:

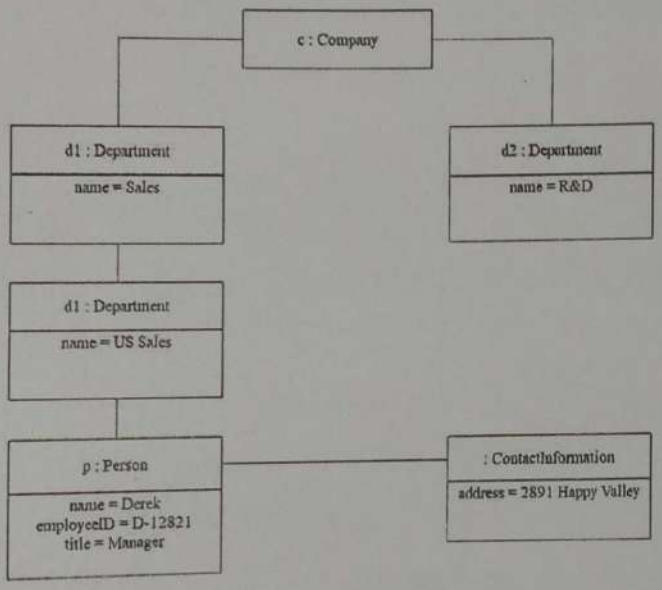
- any activities; (Jobs)
- individual component of system and how they interact with other software components;
- How the system will run;
- How entities interact with others (components and interfaces);
- External User interface.



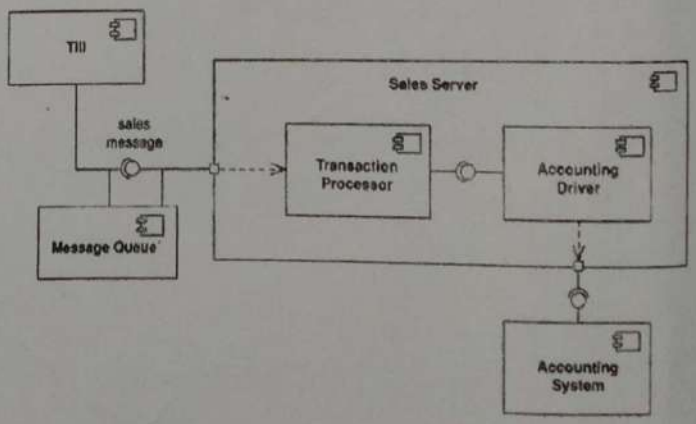
Class Diagram



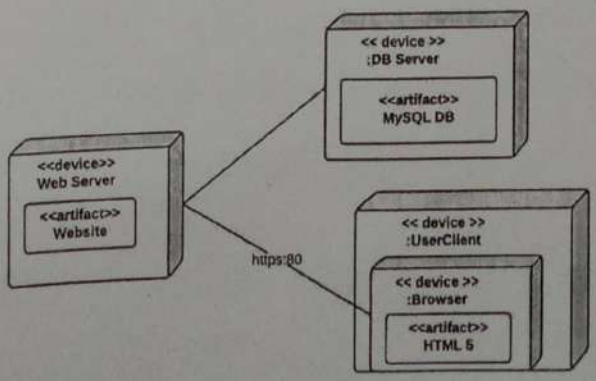
Composite Structure Diagram



Object Diagram



Component Diagram



Deployment Diagram

Diagrams:

The UML diagrams are categorized into structural diagrams, behavioral diagrams and also interaction overview diagrams.

1. Structural Diagrams:

Structural diagrams depict a static view or structure of a system. It is widely used in the documentation of a software architecture. It presents an outline for the system. It embraces:

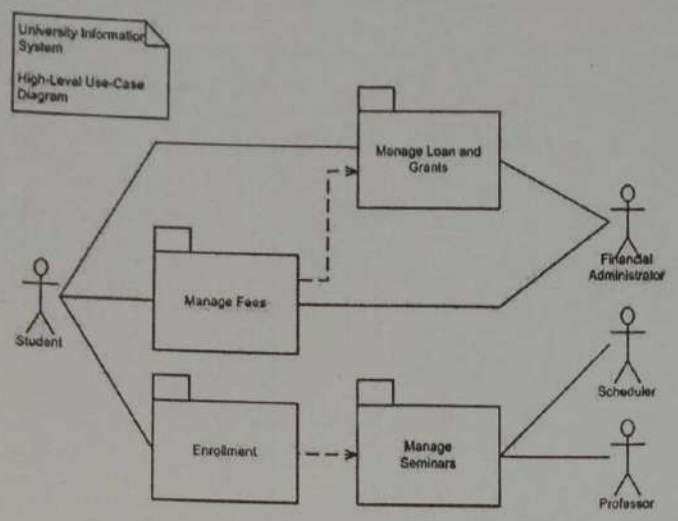
- Class Diagram: It is the backbone of all object-oriented software systems. It depicts the static structure of the system. It displays system's class, attributes and methods. It is helpful in recognizing the relation between different objects as well as classes.

- Composite Structure Diagram: It shows parts within the class. It represents individual parts in a detailed manner when compared to class diagram.

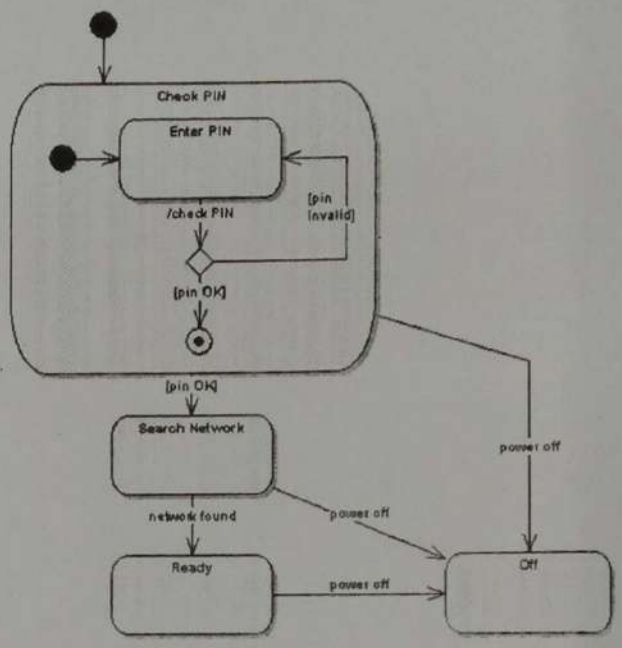
- Object Diagram: It describes the static structure of a system at a particular point in time. It can be used to test the accuracy of class diagrams. It represents distinct instances of classes and relationship between them at a time.

- Component Diagram: It portrays the organization of the physical components within the system. It is used for modeling execution details. It determines whether the desired functional requirements have been considered by the planned development, or not, as it depicts the structural relationship.

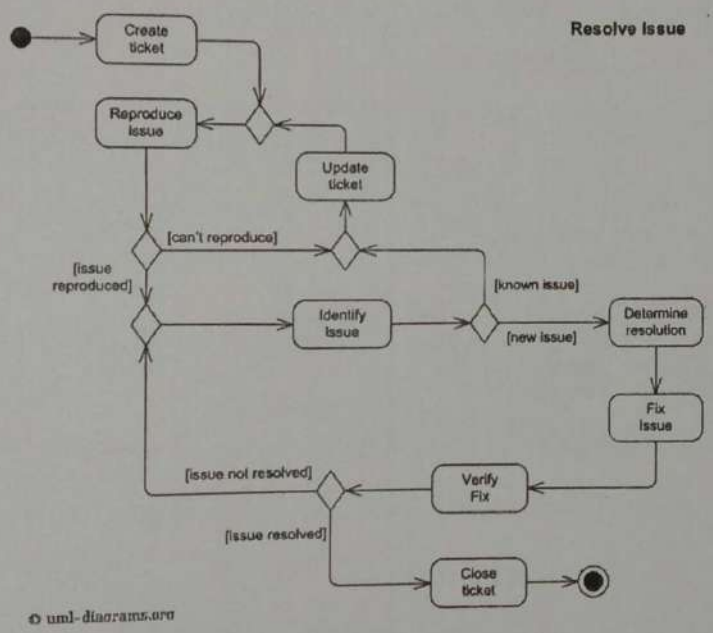
- Deployment Diagram: It presents the software and its hardware by telling what the existing physical components are and what software components are running on them. It is incorporated whenever software is used, distributed or deployed across multiple machines with dissimilar configuration.



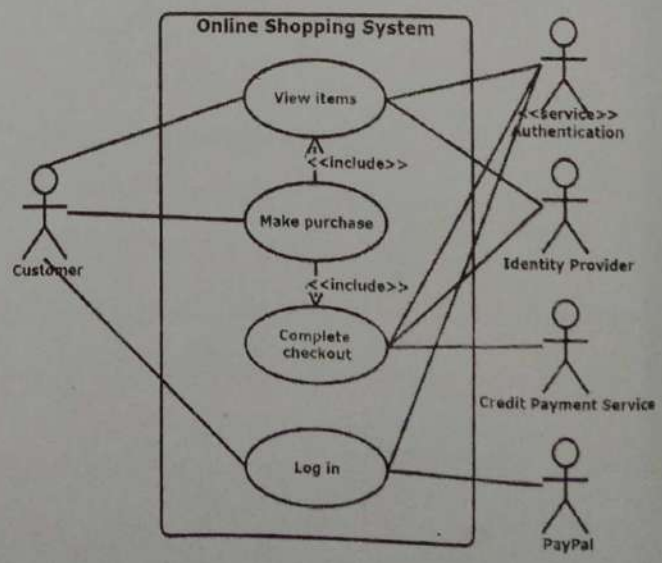
Package Diagram



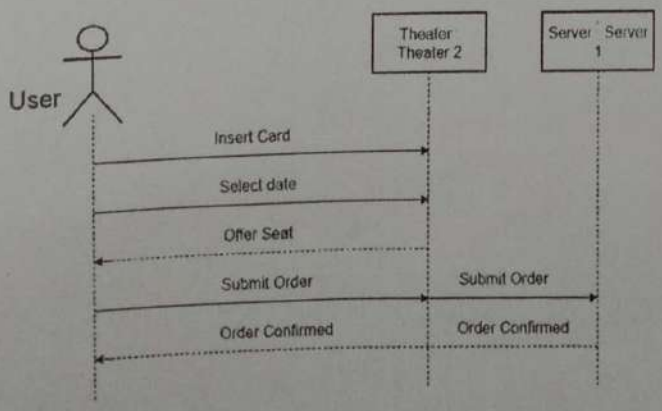
State Machine Diagram



Activity Diagram



Use Case Diagram



Sequence Diagram

• Package Diagram: It is used to illustrate how the packages and their elements are organized. It shows the dependencies between distinct packages.

2. Behavioral Diagram:

Behavioral diagram portrays a dynamic view of a system or the behaviour of a system, which describes the functioning of the system. It defines the interactions within a system. It includes:

• State Machine Diagram: It portrays the system's behaviour utilizing finite state machine transitions. It is also known as State-Charts Diagram. It models the dynamic behaviour of class in response to external stimuli.

• Activity Diagram: It models the flow of control from one activity to other. we can model sequential and concurrent activities. It visually depicts the workflow and what causes an event to occur.

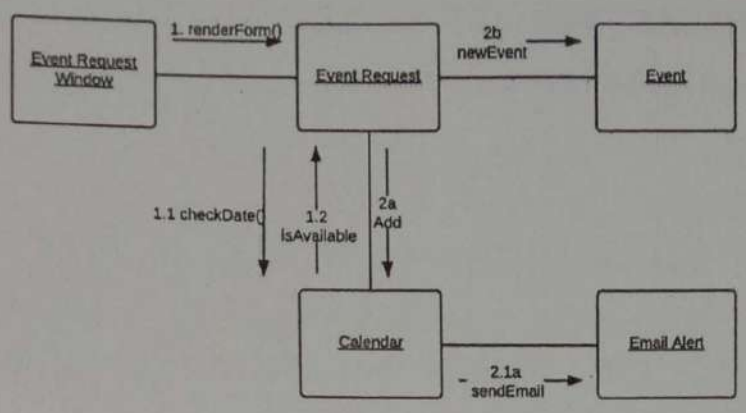
• Use Case Diagram: It represents the functionalities of a system by utilizing actors and use cases. It encapsulates the functional requirement of a system and its association with actors.

3. Interaction Diagram:

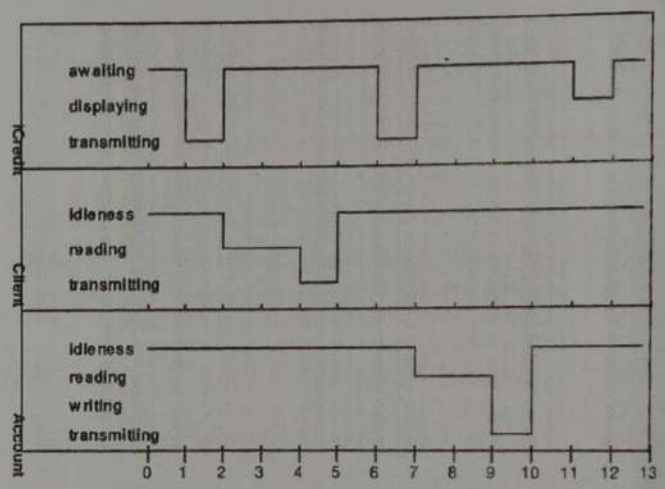
Interaction diagram are subclass of behavioral diagrams that give emphasis on object interactions and also depicts the flow of various use case elements of a system. In simple words, it shows how objects interact with each other and how data flows within them. It consists of:

• Sequence Diagram: It shows the interactions between the objects in terms of message exchanged over time. It delineates in what order and how the object functions in a system.

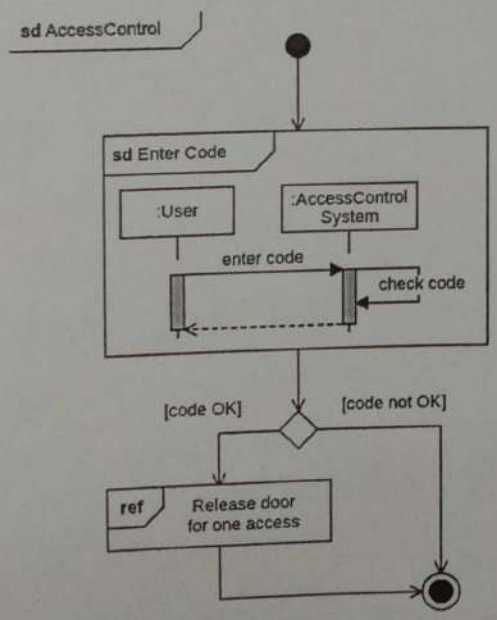
• Communication Diagram: It shows the interchange



Communication Diagram



Timing Diagram



Interaction Overview Diagram

of sequence messages between the objects. It focuses on objects and their relations. It describes the static and dynamic behaviour of system.

- Timing Diagram: It is a special kind of sequence diagram used to depict the object's behaviour over a specific period of time. It governs the change in state and object behaviour by showing the time and duration constraints.

- Interaction Overview Diagram: It is a mixture of activity and sequence diagrams that depicts a sequence of actions to simplify the complex interactions into simple interactions.

18/10

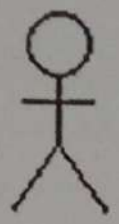
EXPERIMENT-02

Aim : To study the properties of Use case Diagrams. Draw and explain Use Case Diagrams of :
(a) ATM Machine
(b) Library Management System

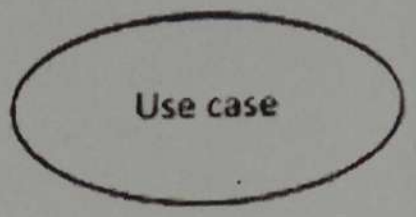
Theory :
A use case diagram is used to represent the dynamic behaviour of a system. It encapsulates the system's functionality by incorporating use cases, actors and their relationships. It models the tasks, services and functions required by a system / subsystem of an application. It depicts the high-level functionality of system and also tells how the user handles a system.

Purpose of Use Case Diagram:
It accumulates the system's requirement, which includes both internal as well as external influences. It represents how an entity from the external environment can interact with a part of the system.
Following are the purpose of use case diagram :
(a) It gathers the system's needs.
(b) It depicts the external view of the system.
(c) Validates system's architecture.
(d) It recognizes the internal as well as external factors that influence the system.
(e) It represents interactions between actors.

Notations of use case diagrams:
1. Actors
Actors are stick figures that represent the people actually employing the uses cases. It is someone who interacts with system function (use case). It is named by a noun. Actors have a responsibility towards the system (inputs) and has expectations from the system (outputs).

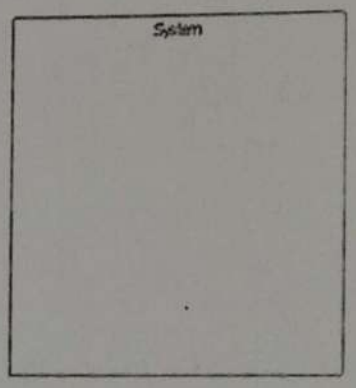


Actor

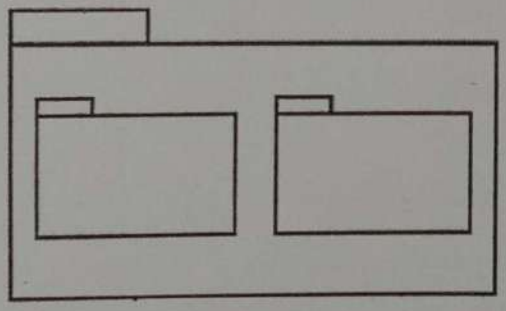


Use Case

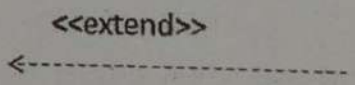
Association



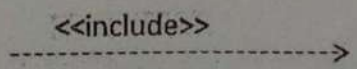
System Boundary



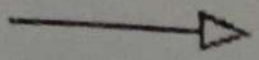
Package



Extends Relationship



Includes Relationship



Generalization Relationship

2. Use Case

Use cases are horizontally shaped ovals that represent the different uses that a user might have. It defines the names of system functions (automated or manual). Each actor must be linked to a use case, while some use cases may not be linked to actors.

3. Communication link / Associations

A line between actors and use cases indicating that the actor and the use case communicate with one another using messages. In complex diagrams, it is important to know which actors are associated with which use case.

4. System Boundary

The system boundary is potentially the entire system as defined in the requirement document. It is a box that sets a system scope to use cases. All use cases outside the box will be considered outside the scope of that system. For large and complex systems, each module may be the system boundary.

5. Packages

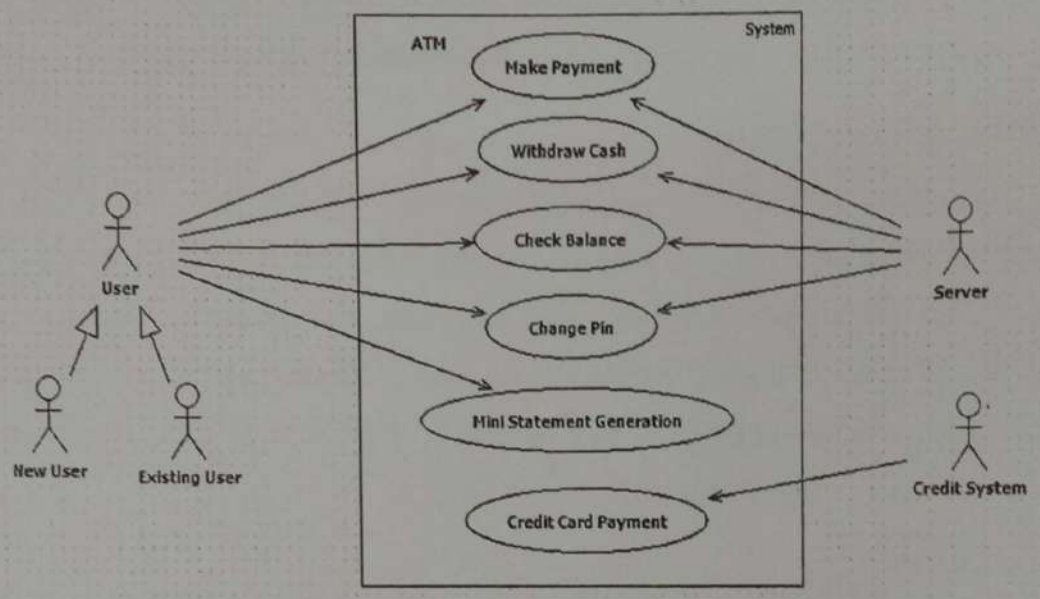
A UML shape that allows you to put different elements into groups. Just as with component diagrams. these groupings are represented as folders.

6. Relationship

A relationship between two use cases is basically modeling the dependency between the two use cases. The reuse of existing use case by using a different type of relationship reduces the overall effort required in developing a system. Use case relationships are:

(a) Extends

Indicidates that a use case may include the behaviour specified by the base use case. It is depicted with a directed arrow having a dotted line. The tip of arrowhead points to the base use case and the child case is connected at the base of the arrow. The stereotype "<< extend >>" identifies as an extend relationship.



ATM Machine Use Case Diagram

(b) Include

when a use case is depicted as using the functionality of another use case, the relationship between the use cases is named as include or uses relationship. A use case includes the functionality of another case as a part of its process flow. A include relationship is depicted with a directed arrow having a dotted line. The tip of arrowhead points the child case and the parent case is connected at the base of the arrow. The stereotype "<< include >>" identifies the relationship as include relationship.

(c) Generalization

The child is used as an enhancement of the parent use case. Generalized cases are shown as a directed arrow with a triangle arrowhead. The child use case is connected at the base of the arrow and the tip of arrow is connected to the parent use case.

Use Case Diagram examples:

(a) ATM Machine:

1. Actors

(a) User (New User and Existing User)

(b) Server

(c) Credit System

2. Use Cases

(a) Make payment - The user can enter the amount to be paid or deposited and then close the session.

(b) Withdraw cash - The user can enter the amount to be withdrawn as per need and the machine then generates a receipt.

(c) Check balance - The user checks the balance in his account and as a next step he can ask the

machine for mini statement generation.

(d) Change Pin - The user may request to change his existing PIN.

All the above use cases involves interaction with the server of the ATM machine system and permissions and verification of details are done by the server only then the session continues otherwise the transaction is cancelled.

(e) Mini statement Generation - The user may demand the mini statement about the account if they want.

(f) Credit card payment - This involves the interaction with credit card system.

3. System Boundary

The system boundary for ATM Machine system consists of make payment, withdraw cash, check balance, change PIN, mini statement generation and credit card payment use cases.

4. Relationship

(a) Generalization - New user and Existing user are generalized as user. The parent actor will have the properties which are common to both child actors and the child actors will have distinct properties other than the ones specified in parent actor.

(b) Library Management System:

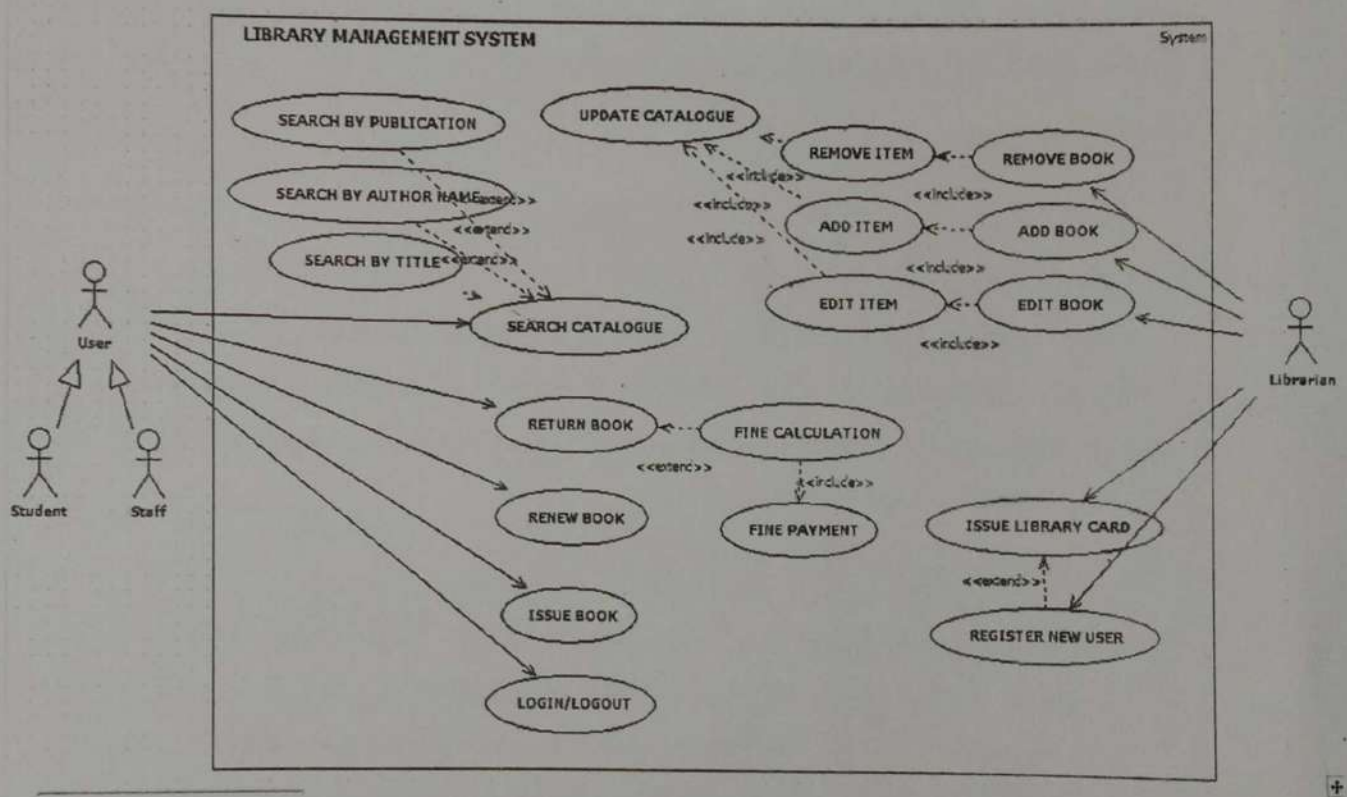
1. Actors

(a) User (student and staff) - responsible for login / logo-out, issue book, renew book, return book and search catalogue.

(b) Librarian - responsible for registration, issue library card, edit book, add book and remove book.

2. Use Cases

(a) Add / Remove / Edit Book - Used by the librarian to add, remove or modify a book or book item



Library Management System Use Case Diagram

(b) Search catalogue - To search the books by title, author name or publication date by user.

(c) Register new account - To add a new member into database.

(d) Issue library card - To issue a new library card for an existing user or create a new card for new user.

(e) Renew Book - To reborrow an already issued book.

(f) Issue Book - To issue a new book.

(g) Return Book - To return a book to the library which was issued to a member.

3. System Boundary

The system boundary for Library Management System consists of login/logout, Issue Book, Renew Book, return book, search catalogue, register new user, issue library card, edit book, add book and remove book use cases.

4. Relationship

(a) Generalization - Student and staff are generalized as user. The parent actor will have the common properties whereas distinct individual properties will be with child case.

(b) Include - Return book use case includes the functionality of fine payment if the extended case of fine calculator is included.

Edit book case includes the edit item case which further includes the functionality to update the catalogue.

Add book / Remove book includes add/remove item case which further includes update catalogue.

(c) Extend - Search catalogue extends with search by title, author name or publication. Any of the functionality may or may not be be included if required by user.

Return book is extended with fine calculator and uses its functionality only when fine is applicable for the user.

Issue library card is extended by register new user and uses the functionality only when the user that demands for card is not already registered in the database.

18/10.

EXPERIMENT-03

Aim: To perform structural view diagram: Class Diagram.
Draw and explain Class Diagram of:
(a) ATM Machine
(b) Library Management System.

Theory:
The class diagram depicts a static view of an application. It represents the types of objects included in the system and the relationships between them. A class consists of its objects and also it may inherit from other classes. A class digaram is used to visualize, describe and document various different aspects of the system, and also construct executable software code.

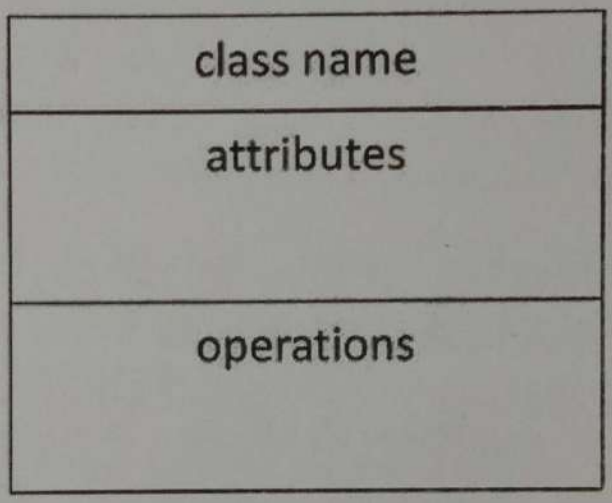
It shows the attributes, classes, functions and relationships to give an overview of the software system. It constitutes class names, attributes and functions in a separate compartment that helps in software development. Since, it is a collection of classes, attributes, interfaces, associations, collaborations and constraints, it is termed as structural diagram.

Purpose of Class Diagram:

The class diagram is the only diagram that is used for construction, and it can be mapped with object-oriented languages.
1. It analyses and designs a static view of an application.
2. It describes the major responsibilities of a system.
3. It is a base for component and deployment diagram.
4. It incorporates forward and reverse engineering.

Benefits of Class Diagram:

1. It can represent the object model for complex systems.
2. It reduces the maintenance time by providing an overview of



Sections of Class Diagram

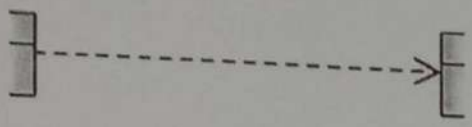
how an application is structured before coding.

3. It provides general schematic of an application for better understanding.
4. It represents a detailed chart by highlighted code, which is to be programmed.
5. It is helpful for the stakeholders and the developers.

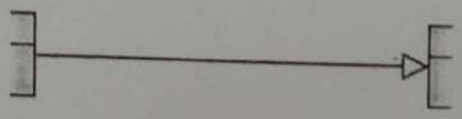
Vital components of Class Diagram:

The class diagram is made up of three sections:

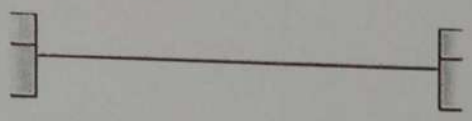
1. Upper Section: The upper section encompasses the name of the class. A class is a representation of similar objects that share same relationships, attributes, operations and semantics. Some rules should be taken into account while representing a class are:
   (a) capitalize initial letter of the class name.
   (b) Place the class name in the center of the upper section.
   (c) A class name must be written in bold format.
   (d) The name of the abstract class should be written in italics format.
2. Middle Section: The middle section constitutes the attributes which describe the quality of the class. The attributes have the following characteristics:
   (a) The attributes are written along with the visibility factors, which are public (+), private (-), protected (#) and package (~).
   (b) The accesibility of an attribute class is illustrated by the visibility factors.
   (c) A meaningful name should be assigned to attributes which explain its usage inside the class.
3. Lower Section: The lower section contains methods or operations. The methods are represented in the form of a list, where each method is written in a single line. It demonstrates how a class interacts with data.



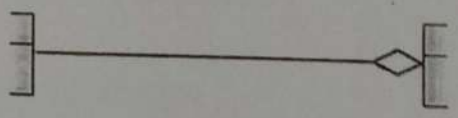
Dependency



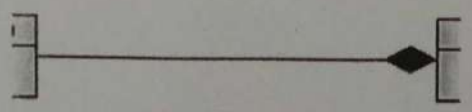
Generalization



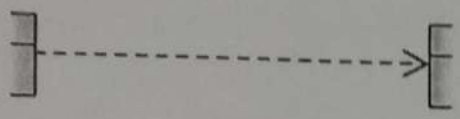
Association



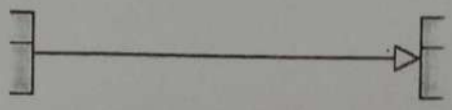
Aggregation



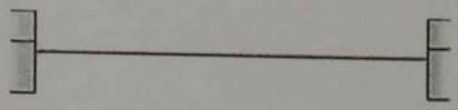
Composition



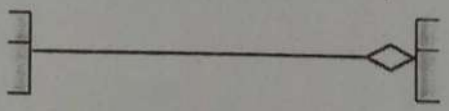
Dependency



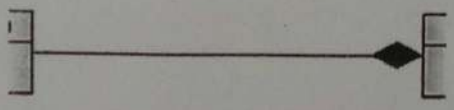
Generalization



Association



Aggregation



Composition

Relationships:

1) Dependency: A dependency is a semantic relationship between two or more classes where a change in one class cause changes in another class. It forms a weaker relationship.

2) Generalization: A generalization is a relationship between a parent class (super class) and a child class (subclass). In this child class is inherited from the parent class.

3) Association: It describes a static or physical connection between two or more objects. It depicts how many objects are there in a relationship.

4) Aggregation: An aggregation is a subset of association, which represents 'has a' relationship. It is more specific than association. It defines a part-whole or part-of relationship. The child can exist independently of its parent class.

5) Multiplicity: It defines a specific range of allowable instances of attributes. In case if a range is not specified, one is considered as a default multiplicity.

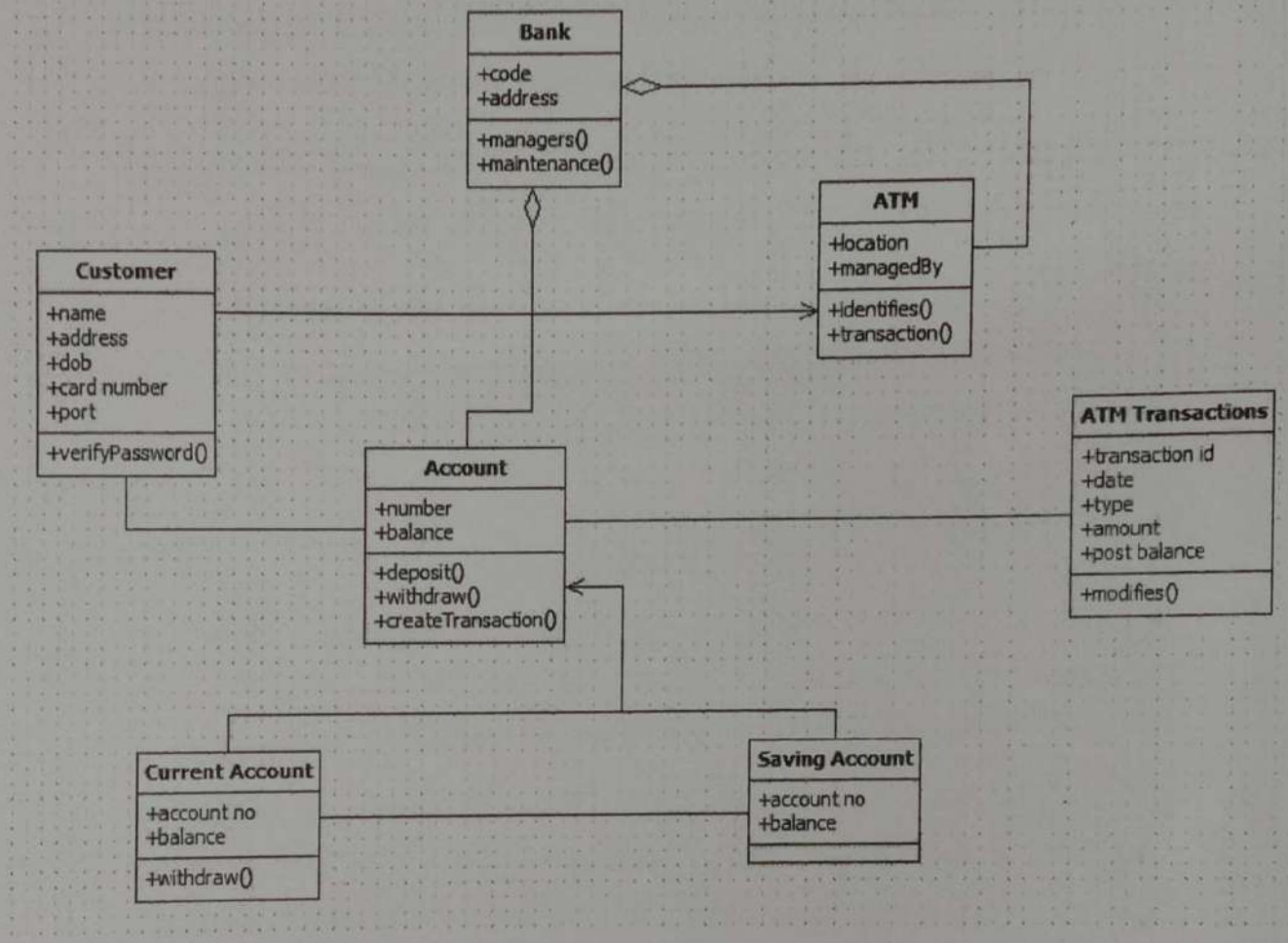
6) Composition: The composition is subset of aggregation. It portrays the dependency between the parent and its child, which means if one part is deleated, then other part also gets discarded. It represents whole-part relationship.

Class Diagram examples:

(a) ATM Machine

The class diagram for an ATM machine consists of the following classes:

- Bank
- ATM
- Customer
- Account
- ATM Transactions
- Current Account



Class Diagram for ATM Machine

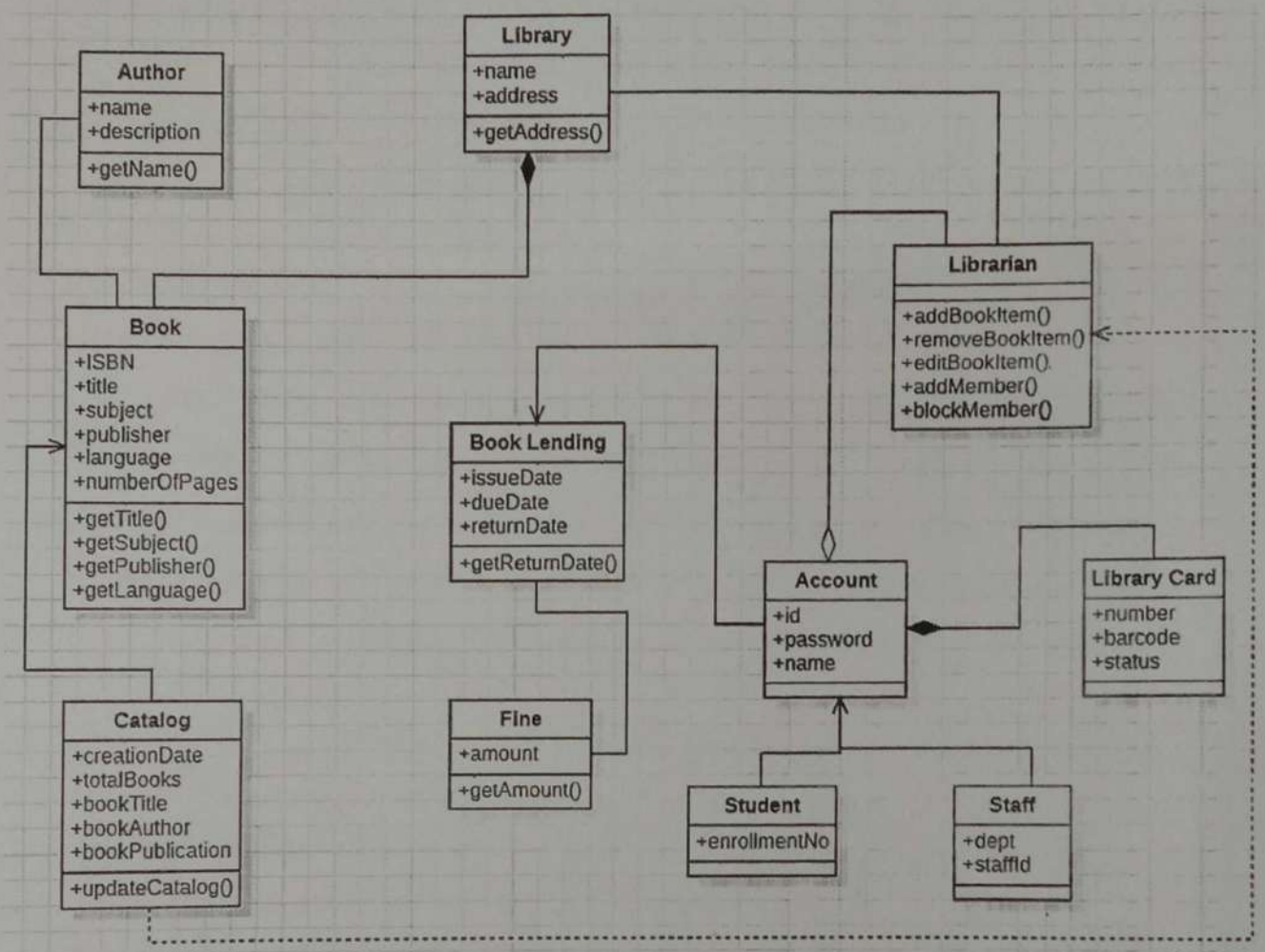
• Savings Account

The bank class represents a physical bank. It has a code assigned to it and an address of where it is located. The bank manages several accounts and maintains them accordingly along with the ATM class for a specific bank. There is association between the two classes because one can have a bank without accounts and accounts without a particular bank. It has ~~a~~ methods called manages() and maintenance().

The ATM class represents a physical ATM that has a location and a bank associated to it which manages it. We can come up with two methods: identifier(), wherein the ATM must cross-check the validity of the customer and is uni-directionally associated with customer class, and transactions() which may include activities like withdrawing depositing or checking balance by the customer in his account.

The customer class represents a real customer. The customer has name, address, date of birth (dob), card number and a port or pin number. It has a method called verifypassw-ord() that is checked when the ATM class calls for validating the customer details before any transaction. The customer class is bidirectionally associated with Account class because for a person to be a customer at the ATM he must have an account.

The Account class represents a bank account. Common attribut-es include account number and balance. You can deposit() and withdraw() cash from the account. In addition-n to this the bank might offer two types of accounts: current Account and Saving Account. These two classes are inherited from Account class. The current account class has account number and shows the balance to withdraw() money. The savings account class is used



Class Diagram for Library Management System

to check the balance in account.

The ATM transaction class shows the details of transaction like transaction id, date, type, amount and post balance. It is used to modify() the account details after every transaction done by the customer.

(b) Library Management System

The class diagram for library management system consists of the following classes:

- Library
- Librarian
- Book
- Author
- catalog
- Fine
- Book lending
- Account
- Library card

The library class is the central part of the organisation for which the software is designed. It has attributes like name to distinguish it from other libraries and address to describe its location. It has a method called getAddress() to extract its location.

The Book class is the basic building block of the system. The books can be identified by title, ISBN, subject, publisher, and the language along with number of pages in the book. The methods getTitle(), getSubject(), getPublisher() and getLanguage() are used to fetch books' details.

The Account has information about the id, password and name of the member. The Account class has two types of accounts: student class and staff class where we have enrollmentNo, dept and staff id as attributes.

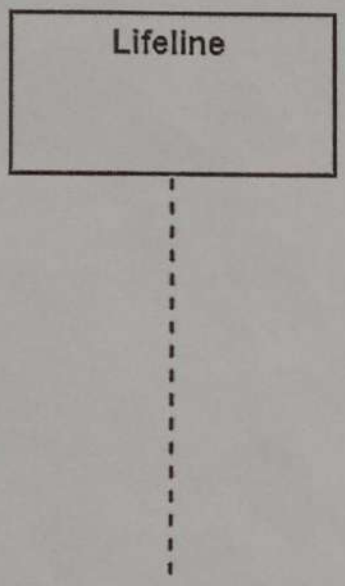
The class library card will be used to identify the members' action of issuing a book or returning book. It has a unique number and barcode and tells the status of member whether active (issue or return or fine) or blocked.

The Book lending class manages the checking-out of books by keeping a record of issuedate, duedate and returndate of the books.

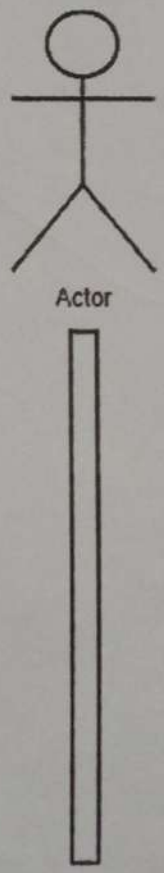
The class catalog contains list of books sorted under certain critria by the librarian class. Our system will support searching for books by Title, Author and Publication. And when the librarian class updates the catalog, it is done by update catalog() method.

The fine class is responsible for collecting fines from the library members.

The Author class encapsulates the author's name and description to maintain record in the library by the Book class.



Lifeline



Actor

EXPERIMENT - 04

Aim: To perform behavioral view diagram: Sequence Diagram.
Draw and explain sequence diagram of:
(a) ATM Machine
(b) Library management System

Theory:
The sequence diagram represents the flow of messages in the system and is also termed as an event diagram. It helps in envisioning several dynamic scenarios. It portrays the communication between any two lifelines as a time-ordered sequence of events, such that these lifelines took part at a run time. In UML, the lifeline is represented by a vertical bar, whereas the message flow is represented by a vertical dotted line that extends across the bottom of the page. It incorporates the iterations as well as branchings.

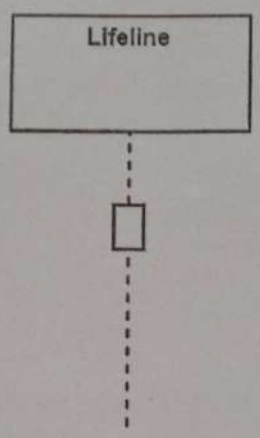
Purpose of sequence Diagram:

1) To model - high level interaction among objects within system.
2) To model interaction among objects inside a collaboration realizing a use case.
3) It either models generic interactions or some certain instances of interactions.

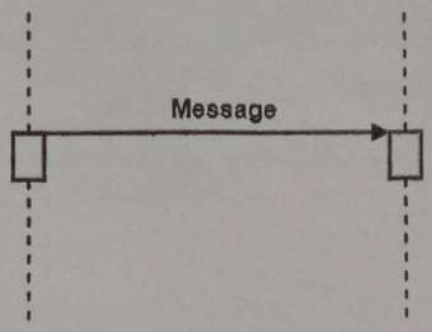
Notations:

1) Lifeline
An individual participant in the sequence diagram is represented by a lifeline. It is positioned at the top of the diagram.

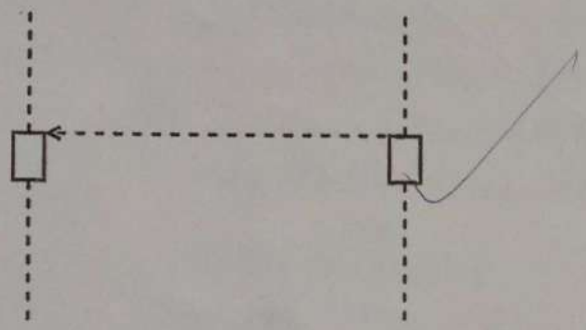
2) Actor
A role played by an entity that interacts with the subject is called actor. It is an out of the scope of the system. It represents the role, which involves human users and external hardware or subjects.



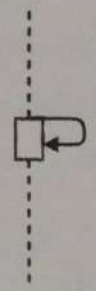
Activation



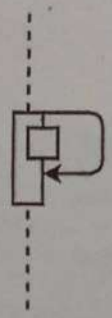
CallMessage



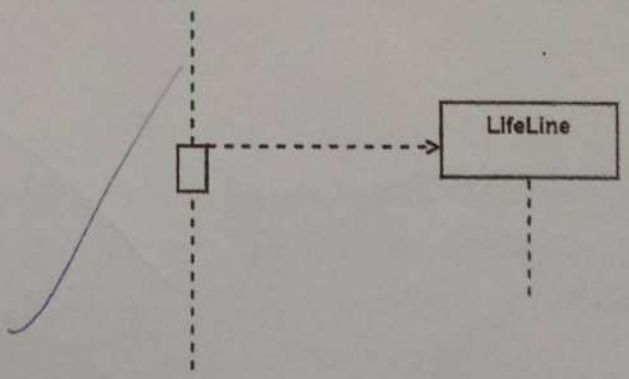
Return Message



Self Message



Recursive Message



Create Message

3) Activation

It is represented by a thin rectangle on the lifeline. It describes that time period in which an operation is performed by an element, such that the top and bottom of the rectangle is associated with the initiation and the completion time, each respectively.

4) Messages

The messages depict the interaction between the objects and are represented by arrows. They are in the sequential order on the lifeline. The core of sequence diagram is formed by messages and lifelines.

Following are the types of messages enlisted below:

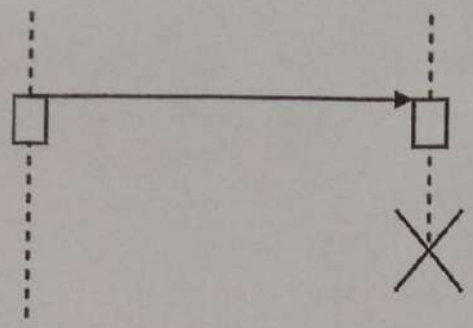
(a) Call message - It is defined as a particular communication between the lifelines of an interaction, which represents that the target lifeline has invoked an operation.

(b) Return message - It defines a particular communication between the lifelines of interaction that represents the flow of information from the receiver of the corresponding called message.

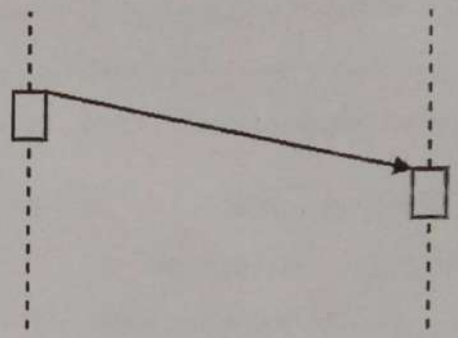
(c) Self message - It describes a communication, particularly between lifeline of an interaction that represents a message of the same lifeline, has been invoked.

(d) Recursive message - A self message sent for a recursive purpose. It can be said that the recursive message is a special case of the self message as it represents the recursive calls.

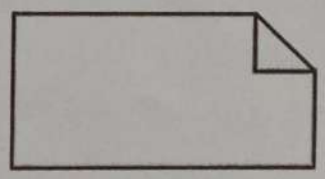
(e) Create message - It describes a communication, particularly between the lifelines of interaction describing that the target (lifeline) has been initiated.



Destroy Message



Duration Message



Note

(f) Destroy message - It describes a communication, particularly between the lifelines of an interaction that depicts a request to destroy the lifecycle of the target.

(g) Duration message - It describes a communication particularly between the life lines of an interaction which portrays the time passage of the message while modeling a system.

5) Note

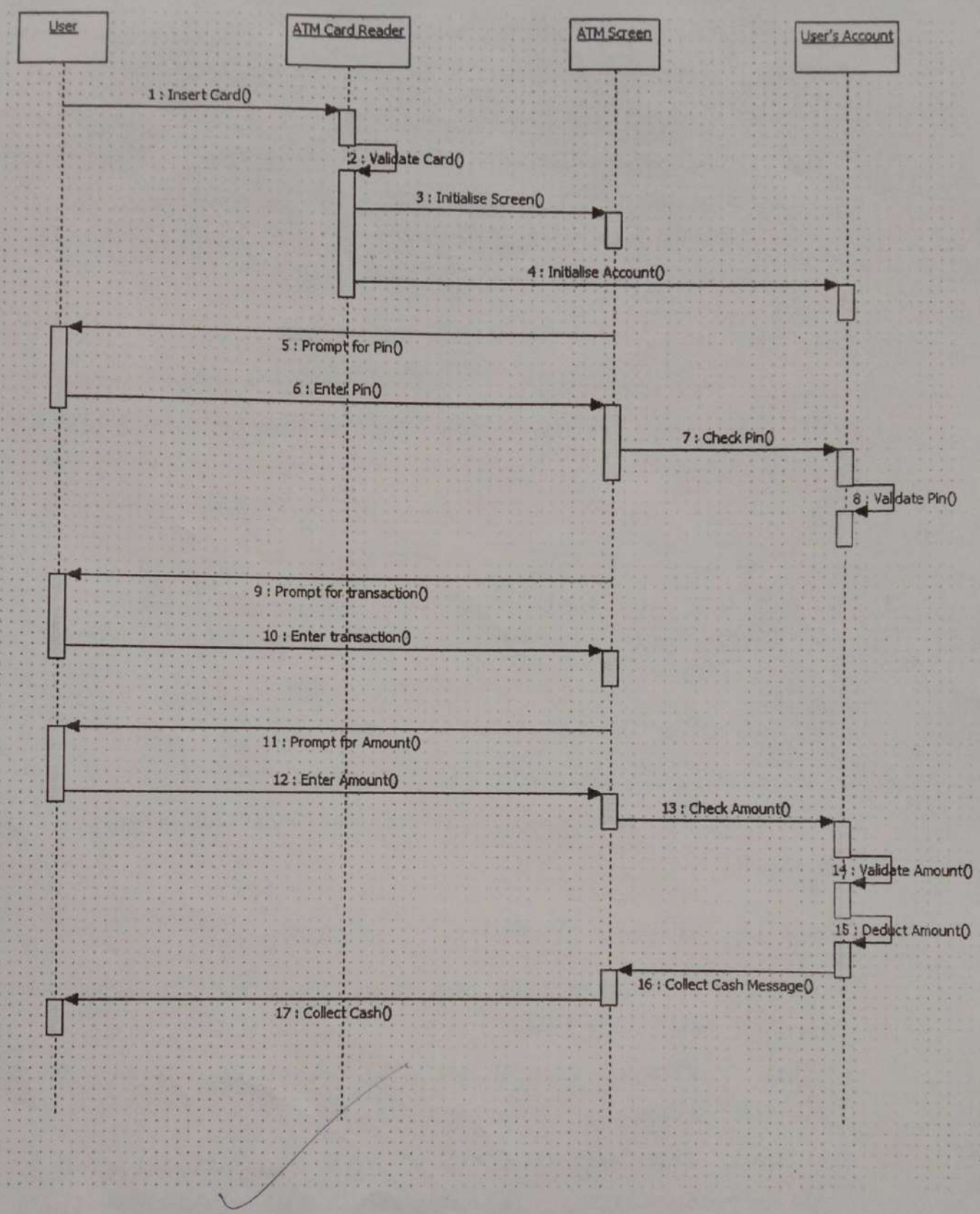
A note is the capabilites of attaching several remarks to the element. It basically carries useful information for the modelers.

Benefits of Sequence Diagram:

1) It explores the real-time application.
2) It depicts the message flow between the different objects.
3) It has easy maintenance.
4) It is easy to generate.
5) It is easy to update as per the new change in the system.

Drawback of Sequence Diagram:

1) In case of too many lifelines, the sequence diagram can get more complex.
2) The incorrect result may be produced, if the order of the flow of messages changes.
3) Since each sequence needs distinct notations for its representations, it may make the diagram more complex.
4) The type of sequence is decided by the type of message.

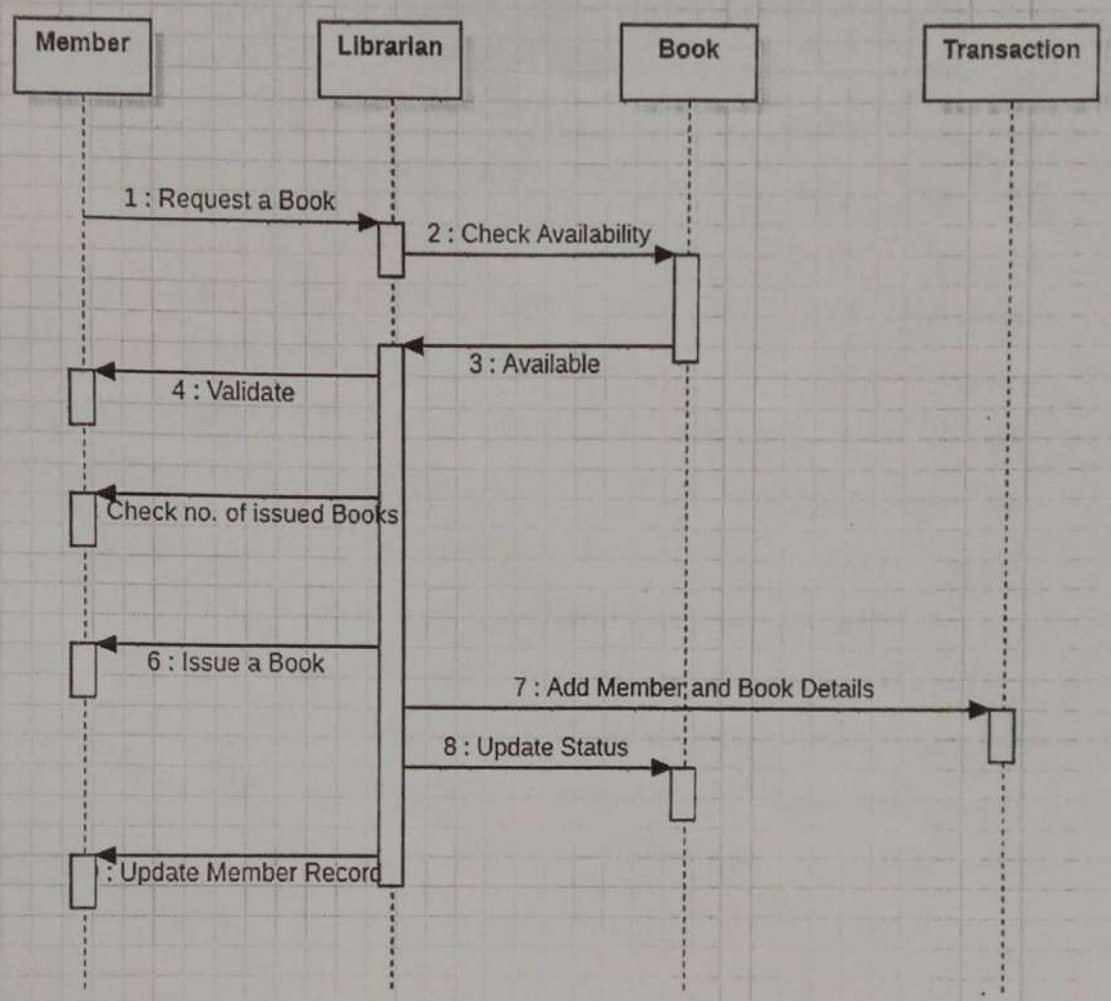


Sequence Diagram of ATM Machine

Sequence Diagrams Examples:

(a) ATM Machine

The sequence diagram illustrates how a bank customer interacts with the ATM to withdraw cash. The client inserts the card and the ATM card reader validates the card. Next, it initialises the ATM screen and user's account. Next, the client gets a prompt for the pin and when the client enters the pin, the ATM checks the pin with the corresponding user's account and validates the pin. The user then receives a prompt for transaction. The client enters the transaction to be taken place, in this case, to withdraw cash. The client then gets a prompt to select the amount and after the client enters the amount the ATM checks the amount with the user's account and validates the transaction if the user has enough funds. The ATM will then deduct the amount from the user's account and dispense the cash for the user to collect.

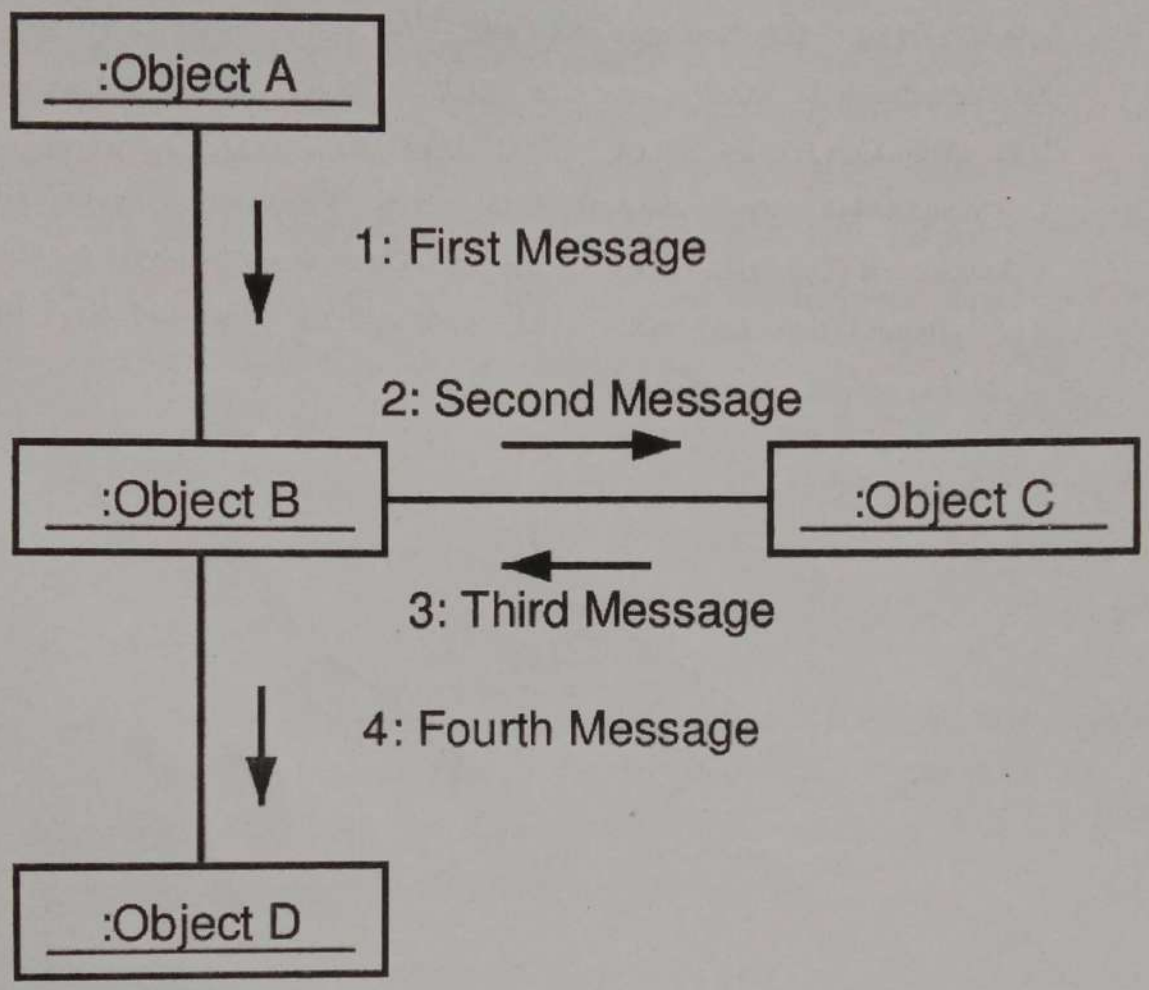


Sequence Diagram of Library Management System

(b) Library Management System

The sequence diagram maps out a library management system. First, the member requests a book from the librarian. The librarian checks the availability of the book and validates the request. Before issuing the book, the librarian checks the number of books the member has checked out. Then the member gets the requested book. The librarian creates the transaction and documents the member and book details. After the librarian creates the transaction, the librarian updates the status of the issued book and member record.

01/11/22



Components of Collaboration Diagram

EXPERIMENT-05

Aim: To perform Behavioral View Diagram: Collaboration diagram. Draw the collaboration diagram of:

(a) ATM Machine

(b) Library Management System

Theory:

The collaboration diagram is used to show the relationship between the objects in a system. Both the sequence and the collaboration diagrams represent the same information but differently. Instead of showing the flow of messages, it depicts the architecture of the object residing in the system as it is based on object-oriented programming.

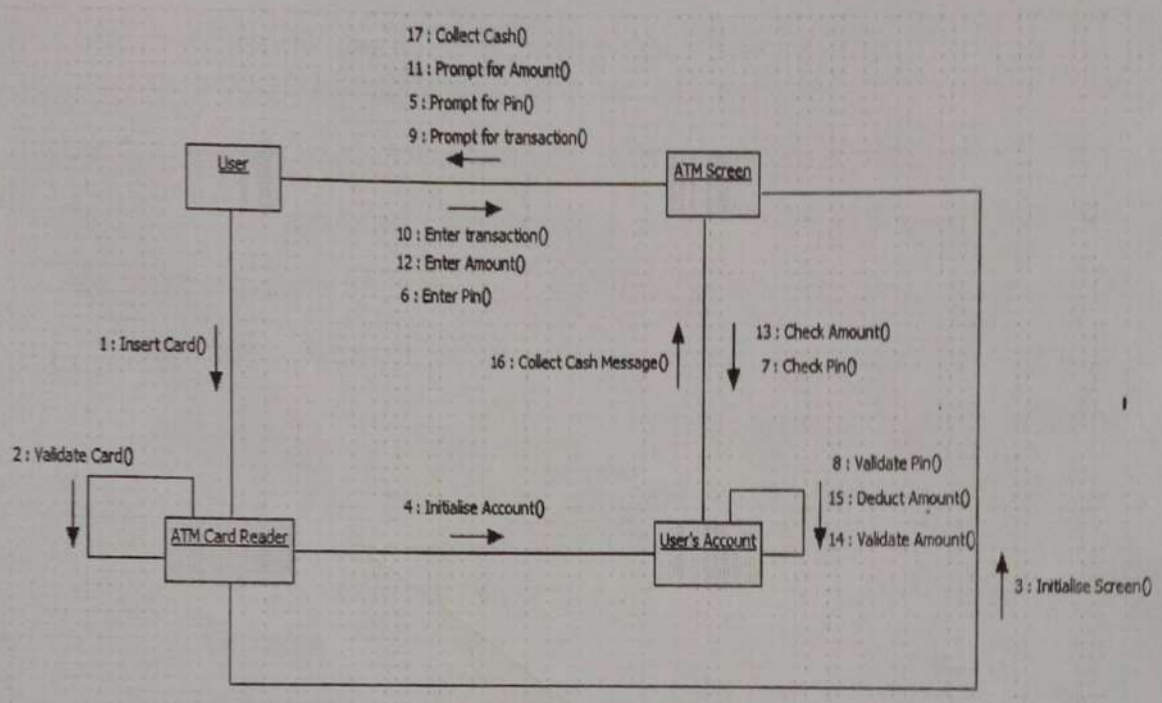
Notations of collaboration Diagram:

1. Objects: The representation of an object is done by an object symbol with its name and class underlined, separated by a colon. It is represented by specifying their name and class. It is not mandatory for every class to appear.

2. Actor: The actor plays the main role as it iniciates or invokes the interaction. Each actor has its respective role and name.

3. links: The link is an instance of association, which associates the objects and actor. It portrays a relationship between the objects through the messages are sent. It is represented by a solid line. The link helps an object to connect with or navigate to another object.

4. Messages: It is communication between objects which carries information and includes a sequence number, so that the activity may take place. It is represented by a labelled arrow, which is placed near a link. The messages are sent from sender to receiver, and direction must be navigate in that



Collaboration diagram of ATM Machine

particular direction. The receiver must understand the message.

Uses:

1. To model collaboration among objects or roles that carry the functionalities of use cases and operations.
2. To capture the interactions that represent the flow of messages between the objects inside the diagram.
3. To support identification of objects participating in use case.
4. To emphasize on structural aspect of an interaction diagram, ie, how lifelines are connected.
5. Since, collaboration diagrams are not that expensive, the sequence diagram can be directly converted to the collaboration diagram.

Drawback:

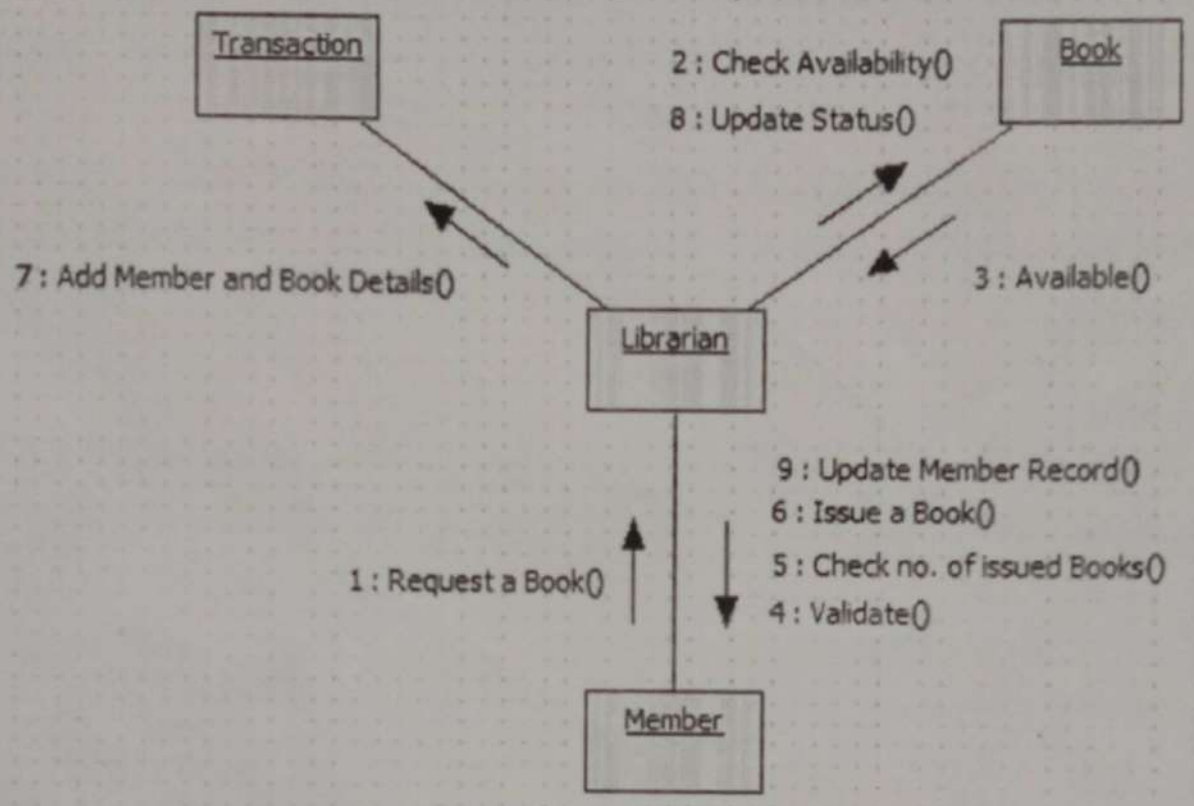
1. Multiple holding objects in the system can make a complex collaboration diagram, as it becomes quite hard to explore the objects.
2. It is time-consuming diagram.
3. After program terminates, the object is destroyed.
4. As the object state changes momentarily, it becomes difficult to keep an eye on every single that has occured inside the object of a system.

Collaboration Diagram Examples:

(a) ATM Machine

Objects: User
ATM card Reader
ATM screen
User's Account

links: User – ATM card Reader
User – ATM screen
ATM card Reader – ATM card Reader (self)



Collaboration Diagram of Library Management System

ATM card Reader - User's Account
ATM card Reader - ATM screen
User's Account - User's Account (self)
User's Account - ATM screen

Messages : 1. Insert card ()
2. Validate card () - self
3. Initialise Screen ()
4. Initialise Account ()
5. Prompt for Pin ()
6. Enter Pin ()
7. check Pin ()
8. Validate Pin () - self
9. Prompt for transaction ()
10. Enter transaction ()
11. Prompt for Amount ()
12. Enter Amount ()
13. check Amount ()
14. Validate Amount () - self
15. Deduct Amount () - self
16. collect cash message ()
17. collect cash ()

(b) Library Management System

Objects : Librarian
Member
Book
Transaction

Links : Librarian - Member
Librarian - Transaction
Librarian - Book

messages: 1. Request a Book ()
2. check Availability ()
3. Available ()
4. Validate ()
5. check no. of Books issued ()
6. Issue a Book()
7. Add member and Book Details ()
8. Update status ()
9. Update member Record()

6/12/22

Account.java 2 ×

Model2 > Account.java > Account

```java
import java.util.*;
public class Account {
    /**
     * Default constructor
     */
    public Account() {
    }
    public void number;
    public void balance;
    public void deposit() {
        // TODO implement here
    }
    public void withdraw() {
        // TODO implement here
    }
    public void createTransaction() {
        // TODO implement here
    }
}
```

Account.java

ATM.java 2 ×

Model2 > ATM.java > ATM

```java
import java.util.*;
public class ATM {
    /**
     * Default constructor
     */
    public ATM() {
    }
    public void location;
    public void managedBy;
    public void identifies() {
        // TODO implement here
    }
    /**
     *
     */
    public void transaction() {
        // TODO implement here
    }
}
```

ATM.java

ATM Transaction.java 9+ ×

Model2 > ATM Transaction.java > ATM

```java
import java.util.*;
public class ATM Transaction {
    /**
     * Default constructor
     */
    public ATM Transaction() {
    }
    public void transaction id;
    public void date;
    public void type;
    public void amount;
    public void post balance;
    public void modifies() {
        // TODO implement here
    }
}
```

ATM Transaction.java

Bank.java 2 ×

Model2 > Bank.java > Bank

```java
import java.util.*;
public class Bank {
    /**
     * Default constructor
     */
    public Bank() {
    }
    public void code;
    public void address;
    public void managers() {
        // TODO implement here
    }
    public void maintenance() {
        // TODO implement here
    }
}
```

Bank.java

Current Account.java 5 ×

Model2 > Current Account.java > Current

```java
import java.util.*;
public class Current Account extends Account {
    /**
     * Default constructor
     */
    public Current Account() {
    }
    public void account no;
    public void balance;
    public void withdraw() {
        // TODO implement here
    }
}
```

Current Account.java

Customer.java 6 ×

Model2 > Customer.java > Customer

```java
import java.util.*;
public class Customer {
    /**
     * Default constructor
     */
    public Customer() {
    }
    public void name;
    public void address;
    public void dob;
    public void card number;
    public void port;
    public void verifyPassword() {
        // TODO implement here
    }
}
```

Customer.java

Saving Account.java 5 ×

Model2 > Saving Account.java > Saving

```java
import java.util.*;
public class Saving Account extends Account {
    /**
     * Default constructor
     */
    public Saving Account() {
    }
    public void account no;
    public void balance;
}
```

Saving Account.java

Code Generation for ATM Machine class diagram

EXPERIMENT - 06

Aim: To generate code using a given class diagram and write the steps involved in code generation.

Theory:

Steps involved in code generation are:

1) construct a class diagram or choose a class diagram to generate its code.
2) Go to tools, see if the language in which you want the code in present or not.
3) If the language is not present. Go to extension manager and install plugins for that language. once installed restart starUML.

   If the language is already present below then proceed to step 4.

4) Click on the language you want the code in, and further click on 'generate code'.
5) A dialogue box appears. Select the required checkboxes which contains the class diagram and click OK.
6) Now slect the location where you want your code to be saved and click select folder.
7) This will generate the code in the desired location.

Code generation examples:

(a) ATM Machine

classes: Bank, ATM, Account, customer, ATM Transaction, current Account, Saving Account

Parent class: Account

child class: current Account, Saving Account

Account.java 3 ×

Model1 > Model1 > Account.java > Account

```java
import java.util.*;
public class Account {
    /**
     * Default constructor
     */
    public Account() {
    }
    public void id;
    public void password;
    public void name;
    public void Operation1() {
        // TODO implement here
    }
}
```

Account.java

Author.java 2 ×

Model1 > Model1 > Author.java > Author > description

```java
import java.util.*;
public class Author {
    /**
     * Default constructor
     */
    public Author() {
    }
    public void name;
    public void description;
    public void getName() {
        // TODO implement here
    }
}
```

Author.java

Book Lending.java 6 ×

Model1 > Model1 > Book Lending.java > Book > issueDate

```java
import java.util.*;
public class Book Lending {
    /**
     * Default constructor
     */
    public Book Lending() {
    }
    public void issueDate;
    public void dueDate;
    public void returnDate;
    public void getReturnDate() {
        // TODO implement here
    }
}
```

Book Lending.java

Catalog.java 5 ×

Model1 > Model1 > Catalog.java > Catalog > bookPublication

```java
import java.util.*;
public class Catalog {
    /**
     * Default constructor
     */
    public Catalog() {
    }
    public void creationDate;
    public void totalBooks;
    public void bookTitle;
    public void bookAuthor;
    public void bookPublication;
    public void updateCatalog() {
        // TODO implement here
    }
}
```

Catalog.java

Book.java 6 ×

Model1 > Model1 > Book.java > Book > getPublisher()

```java
import java.util.*;
public class Book {
    /**
     * Default constructor
     */
    public Book() {
    }
    public void ISBN;
    public void title;
    public void subject;
    public void publisher;
    public void language;
    public void numberOfPages;
    public void getTitle() {
        // TODO implement here
    }
    public void getSubject() {
        // TODO implement here
    }
    public void getPublisher() {
        // TODO implement here
    }
    public void getLanguage() {
        // TODO implement here
    }
}
```

Book.java

Fine.java 1 ×

Model1 > Model1 > Fine.java > Fine > amount

```java

import java.util.*;
public class Fine {
    /**
     * Default constructor
     */
    public Fine() {
    }
    public void amount;
    public void getAmount() {
        // TODO implement here
    }

}
```

Fine.java

Library.java 2 ×

Model1 > Model1 > Library.java > Library > getAddress()

```java

import java.util.*;
public class Library {
    /**
     * Default constructor
     */
    public Library() {
    }
    public void name;
    public void address;
    public void getAddress() {
        // TODO implement here
    }
}
```

Library.java

Library Card.java 6 ×

Model1 > Model1 > Library Card.java > Library > number

```java

import java.util.*;
public class Library Card {
    /**
     * Default constructor
     */
    public Library Card() {
    }
    public void number;
    public void barcode;
    public void status;
}
```

Library Card.java

Staff.java 2 ×

Model1 > Model1 > Staff.java > Staff > staffId

```java

import java.util.*;
public class Staff {
    /**
     * Default constructor
     */
    public Staff() {
    }
    public void dept;
    public void staffId;
}
```

Staff.java

Librarian.java 1 ×

Model1 > Model1 > Librarian.java > Librarian > blockMember()

```java
import java.util.*;
public class Librarian {
    /**
     * Default constructor
     */
    public Librarian() {
    }
    public void Attribute1;
    public void addBookItem() {
        // TODO implement here
    }
    public void removeBookItem() {
        // TODO implement here
    }
    public void editBookItem() {
        // TODO implement here
    }
    public void addMember() {
        // TODO implement here
    }
    public void blockMember() {
        // TODO implement here
    }
}
```

Librarian.java

Student.java 1 ×

Model1 > Model1 > Student.java > Student > enrollmentNo

```java

import java.util.*;
public class Student {
    /**
     * Default constructor
     */
    public Student() {
    }
    public void enrollmentNo;
}
```

Student.java

Code Generation for Library Management System

(b) Library Management System

class: Author, Library, Book, Librarian, Book lending catalog, Fine, Account, Student, Staff, Library card.

13/12/22

EXPERIMENT - 07

Aim: To draw the behavioral view diagram: State - Chart Diagram.

Theory:

The state machine diagram is also called Statechart or State Transition Diagram, which shows the order of states undergone by an object within the system. It captures the software system's behaviour. It models the behaviour of a class, a subsystem, a package, and a complete system.

It tends out to be an effective way of modelling the interactions and collaborations in the external entities and the system. It models event-based systems to handle the state of an object. It also defines several distinct states of a component within the system.

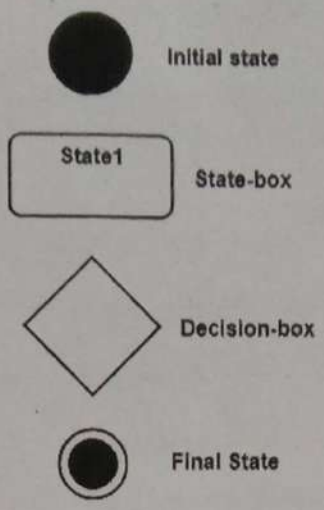
Types of state machine diagrams:

(i) Behavioral state machine - records behaviour of object within the system. It depicts an implementation of a particular entity.

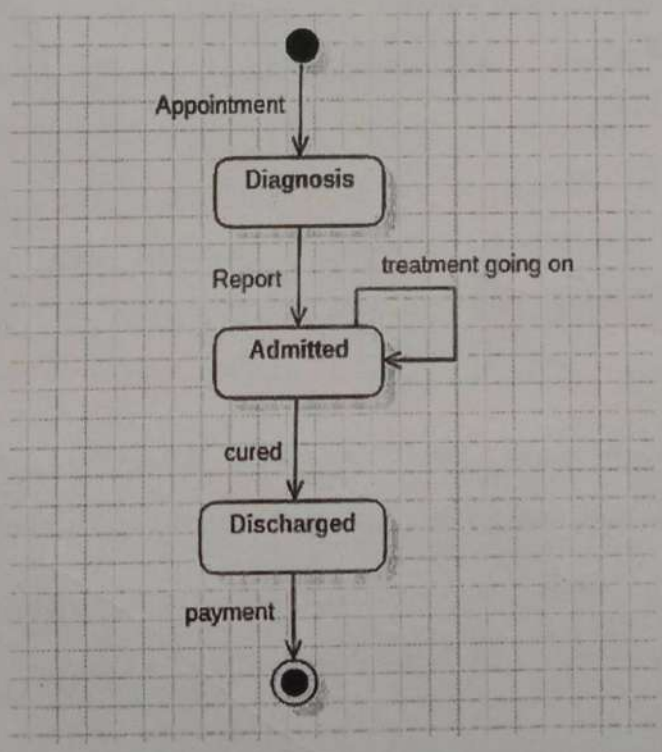
(ii) Protocol state machine - depicts the change in state of protocol and parallel changes within the system. But it does not portray the implementation of particular component.

It blueprints an interactive system that response back to either the internal events or the external events. The execution flow from one state to another is represented by a state machine diagram. It visualizes an object from its creation to termination.

The main purpose is to depict each state of an individual object.



Notations of State-Chart Diagram



State-Chart Diagram for Patient

Notations :

(1) Initial State : It defines the initial state (beginning) of a system and is represented by a black filled circle.

(2) Final State: It represents the final state (end) of a system. It is denoted by a filled circle present within a circle.

(3) Decision Box : It is of diamond shape that represents the decision to be made on the basis of an evaluated guard.

(4) Transition : A change of control from one state to another due to the occurence of some event is termed as a transition. It is represented by an arrow labeled with an event due to which the change has ensued.

(5) State Box : It depicts the conditions or circumstances of a particular object of a class at a specific point of time. A rectangle with round corners is used to represent the state box.

Uses :

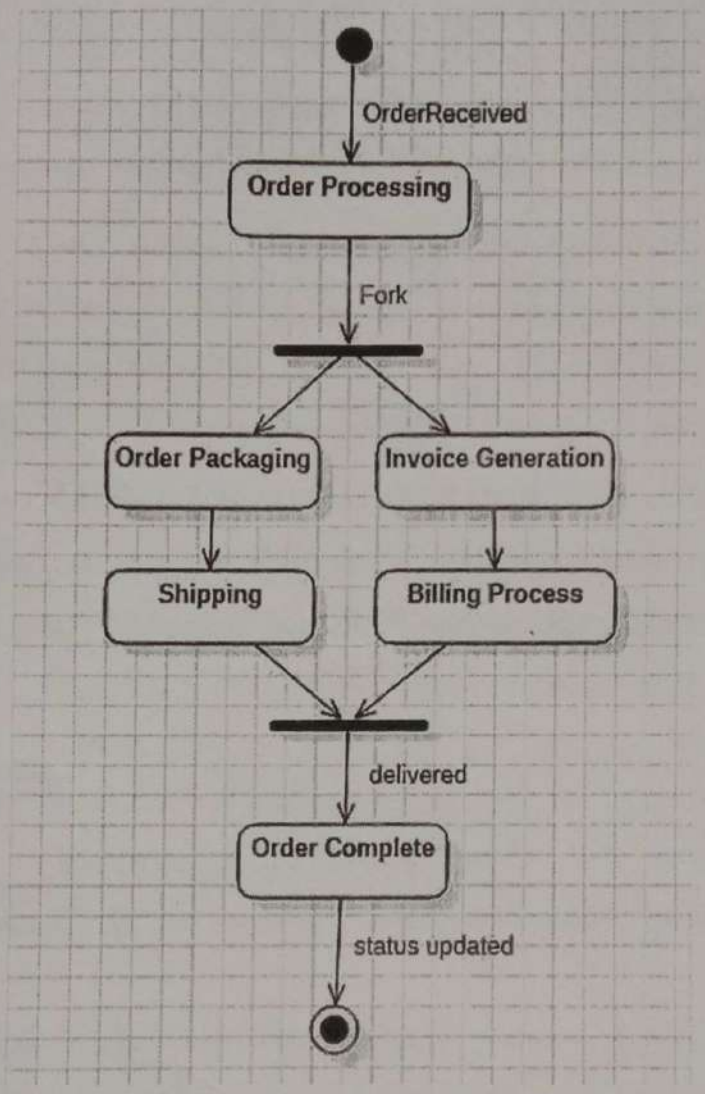
(1) For modelling the object states of a system.
(2) For modelling the reactive system as it contains reactive objects.
(3) For pinpointing events responsible for state transitions.
(4) For implementing forward and reverse engineering.

Examples :

(a) Patient

States : Initial state, diagnosis, admitted, discharged and final state.

Transitions : Patient takes an 'appointment' in the hospital and undergoes diagnosis. Based on the 'report' the patient is admitted. If the patient is 'cured', he is ready to be discharged, otherwise, the 'treatment



State-Chart Diagram for Order-Delivery System

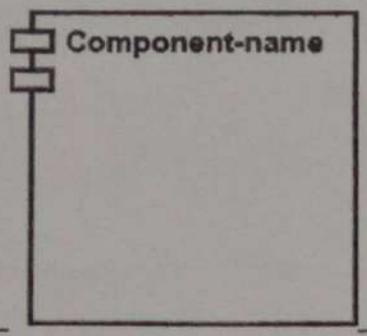
will keep going on' and the patient remains admitted. After the patient is 'cured', he gets discharged and proceeds with payment to the final state.

(b) Order - Delivery System

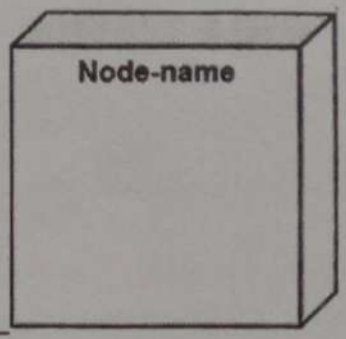
States : Initial state, order processing, order packaging, shipping, Invoice generation, billing process, order complete and final state.

Transitions : Once the 'order is received' by the vendor it reaches the state of order processing. The process further forks or splits in two concurrent processes, ie, order packaging and Invoice generation. Once the order is packed it goes for shipping and simultaneously after the invoice generation the billing process takes place. The shipping and billing process joins for the order to be 'delivered' and reach the order complete state which then results in 'status update' at the users end to concludes to final state.

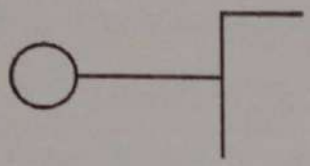
29/12



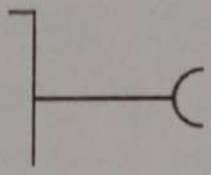
Component



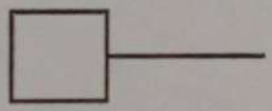
Node



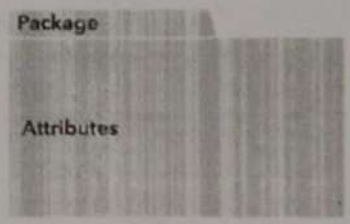
Provided Interface



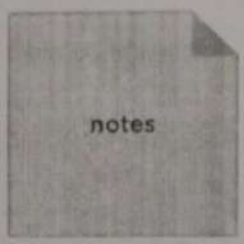
Required Interface



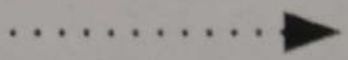
Port



Package



Note



Dependency

EXPERIMENT-08

Aim: To perform the implementation view diagram: Component Diagram for the system.

Theory:

A component diagram is used to break down a large object-oriented system into the smaller components, so as to make them more manageable. It models the physical view of a system such as executables, files, libraries, etc... that resides within the node.

It visualises the relationships as well as the organization between the components present in the system. It helps in forming an executable system. A component is a single unit of the system, which is replacable and executable. The implementation details of a component are hidden, and it necessitates an interface to execute a function. It is like a black box whose behaviour is explained by the provided and required interfaces.

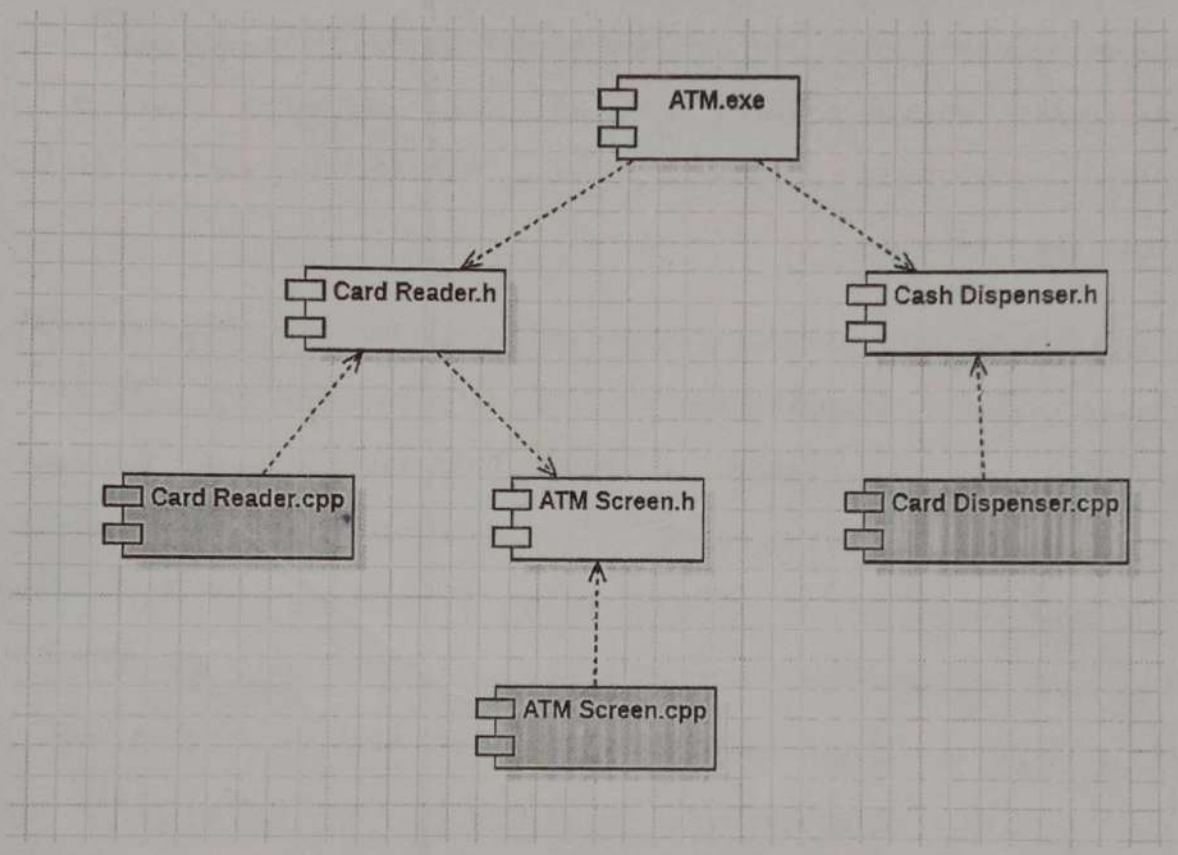
Notations:

(1) Component - an entity required to execute a function. A component provides and consumes behavior through interfaces, as well as through other components.

(2) Node - represents hardware or software objects, which are of a higher level than components.

(3) Interface - shows input or material that a component either receives or provides. Interface can be:

(a) Provided interface - where component produces information used by the required interface of



Component Diagram for ATM System

another component.

(b) Required interface - where component requires information in order to perform its proper function.

(4) Port - specifies a separate interaction point between the component and environment.

(5) Package - groups together multiple elements of the system and is just a file folders which can be drawn around several components.

(6) Note - allows to add a meta-analysis to diagram.

(7) Dependency - shows one part of your system depends on another.

Purpose:

(1) It envisions each component of a system and links between several connections.

(2) Depicts relationships and organization of components.

(3) Helpful in testing a system.

(4) To divide a system into multiple components according to functionality.

(5) To model database schemas.

(6) To model the system's source code.

Example:

(a) ATM System

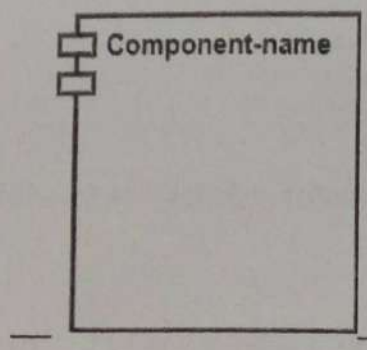
ATM.exe depends on CardReader.h and Cash Despenser.h.

card Reader.cpp depends on card Reader.h.

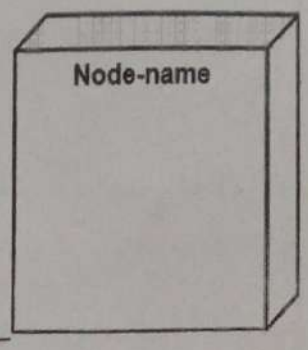
ATM screen.cpp and card Reader.h depends on ATM Screen.h.

Card dispenser.cpp depends on card Dispenser.h.

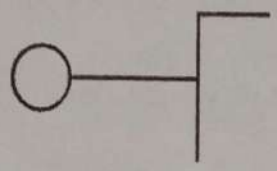
27/12



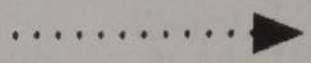
Component



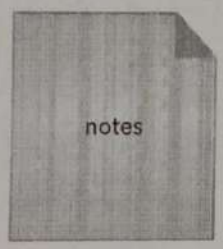
Node



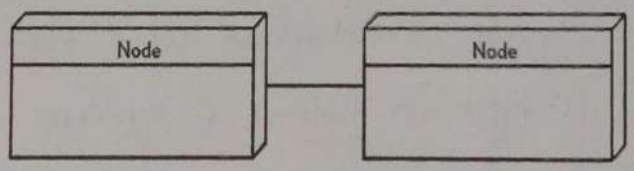
Interface



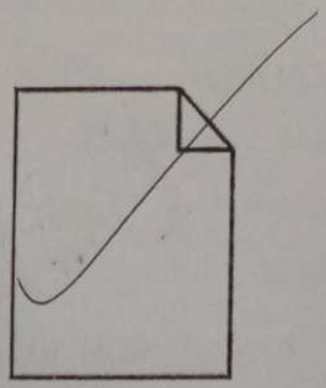
Dependency



Note



Communication Path



Artifact

EXPERIMENT-09

Aim: To perform the environmental view diagram: Deployment diagram for the system.

Theory:

The deployment diagram visualises the physical hardware on which the software will be deployed. It portrays the static deployment view of the system. It involves the nodes and their relationships.

It ascertains how software is deployed on the hardware. It maps the software architecture created in design to the physical system architecture, where the software will be executed as a node. Since it involves many nodes, the relationship is shown by utilizing communication paths.

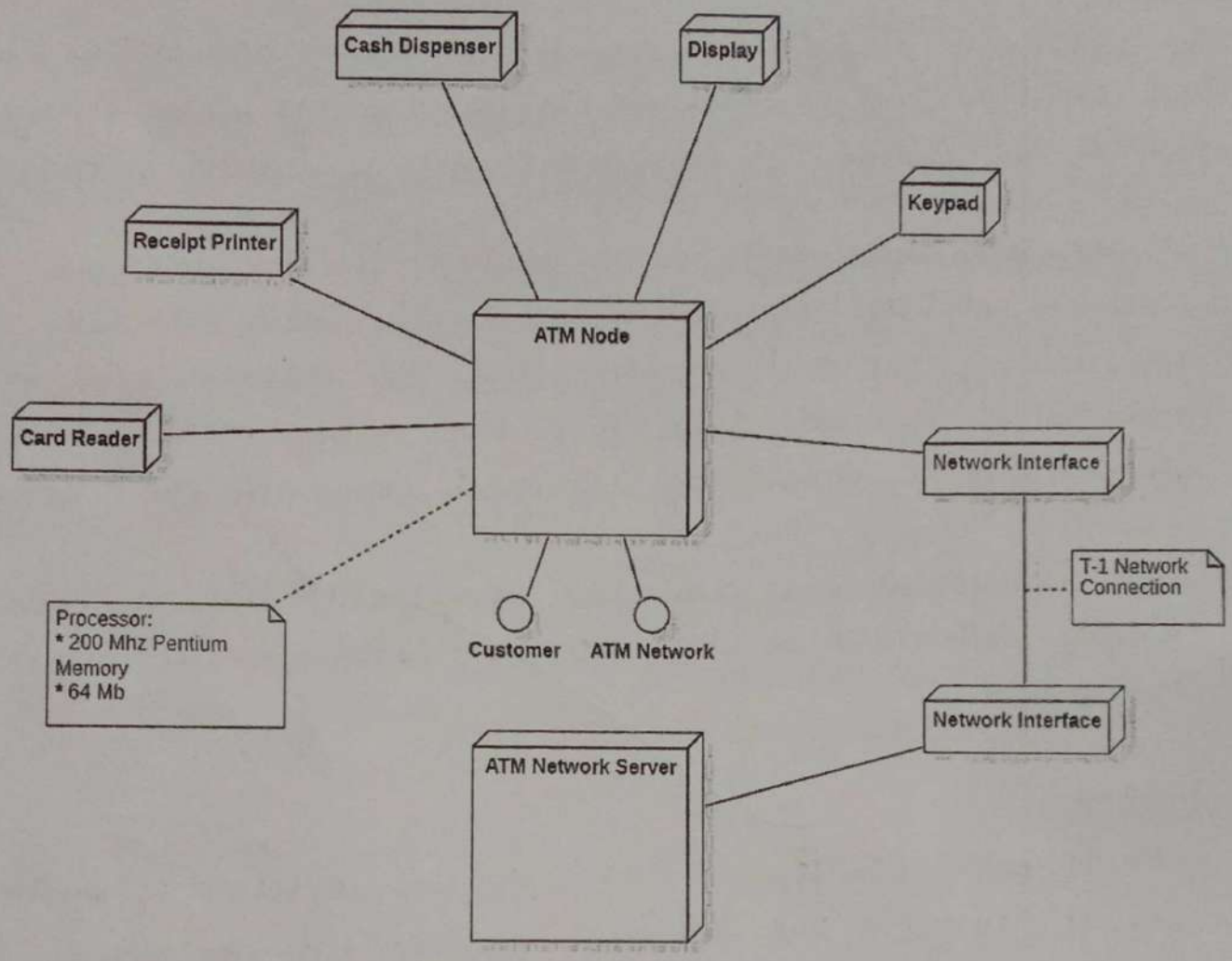
Both deployment diagram and component diagram are closely interrelated as they focus on software and hardware components.

Purpose:

(1) To put attention to the hardware topology of system.
(2) To represent the hardware components on which software components are installed.
(3) To describe processing of nodes at runtime.
(4) It involves high performance, scalability, maintainability, portability and easily understandable.
(5) To model distributed networks and systems.
(6) For modelling embedded systems.

Notations:

(1) Artifact - A product developed by software or deployment and operation of a system. Example: source file, executable file, table in database or document.



Deployment Diagram for ATM Machine

(2) Component - indicates the software elements.

(3) Dependency - indicates that one component or node is dependent on another.

(4) Node - a hardware or software component / object. They are of two types :

(a) Device node - computing resources with processing capability and ability to execute a program. Example: PCs, mobile phones.

(b) Execution environment - node - any computer system that resides with a device node, like JVM

(5) Communication path - represents communication between two nodes.

Example :

(a) ATM Machine

Nodes - ATM Node, Card Reader, Receipt Printer, Cash Dispenser, Display, keypad, network interface and ATM network server.

Interface - customer interface, network interface

3/1/23