Cavalier Institute - https://cavalierinstitutions.com

Date	Oct-08-2024	Session No	6

Topic : Data Modeling (Software Engineering)

References : https://www.lucidchart.com/pages/er-diagrams

Data modeling in software engineering helps in organizing the structure of a system's data to ensure efficient storage, retrieval, and relationships between entities. By using concepts like entities, relationships, normalization, and ER diagrams, software developers can design databases that are both efficient and scalable.

In software engineering, data modeling is the process of creating a visual representation of a system's data and its structure. It defines how data is stored, accessed, and related within a system. The goal is to establish a blueprint for the data architecture, helping developers understand the system's data flow and database design. Let's explore key data modeling concepts with examples:

1. Entities

An entity represents a real-world object or concept that has data stored about it. Each entity is characterized by attributes.

- Example: In a student management system, entities could be `Student`, `Course`, and `Instructor`.

2. Attributes

Attributes are the properties or characteristics of an entity. They define the data that each entity stores.

- Example: For the entity `Student`, attributes could include `StudentID`, `Name`, `DateOfBirth`, `Email`.

3. Relationships

Relationships describe how entities are related to each other. These relationships can be one-to-one, one-to-many, or many-to-many.

- Example: In a school system:
  - A one-to-many relationship: One `Instructor` teaches many `Courses`.
  - A many-to-many relationship: Many `Students` can enroll in many `Courses`.

4. Primary Key

A primary key is a unique identifier for an entity, ensuring that each record in a table can be uniquely identified.

- Example: For the `Student` entity, `StudentID` could be the primary key, uniquely identifying each student.

5. Foreign Key

A foreign key is an attribute in one entity that refers to the primary key of another entity, establishing a relationship between the two entities.

- Example: In a `Course` entity, `InstructorID` could be a foreign key that refers to the `InstructorID` in the `Instructor` entity, linking courses to their respective instructors.

6. Normalization

Normalization is the process of organizing data to reduce redundancy and dependency. It involves dividing large tables into smaller ones and establishing relationships between them to ensure data integrity.

- Example: Suppose there is a single table that stores both student and course information, leading to data redundancy (e.g., storing the same course details for multiple students). By normalizing the table, the `Student` and `Course` entities **would** be split into separate tables, and a new `Enrollment` table would handle the relationship between students and courses.

7. Cardinality

Cardinality refers to the number of instances of one entity that can be related to another entity.

- Example: In a library system:
  - A one-to-many cardinality: One `Library` has many `Books`.
  - A many-to-many cardinality: Many `Members` can borrow many `Books`.

8. ER Diagrams (Entity-Relationship Diagrams)

An ER Diagram is a visual representation of the entities, their attributes, and relationships within a system. It's widely used in data modeling to describe how data is structured and interrelated.

- Example: In an e-commerce system:
  - Entities: `Customer`, `Order`, `Product`
  - Relationships: `Customer` places many `Orders`; `Order` contains many `Products`.

9. Data Types

Data types define the kind of data that attributes can store (e.g., integer, string, date).

- Example: In a banking system:
  - `AccountNumber` could be an integer.
  - `AccountHolderName` would be a string.
  - `DateOfTransaction` could be of date type.

10. Schema

A schema represents the structure of the database, including the tables, fields, and relationships between tables. It defines how data is organized in the system.

- Example: In a payroll system, the schema may consist of tables such as `Employee`, `Salary`, `Department`, each with its own attributes and relationships.

Example Scenario: Library Management System

Let’s build a data model for a Library Management System:

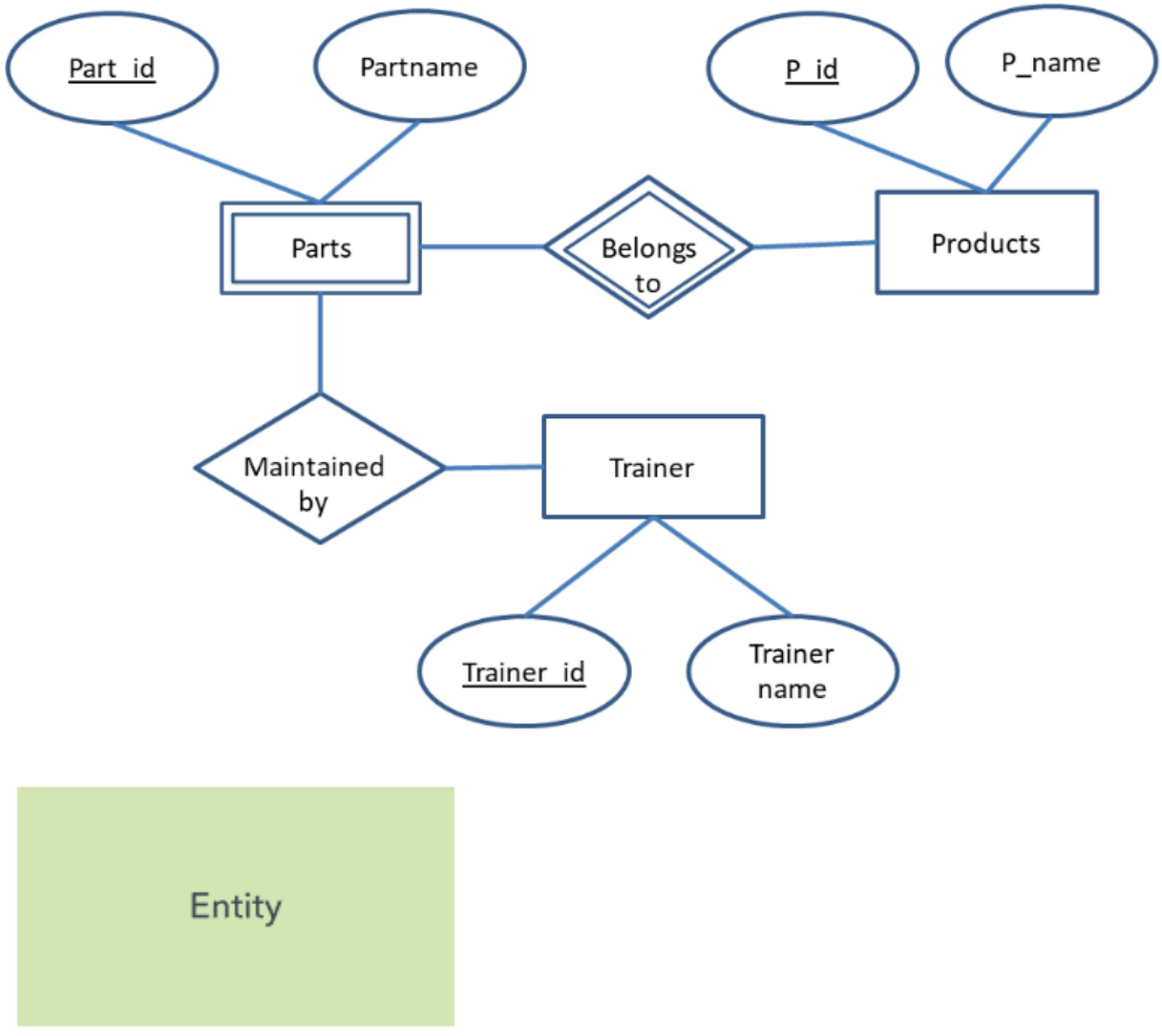
- Entities:
  - `Book` (attributes: `BookID`, `Title`, `Author`)
  - `Member` (attributes: `MemberID`, `Name`, `Email`)
  - `Loan` (attributes: `LoanID`, `IssueDate`, `ReturnDate`)

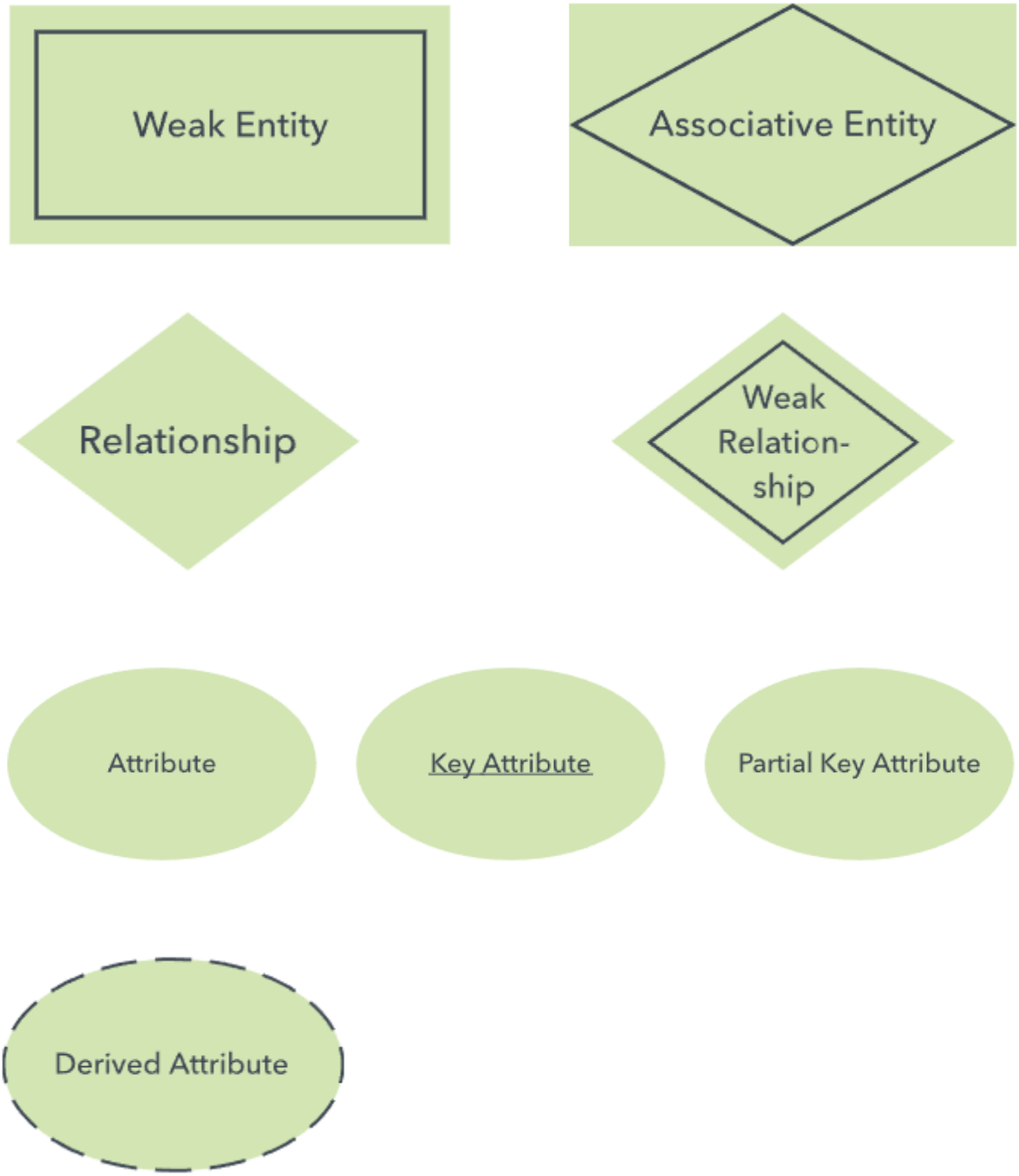
- **Relationships:**
    - A `Member` can borrow many `Books`, and a `Book` can be borrowed by many `Members` (many-to-many relationship), managed through the `Loan` entity.

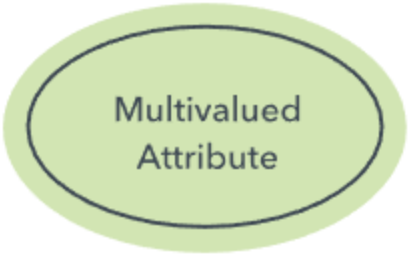
The ER diagram for this system would show:

- `Member` and `Book` entities with their respective attributes.
- A `Loan` entity acting as a bridge table connecting `Member` and `Book`.

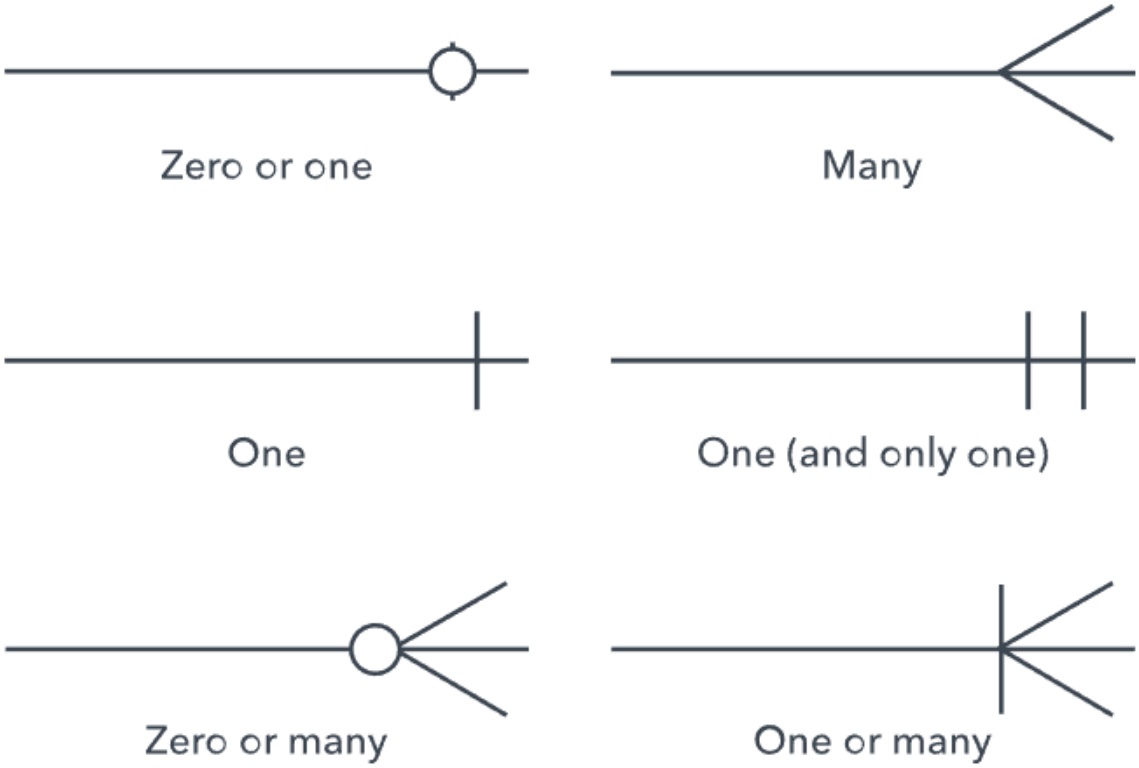
ER Diagram



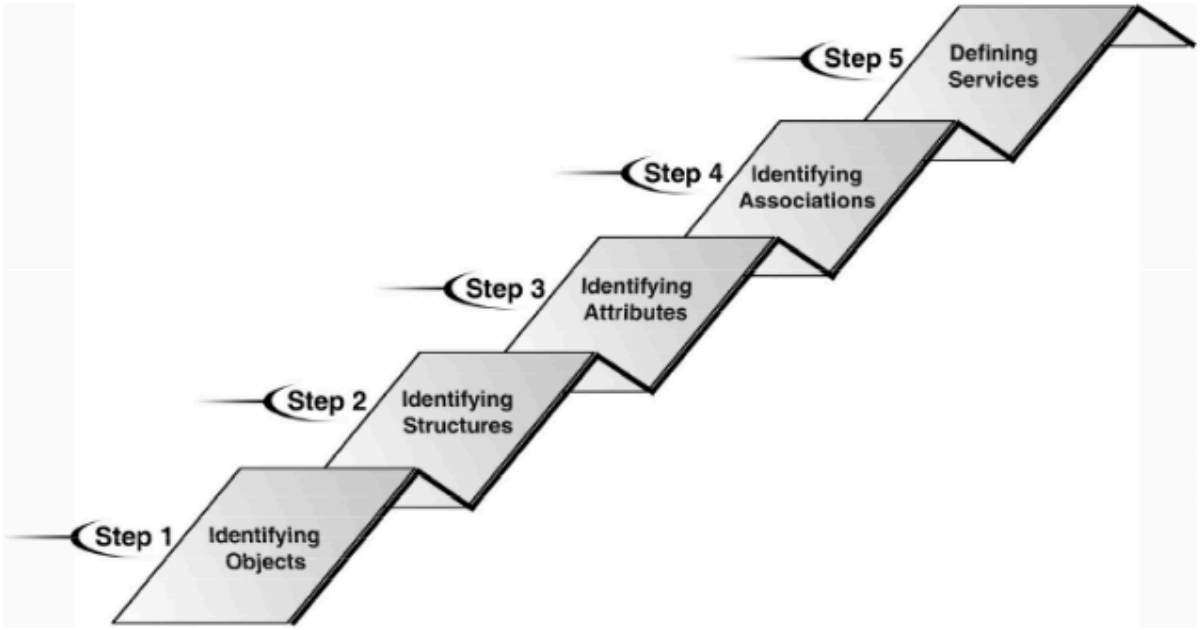

Weak Entity
Associative Entity
Relationship
Weak Relation-ship
Attribute
Key Attribute
Partial Key Attribute
Derived Attribute

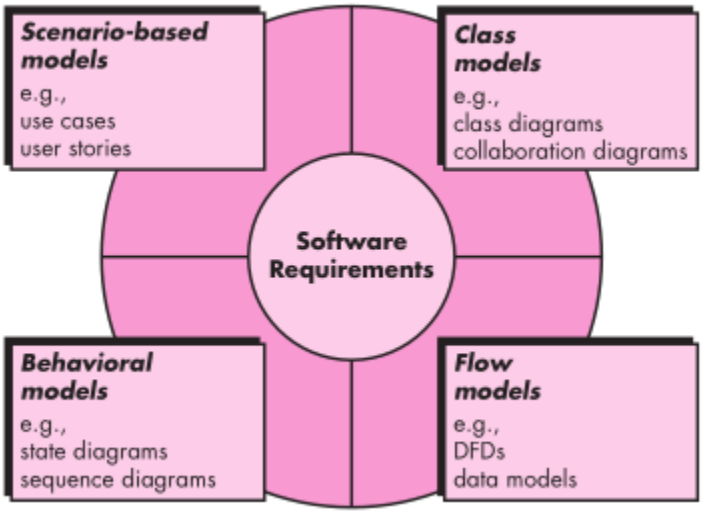


Cardinality



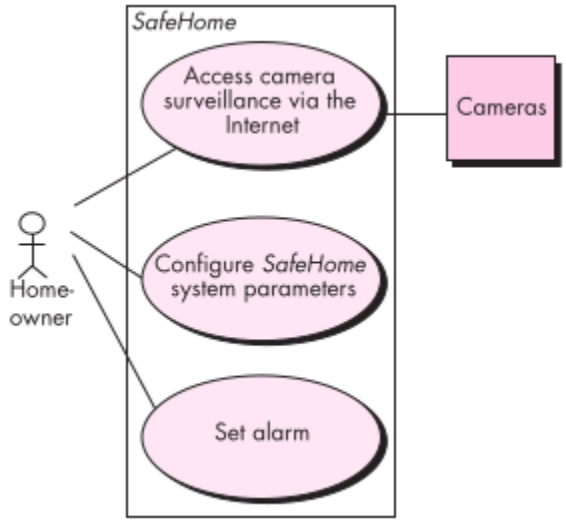
OOP analysis



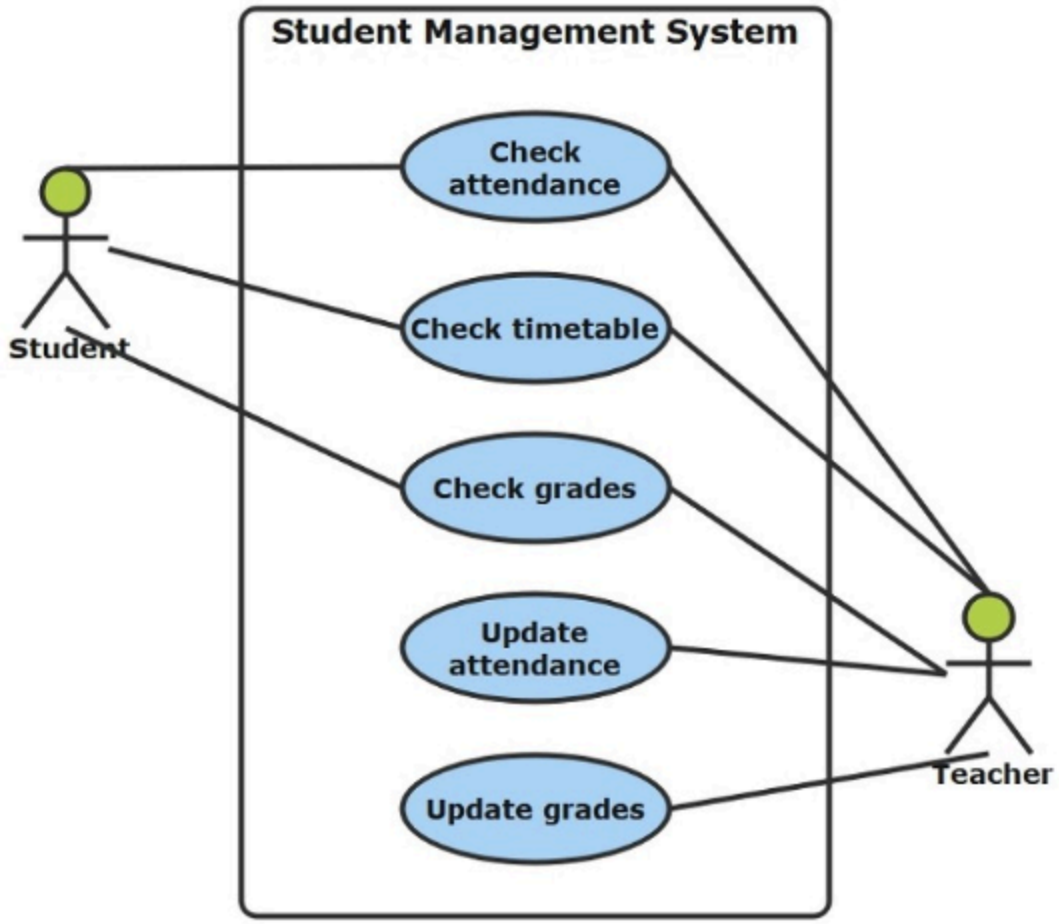


Use case diagrams

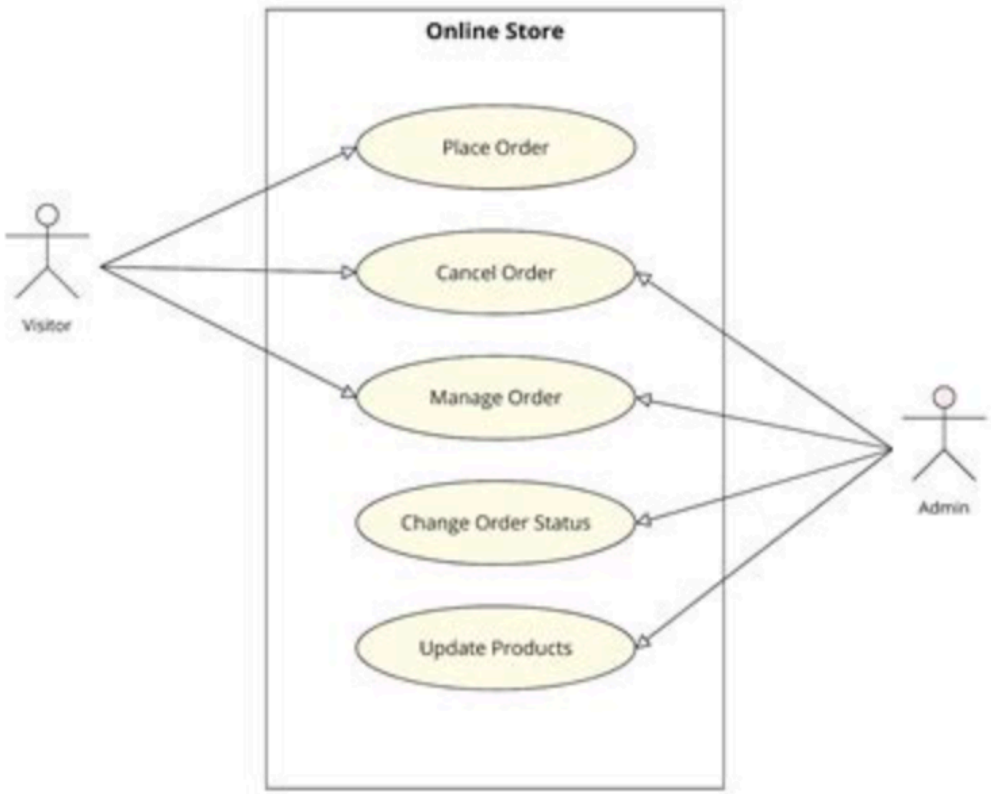
Safe home systems



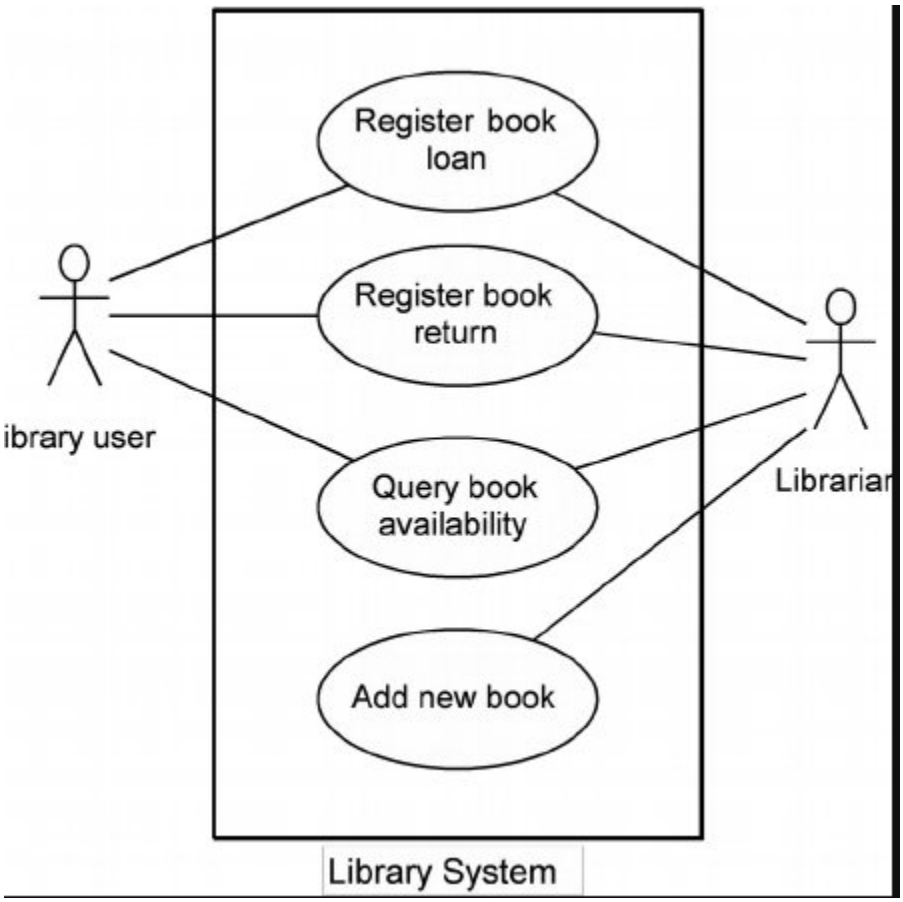
Student Management System

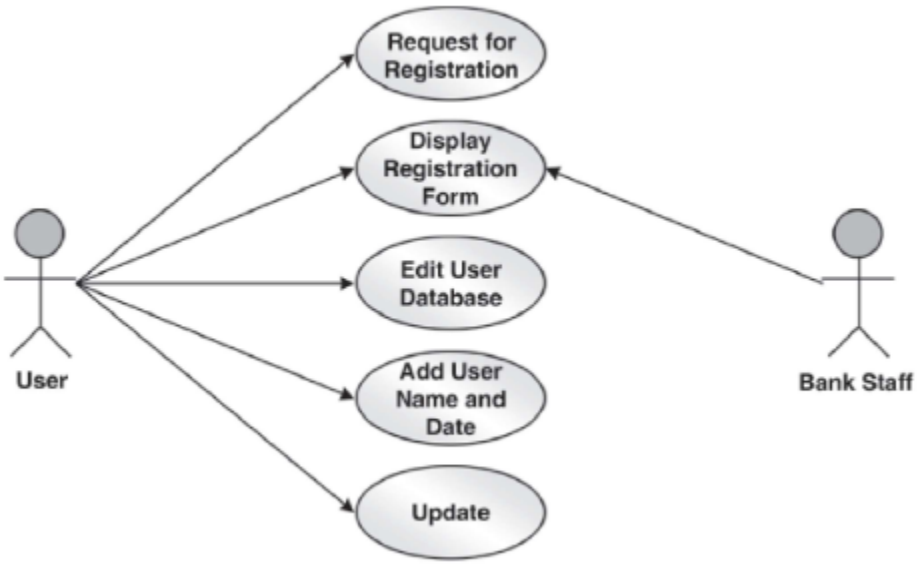

Student Management System
Check attendance
Check timetable
Check grades
Update attendance
Update grades
Student
Teacher

Online store

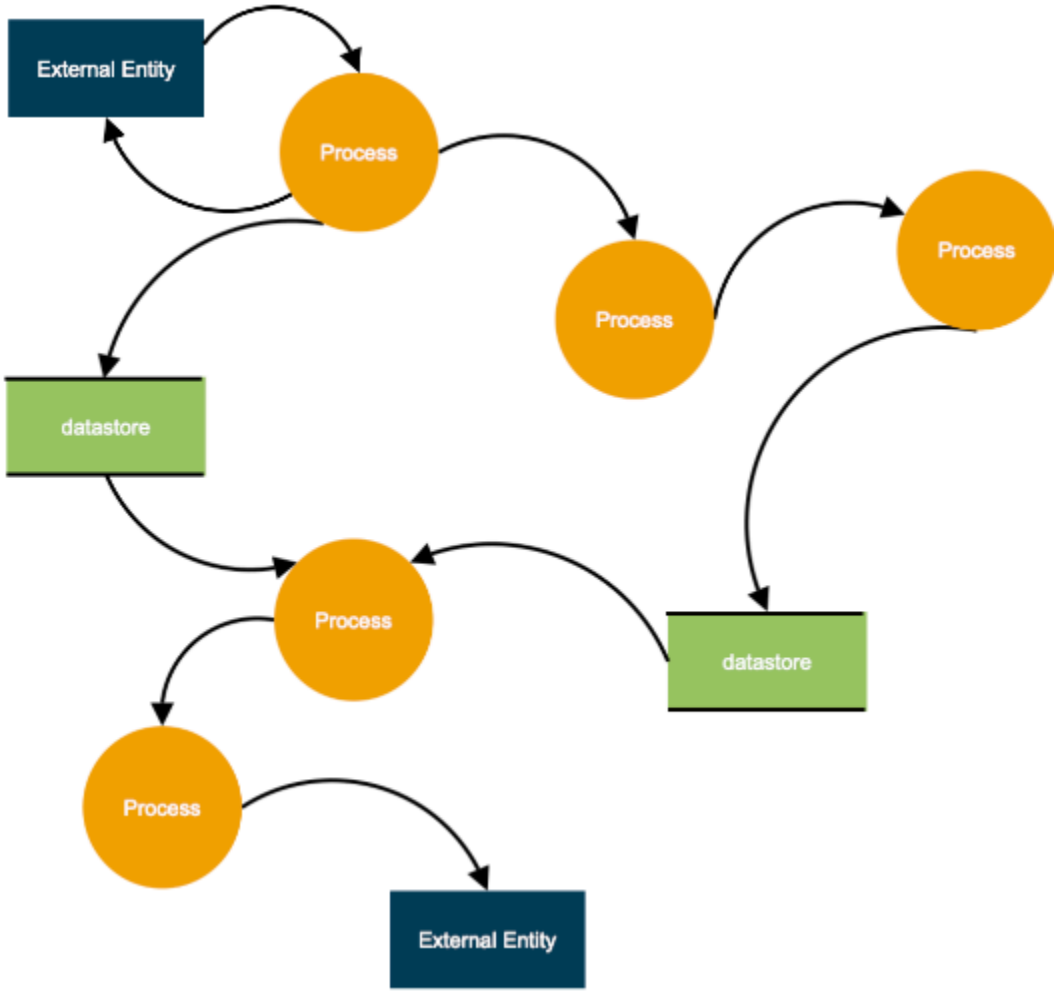


Library System





DFD Diagram


External Entity
Process
Process
Process
datastore
Process
datastore
Process
External Entity

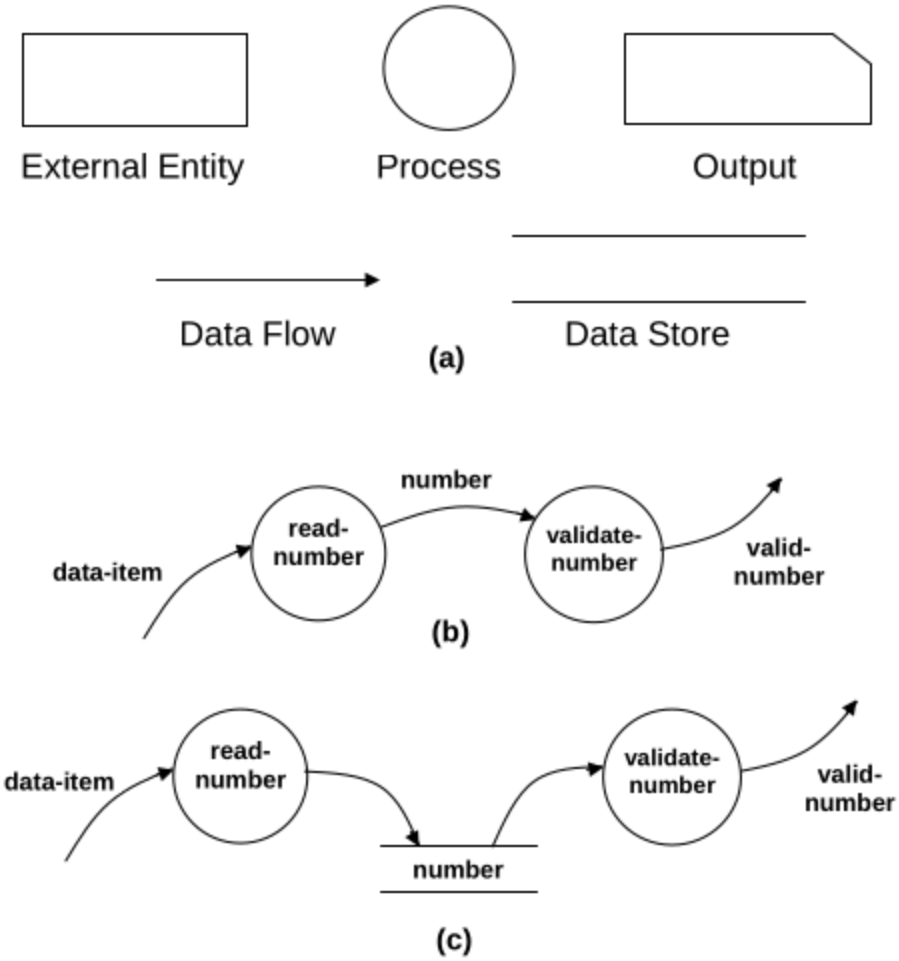


Fig. 5.1 (a) Symbols used for designing DFDs
(b), (c) Synchronous and asynchronous data flow

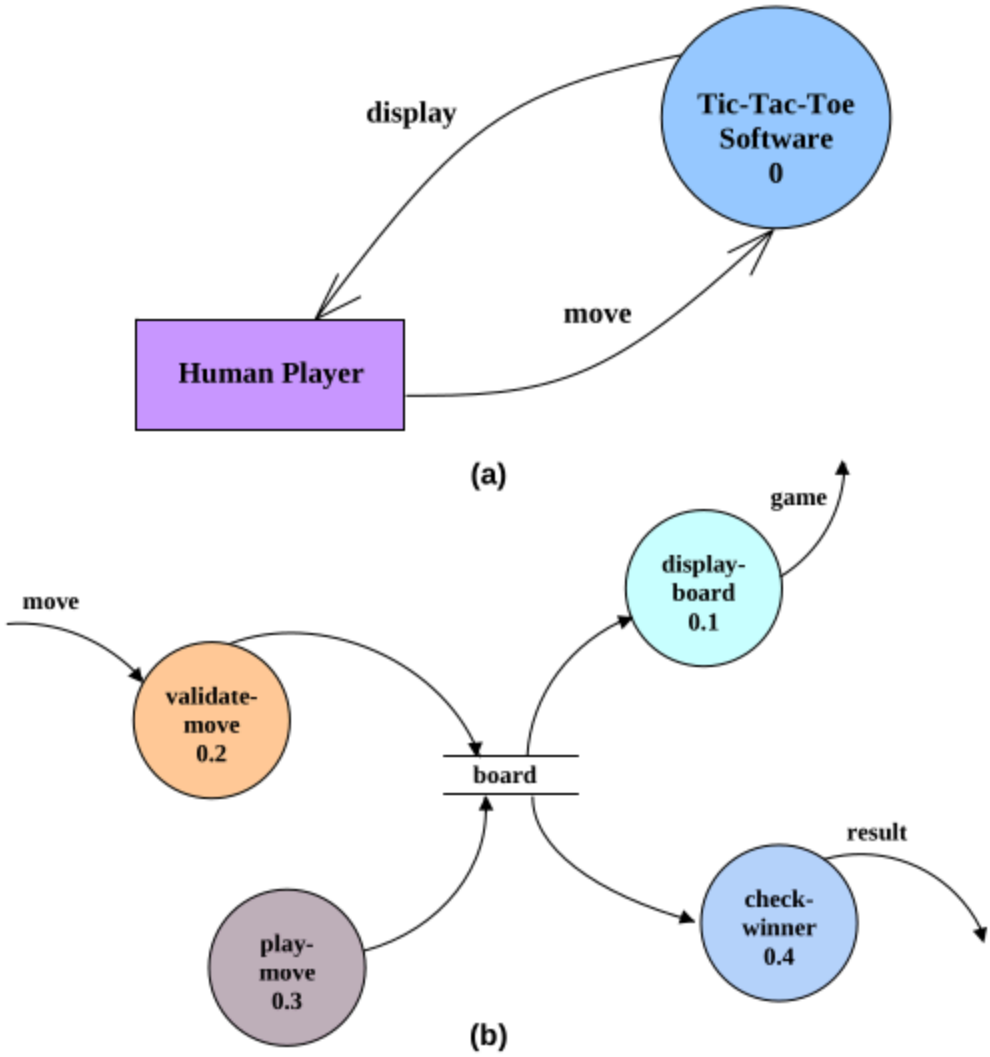
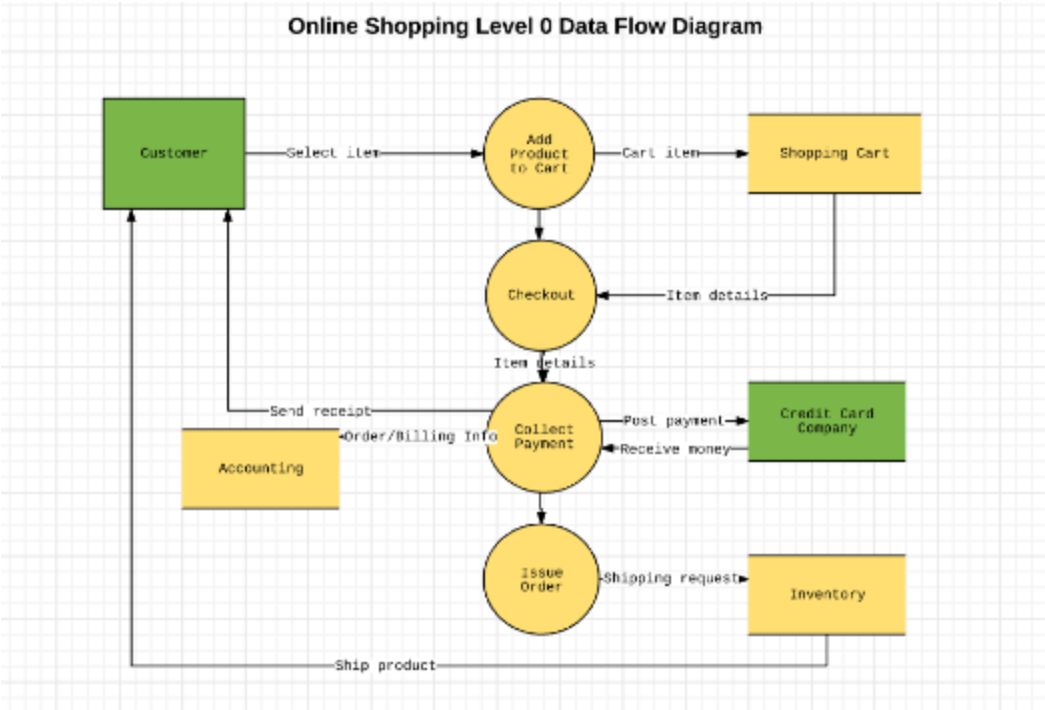


Fig 5.2 (a), (b) Level 0 and Level 1 DFD for Tic-Tac-Toe game described in Example 1



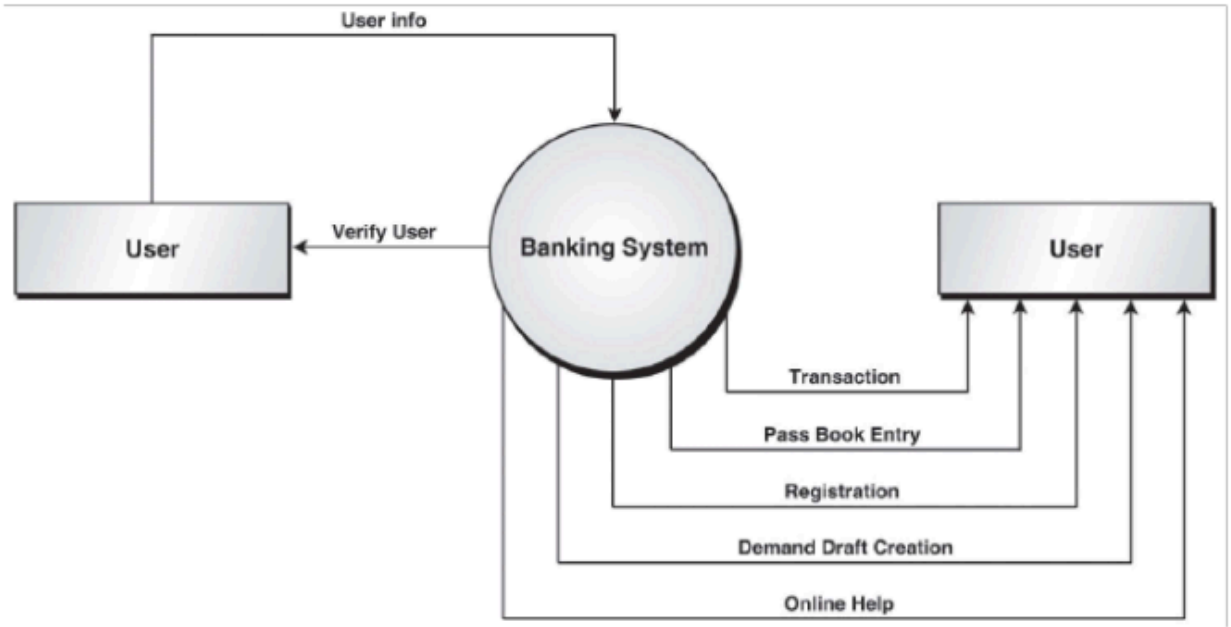


***Fig. 6.2** Level 0 DFD of Banking System*

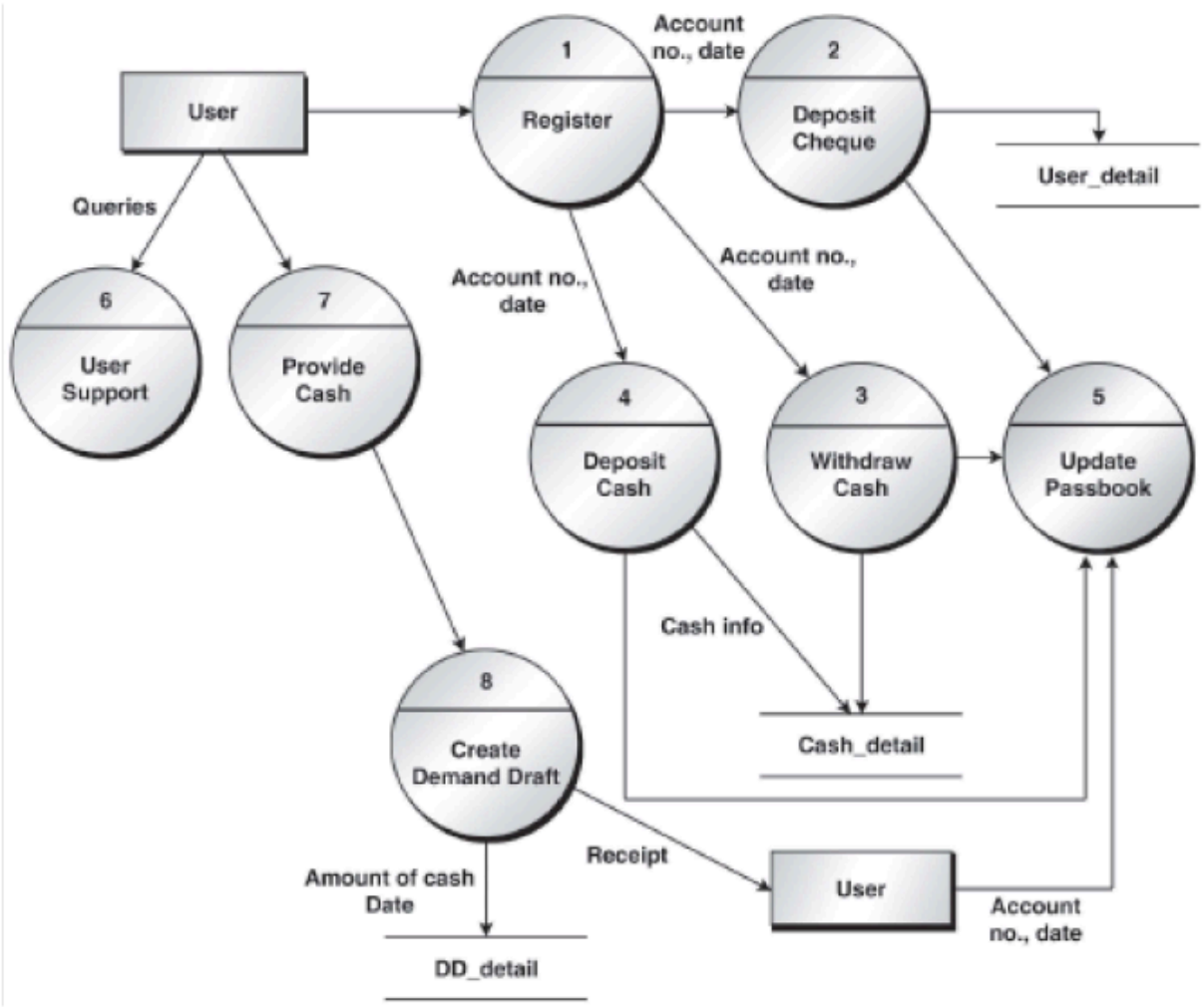


Fig. 6.3 *Level 1 DFD to Perform Transaction*

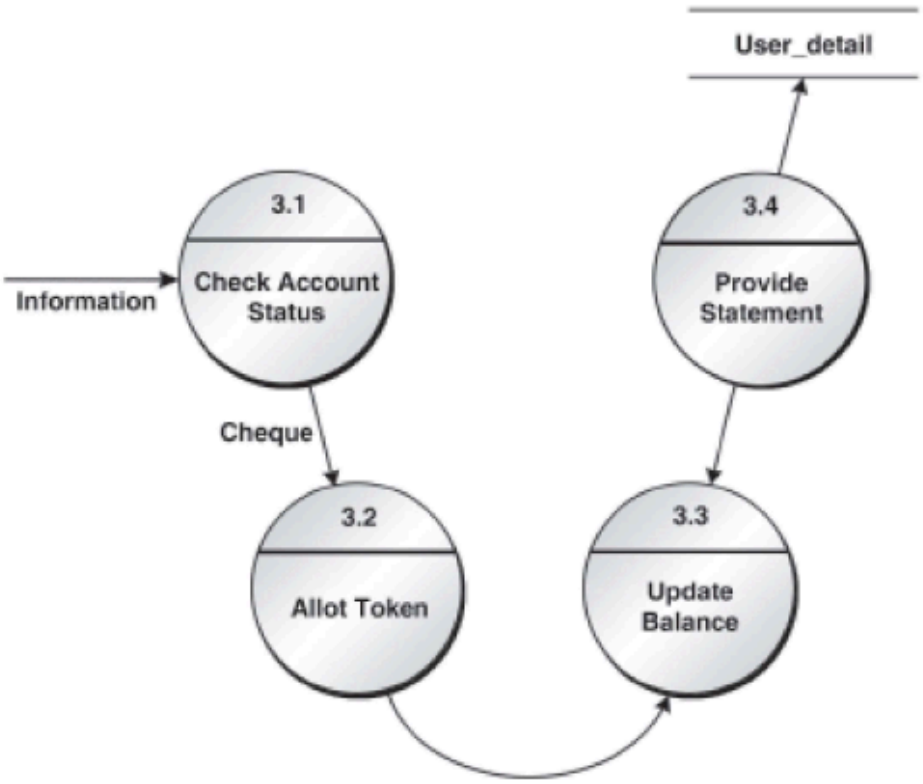


Fig. 6.4 *Level 2 DFD to Withdraw Cash*

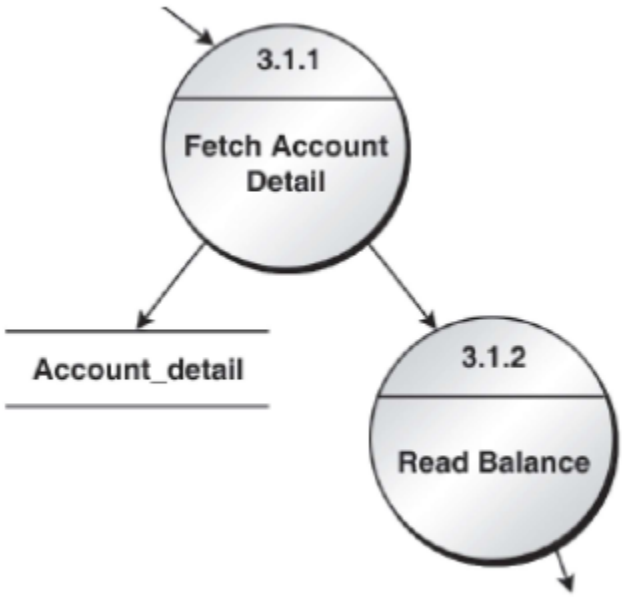
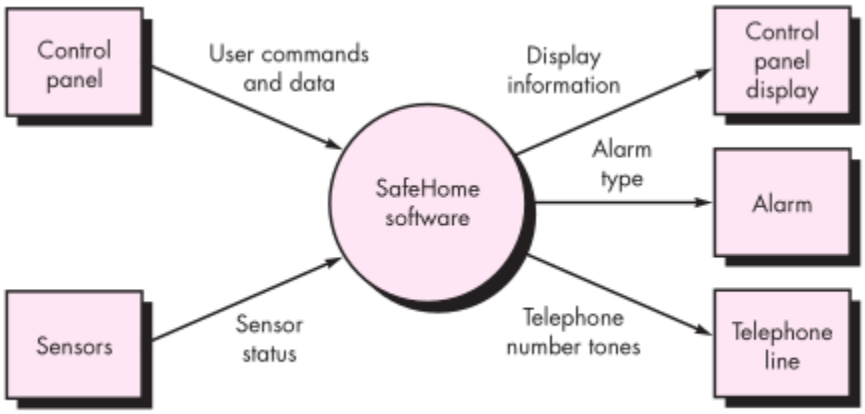
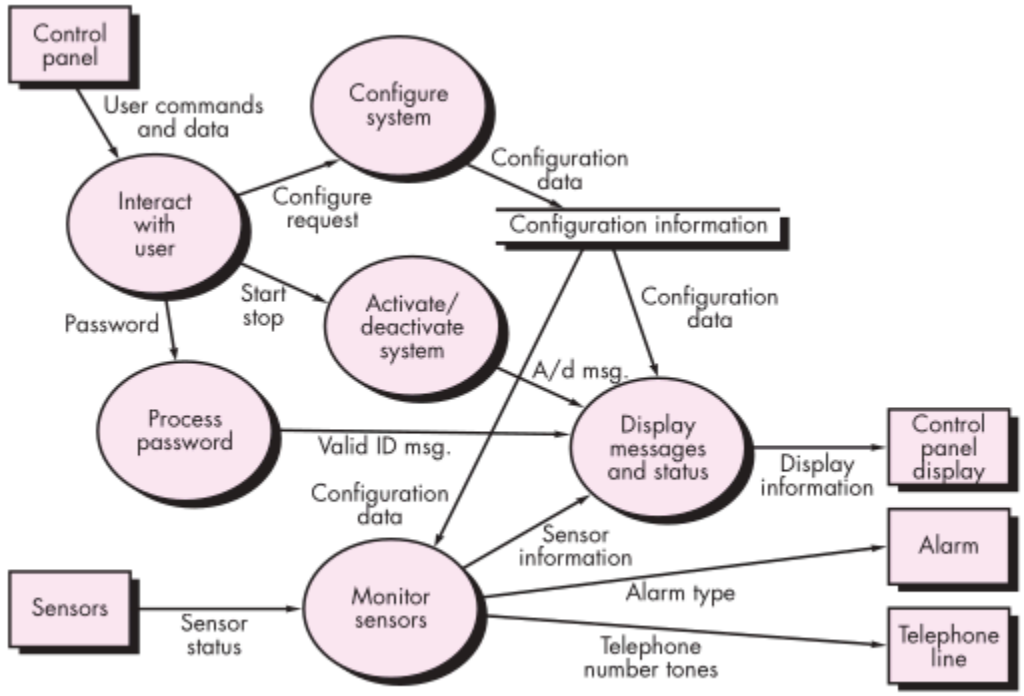


Fig. 6.5 *Level 3 DFD to Check Account Status*

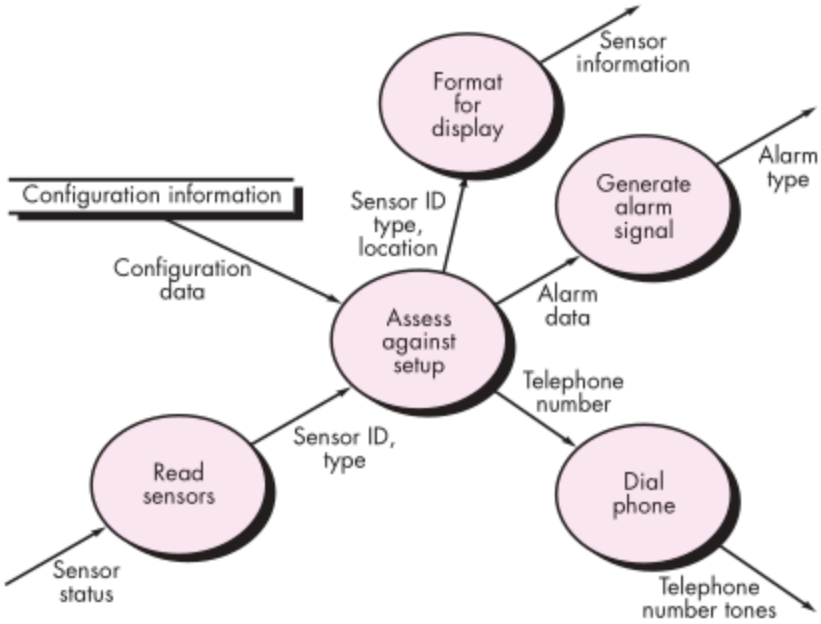
DFD diagram for safehome security function



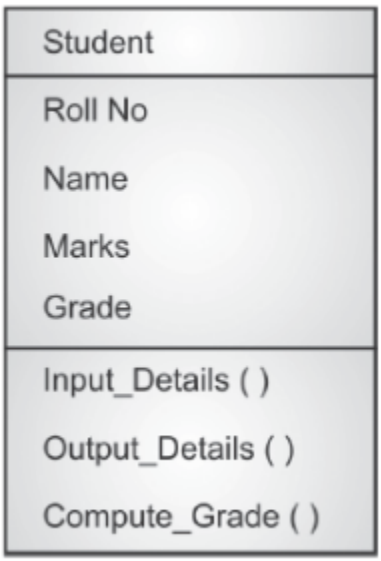
Level1 DFD - safehome

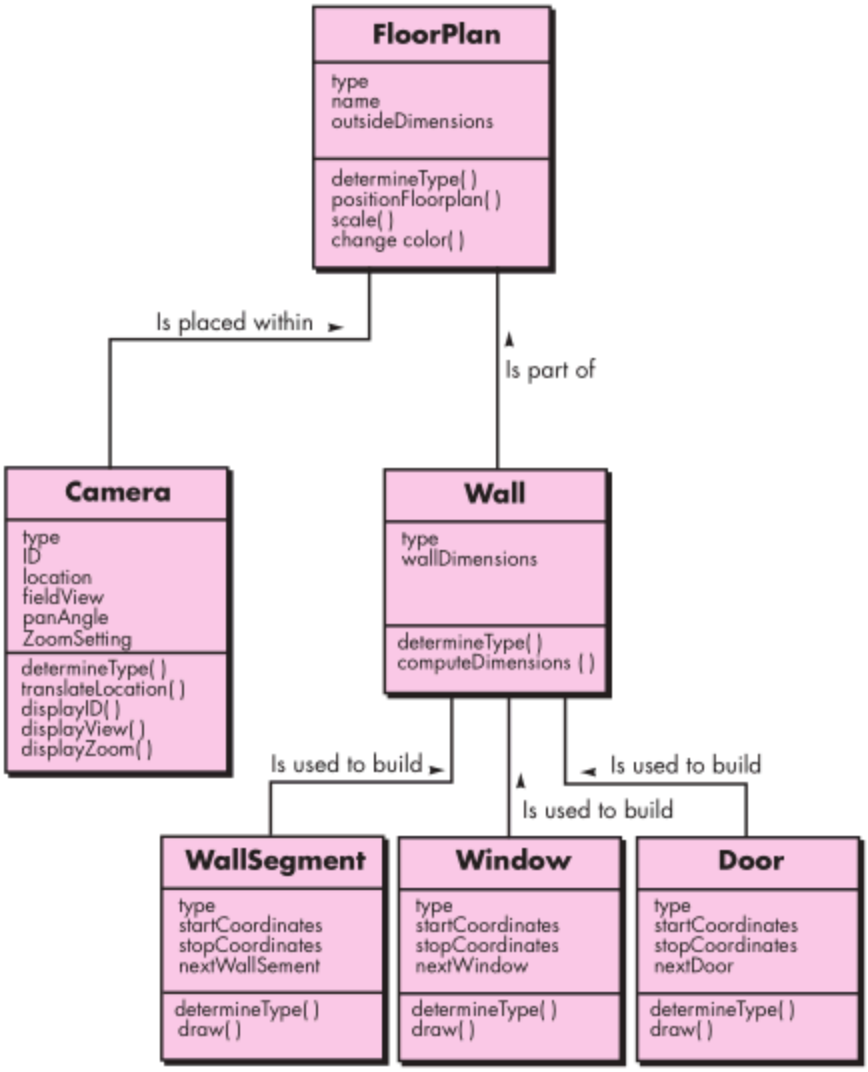


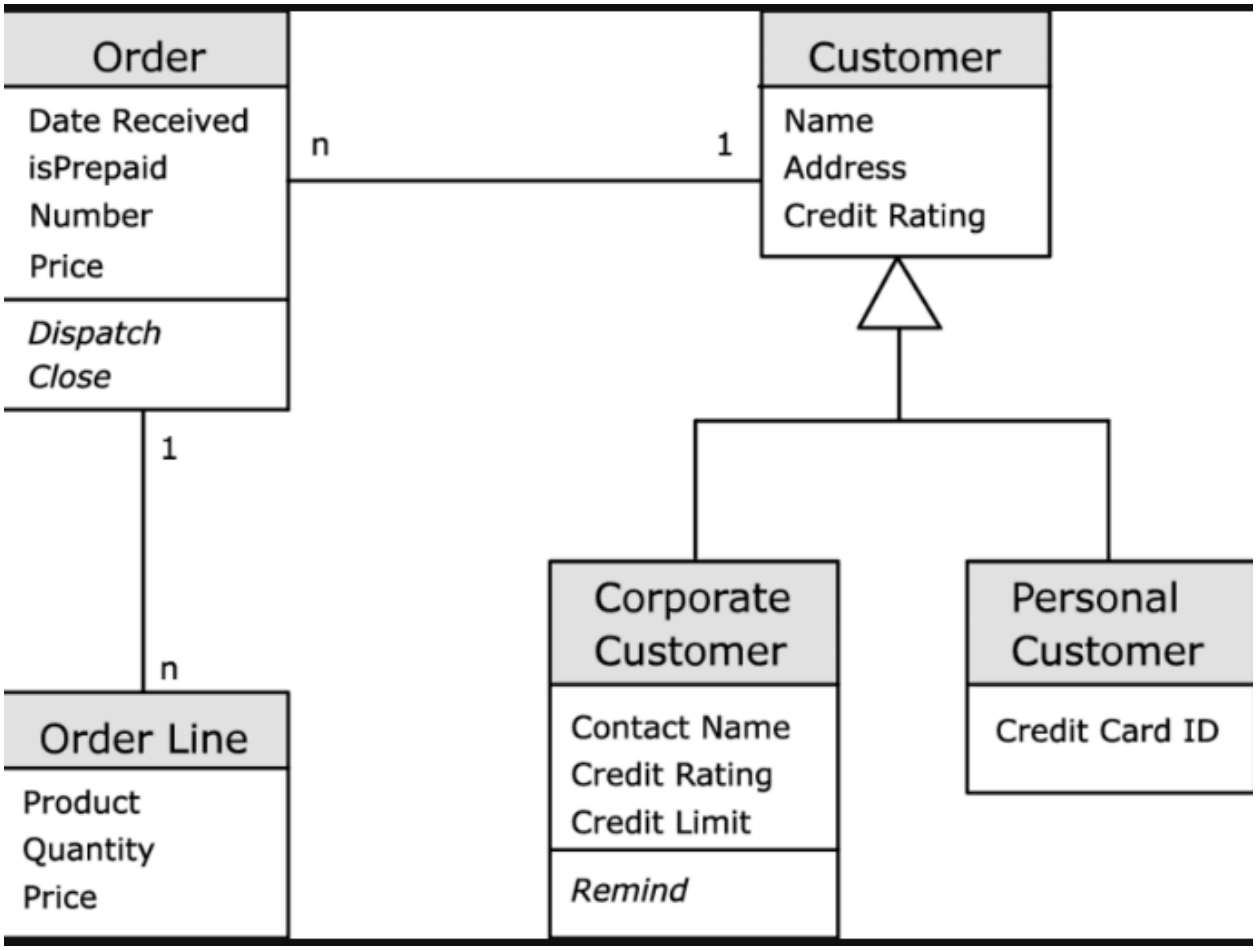
Level2 DFD - monitor sensors process



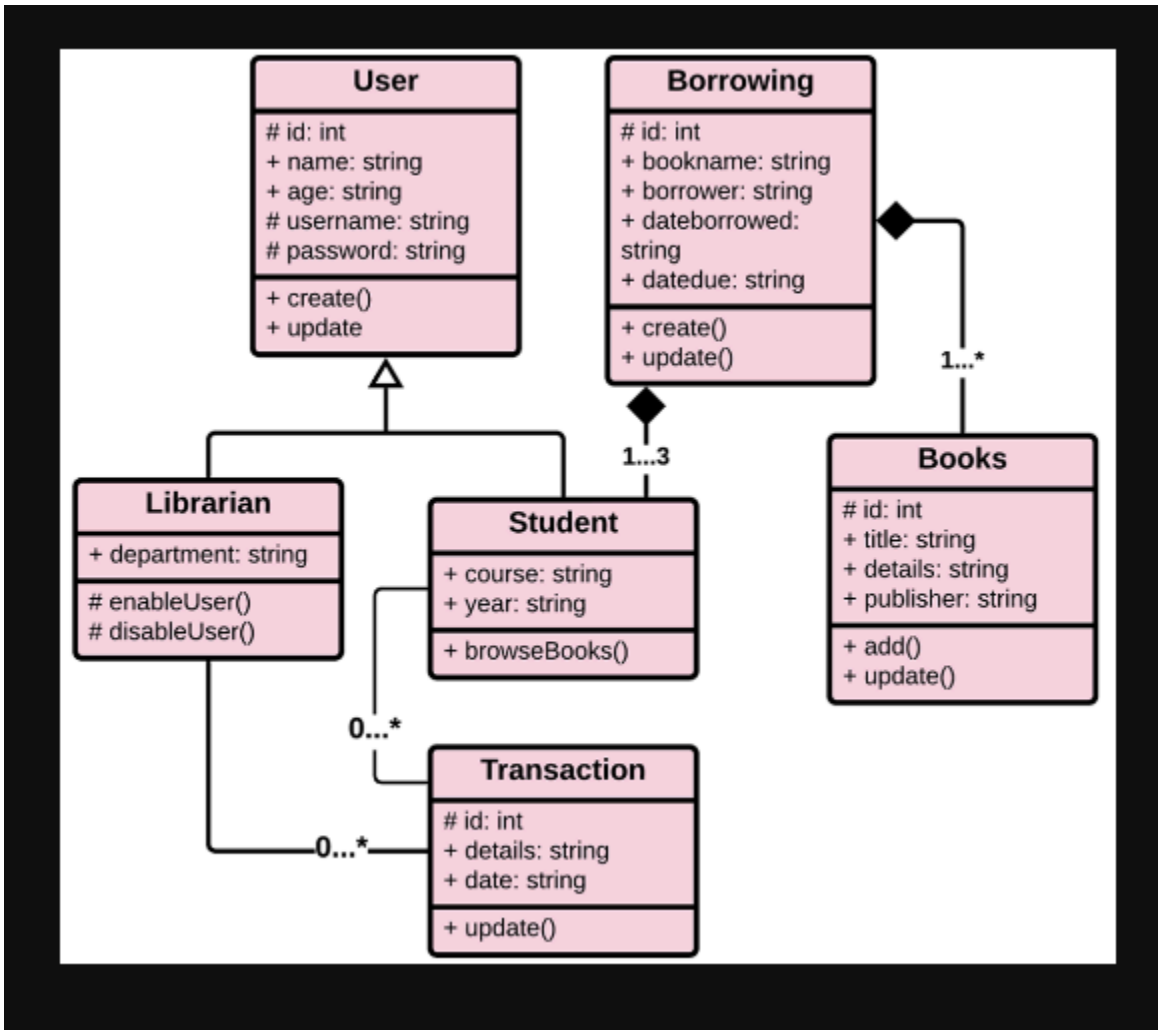
Class diagram

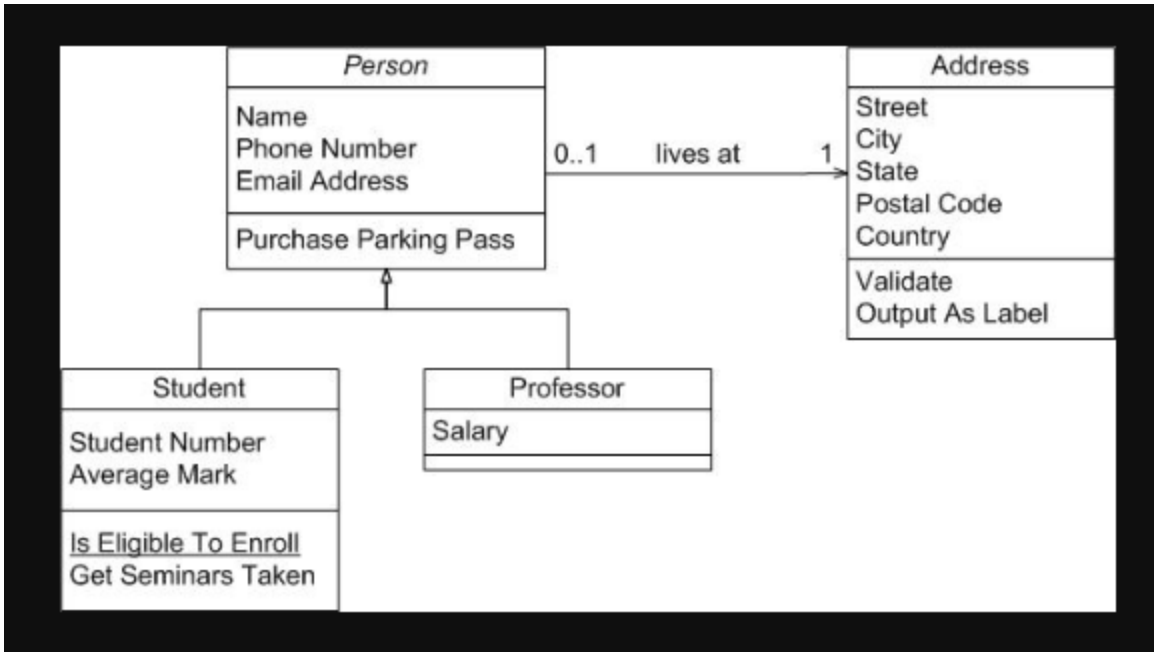



FloorPlan
type
name
outsideDimensions
determineType( )
positionFloorplan( )
scale( )
change color( )
Is placed within
Is part of
Camera
type
ID
location
fieldView
panAngle
ZoomSetting
determineType( )
translateLocation( )
displayID( )
displayView( )
displayZoom( )
Wall
type
wallDimensions
determineType( )
computeDimensions ( )
Is used to build
Is used to build
Is used to build
WallSegment
type
startCoordinates
stopCoordinates
nextWallSement
determineType( )
draw( )
Window
type
startCoordinates
stopCoordinates
nextWindow
determineType( )
draw( )
Door
type
startCoordinates
stopCoordinates
nextDoor
determineType( )
draw( )


Order
Date Received
isPrepaid
Number
Price
Dispatch
Close
n
1
Customer
Name
Address
Credit Rating
1
n
Order Line
Product
Quantity
Price
Corporate Customer
Contact Name
Credit Rating
Credit Limit
Remind
Personal Customer
Credit Card ID

Library Management System





Behavioral Modeling

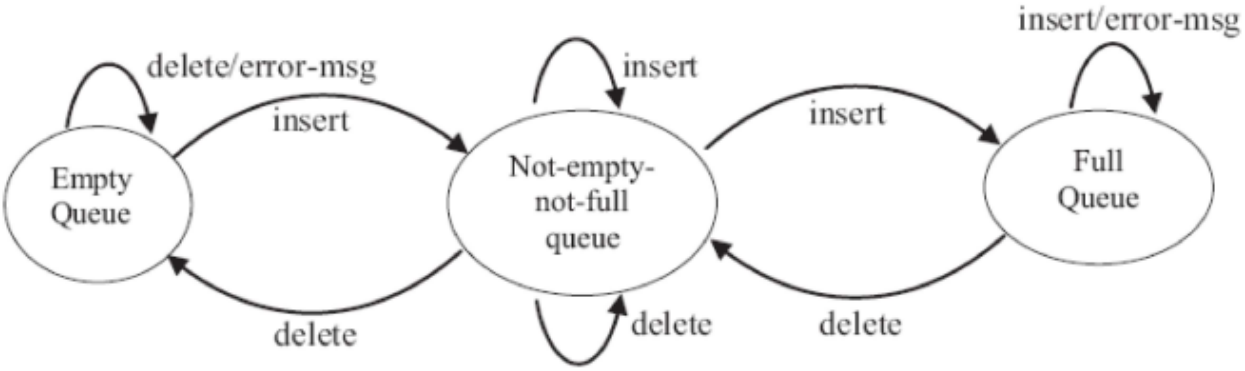
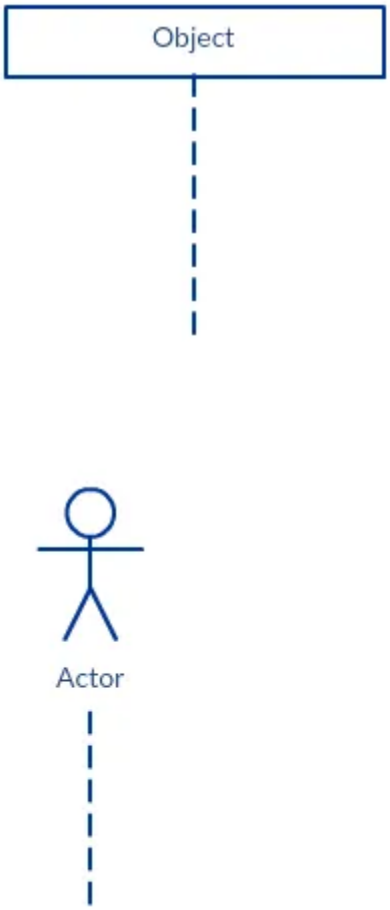
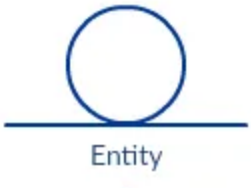


***Fig. 6.10** State Transition Diagram for Queue Object*

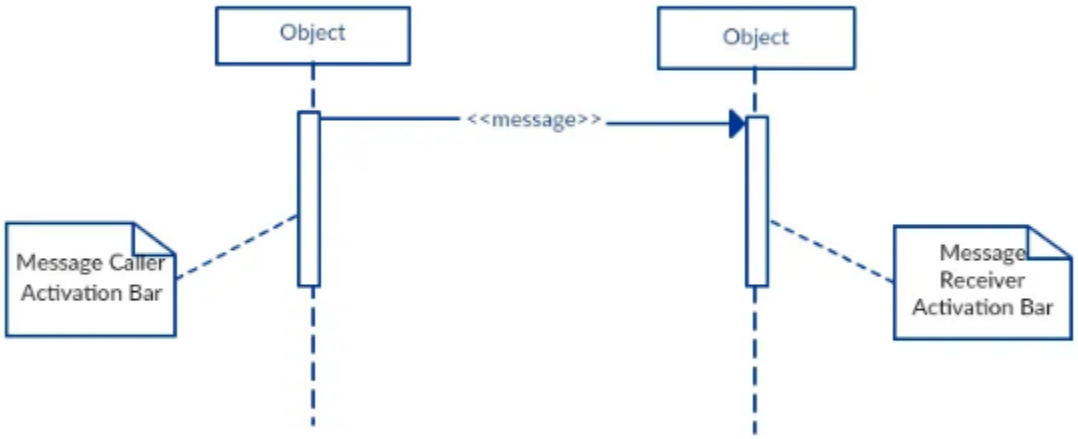
Sequence Diagrams


Object
Actor

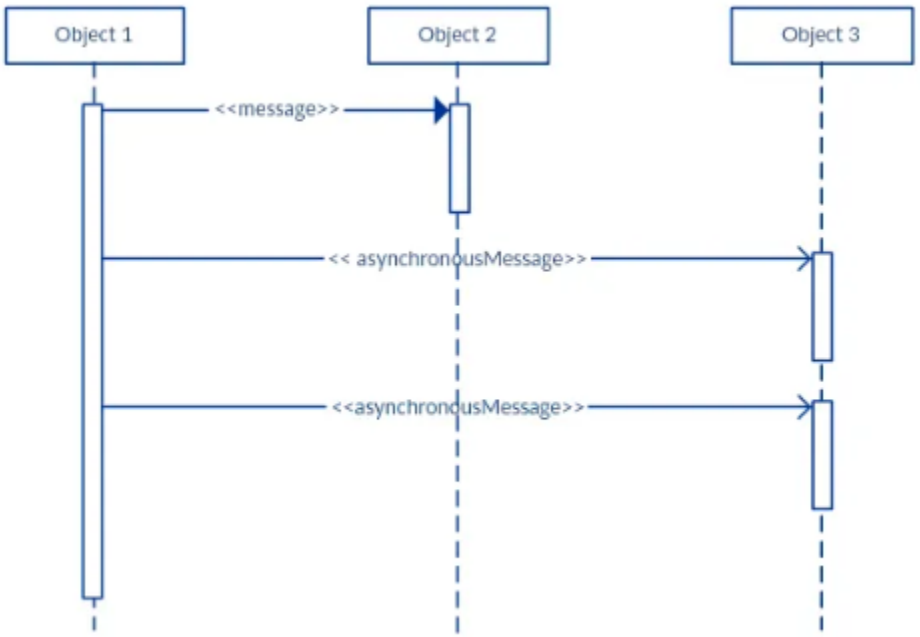

Entity


Boundary

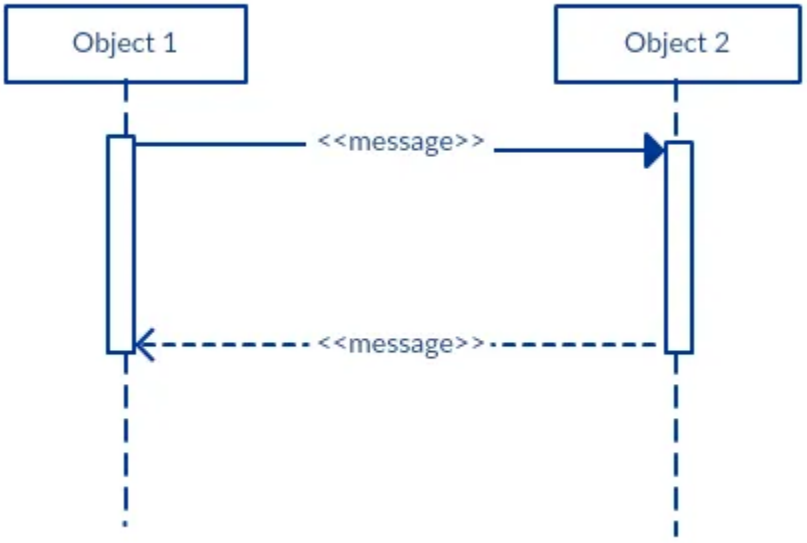

Control


Object
Object
<<message>>
Message Caller
Activation Bar
Message
Receiver
Activation Bar

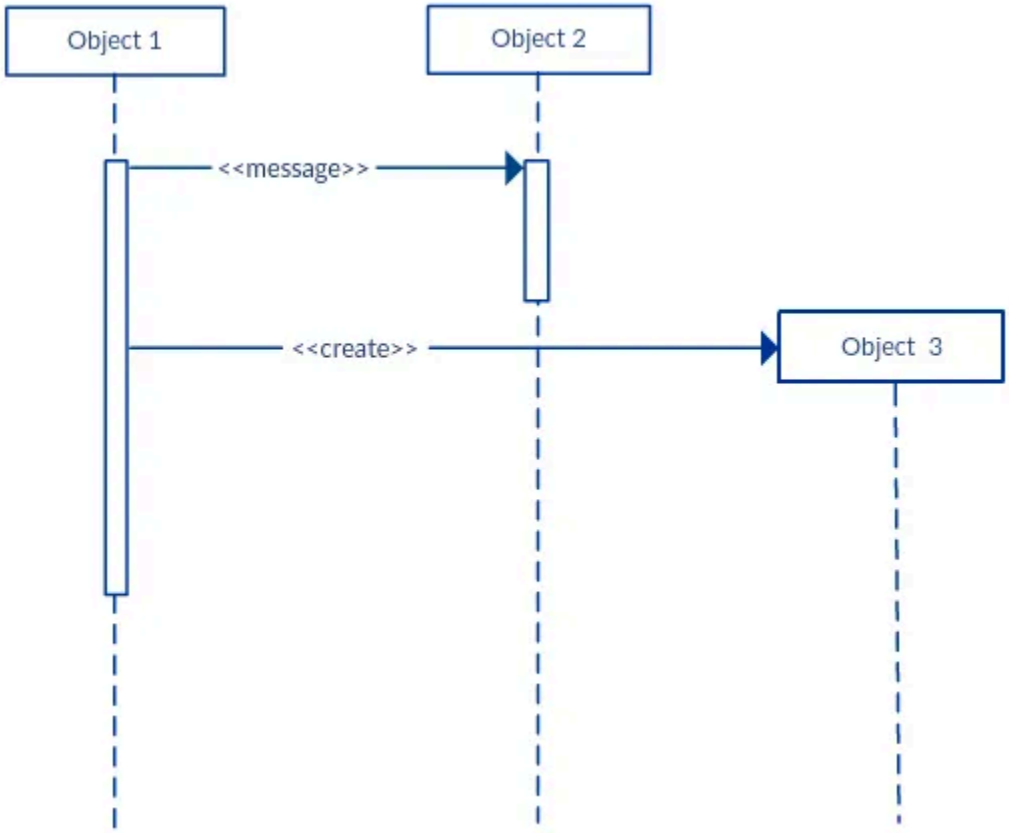
Message


Object 1
Object 2
Object 3
<<message>>
<< asynchronousMessage>>
<<asynchronousMessage>>

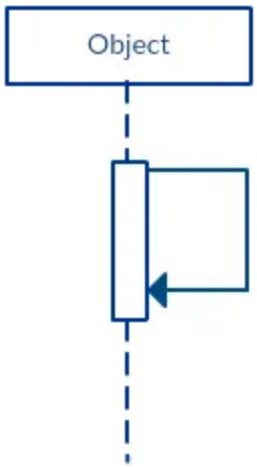
Return Message



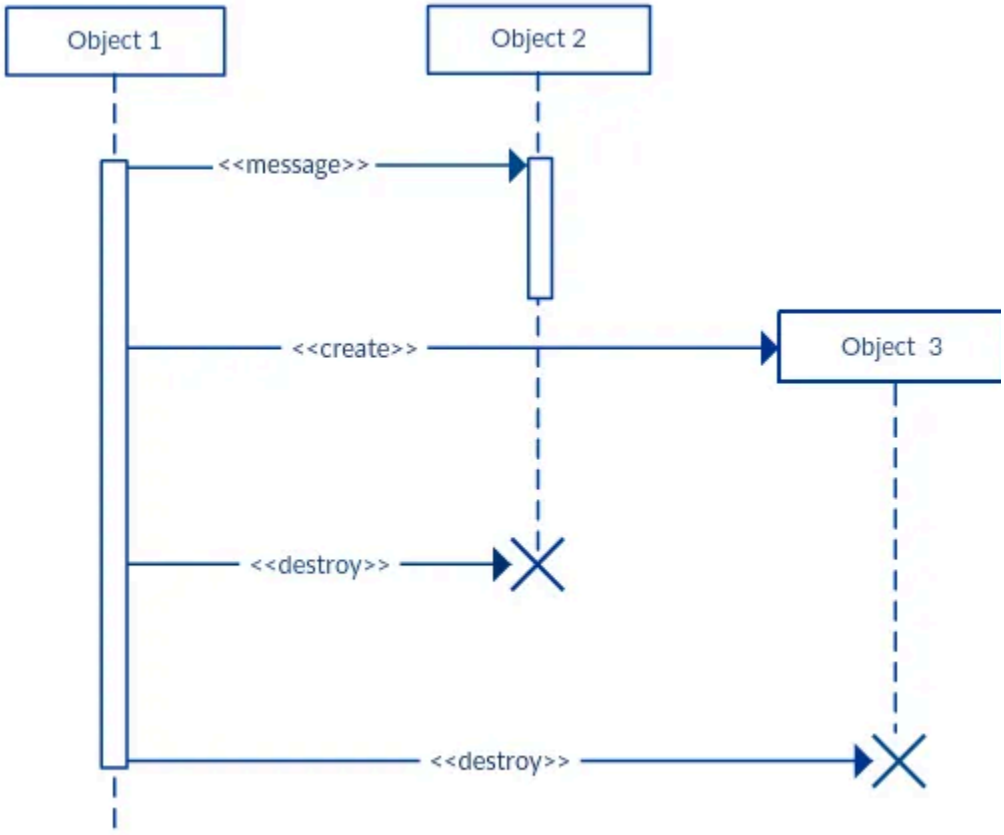
Create message



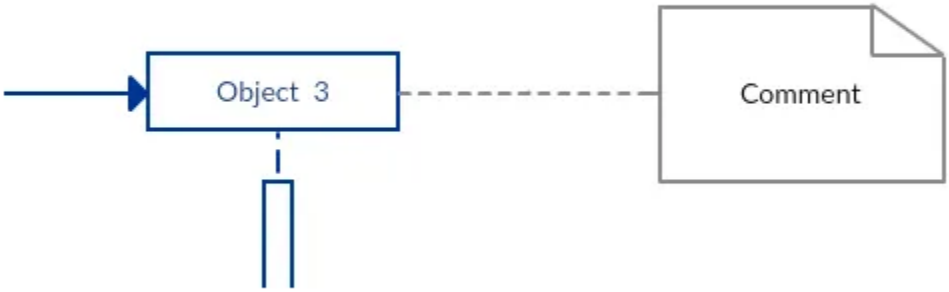
Message Destruction

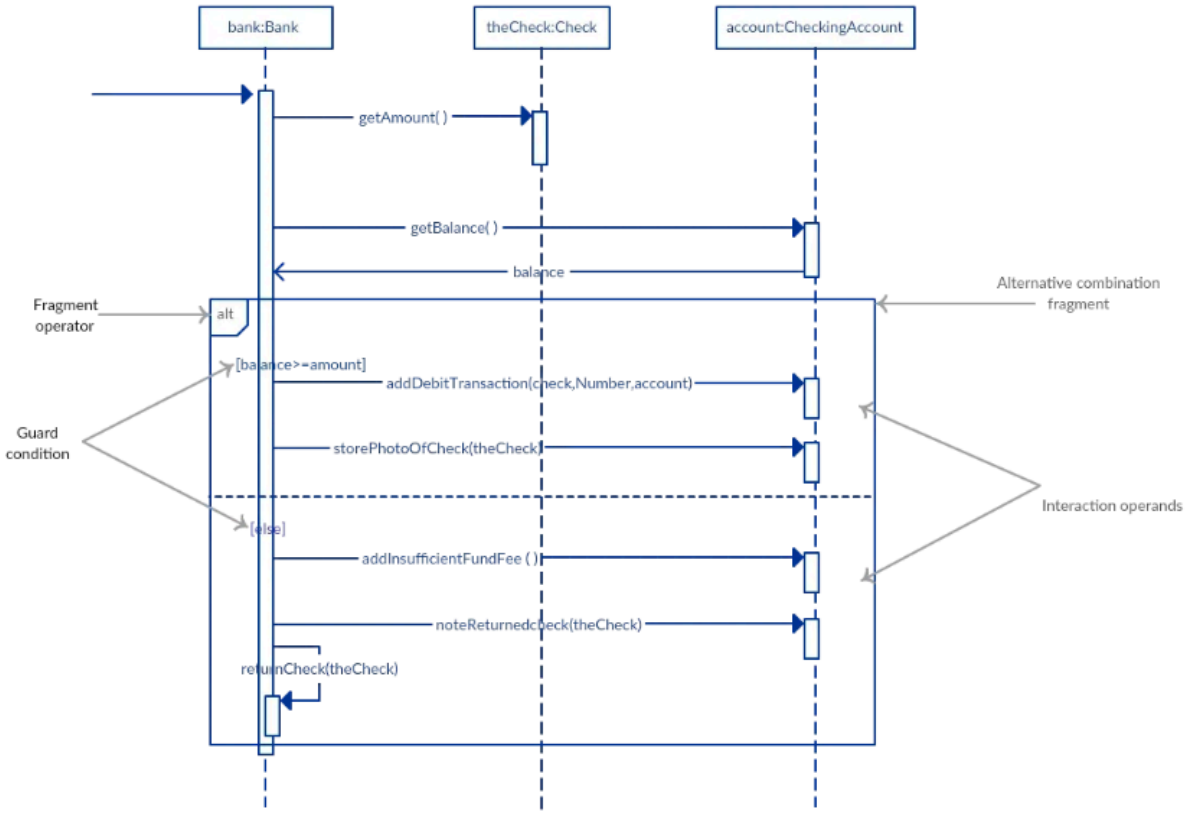


Reflexive message : when object sends message to itself

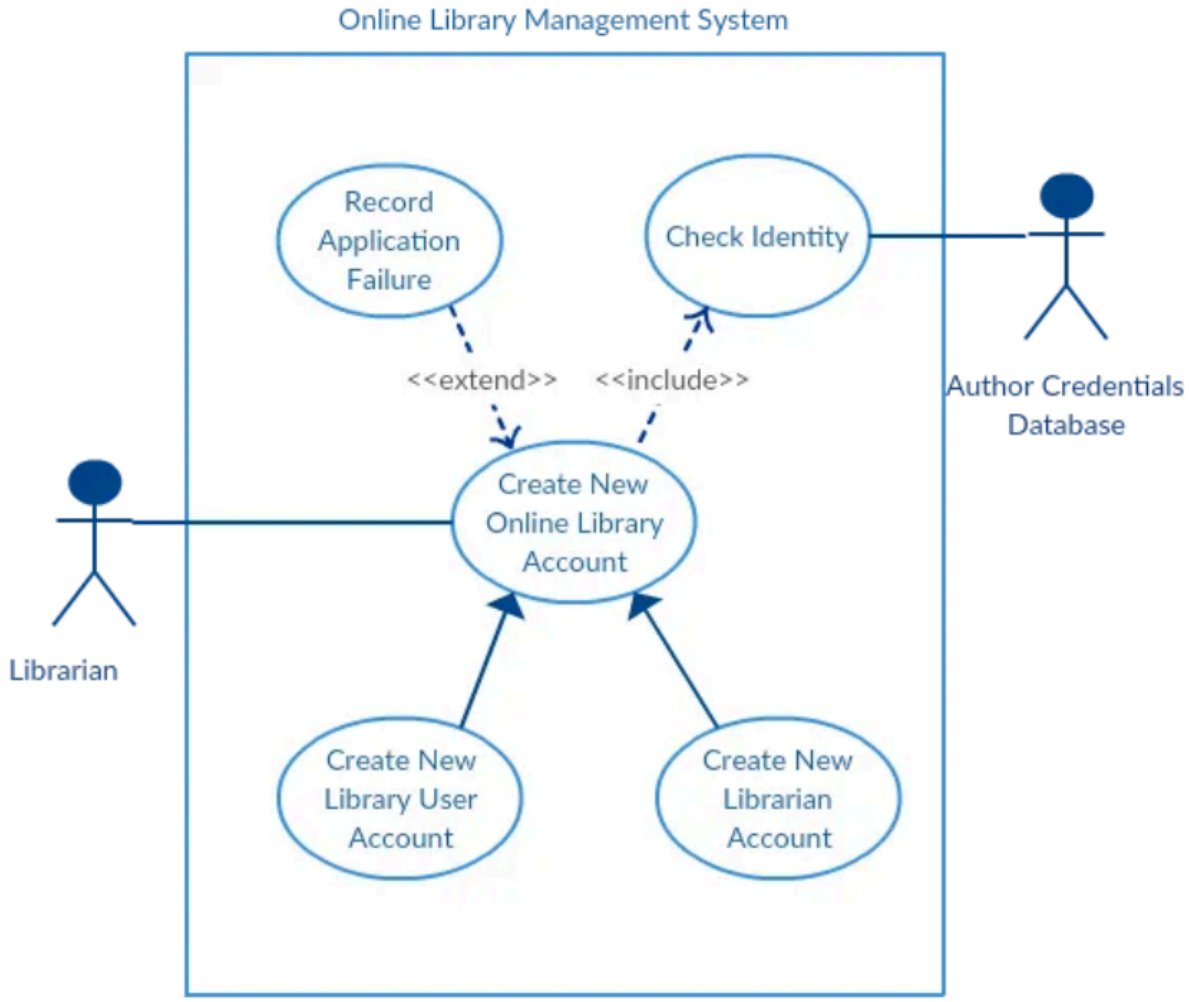


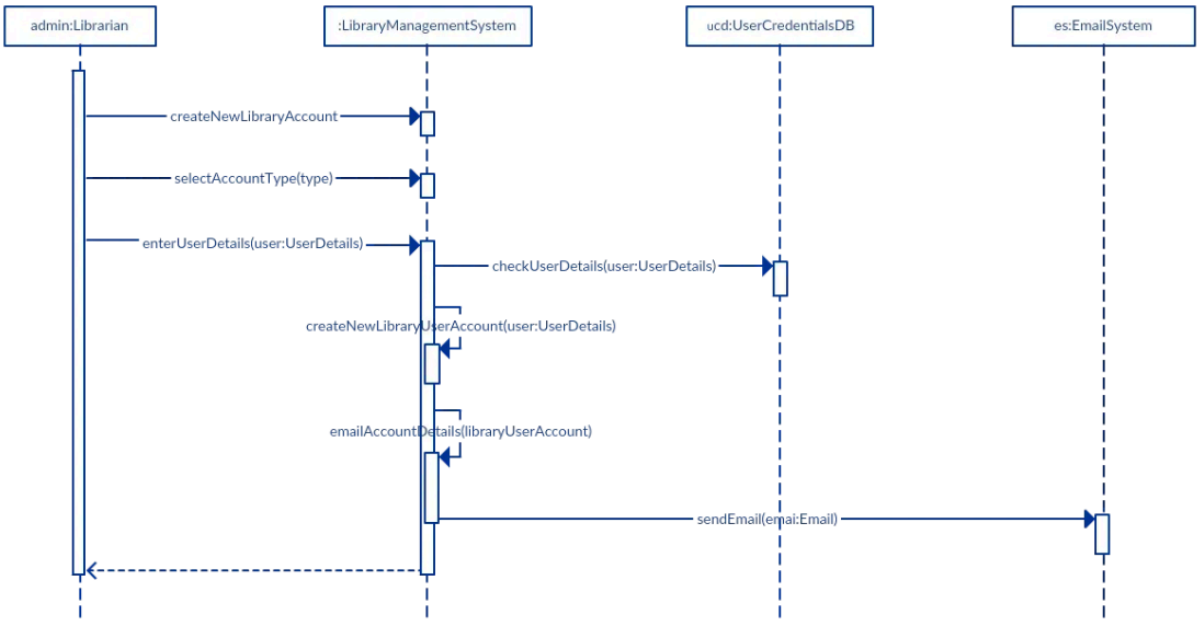
Comments


Object 3
Comment

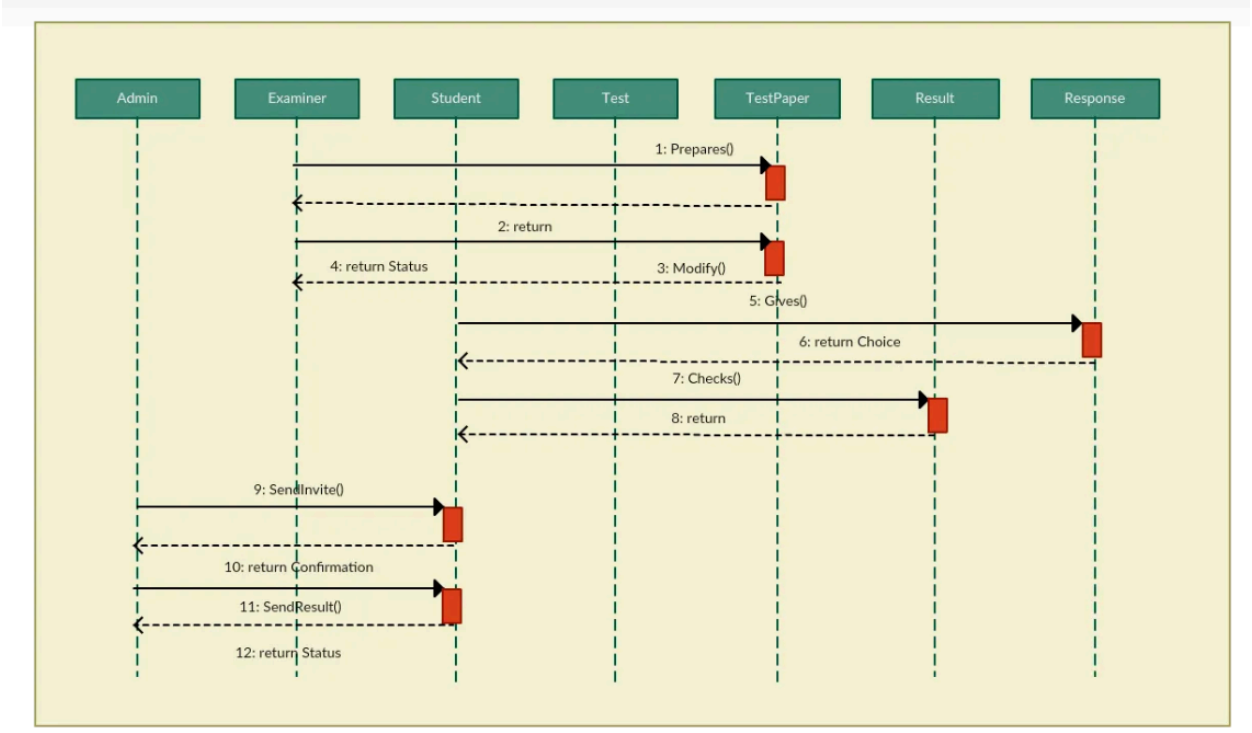

bank:Bank
theCheck:Check
account:CheckingAccount
getAmount( )
getBalance( )
balance
Alternative combination fragment
Fragment operator
alt
[balance>=amount]
addDebitTransaction(check,Number,account)
Guard condition
storePhotoOfCheck(theCheck)
Interaction operands
[else]
addInsufficientFundFee ( )
noteReturnedCheck(theCheck)
returnCheck(theCheck)

Creating a Sequence diagram from use case diagram

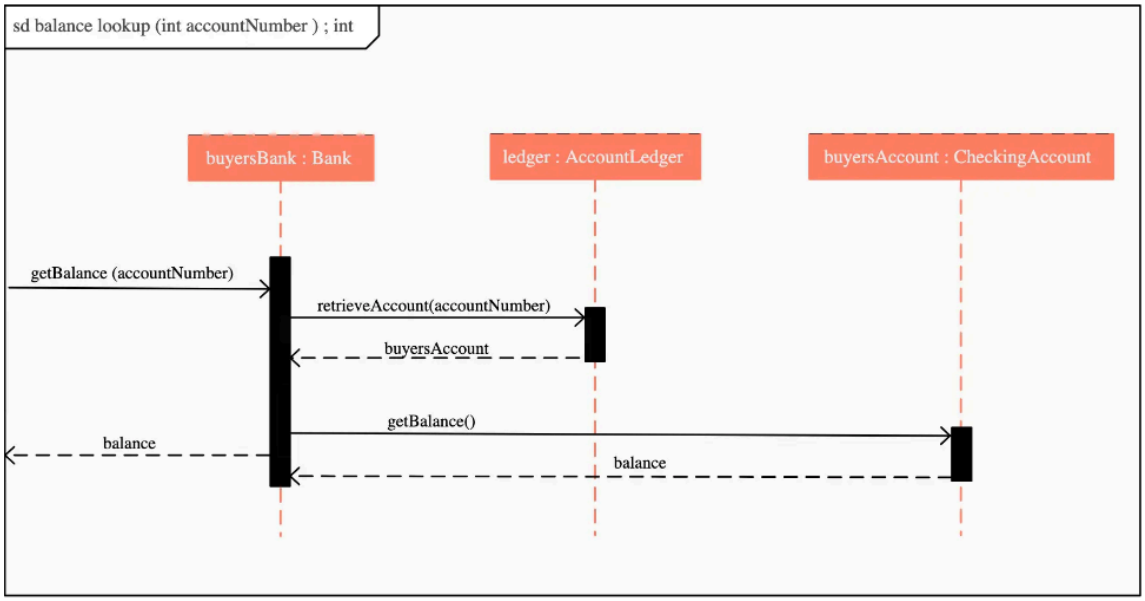




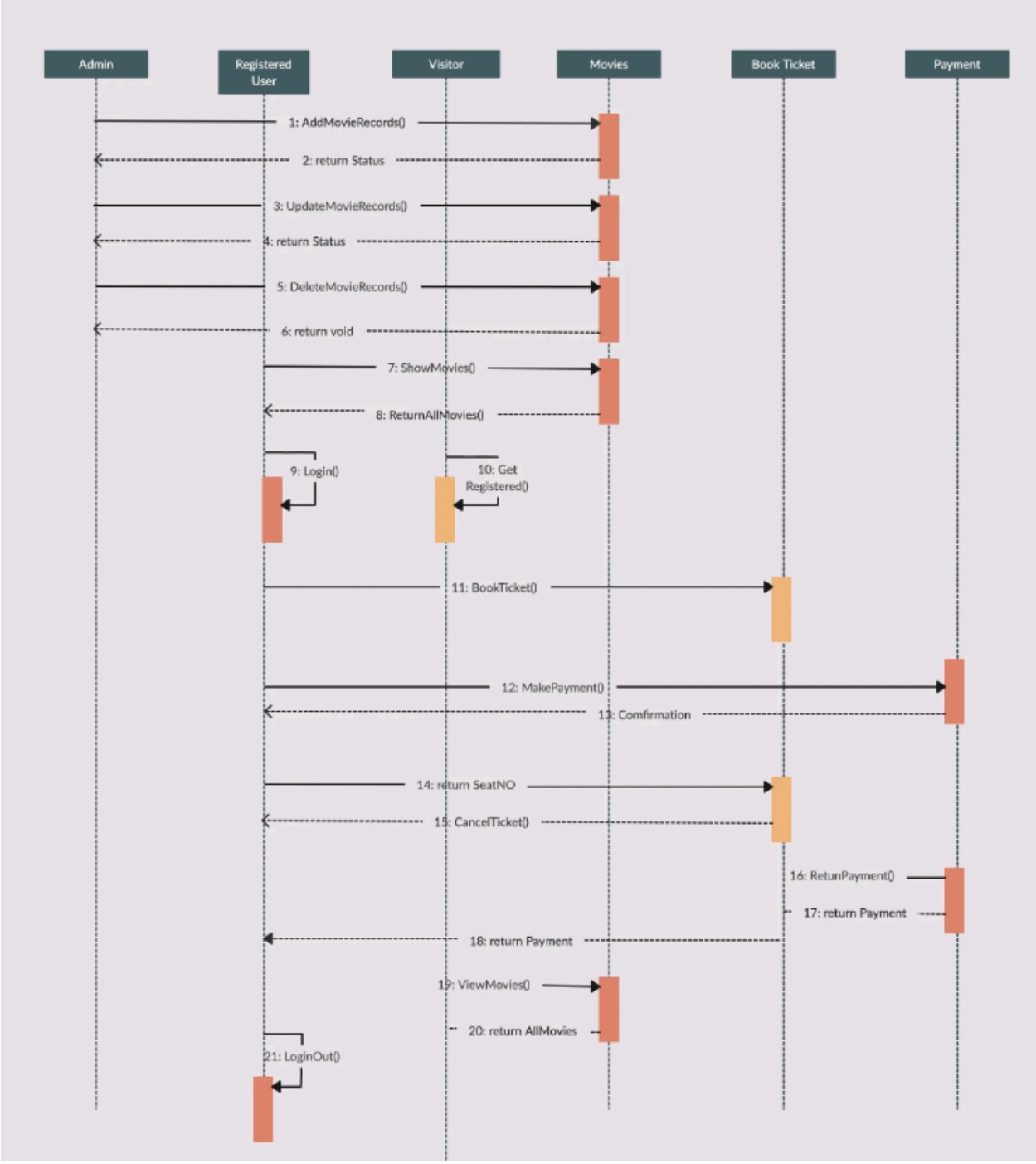
Sequence Diagram for Online Exam



Balance Lookup



Movie Ticket Booking System



END