Chapter 5:
Object Models

Introduction

- Object-oriented (OO) approach to the whole software development process is now commonly used, expressing **requirements** using OO model **designing** by using objects and **developing** the system in **OO programming language**, such as java or C++ .
- Models developed during the **Analysis-phase** represent **both data and its processing**, thus they combine data-flow and semantic -data models.

Advantages of OO

- **Object model good in showing how entities may be classified and composed of other**, this is

  - Very true for **tangible entities**; such as, car, aircraft, book, student, employee (these have clear attributes).

  - However this process becomes harder to model **abstract entities** such as medical-records, and word processing.

OO model simplifies the transition from Analysis to design and then to programming.

➔ ***Analyst should model real-world entities using object-classes not instantiations of the classes .***

Disadvantages of OO

- Identifying objects Vs object-classes is the most difficult area of OO development. This concerns the developers.

- Understanding OO concepts is difficult concerning users.

Various Methods of OO Analysts were Proposed

(i). **Coad and Yourdon**…..1990

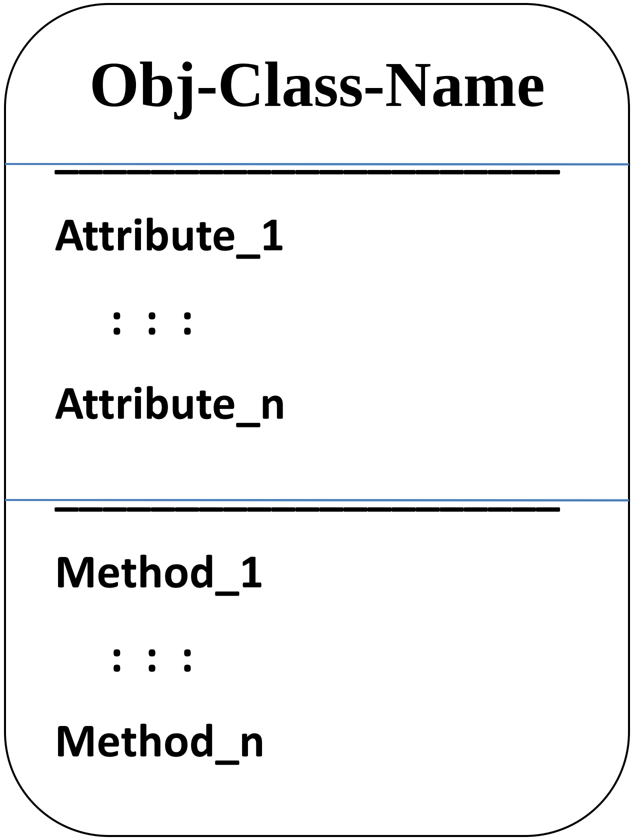
(ii). **Rumbaugh**, et al……...1991

(iii). **Jacobsen**, et al………..1993

(iv). **Booch**……………………...1994

- **Rumbaugh**, **Jacobsen** and **Booch** methods have been integrated into Unified–Modeling–Language (UML) in mid 1990s, then it had become a **standard** for object modeling.

UML Class Diagram



Obj-Class-Name

Attribute_1

: : :

Attribute_n

Method_1

: : :

Method_n

Attributes & Methods Directives

+Attribute: Public Attribute

-Attribute: Private Attribute

#Attribute: Protected Attribute

+Method: Public Method

-Method: Private Method

#Method: Protected Method

Public Vs **Private**

+Attribute/+Method→ **Public:** Allowing access from any other object.

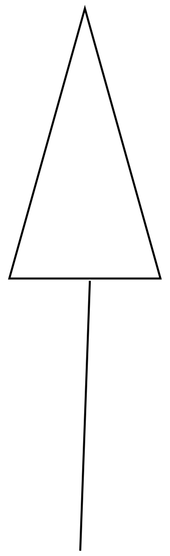
-Attribute/-Method → **Private:** Allowing access only within the scope of that object.

#Attribute/#Method → “***Mixed***”: A class and its subclasses may access this element

(In C++ this is called protected)

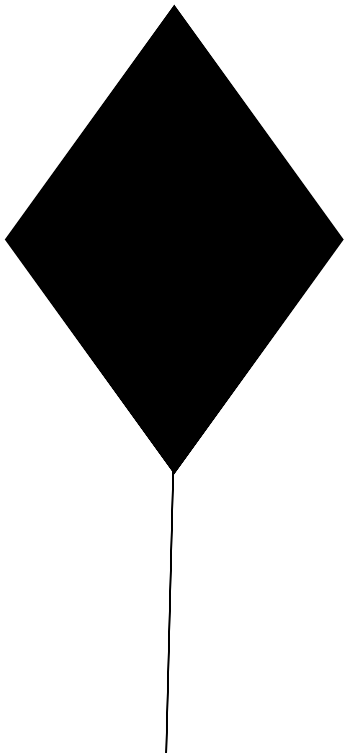
Inheritance

- The Symbol: Unfilled Triangular Arrow head
- Generalization Relation Between Subclass and Superclass (**Inheritance**)



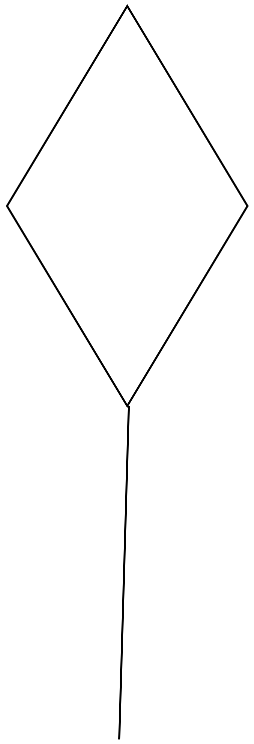
Composition

- Filled (or Black) Diamond (strong) **<u>Composition</u>**



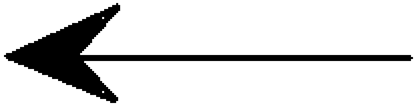
Aggregation

- Unfilled (white) Diamond (weak)
- **Aggregation**



Association

- Filled Arrow head **Association** showing the direction of the navigation



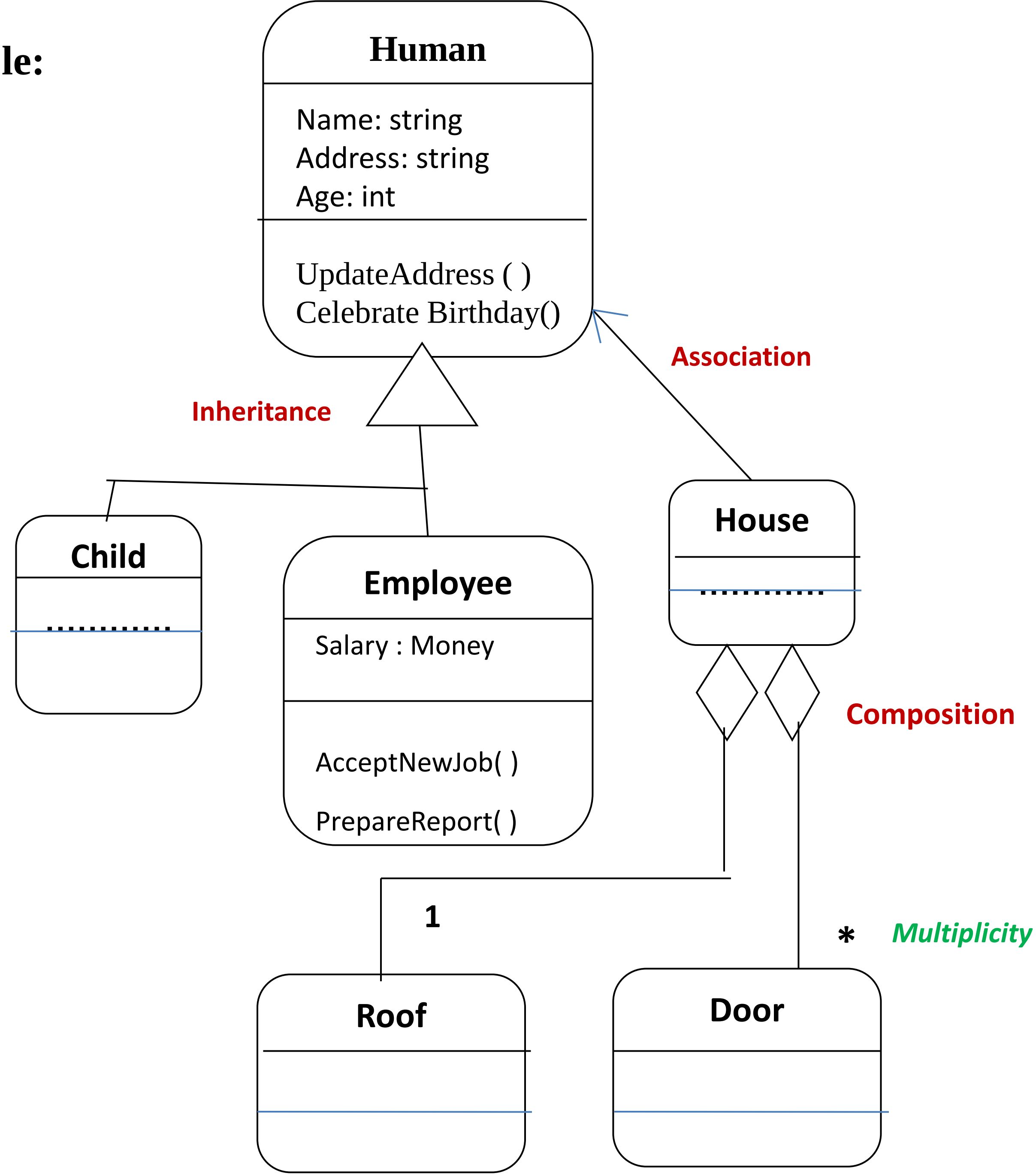
Multiplicity Symbols

*……Multiplicity Unspecified number.

1…….Multiplicity Exactly ONE

Note : if the class name in “***ITALIC***” FONT then the class is an abstract also applies on methods.

Example:



Human

Name: string
Address: string
Age: int

UpdateAddress ( )
Celebrate Birthday()

Inheritance

Association

Child

....

Employee

Salary : Money

AcceptNewJob( )

PrepareReport( )

House

....

Composition

1

*

Multiplicity

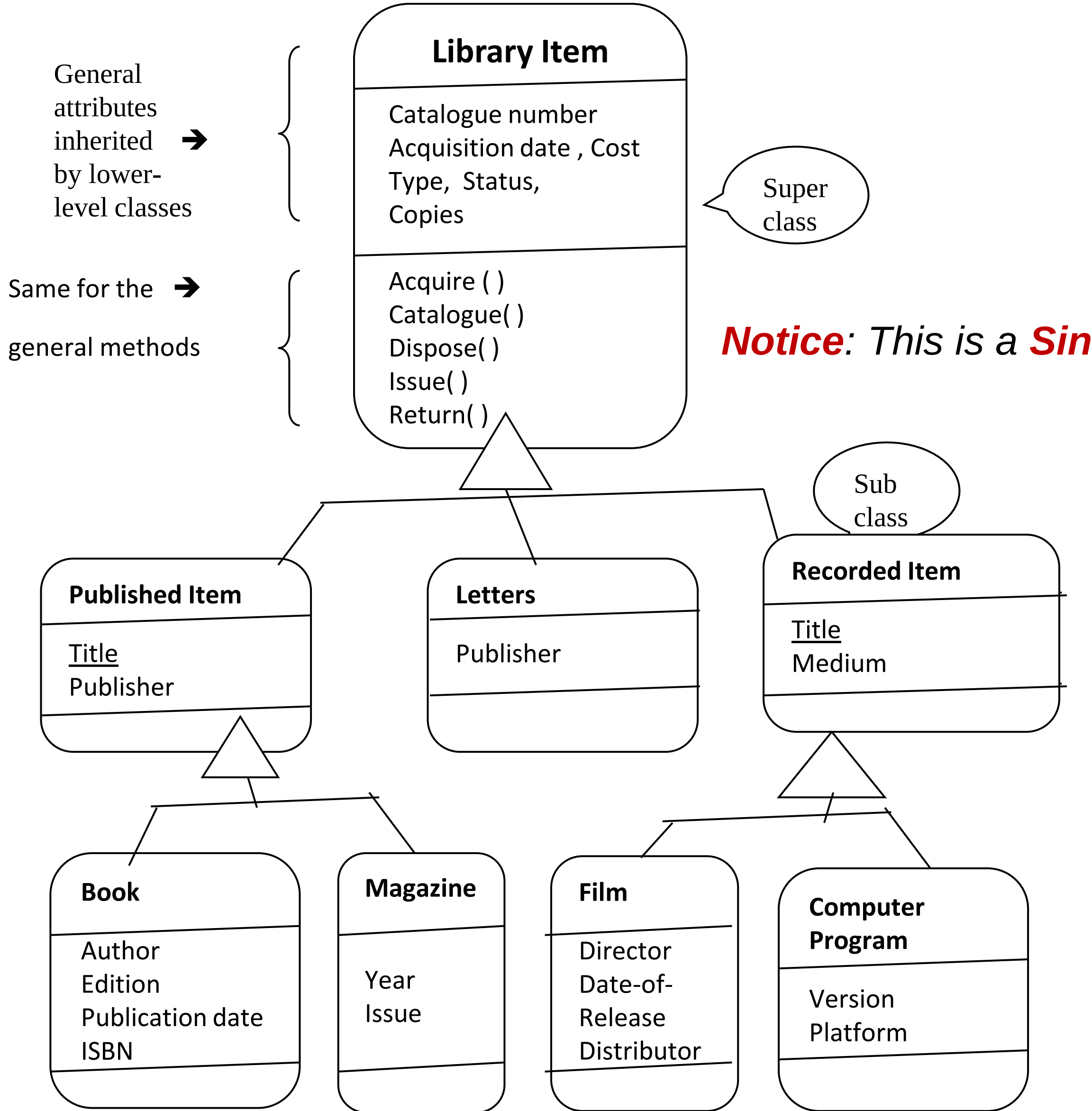
Roof

Door

(i). Inheritance Model

- Identify object-classes that are important in the domain being studied during analysis.
- Object-classes then classified in a hierarchy shows how an object class is related to other classes through common attributes and services.
- The hierarchy has the most general classes at the top, and more specialized objects have their own attributes and services.
- These specialized objects may have (in addition to the inherited ones) their own attributes and services.

Example-1 (Library Items): A library holds books, music, recordings of films, magazines and news papers etc . The inheritance hierarchy may look like.



General attributes inherited by lower-level classes ➔

Same for the ➔ general methods

Library Item

Catalogue number
Acquisition date , Cost
Type, Status,
Copies

Acquire ( )
Catalogue( )
Dispose( )
Issue( )
Return( )

Super class

Sub class

Published Item

Title
Publisher

Letters

Publisher

Recorded Item

Title
Medium

Book

Author
Edition
Publication date
ISBN

Magazine

Year
Issue

Film

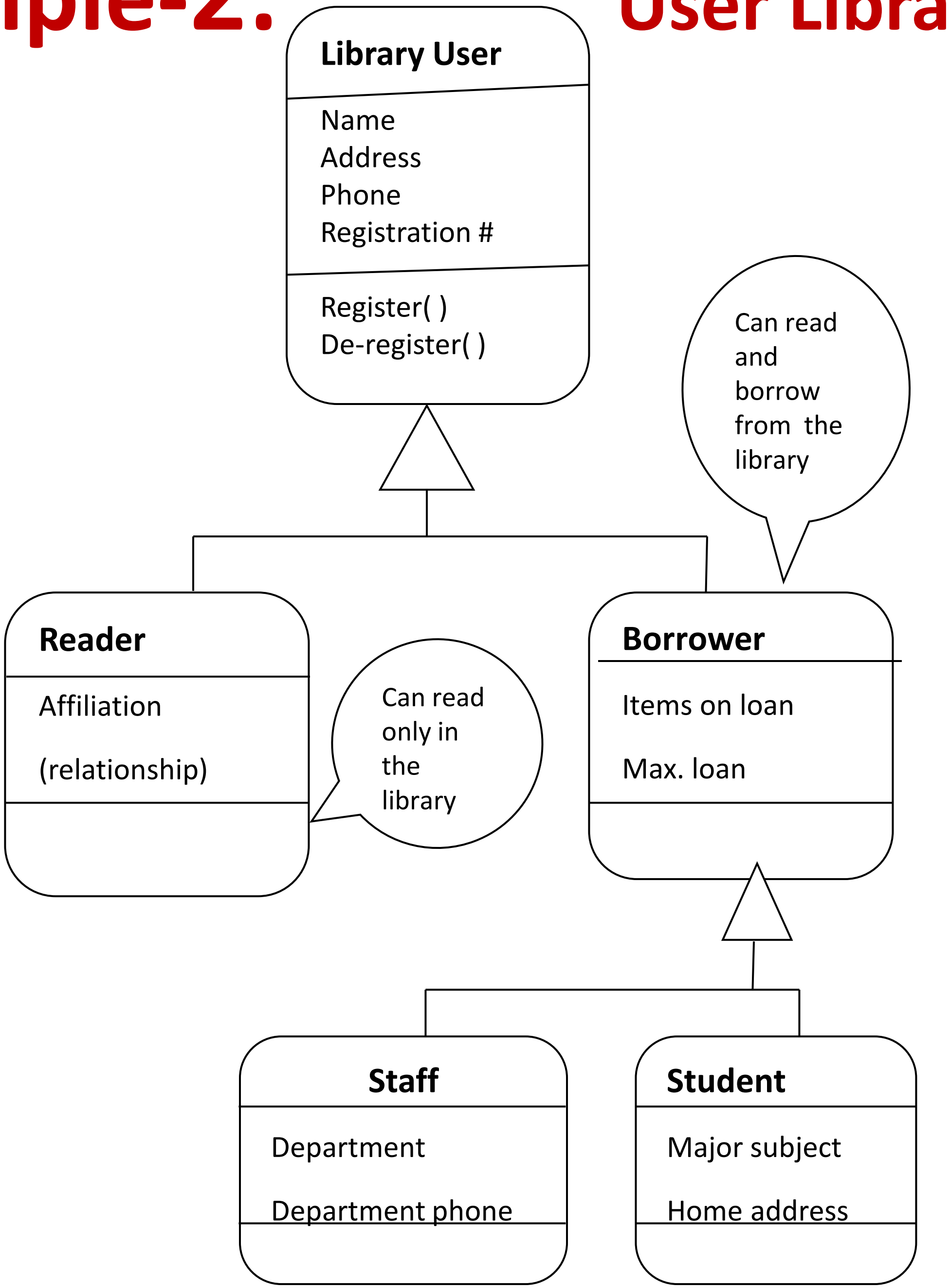
Director
Date-of-Release
Distributor

Computer Program

Version
Platform

Notice*: This is a *Single-Parent-Inheritance

Example-2: User Library Class Hierarchy



Library User

Name
Address
Phone
Registration #

Register( )
De-register( )

Reader

Affiliation

(relationship)

Can read only in the library

Borrower

Items on loan

Max. loan

Can read and borrow from the library

Staff

Department

Department phone

Student

Major subject

Home address

***Notice**: This example is also*
Single-Parent-Inheritance

Example-3: Library Items (with Multiple-Parent-Inheritance)

- Difficulties with multiple-parent- inheritance (in general)
  - Objects may inherit unnecessary attributes.
  - Resolving name clashes when two or more super-class

  attributes (parents attributes) have the same name but different meaning. This might be Ok to solve during modeling-level however, it is header to sort out at implementation-level.

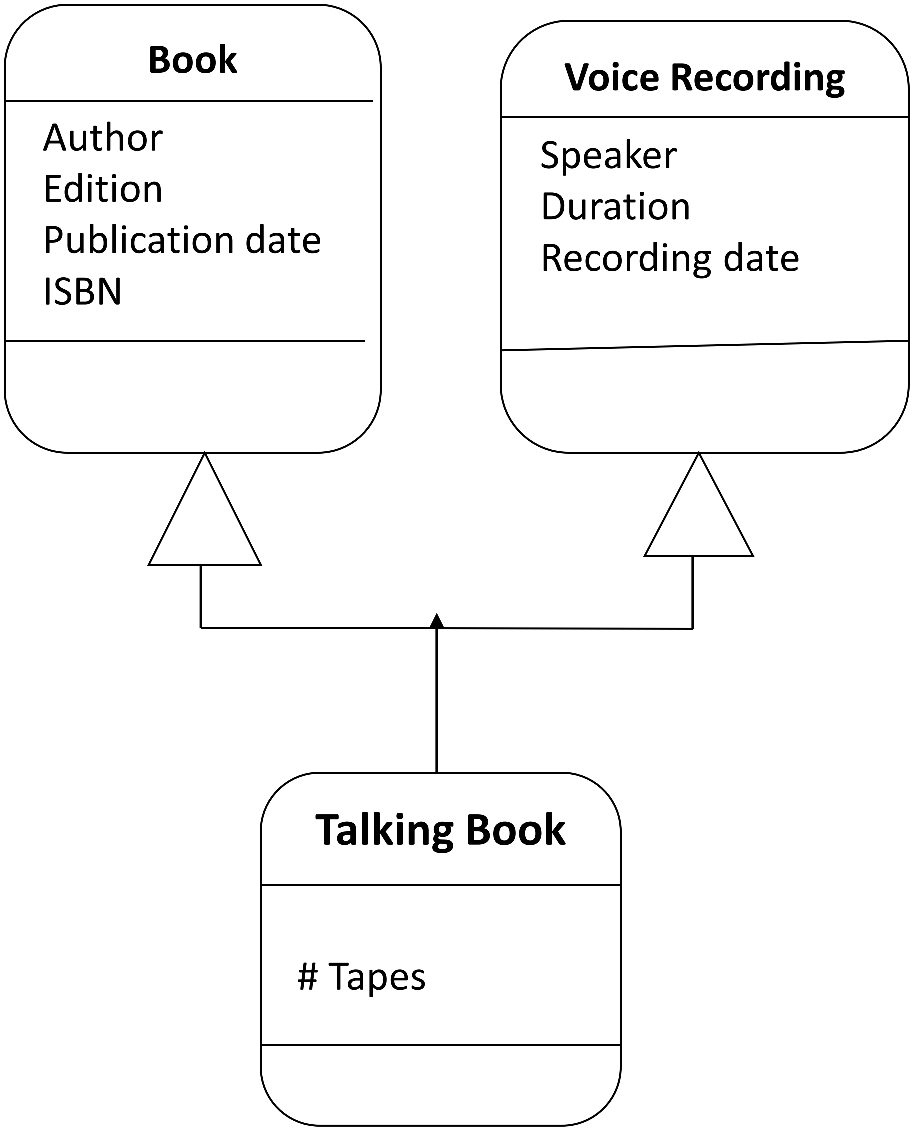
  Thus,

  @ Modeling-level → Ok

  @ Programming-level → Hard

  - Re-organizing the structure become a difficult task when the structure is modified/changed.

Example-3: Multiple Parent Inheritance *(Cont.)*



Book

Author
Edition
Publication date
ISBN

Voice Recording

Speaker
Duration
Recording date

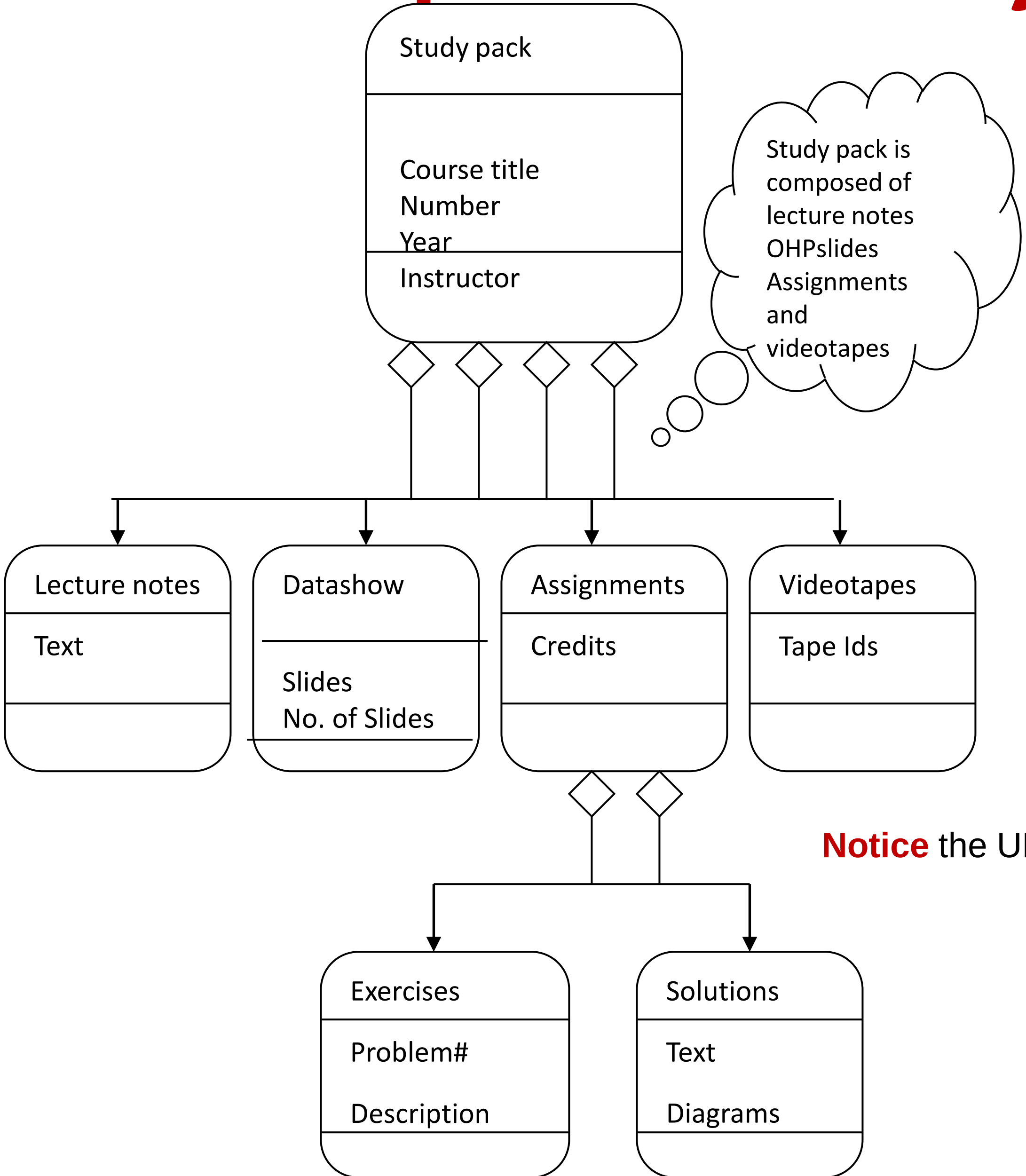
Talking Book

# Tapes

(ii). Object Aggregation

- Some objects are groupings of other objects, meaning an object is an aggregate of a set of other objects.

Example-1: A Study-Pack



Notice the UML notation to represent "**aggregation**" is diamond ◇

Class Diagram Example – Requirements

- Bank system is a huge system. This is part of it only, the customer owns bank accounts, and customer can ask Customer care support to create current account and saving account. Customer can withdraw money and deposit money also. Customer can take loans also if the bank manager accepts that. Bank has ATM, ATM consist of HW (Keypad, Buttons) and SW(Embedded OS)

(iii). Object Behavior Modeling

- We model the behavior of object by showing the operations (method) provided by those objects.
- In UML we model behavior using **scenarios** that are represented as **use-cases** this can be done by using UML **sequence-diagram**.
- The sequence-diagram shows the sequence of actions involved in a use-case.

Use-Cases

- A fundamental feature of UML for describing object-oriented system models.
- A use-case Identifies:
  - The type of interaction.
  - The actors involved.
  - The direction associated with the interaction (used symbols ⇄)

Importance of Use-case

- **To identify FUNCTIONS and how ROLES interact with them -** Primary purpose.
- **To provide a HIGH LEVEL VIEW of the system** – Especially useful when presenting to managers or stakeholders. (highlight roles & functionality without going deep into inner workings of the system).

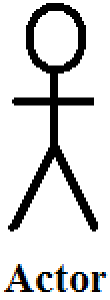
Use-Case Diagram Components

➔ Use case diagrams consist of 4 objects.

- Actor
- Use case
- System
- Package

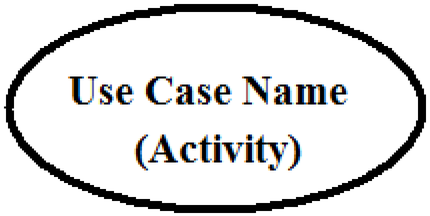
Actor

- Actor in a use case diagram is **any entity that performs a role** in one given system. This could be a person, organization or an external system and usually drawn like skeleton shown below.



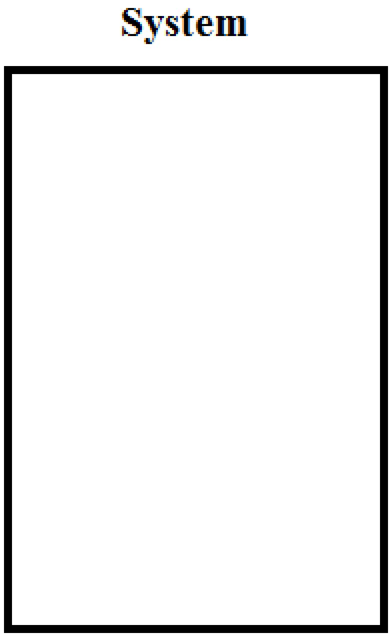
Use-Case

- A use case **represents a function, activity or an action within the system**. Its drawn as an oval and named with the function.



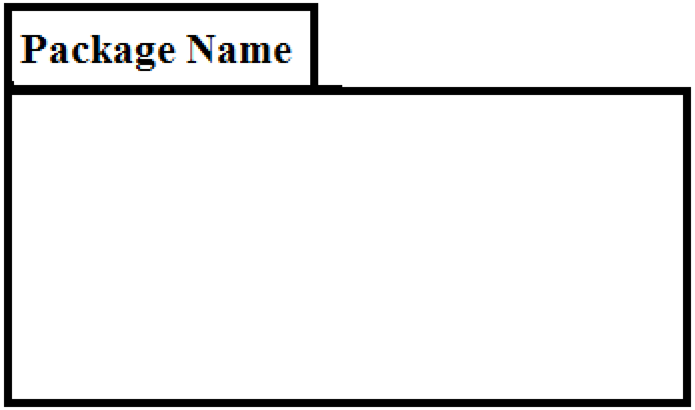
System *(Optional)*

- System is used to **define the scope of the use case** and drawn as a rectangle. Useful to visualize large systems. For example you can create all the use cases and then use the system object to define the scope covered by your project.



Package *(Optional)*

- Package is useful in complex diagrams. Packages are **used to group together use cases**. They are drawn like the image shown below.



Relationships in Use Case Diagrams

There are five types of relationships in a use case diagram. They are:

1. **Association** between an actor and a use case
2. **Generalization** of an **actor**
3. **Extend** relationship between two use cases
4. **Include** relationship between two use cases
5. **Generalization** of a **use case**

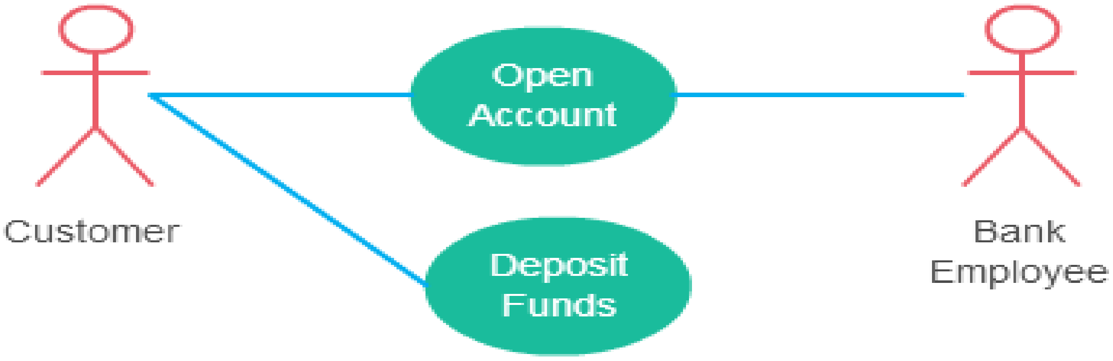
Association Between Actor &Use Case

- Straightforward and present in every diagram.

  Notes:

- An actor must be associated with at least one use case.
- An actor can be associated with multiple use cases.
- Multiple actors can be associated with a single use case

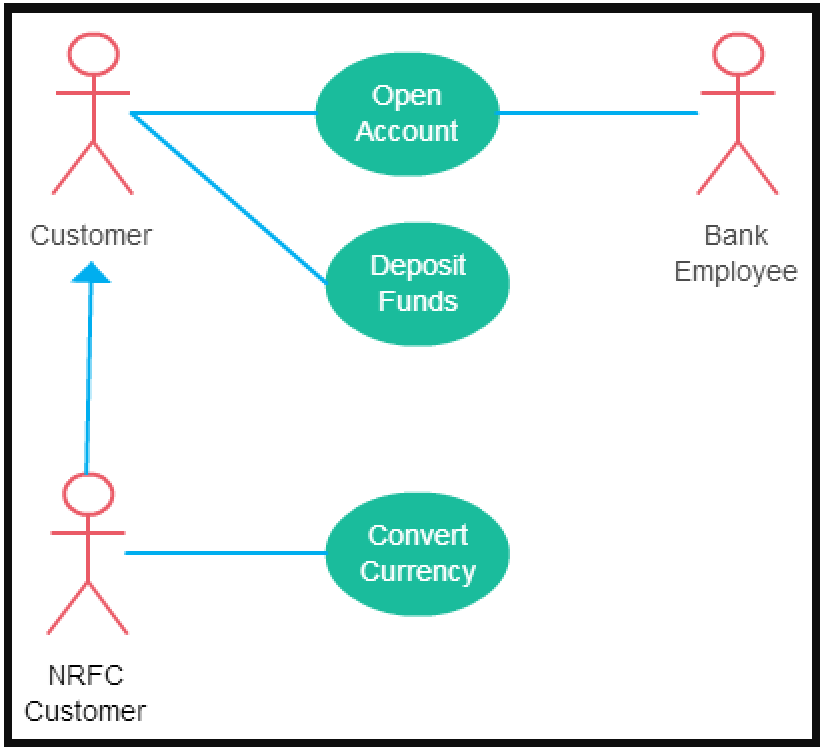
Illustration -Example in a Bank System



Different ways association relationship appears in use case diagrams

Generalization of an **Actor**

- **Generalization** of an **actor** means that one actor can inherit the role of an other actor. The descendant (Child) inherits all the use cases of the ancestor (Parent). The descendant have one or more use cases that are specific to that role. Lets expand the previous use case diagram to show the generalization of an actor.

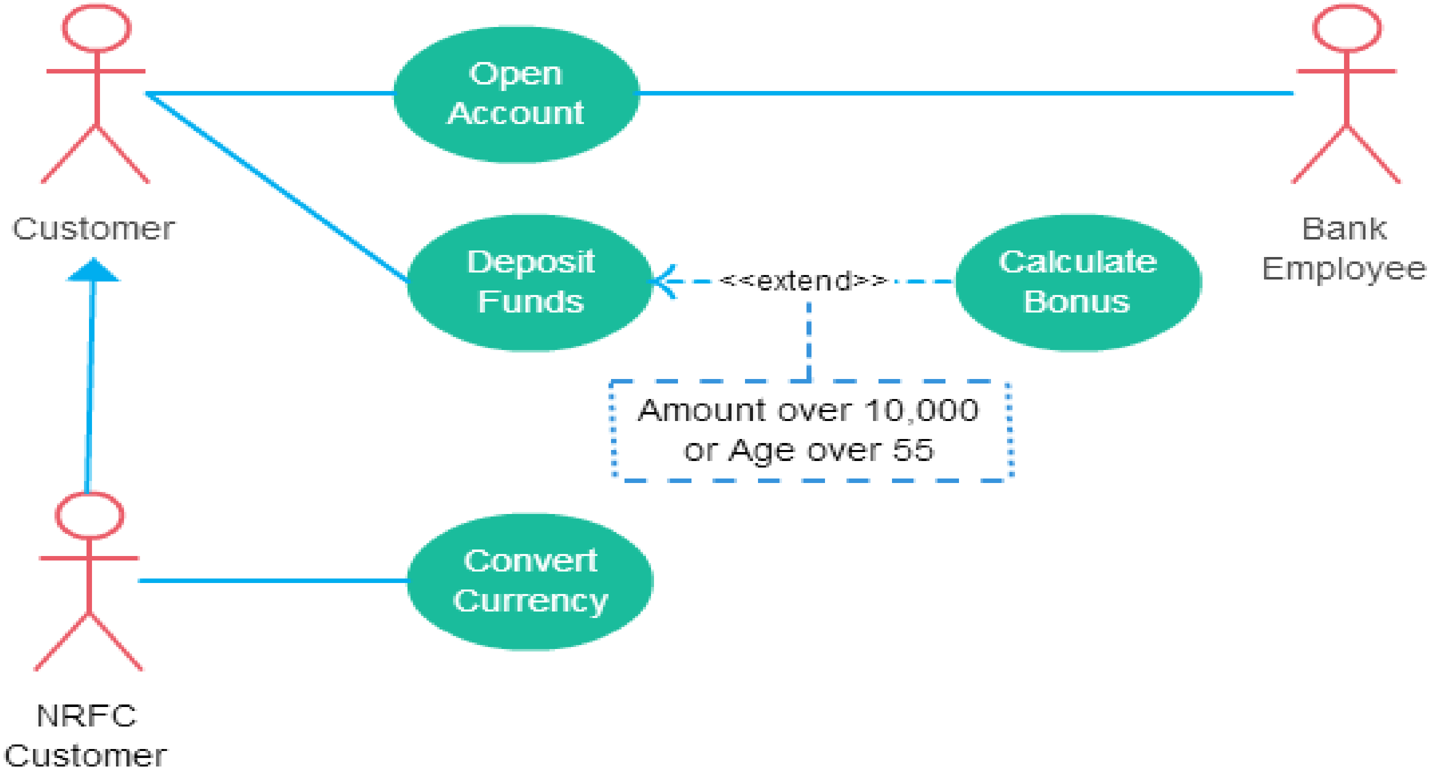


Extend Relationship Between Two Use Cases

- As the name implies it extends the **base use case** and adds more functionality to the system. Here are few things to consider when using the <<**extend**>> relationship.
  - **The extending use case is dependent on the extended (base) use case**.
  - **The extending use case is usually optional** and can be triggered conditionally.
  - **The extended (base) use case must be meaningful on its own**. This means it should be independent and must not rely on the behavior of the extending use case.

Example: Extended Use-Case Relationship

Extend Use-Case Relationship -In a Bank System



Extend relationship in use case diagrams

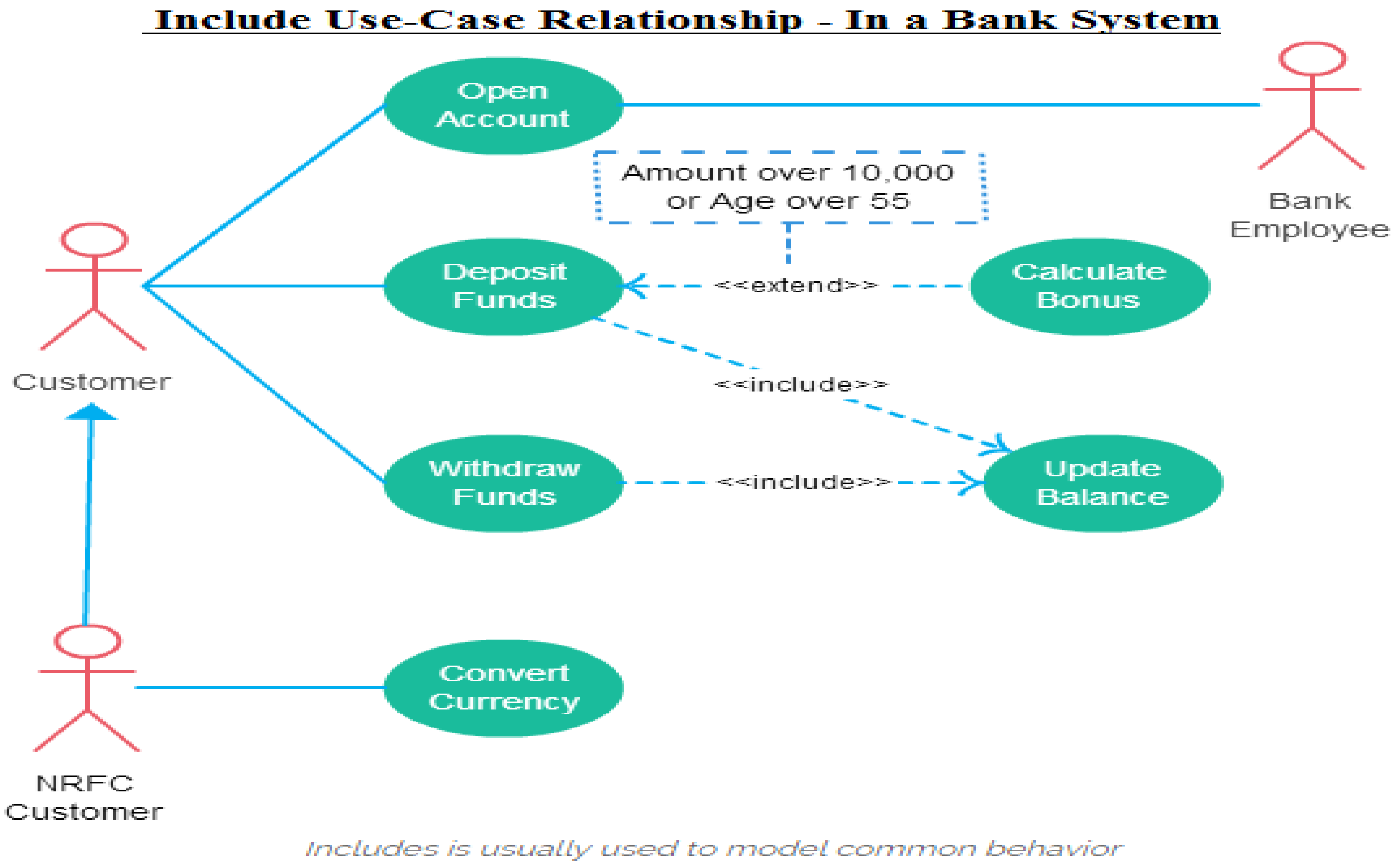
Include Relationship Between Two Use Cases

- **Include** relationship show that the behavior of the included use case is part of the including (base) use case. The main reason for this is to reuse the common actions across multiple use cases. In some situations this is done to simplify complex behaviors.

➔Two things to consider when using the <<**include**>> relationship:

- The base use case is incomplete without the included use case.
- The included use case is mandatory & not optional.

Example: Include Use-Case Relationship



Generalization of a **Use Case**

- This is similar to the generalization of an actor. The behavior of the ancestor (Parent) is inherited by the descendant (Child). This is used when there are common behavior between two use cases and also specialized behavior specific to each use case.
- In the previous banking **example:**

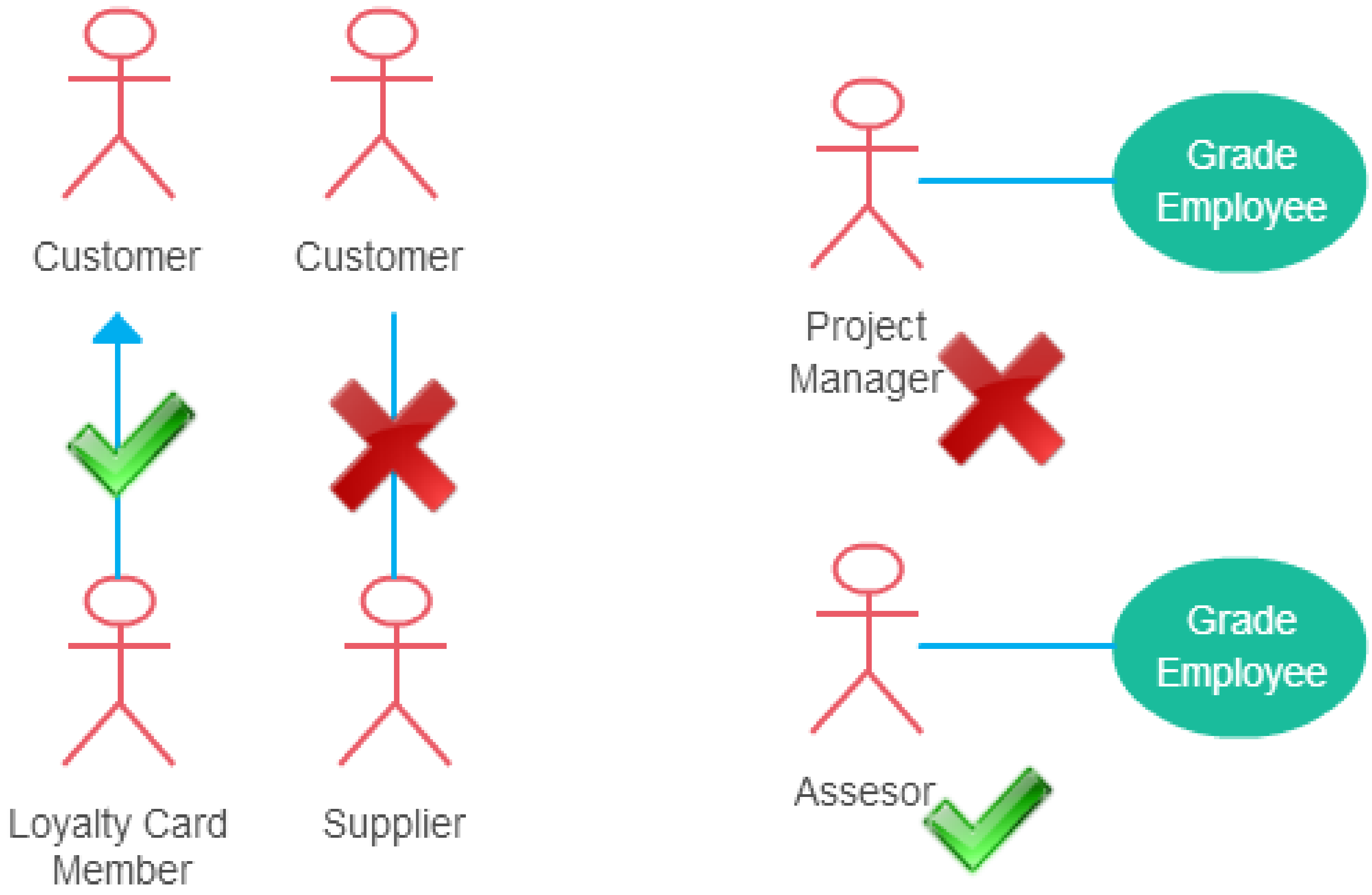
  There might be an use case called "**Pay Bills**".

  This can be generalized to "**Pay by Credit Card**", "**Pay by Bank Balance**" etc.

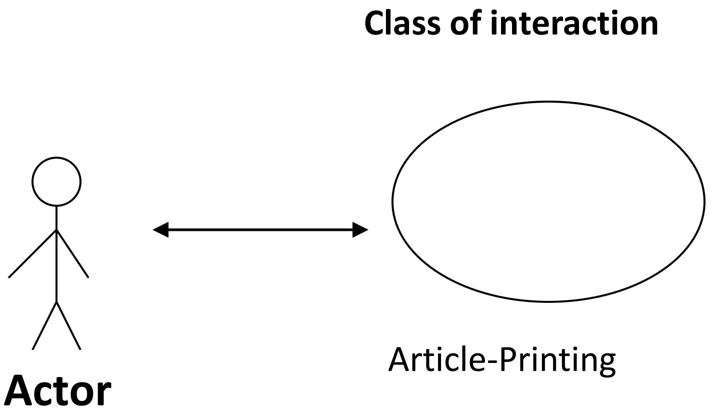
Use-Case Diagram Guidelines

- Give **meaningful business relevant names** for actors
  - **Eg**: Airline-Company is better than United
- **Primary actors** should be **to the left side** of the diagram
- **Actors model roles** (not positions)
  - **Eg:** In a hotel both the front office executive and shift manager can make reservations. So something like “Reservation Agent” should be used for actor name to highlight the role.
- **External systems are actors**
- **Actors don’t interact** with **other actors**
- **Place inheriting actors below** the **parent actor**

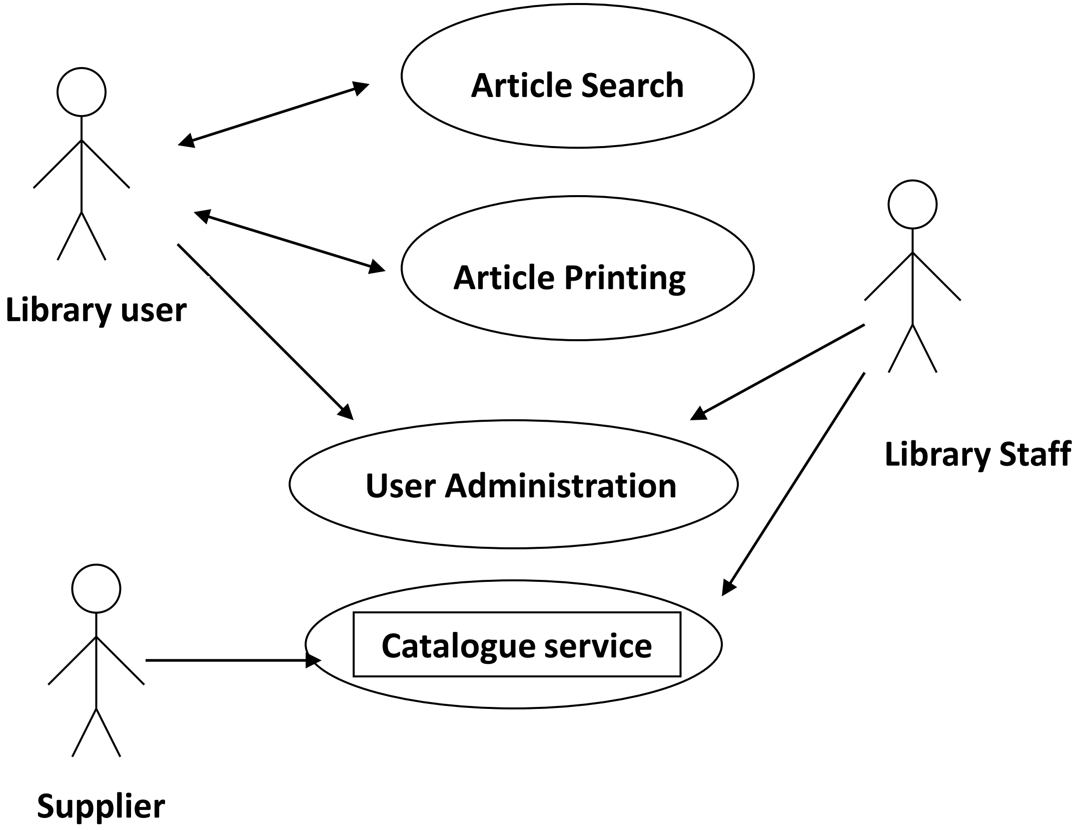
Invalid and Valid Examples



Simple example of a use-case for the Article-Printing Facility in **LIBSYS**:



More examples of use-cases diagram from the LIBSYS



Article Search

Article Printing

User Administration

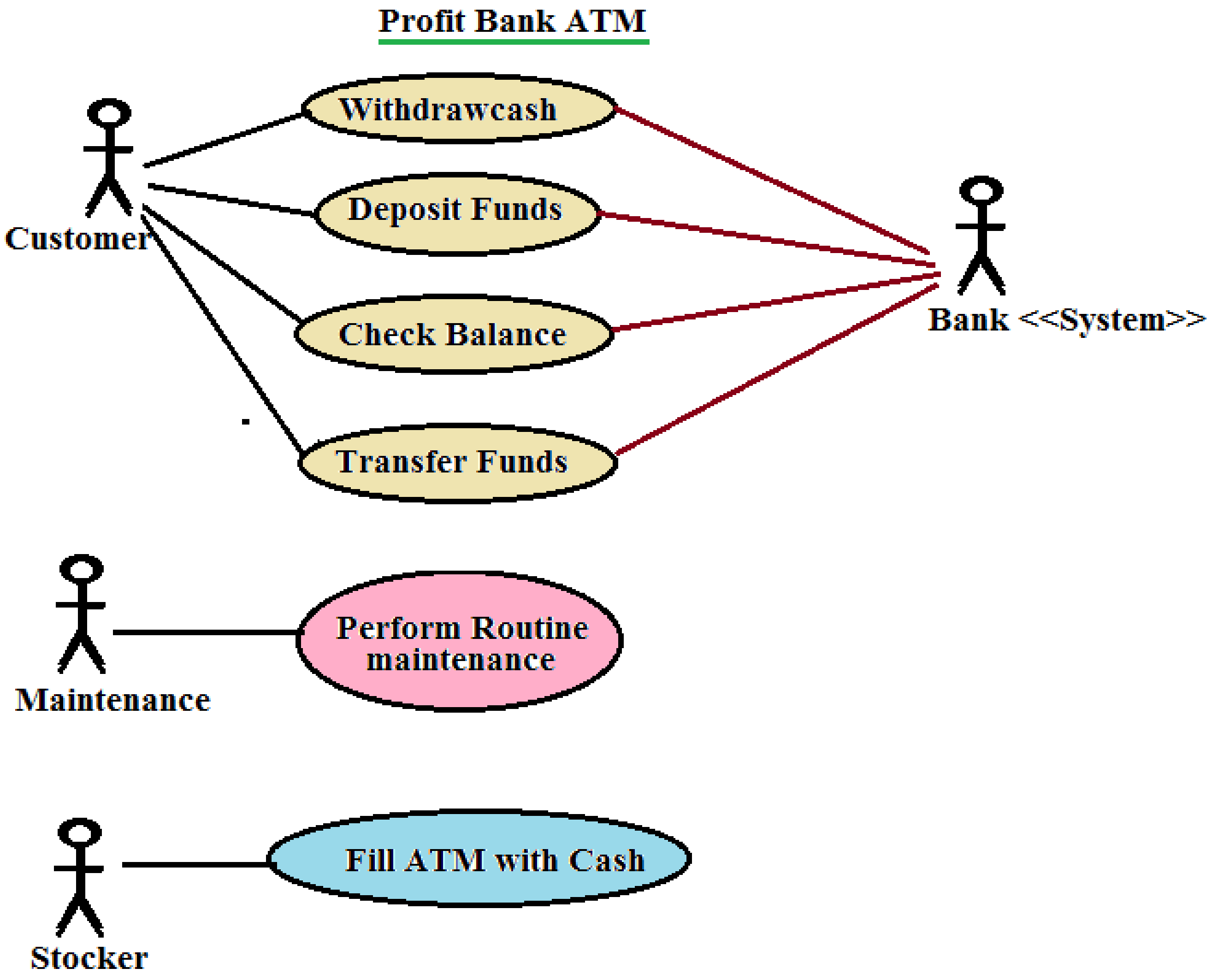
Catalogue service

Library user

Library Staff

Supplier

ATM Use-Case Diagram Example



Difference between Extend & Include

- **Extend**

  Means to expand the current (original) use-case to another (additional) use-case, and Extend is usually optional.

  Example (1): If you REGISTER on a Website, you can **extend** to do REPLY. But replying is not a mandatory, it is optional. Thus, the arrow direction goes to the original use-case LOGIN. Here is the illustration:

  Log in <---<extend>------- add reply

  Example (2): **View profile <----<extend>----- edit profile**

- **Include**

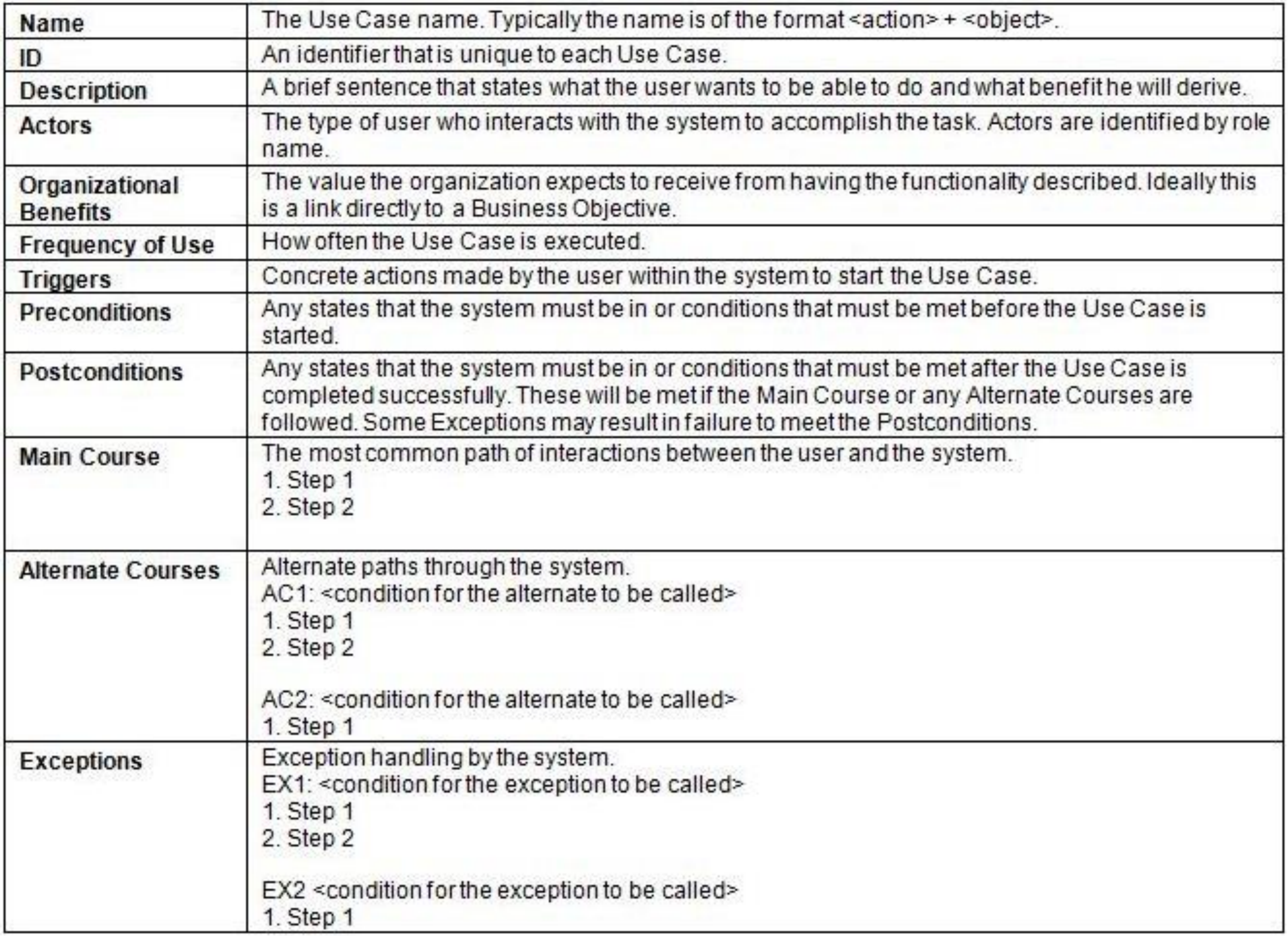
  It means, adding another use-case. It is usually mandatory, in order to execute the original use-case we must include the execution of an additional one first.

  Example (1): To be able to reply from within a website you have to be logged in first, meaning you have to include the login use-case before being able to reply.

  Example (2): **Delete reply ----<include>----->view reply**

  Example (3): **Change password ----<include>----->Log in**

Use-Case Business Requirements Model Template



Name	The Use Case name. Typically the name is of the format <action> + <object>.
ID	An identifier that is unique to each Use Case.
Description	A brief sentence that states what the user wants to be able to do and what benefit he will derive.
Actors	The type of user who interacts with the system to accomplish the task. Actors are identified by role name.
Organizational Benefits	The value the organization expects to receive from having the functionality described. Ideally this is a link directly to a Business Objective.
Frequency of Use	How often the Use Case is executed.
Triggers	Concrete actions made by the user within the system to start the Use Case.
Preconditions	Any states that the system must be in or conditions that must be met before the Use Case is started.
Postconditions	Any states that the system must be in or conditions that must be met after the Use Case is completed successfully. These will be met if the Main Course or any Alternate Courses are followed. Some Exceptions may result in failure to meet the Postconditions.
Main Course	The most common path of interactions between the user and the system. 1. Step 1 2. Step 2
Alternate Courses	Alternate paths through the system. AC1: <condition for the alternate to be called> 1. Step 1 2. Step 2 AC2: <condition for the alternate to be called> 1. Step 1
Exceptions	Exception handling by the system. EX1: <condition for the exception to be called> 1. Step 1 2. Step 2 EX2 <condition for the exception to be called> 1. Step 1

Example (1): of a Use-case taken from Last Semester’s Graduation project (IOT)

Use case	**Measuring Pressure on the Pillow**
Actor	The Sensor
Description	The Sensor Measures the weight on the pillow when a person lays down his head on the pillow to get some sleep.
Normal flow	- The Sensor measures the pressure (weight) - The Sensor sends the weight value to the system
Precondition	A person lays his/her head on the pillow
Post-condition	The weight value has been Sent to the system
Exception	No connection with sensor (Send notification message)

Example (2): of a Use-case taken from Last Semester’s Graduation project (IOT)

Use case	**Vibration Alarm**
Actor	System
Description	A person sets the alarm at specific time and the system shall vibrate inside the pillow on that time.
Normal flow	When the real time matches the alarm time then the pillow will vibrate.
Precondition	Set alarm time
Post-condition	The Person did wake up, after the vibration.
Exceptions	No Vibration (Defect with the device) No Vibration Wrong setting (PM/AM)

Scenarios

- Starts with an outline of the interaction and during elicitation (bring out) details are added to create a complete description of that interaction.
- A scenario consists of:
  - **Start State**; describes what the system and users expect when it start.
  - **Flow of events (normally);** describe normal flow of events.
  - **Possible Faults and Handling**; describe the handling of unexpected or faulty events.
  - **Other Activities**; describes other simultaneous activities of the system
  - **Finish State**; describes the system when it finishes the task.
- Scenarios may be written as:

(1). Text supplemented by diagrams, screen-shots, etc.

(2). More structured approach such as events scenario or use-cases.

Example of Text-scenario:

A user of the LIBSYS printing an article in a medical-journal (free for subscribers, and charge/fee for others)

- **Initial Assumption**:

  The user logged-on to the LIBSYS and located the journal containing the article that need to be printed.

Text-scenario (page-2)

– **Normal Activities:**

- The user selects the article.
- The system prompts the subscription-code or method of payment if not a subscriber.
- The user fill in a copyright from and submit it.
- The PDF version of the article is downloaded to the LIBSYS working area on the user’s PC.
- The user prints a copy on a selected printer

Text-scenario (page-3)

What Can Go Wrong:

- The user may fail to fill in copyright info. Correctly.

→ Rejected from, if not corrected in 2nd attempt.

- The payment may be rejected by the system because of expired CR-Card for example.

→ User-request is Rejected.

- The download may fail.

→ The system Retry until success or termination by the user.

▪ The article may not print if flagged “Read-Only“ in this case it is deleted.

Text-scenario (page-4)

- **Other Activities:**

- Simultaneous download of other articles is taking place on the system.

- **Systems State on Completion:**

- User is logged on. The downloaded article has been printed and deleted from LIBSYS workspace.

In class excise

- Draw use case diagram for :
- Bank system is a huge system. This is part of it only, the customer owns bank accounts, and customer can ask Customer care support to create current account and saving account. Customer can withdraw money and deposit money also. Customer can take loans also if the bank manager accepts that. Bank has ATM, ATM consist of HW (Keypad, Buttons) and SW(Embedded OS)

Sequence Diagram

- It show the "**Actors**" involved in the interaction the "**Objects**" they interact with and the "**Operations**" associated with these objects.

- Sequence diagrams (or **collaboration diagrams**) in the UML are used to model interaction between objects.

What do Sequence Diagrams model?

- Capture the interaction between objects in the context of a collaboration
- Show object instances that play the roles defined in a collaboration
- Show the order of the interaction visually by using the vertical axis of the diagram to represent time of what messages are sent and when
- Show elements as they interact over time, showing interactions or interaction instances
- Do not show the structural relationships between objects

What do Sequence Diagrams model? (Cont.)

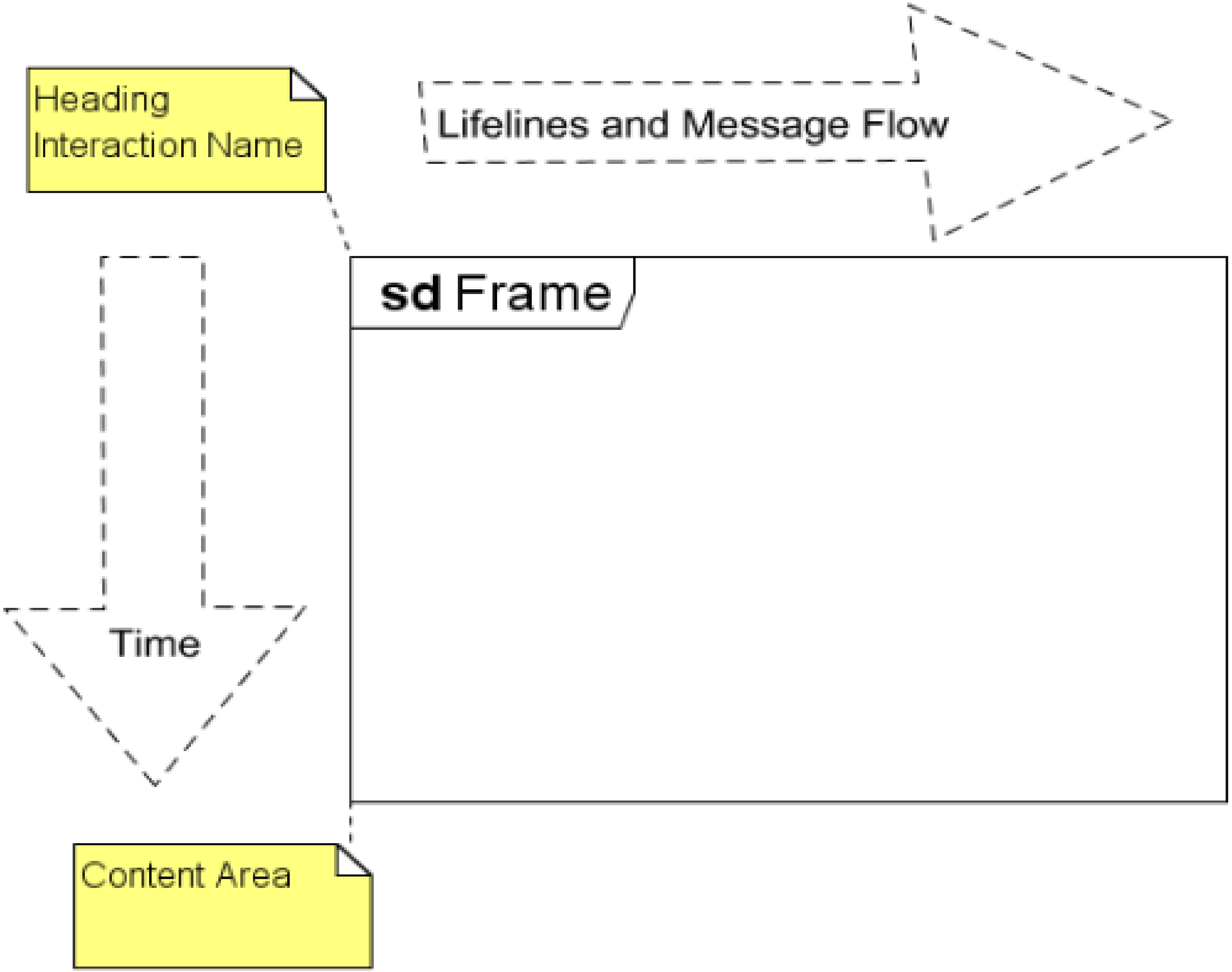
- **Model high-level interaction between active objects in a system (or object instances).**
- **Either model generic interactions (showing all possible paths through the interaction) or specific instances of a interaction (showing just one path through the interaction)**
- **Capture high-level interactions between user of the system and the system, between the system and other systems, or between subsystems.**

Sequence Diagrams Pieces

- Frames
- Lifelines
- Messages and Focus Control

And more!

Frames


Heading
Interaction Name
Lifelines and Message Flow
sd Frame
Time
Content Area

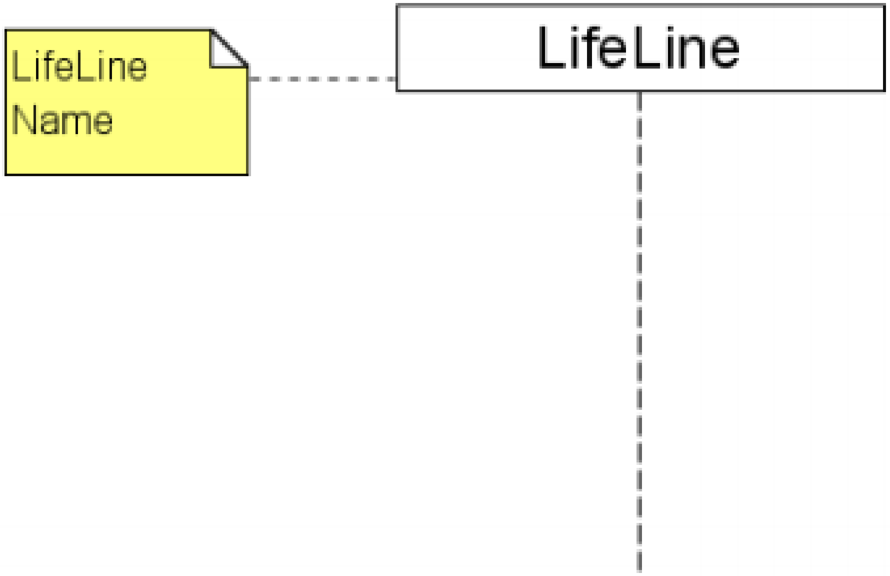
Sequence Diagrams Dimensions

Time: The **vertical axis** represents **time** proceedings (or progressing) down the page. Note that Time in a sequence diagram is all a about ordering, not duration. The vertical space in an interaction diagram is not relevant for the duration of the interaction.

Objects. The **horizontal axis** shows the **elements** that are involved in the interaction. Conventionally, the objects involved in the operation are listed from left to right according to when they take part in the message sequence.

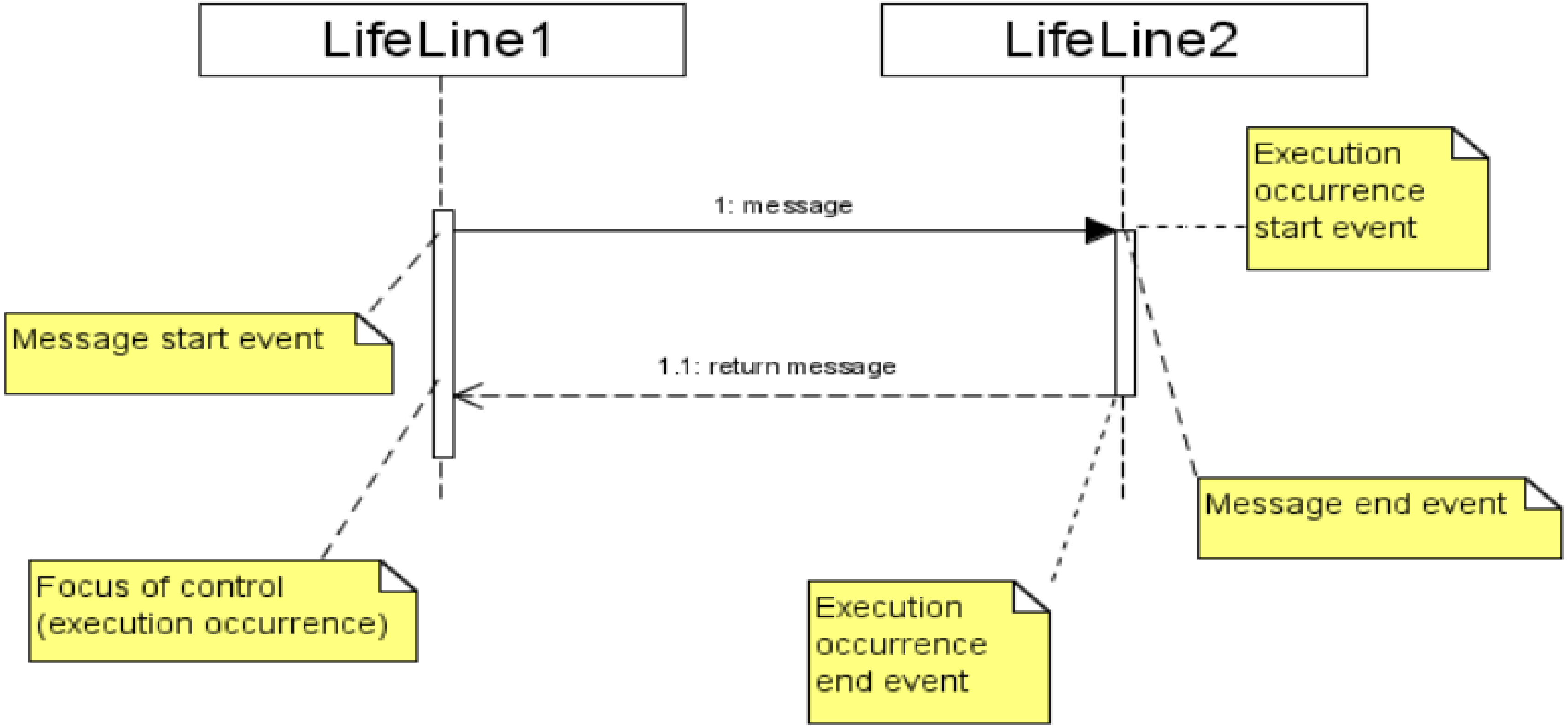
➔***However, the elements on the horizontal axis may appear in any order.***

Lifelines



- Sequence diagrams are organized according to time.
- Each participant has a corresponding lifeline.
- Each vertical dotted line is a lifeline, representing the time that an object exists.

Messages and Focus of Control

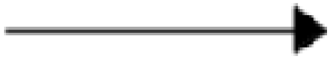
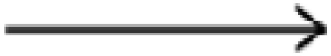
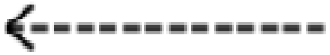

LifeLine1
LifeLine2
1: message
1.1: return message
Execution occurrence start event
Message start event
Message end event
Focus of control (execution occurrence)
Execution occurrence end event

- Focus of control (execution occurrence): an execution occurrence (shown as tall, thin rectangle on a lifeline) represents the period during which an element is performing an operation. The top and the bottom of the of the rectangle are aligned with the initiation and the completion time respectively.
- An Event is any point in an interaction where something occurs.

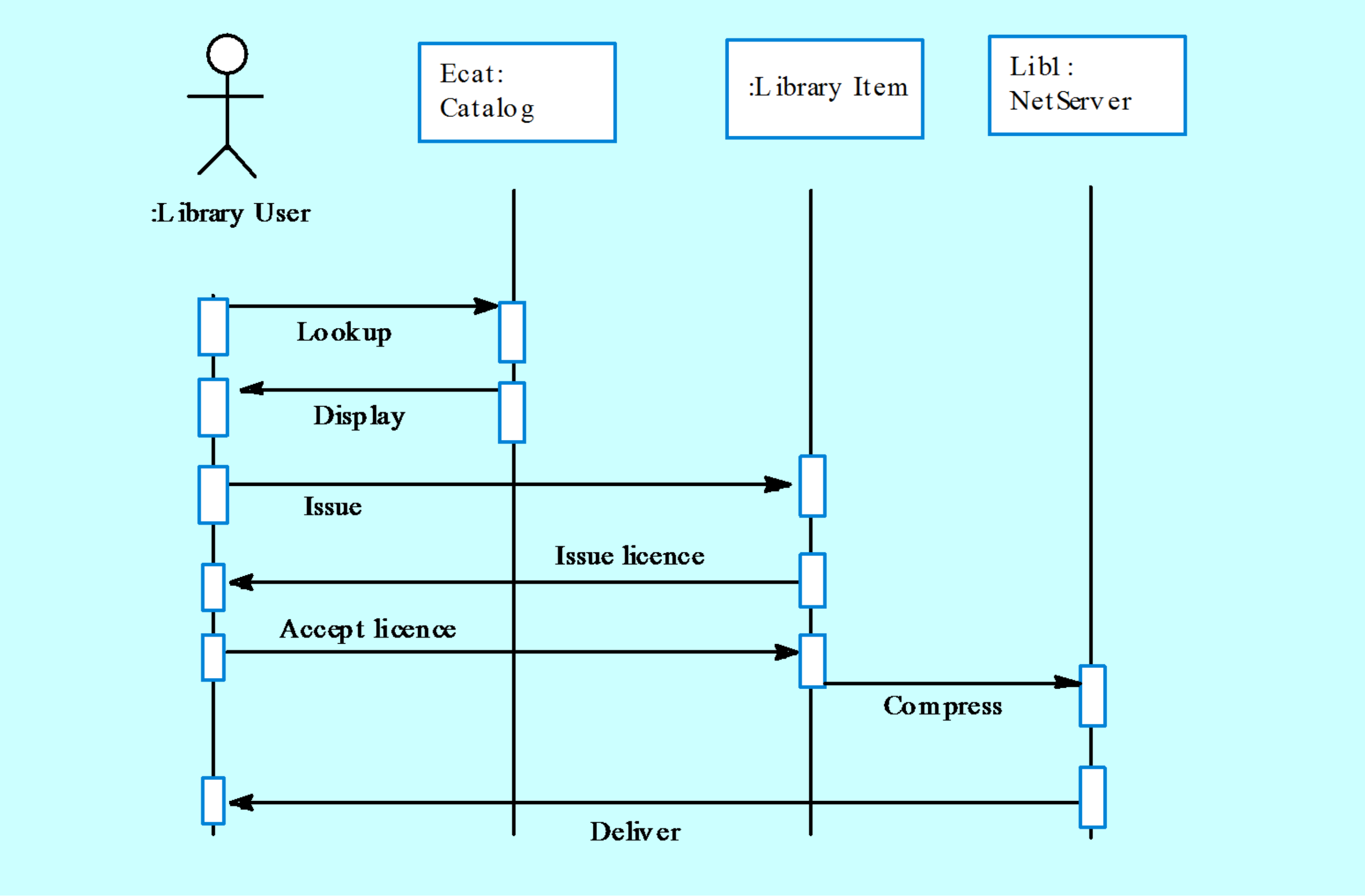
Messages

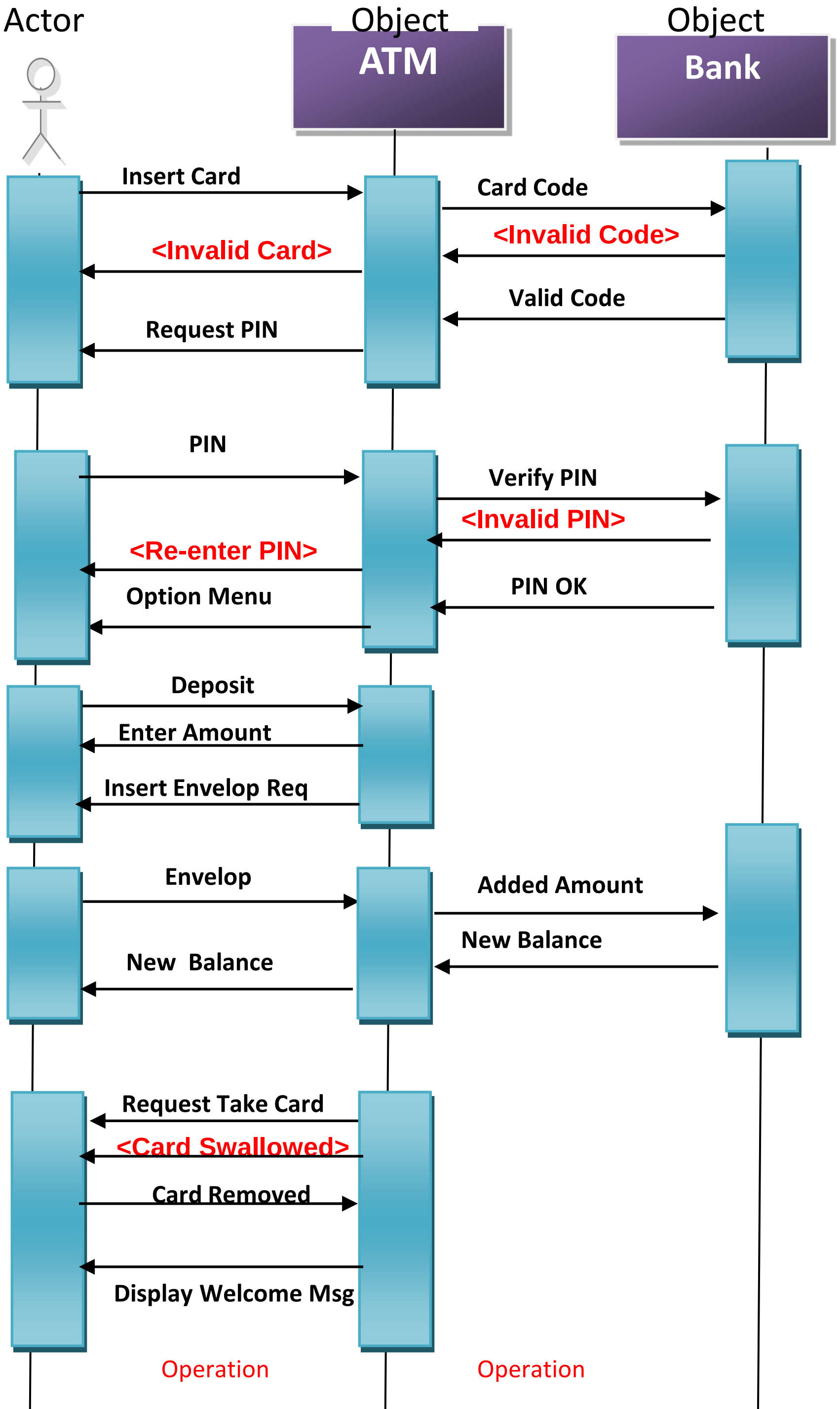
- **Messages (or signals)** on a sequence diagram are specified using an arrow from the participant (message caller) that wants to pass the message to the participant (message receiver) that is to receive the message
- **A Message (or stimulus)** is represented as an arrow going from the sender to the top of the focus of control (i.e., execution occurrence) of the message on the receiver’s lifeline

Message Type Notations

Message	Description
	**Synchronous:** A synchronous message between active objects indicates wait semantics; the sender waits for the message to be handled before it continues. This typically shows a method call.
	**Asynchronous:** With an asynchronous flow of control, there is no explicit return message to the caller. An asynchronous message between objects indicates no-wait semantics; the sender does not wait for the message before it continues. This allows objects to execute concurrently.
	**Reply:** This shows the return message from another message.

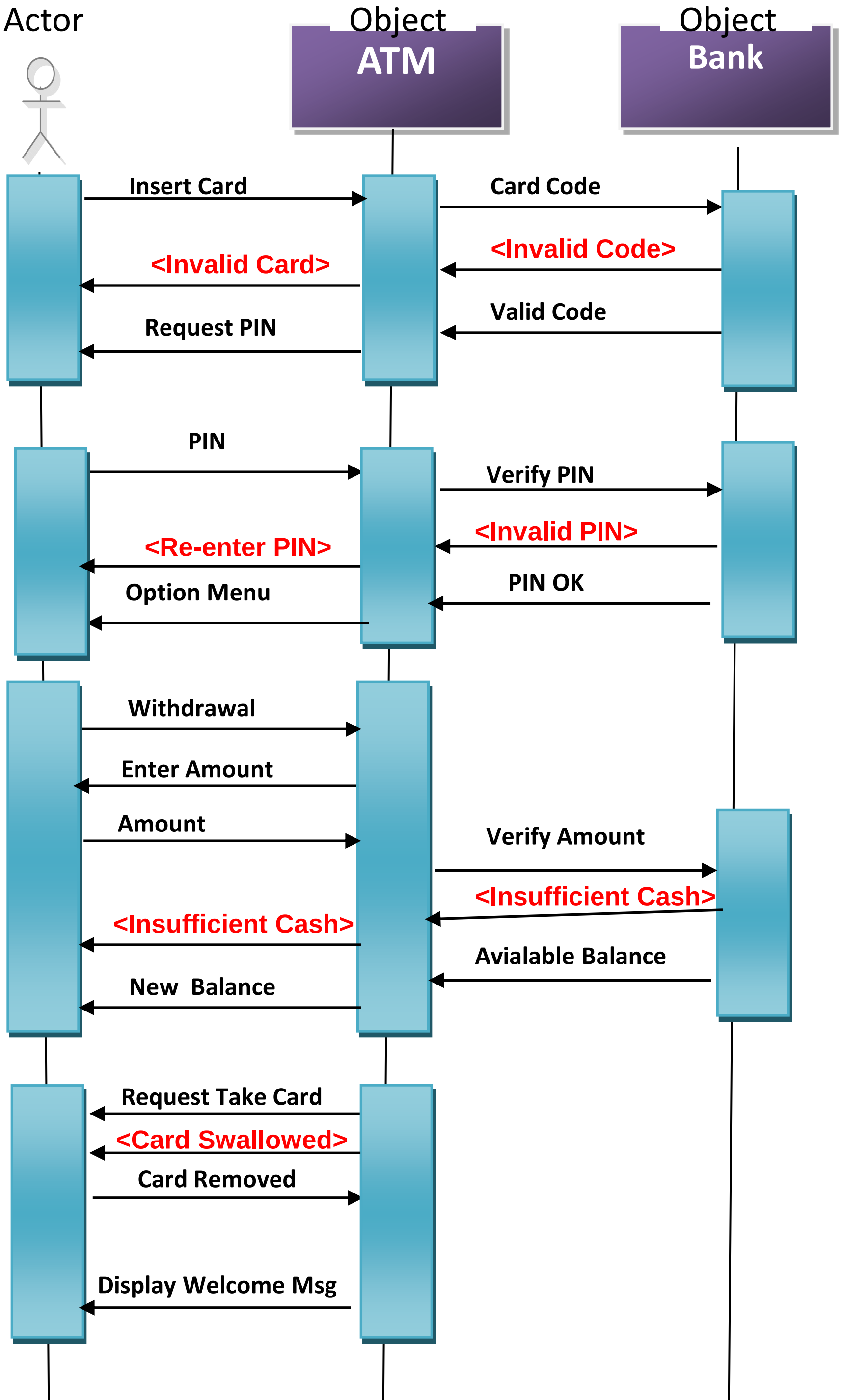
Issue of electronic items





Sequence Diagram:

ATM Deposit Transaction



Sequence Diagram:

ATM Withdrawal Transaction