USE CASE DIAGRAM

Scenario Based

Requirements Specification vs Analysis Model

Both focus on the requirements from the user's view of the system

- The requirements specification uses natural language (derived from the problem statement)
- The analysis model uses a formal or semi-formal notation (we use UML)

USE CASE DIAGRAM

- The use case represents the different ways in which a system can be used by the users
- A **use case** describes interactions of actors with the target system to perform a **function**.
- The use case model can be documented by drawing a use case diagram and writing an accompanying text elaborating the drawing (use case specification)

USE CASE DIAGRAMS

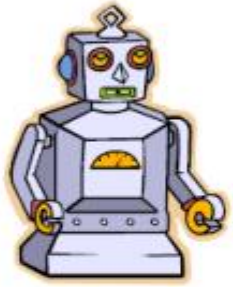
- Use case diagrams are used to visualize, specify, construct, and document the (intended) behavior of the system, during requirements capture and analysis.
- Provide a way for developers, domain experts and end-users to Communicate.
- Serve as basis for testing.
- The use case diagram consists of
  - Actor
  - Use cases
  - Relationships

ACTORS

External objects that produce/consume data:

- Must serve as sources and destinations for data
- Must be external to the system

Humans



Machines

External systems

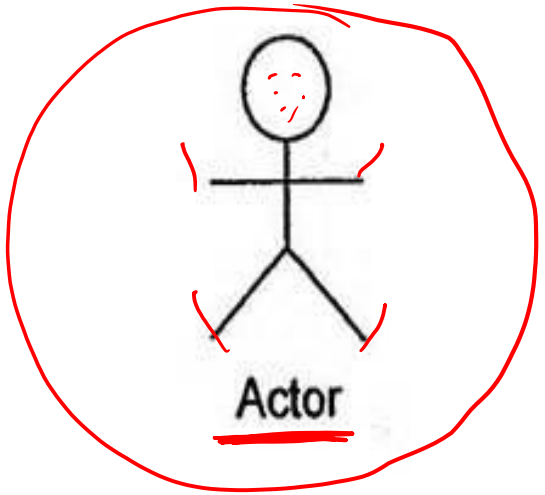
Organizational Units

Sensors

ACTOR

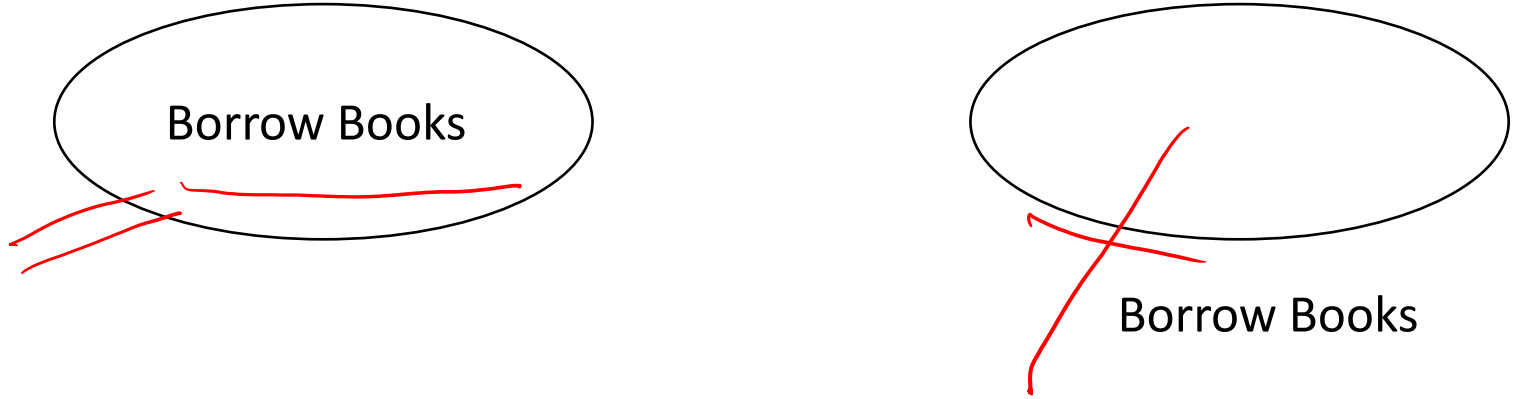
- Each Actor must be linked to a use case
- Represented by stick figure
- Labelled using a descriptive noun or phrase

  Eg: Librarian, Doctor, Student, Customer



USE CASE

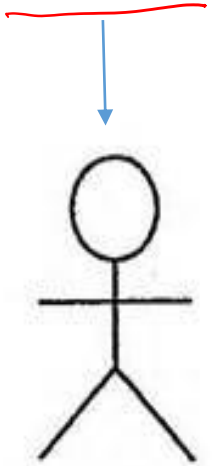
- Labelled using a descriptive verb-noun phrase
- Represented using an ellipse with name of the use case written inside the ellipse or written below the ellipse



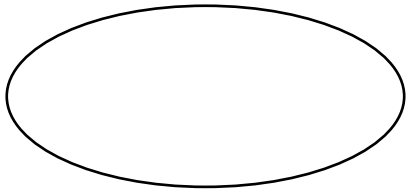
- Consider the following scenario:

"A patient calls the clinic to make an appointment for a yearly checkup"

- *A patient calls the clinic to make an appointment for a yearly checkup.*

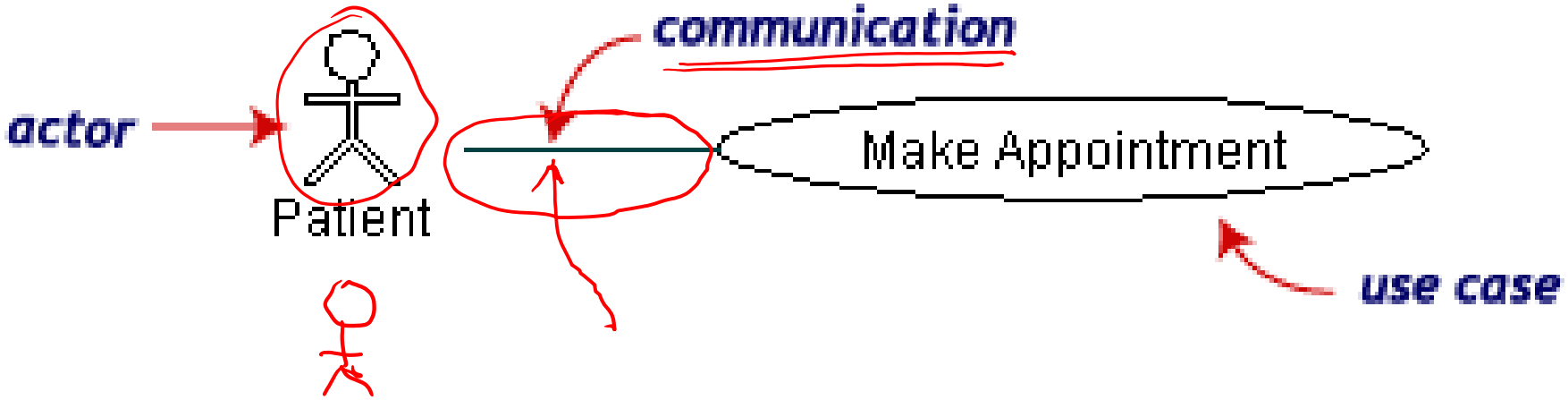


Actor



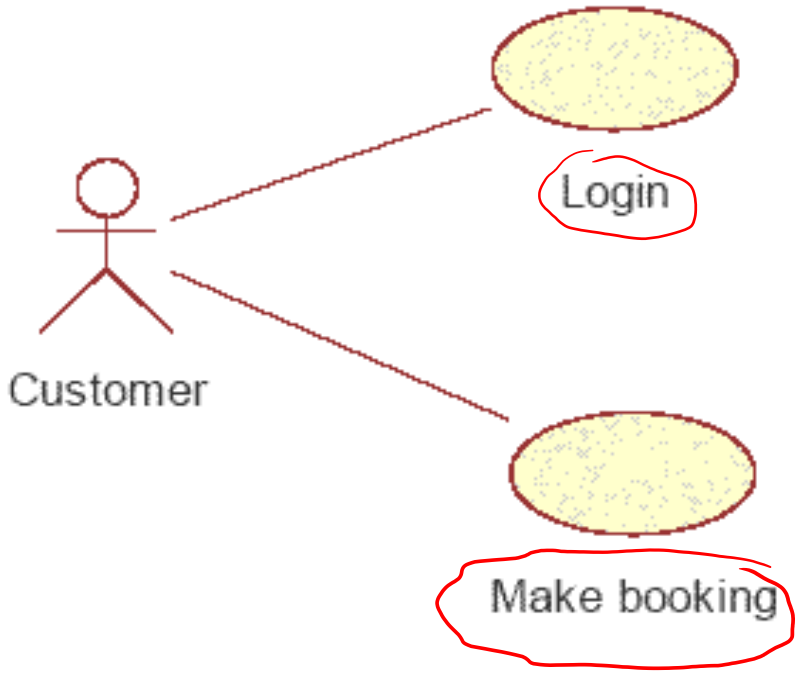
Use case

- The picture below is a **Make Appointment** use case for the medical clinic.
- The actor is a **Patient**. The connection between actor and use case is a **communication association** (or **communication** for short).



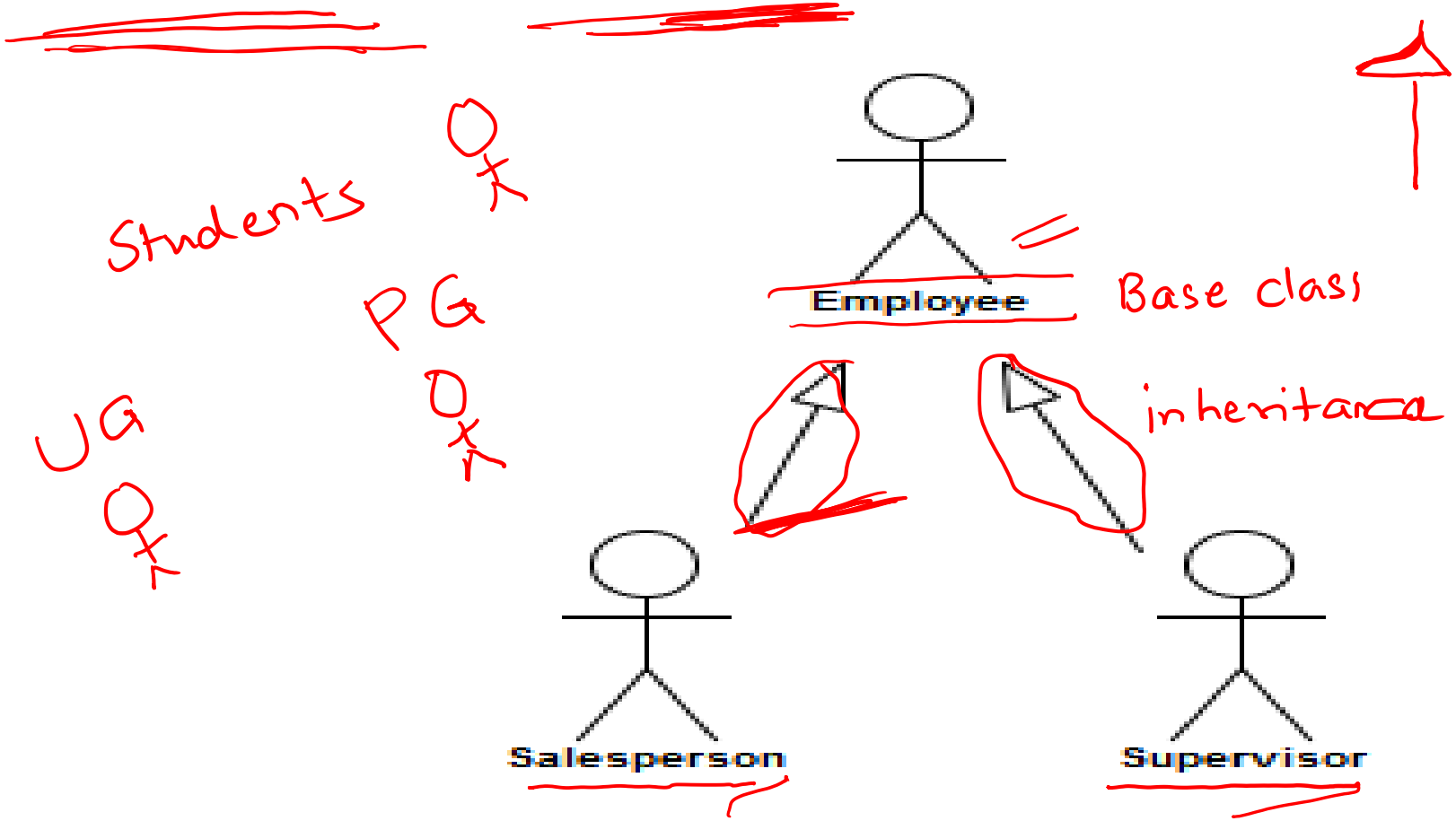
RELATIONSHIPS

- Represents communication between actor and use case
- Depicted by a line connecting actor and use case



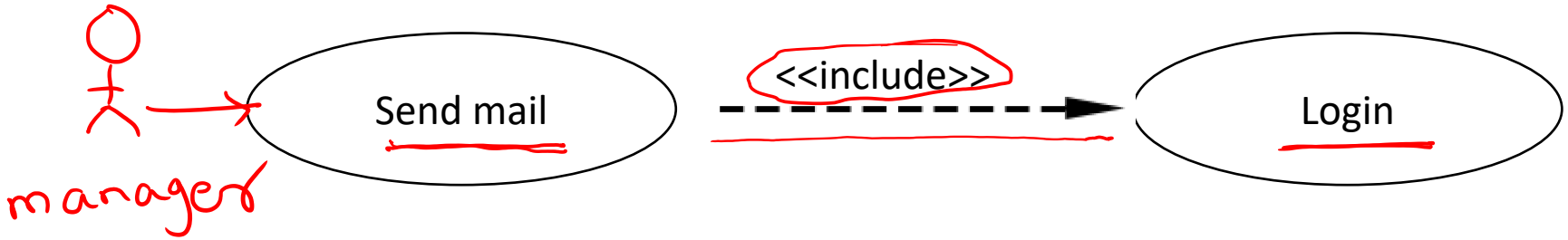
- Other Types of Relationships for Use Cases
  - Generalization (in old version 1.0 but removed from 2.0)
  - Include
  - Extend

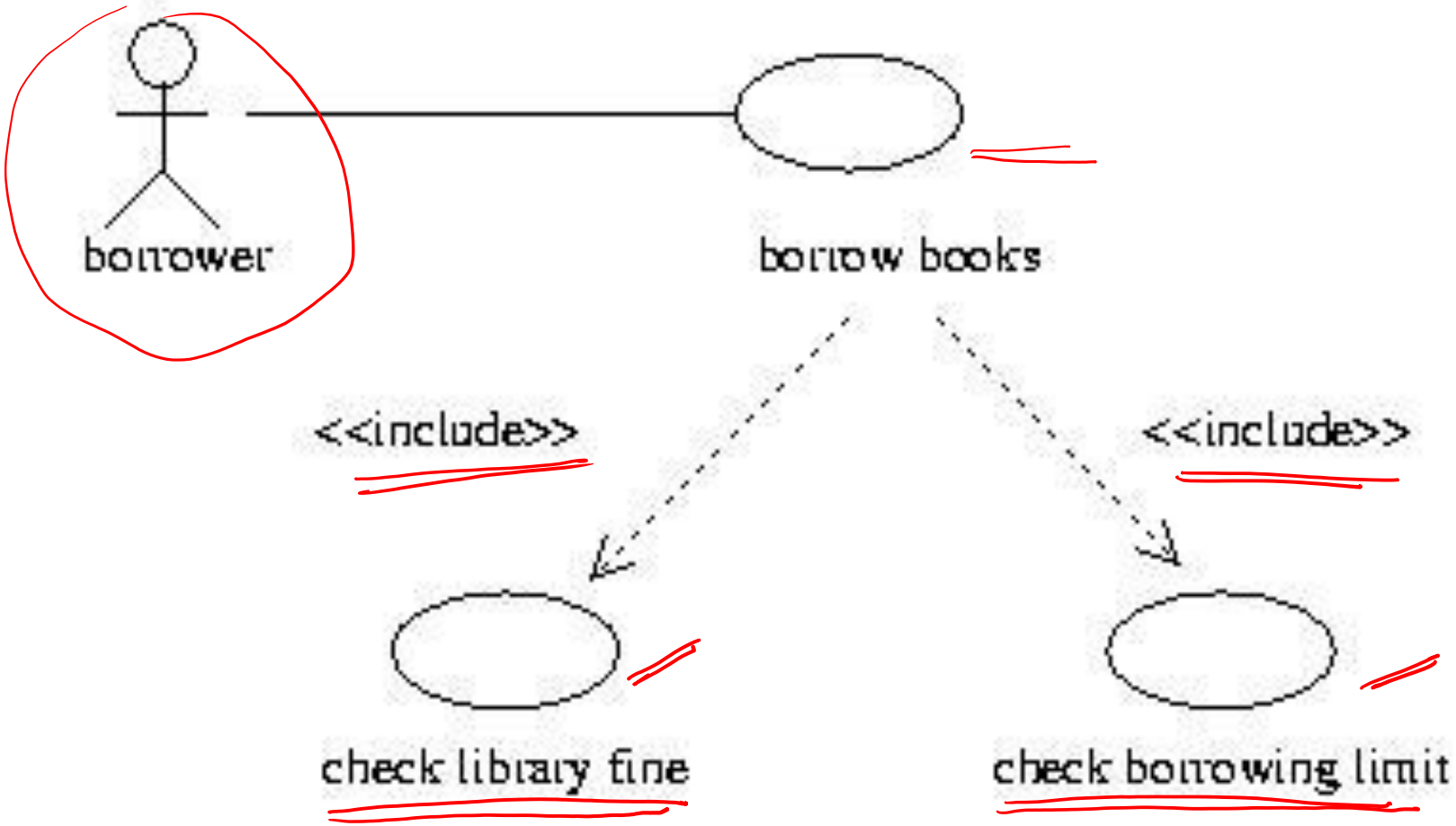
- Generalization between actors



Include Relationship

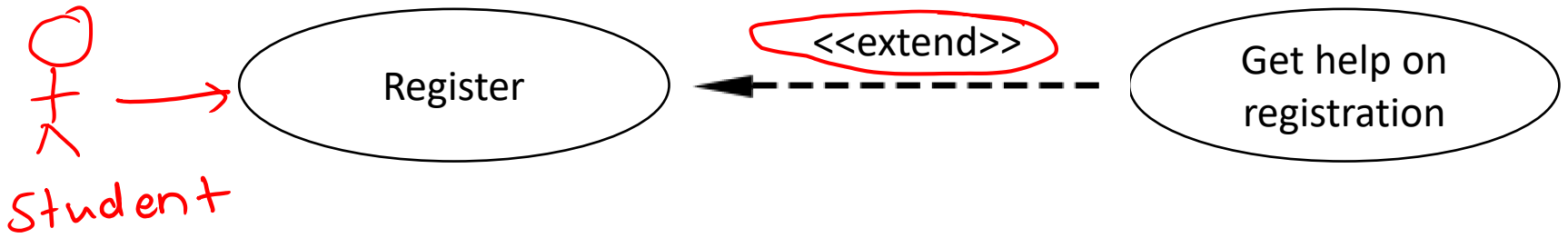
- Represents the inclusion of the functionality of one use case within another
- Arrow is drawn from the base use case to the used use case
- Write << include >> above arrowhead line

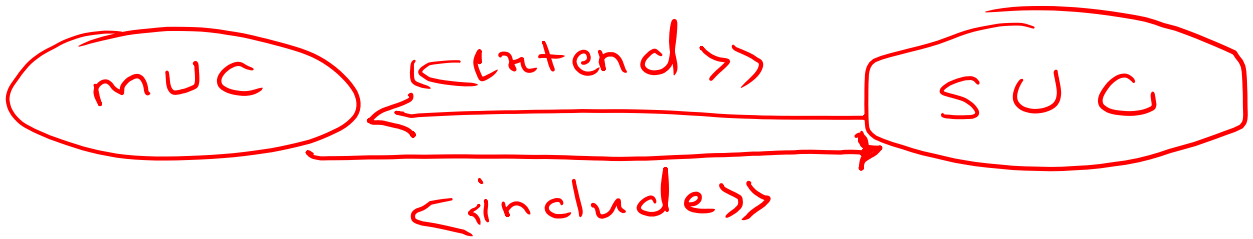


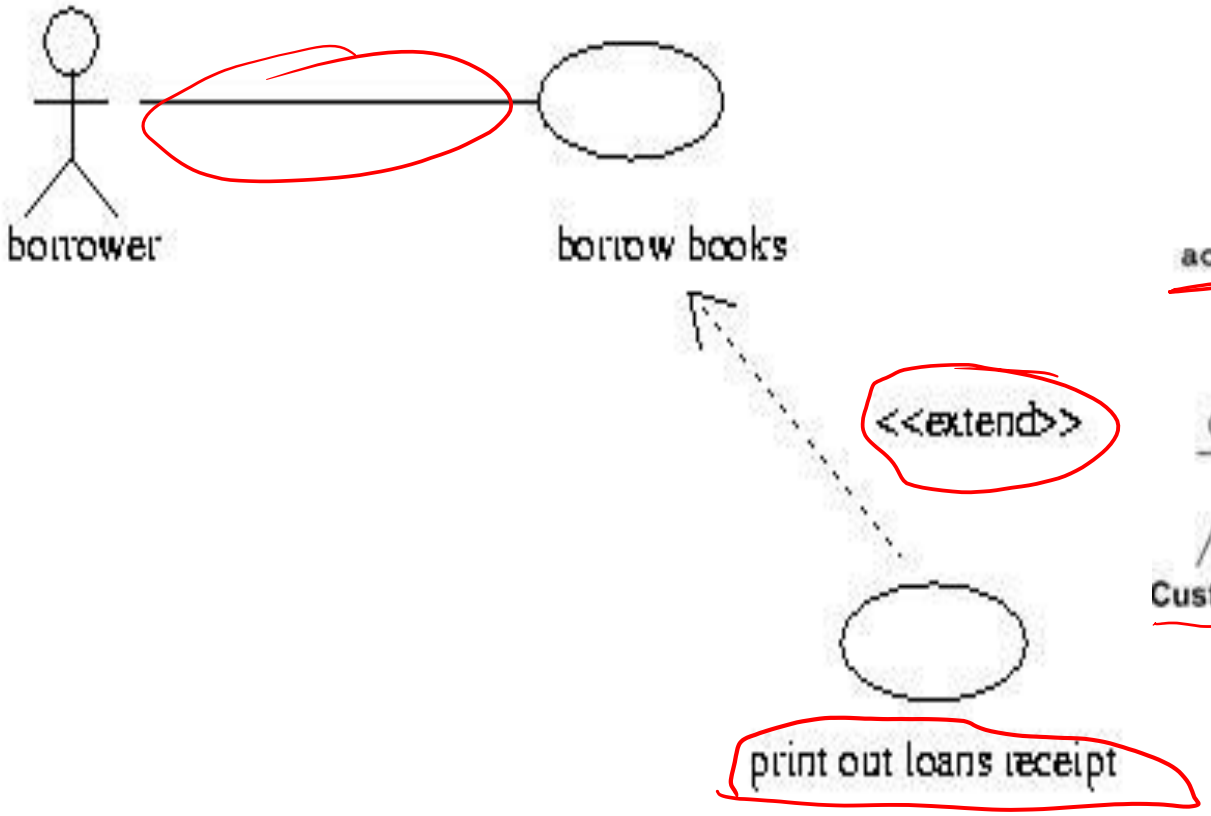

borrower
borrow books
<<include>>
<<include>>
check library fine
check borrowing limit

Extend relationship

- Represents the extension of the use case to include optional functionality
- Arrow is drawn from the extension use case to the base use case
- Write << extend >> above arrowhead line







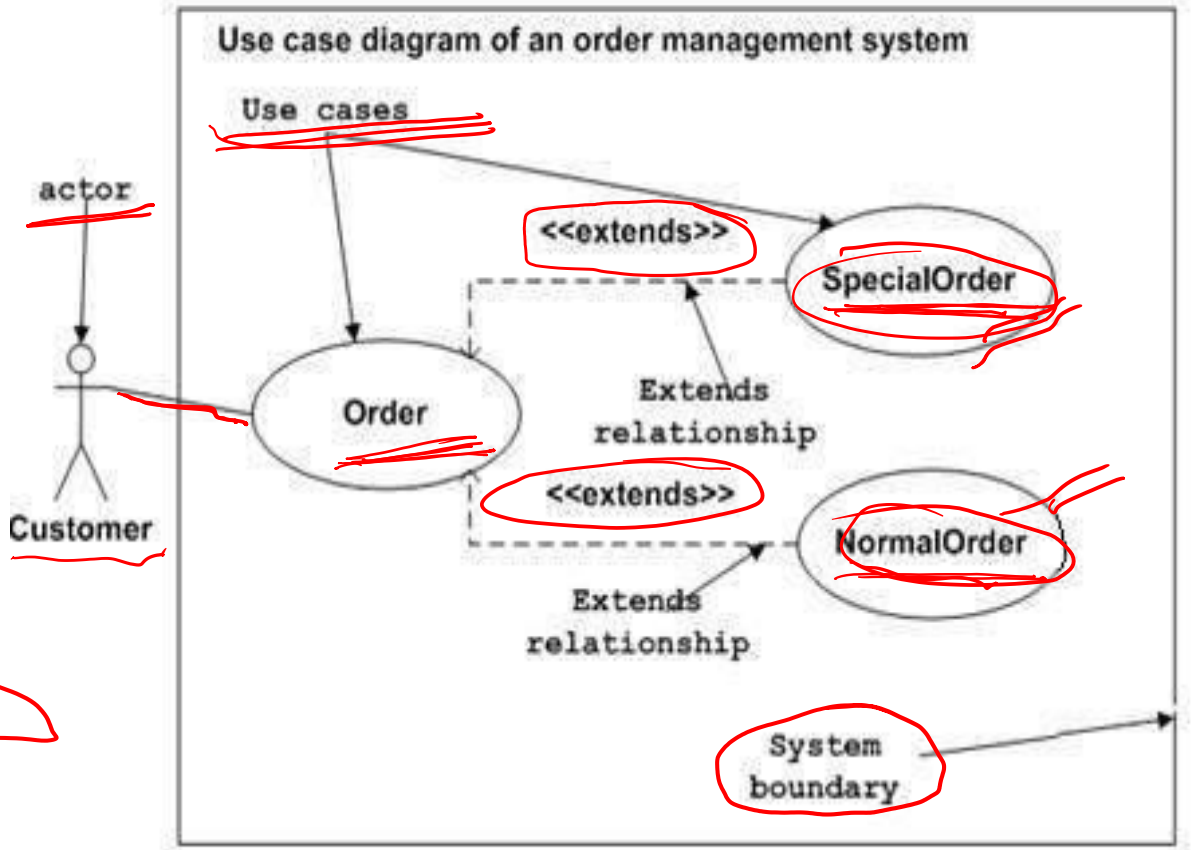
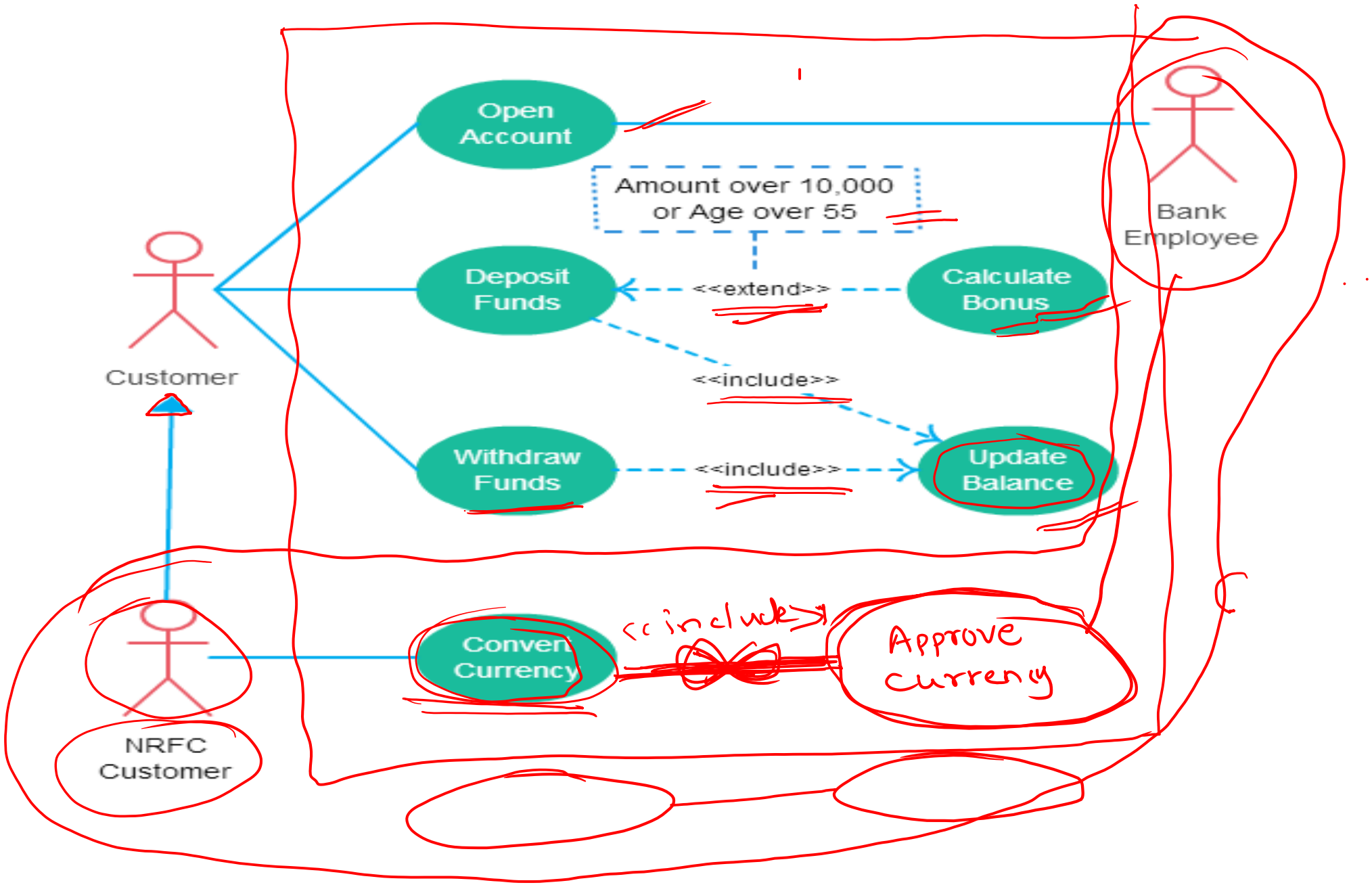
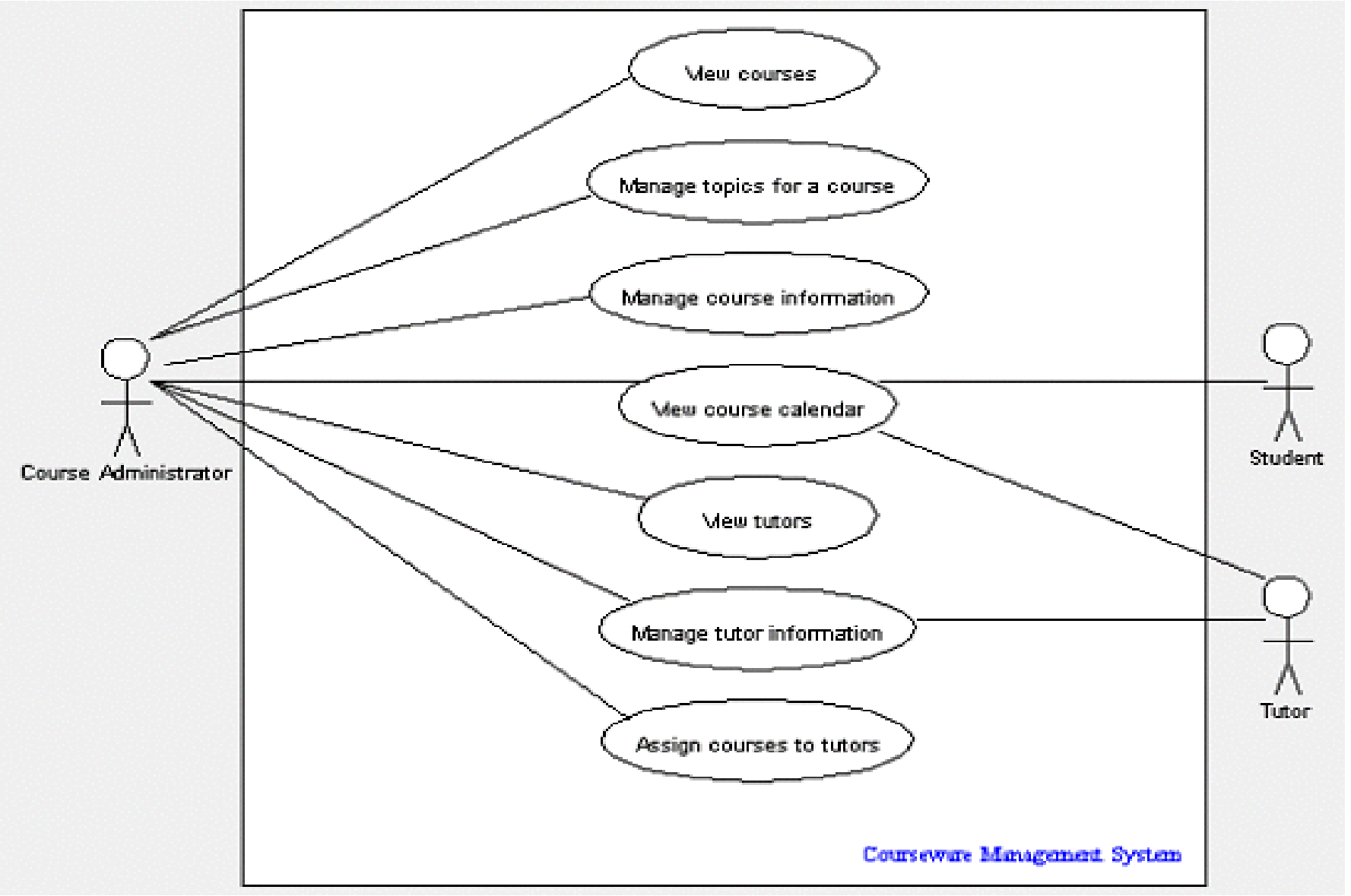


Figure: Sample Use Case diagram


Open Account
Amount over 10,000 or Age over 55
Bank Employee
Deposit Funds
<<extend>>
Calculate Bonus
Customer
<<include>>
Withdraw Funds
<<include>>
Update Balance
Convert Currency
<<include>>
Approve currency
NRFC Customer

USE CASE DIAGRAM EXAMPLE

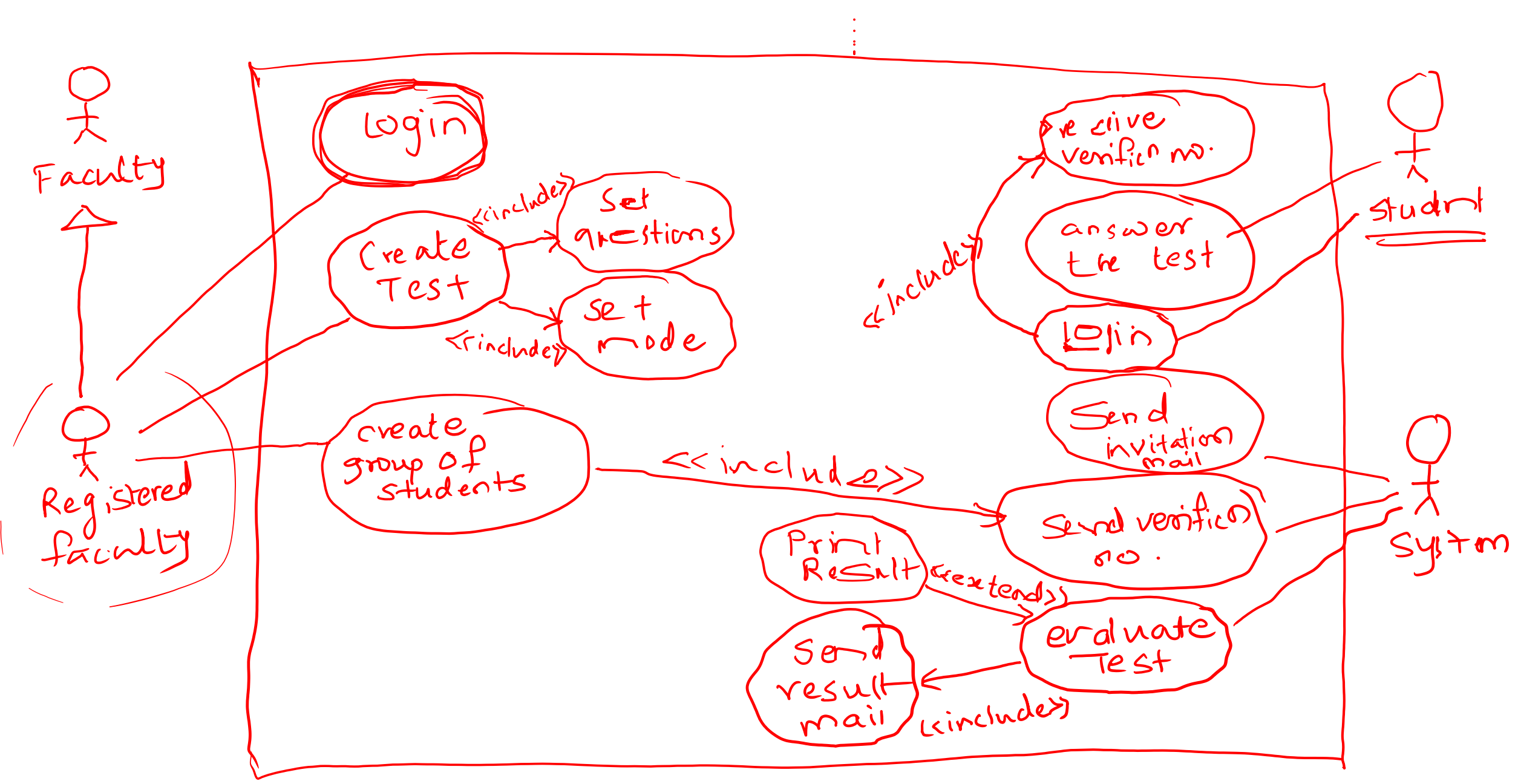


Problem Statement 1

The System is intended for faculty members to conduct online examination. Every registered faculty member can login to the system, create test, set the questions, set the mode for each test etc. Faculty member also decides which groups of students have to take a particular test and accordingly creates group of students. The system should send mail to the students inviting them to take the test. Students can login using the verification number received and then answer the test questions. The System should evaluate student answers and send the result via mail or can print the result.

Problem Statement 1

The System is basically intended for faculty members to conduct online examination. Every registered faculty member can login to the system, create test, set the questions, set the mode for each test etc. Faculty member also decides which groups of students have to take a particular test and accordingly creates group of students. The system should send mail to the students inviting them to take the test. Students can login using the verification number received and then answer the test questions. The System should evaluate student answers with that specified by faculty and send the result via mail.


Faculty
Registered faculty
Login
Create Test
<<include>>
Set questions
<<include>>
set mode
create group of students
<<include>>
receive verification no.
answer the test
<<include>>
Login
Send invitation mail
Send verification no.
Print Result
<<extend>>
evaluate Test
Send result mail
<<include>>
Student
System

Problem Statement 2

An **auto rental company** wants to develop an automated system that would handle car reservations, customer billing, and car auctions. Usually a customer reserves a car, picks it up, and then returns it after a certain period of time. At the time of pick up, the customer has the option to buy or waive collision insurance on the car. When the car is returned, the customer receives a bill and pays the specified amount. In addition to renting out cars, every six months or so, the auto rental company auctions the cars that have accumulated over 20,000 miles.

Problem Statement 3

Draw the use-case diagram for Hotel Information System. There are two types of customers: Tour-group customers and Individual customers. Both can book, cancel, check-in and check-out of a room by Phone or via the Internet. There are booking process clerk and reception staff who manage it. A customer can pay his bill by credit card or by cash. After frequent booking of hotel more than 5 times, customer can go for booking of special package of stay for 6^{th} time. Clerk also maintains the resources available in the hotel and reception staff takes care of feedback collection from the customer.

Problem Statement 4

A student may register for classes during a specified registration period. To register, a student must see their advisor. The advisor must approve each course that the student has selected. The advisor will use the registration system to determine if the student has met the course prerequisites, is in good academic standings and is eligible to register. If the advisor approves the courses, the advisor enters the student's college id into the course registration system. The course registration number for each course is entered. The course description, course number and section for those courses will automatically display. The system will check for schedule conflicts before saving the registrations. A bill for the courses will print in the financial administrator’s office. The student should proceed to pick it up. Faculty can use the registration system to check enrollments in their classes, get a class list, check a student's transcript, look up a student's phone number and other such student information. The registrar can use the registration system to enter new classes for an upcoming semester, cancel a class, and check conflicts in classroom/faculty assignments. Admissions use the registration system to add new students. Enrollment services use the registration system to report on retention, update student information, and check fulfillment of graduation requirements for those students planning to graduate.

Types of requirements

- Functional requirements
  - Describe the interactions between the system and its environment independent from the implementation

    “An operator must be able to define a new game. “
- Nonfunctional requirements
  - Aspects not directly related to functional behavior.

    “The response time must be less than 1 second”
- Constraints
  - Imposed by the client or the environment
    - “The implementation language must be Java “
  - Called “Pseudo requirements” in the text book.

Functional vs. Nonfunctional Requirements

Functional Requirements

- Describe user tasks that the system needs to support
- Phrased as actions
  - "Advertise a new league"
  - "Schedule tournament"
  - "Notify an interest group"

Nonfunctional Requirements

- Describe properties of the system or the domain
- Phrased as constraints or negative assertions
  - "All user inputs should be acknowledged within 1 second"
  - "A system crash should not result in data loss".

Types of Nonfunctional Requirements

- Usability
- Reliability
  - Robustness
  - Safety
- Performance
  - Response time
  - Scalability
  - Throughput
  - Availability
- Supportability
  - Adaptability
  - Maintainability

Quality requirements

- Implementation
- Interface
- Operation
- Packaging
- Legal
  - Licensing (GPL, LGPL)
  - Certification
  - Regulation

Constraints or
Pseudo requirements

Nonfunctional Requirements: Examples

- “Spectators must be able to watch a match without prior registration and without prior knowledge of the match.”
  - *Usability Requirement*
- “The system must support 10 parallel tournaments”
  - *Performance Requirement*
- “The operator must be able to add new games without modifications to the existing system.”
  - *Supportability Requirement*

USE CASE SPECIFICATION

(detailed version of a use case)

Use Case Specification

1. **Brief Description** - (about the use case)
2. **Actors** – list out the actors involved
3. **Preconditions** - A textual description that defines any constraints on the system at the time the use case may start.
4. **Basic flow of events** - describes what "normally" happens when the use case is performed.
5. **Alternative flows** - covers behavior of an optional or exceptional character relative to normal behavior, and also variations of the normal behavior. Think of the alternate flows of events as "detours" from the basic flow of events.
6. **Postconditions** - A textual description that defines any constraints on the system at the time the use case will terminate.

Use-Case Specification: Withdraw Cash

1. Brief Description

This use case describes how a Bank Customer uses an ATM to withdraw money from a bank account.

2. Actors

- Customer
- Bank

3. Preconditions:

- The bank Customer must possess a **bank card**.
- The network connection to the **Bank System** must be active.
- The system must have at least some cash that can be dispensed.
- The cash withdrawal **service option** must be available.

4. Basic Flow of Events

4.1 Insert Card

- The use case begins when the actor **Customer** inserts his/her **bank card** into the card reader on the ATM.
- The system allocates an **ATM session identifier** to enable errors to be tracked and synchronized between the ATM and the Bank System.

4.2 Read Card

- The system reads the **bank card information** from the card.

4.3 Authenticate Customer

- Authenticate the use of the **bank card** by the individual using the machine

4.4 Select Withdrawal

- The system displays the **service options** that are currently available on the machine.
- The Customer selects to withdraw cash.

4.5 Select Amount

- The system prompts for the amount to be withdrawn by displaying the list of standard withdrawal amounts.
- The Customer selects an amount to be withdrawn.

4.6 Confirm Withdrawal

- Customer gives conformation to withdraw cash

4.7 Eject Card

- The system ejects the Customer's **bank card**.
- The Customer takes the **bank card** from the machine.

4.8 Dispense Cash

- The system dispenses the requested amount of cash to the Customer.
- The system records a **transaction log** entry for the withdrawal.

4.9 Use Case Ends

- The use case ends.

5.Alternative Flows

5.1 Invalid User

5.2 Amount Exceeds Withdrawal limit

5.3 Insufficient cash

5.4 Money not removed

6. Postconditions

- The ATM has returned the card and dispensed the cash to the Customer and the withdrawal is registered on the Customer’s account.
- The ATM has returned the card to the Customer and no withdrawal is registered on the Customer’s account.
- The ATM has returned the card but has not supplied the amount of cash registered as withdrawn from the Customer’s account. The discrepancy is registered in the ATM’s log.
- The ATM has kept the card, no withdrawal has registered on the Customer’s account and the Customer has been notified where to contact for more information.

Write a use case specification for

i) **book ticket** use case in a **railway ticket booking system.**

ii) **Order item** use case in an **online shopping website.**

Millennium Travel Corporation (MTC) travel agency plans to become a market leader by augmenting its human travel agents with an automated travel agent system for processing flight reservations. The automated travel agent will intermediate between travelers and the MTC corporate computing system, which interfaces with commercial airline reservation services. Like a human travel agent, it will assist travelers in booking, changing, and canceling flight reservations. If, for any reason, a traveler making a flight reservation travel request prefers human assistance, she will have the option to interact directly with a human travel agent.

The MTC automated travel agent system will process a wide range of flight reservation service requests. These include but are not limited to: inquiring about flights and airfares, making, changing, and canceling traveler profiles and accounts, booking, changing, confirming, and canceling flight reservations, generating travel itineraries.

A user with a valid system account and a valid travel account logs in to the system, requests to book a flight reservation, selects a flight, selects a payment method, and specifies delivery services for the flight tickets and travel itineraries.

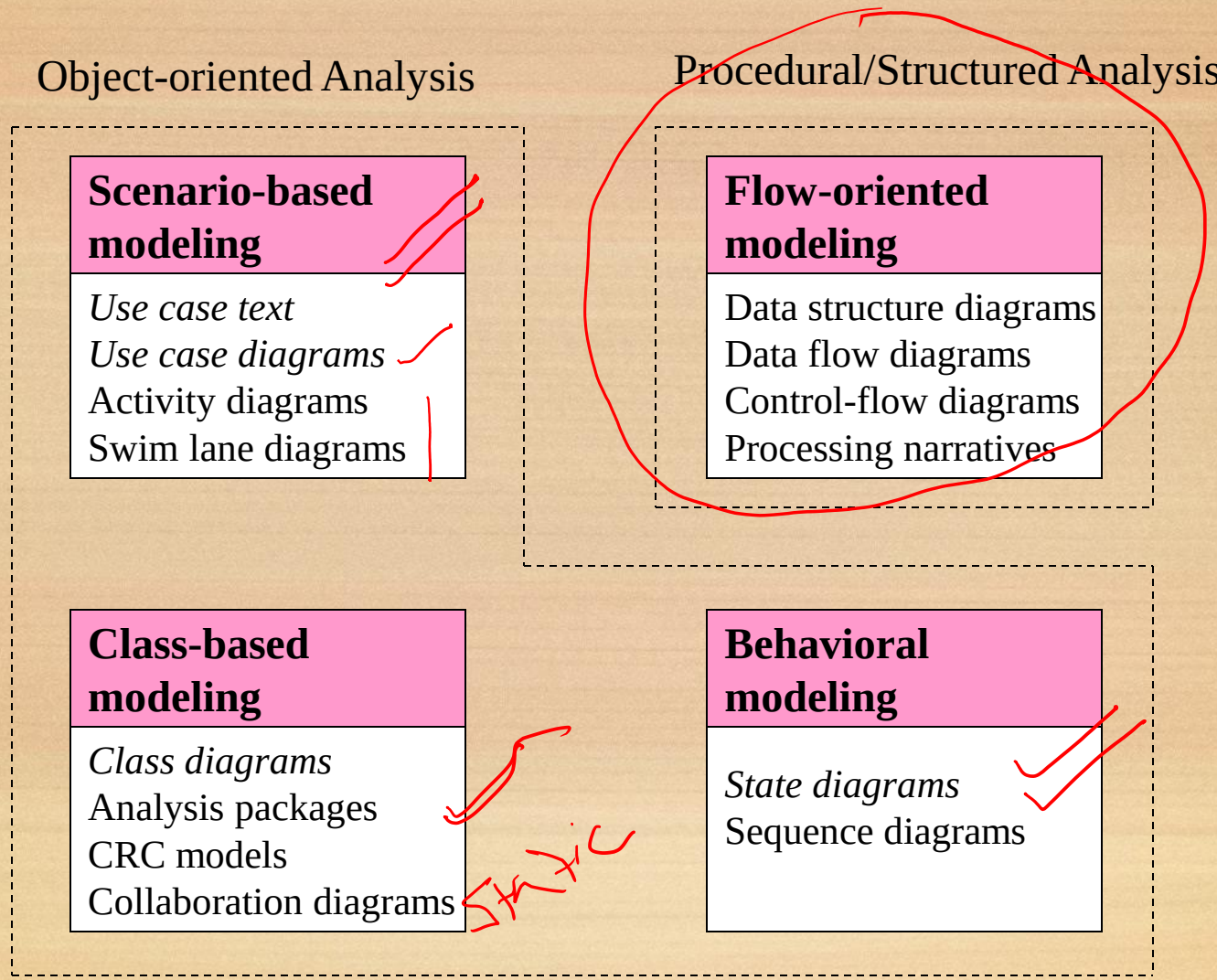
The travel agent system must be capable of providing fast, accurate, and courteous ("user friendly") services for all requests supported. The system must be able to answer inquiries about flights and fares, generate, modify, and cancel traveler profiles and travel accounts, make, change, complete, and cancel reservations, obtain payment method and verify traveler credit line, generate travel itineraries and arrange for delivery of flight tickets and flight itineraries.

Scenario-Based Modeling
Activity Diagram : Control Flow

Analysis Modeling Approaches

- Structured analysis
  - Considers data and the processes that transform the data as separate entities
  - Data is modeled in terms of only attributes and relationships (but no operations)
  - Processes are modeled to show the 1) input data, 2) the transformation that occurs on that data, and 3) the resulting output data
- Object-oriented analysis
  - Focuses on the definition of classes and the manner in which they collaborate with one another to fulfill customer requirements

Elements of the Analysis Model

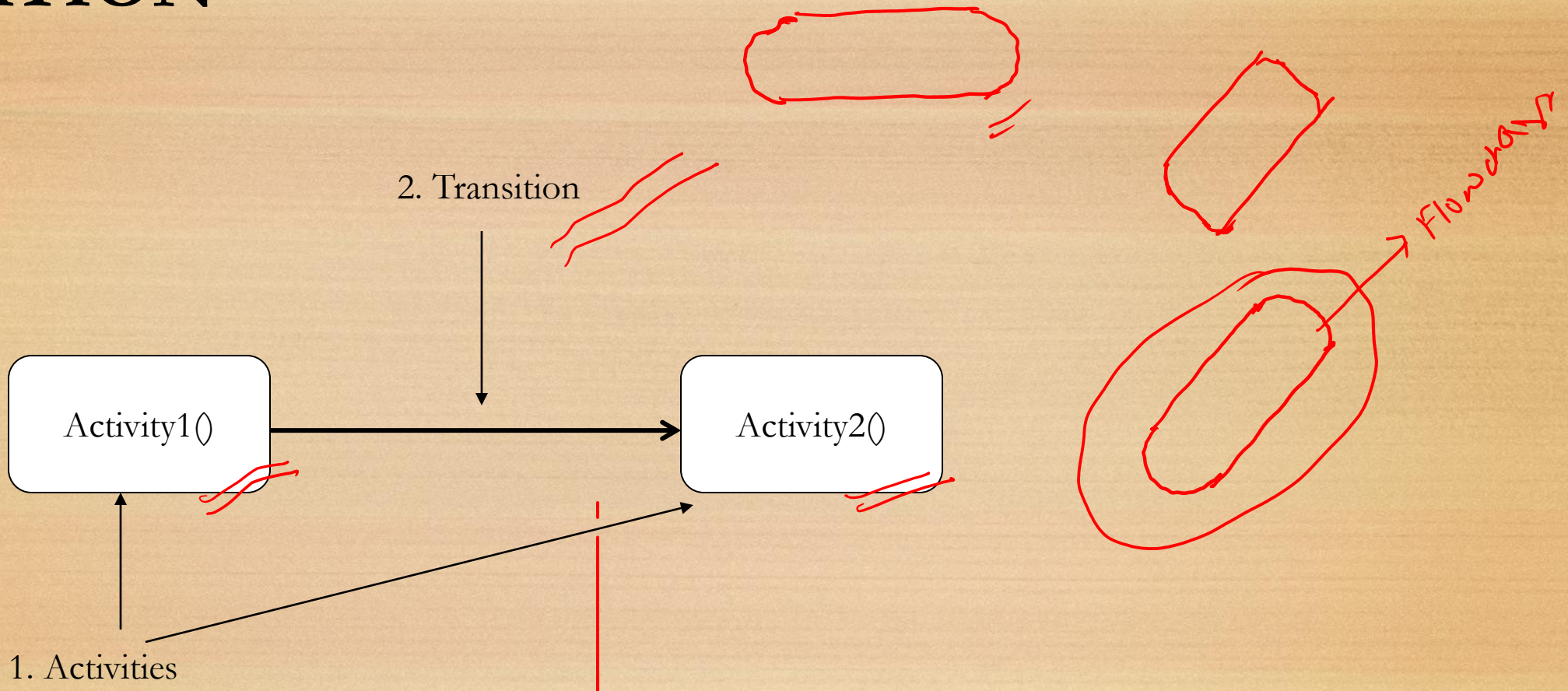


Developing an Activity Diagram

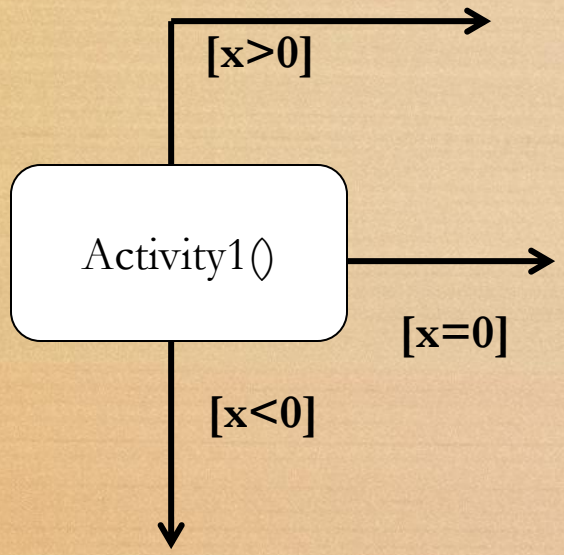
Activities

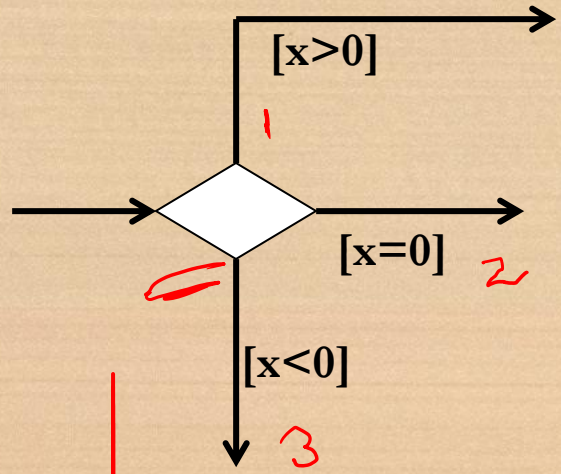
- Describes how activities are coordinated.
- Is particularly useful when you know that an operation has to achieve a number of different things, and you want to model what the essential dependencies between them are, before you decide in what order to do them.
- Record the dependencies between activities, such as which things can happen in parallel and what must be finished before something else can start.
- Represents the workflow of the process.

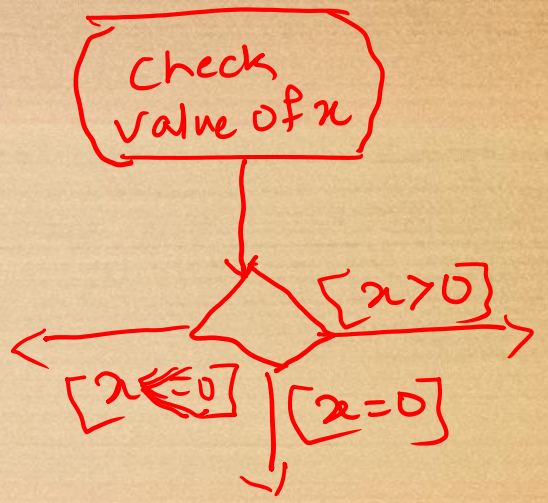
NOTATION



NOTATION - 2



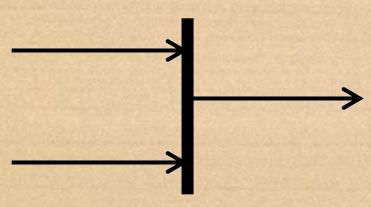




3. Decision Diamond

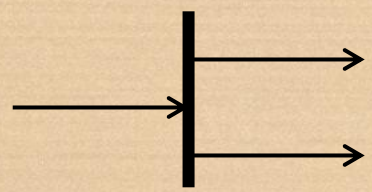
[ ]
Guard condn

NOTATION - 3



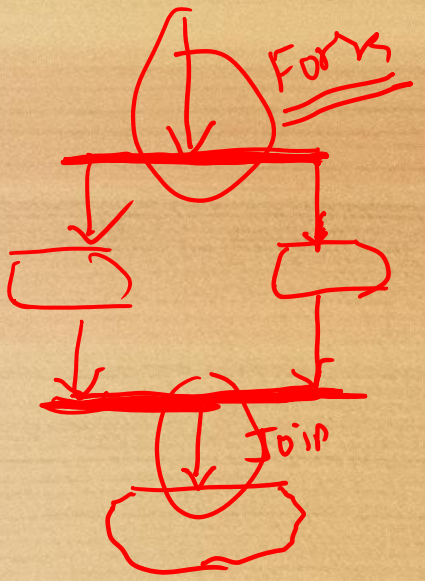
4.1 Synch. Bar (Join)

2



4.2 Splitting Bar (Fork)

1

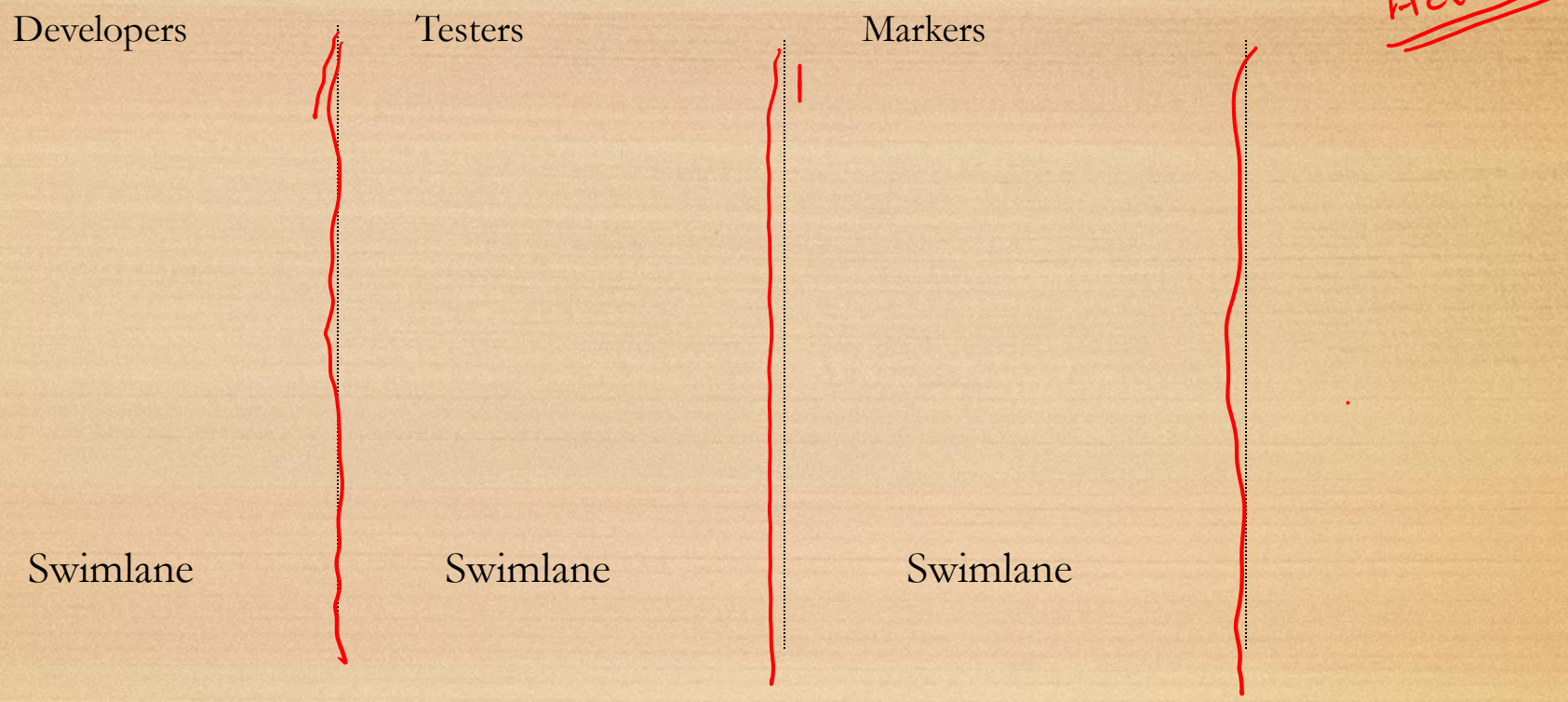


Notation - 3

5. Start & Stop Markers

Notation - 4

- Simple Activity diagram
- Swimlane diagram ✓
  - Actors



Application/Department/Group/Role Boundaries

Activity Diagrams

- Activity diagrams commonly contain
  - Activity states and action states
  - Transitions
  - Objects

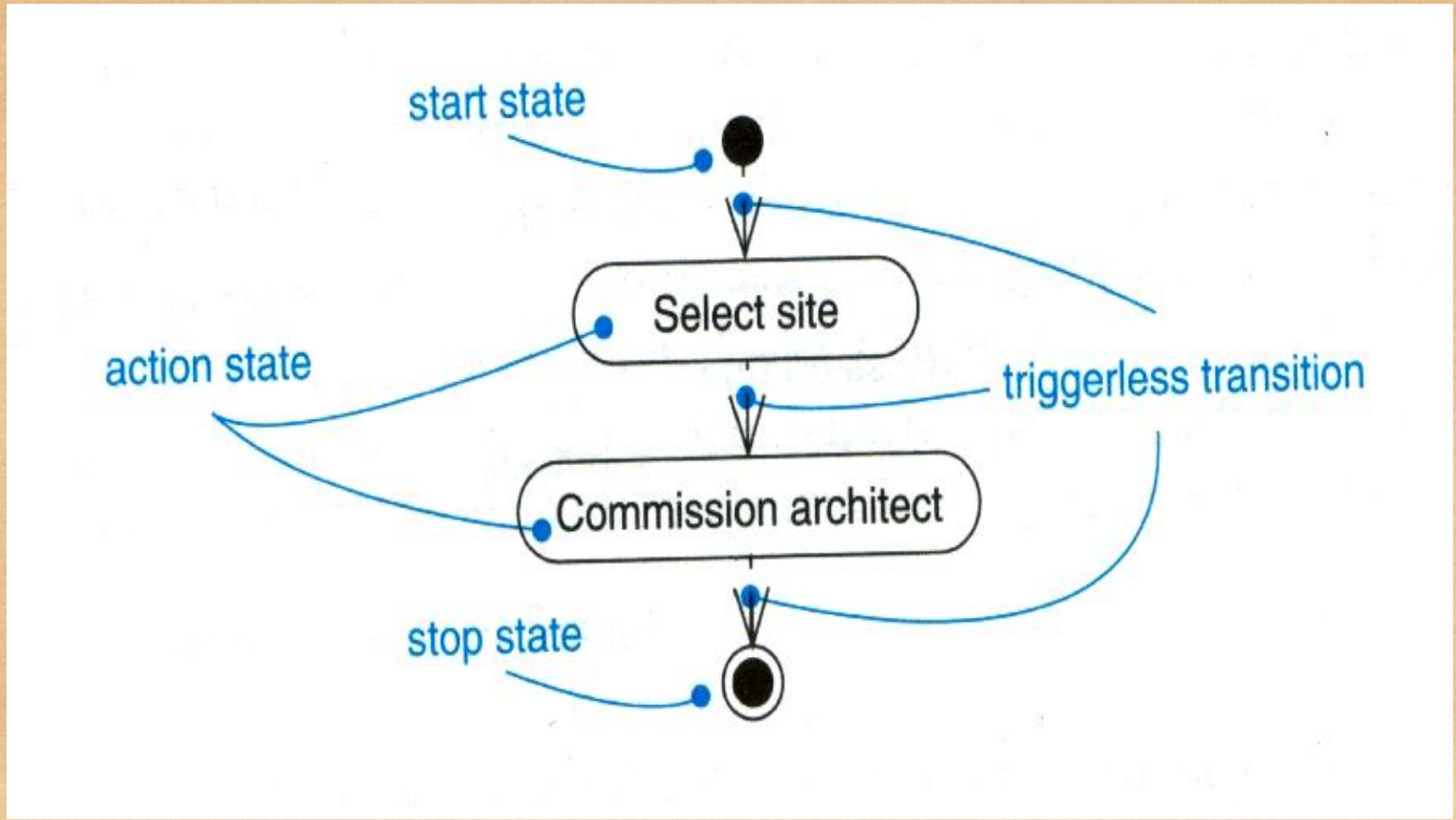
Action States and Activity States

- Action states are atomic and cannot be decomposed
  - Work of the action state is not interrupted
- Activity states can be further decomposed
  - Their activity being represented by other activity diagrams
  - They may be interrupted

Transitions

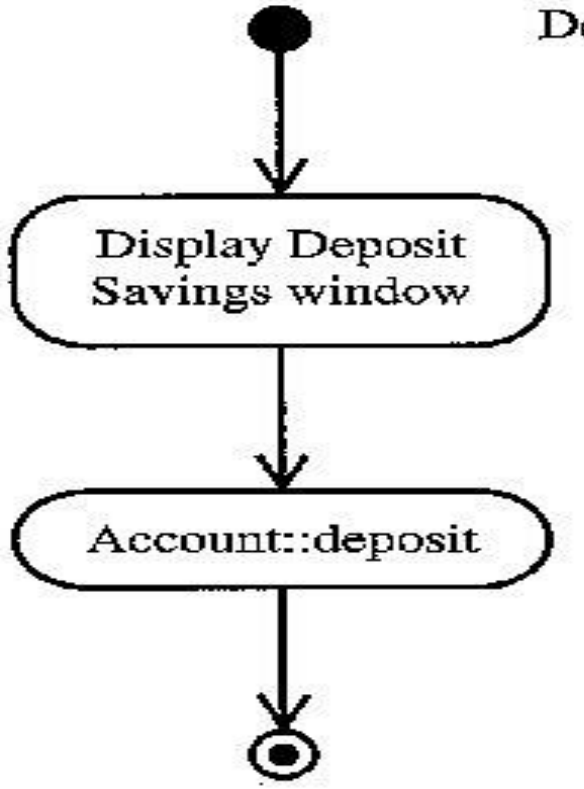
- When the action or activity of a state completes, flow of control passes immediately to the next action or activity state
- A flow of control has to start and end someplace
  - initial state -- a solid ball
  - stop state -- a solid ball inside a circle

Transitions



Activity Diagram: Example (1)

FIGURE 12–24
Activity diagram for processing a deposit to a savings account.



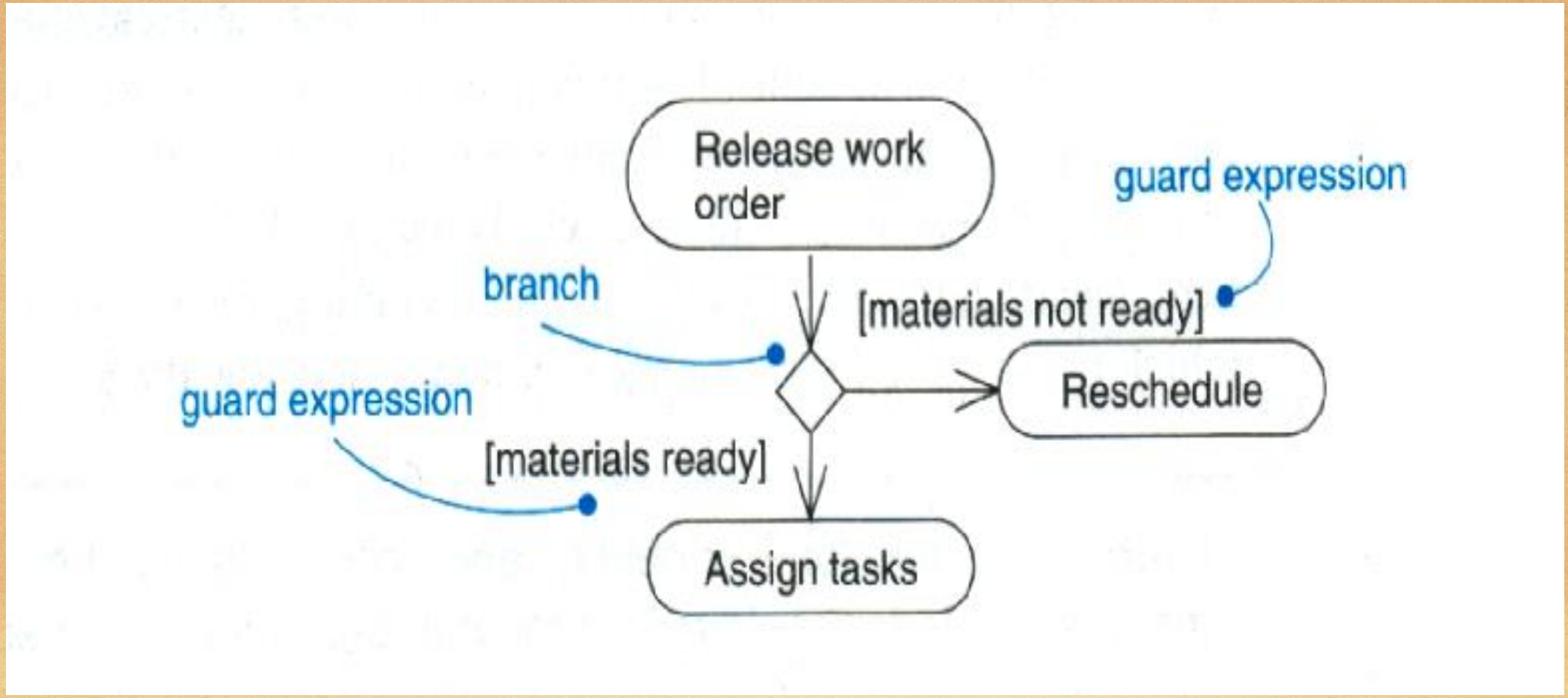
Deposit Savings button is pressed in the BankUI window

Account::deposit (anAmount)

Branching

- A branch specifies alternate paths taken based on some Boolean expression
- A branch may have one incoming transition and two or more outgoing ones

Branching


Release work order
guard expression
branch
[materials not ready]
guard expression
Reschedule
[materials ready]
Assign tasks

Activity Diagram: Example

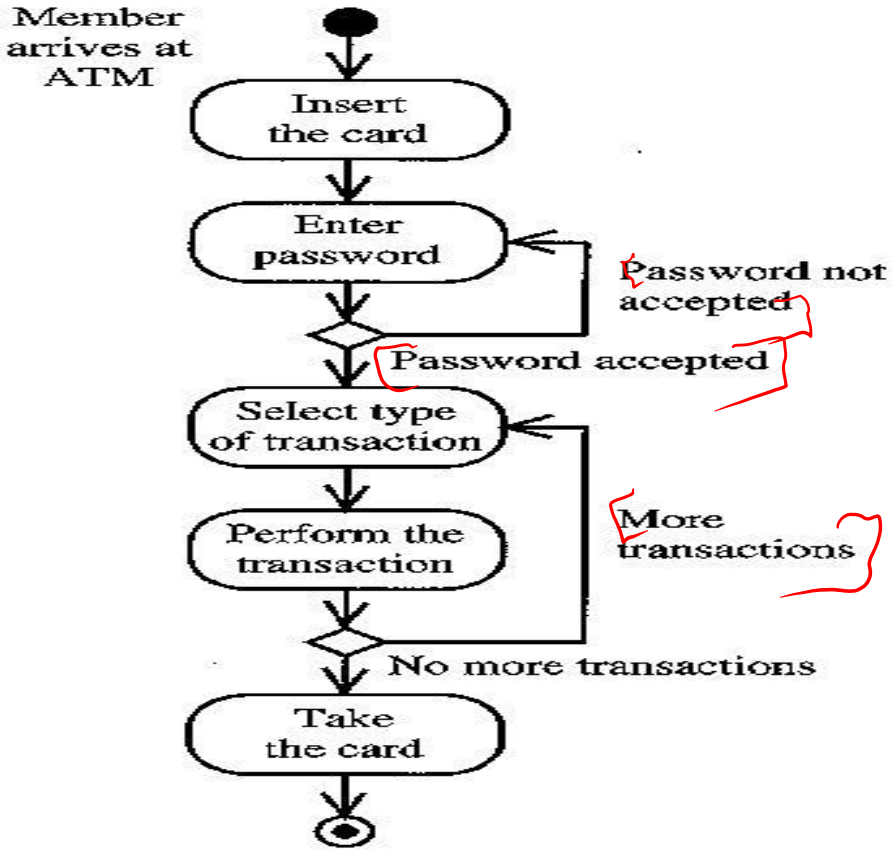


FIGURE 6–8
Activities involved in an ATM transaction.

Forking and Joining

- Use a synchronization bar to specify the forking and joining of parallel flows of control
- A synchronization bar is rendered as a thick horizontal or vertical line

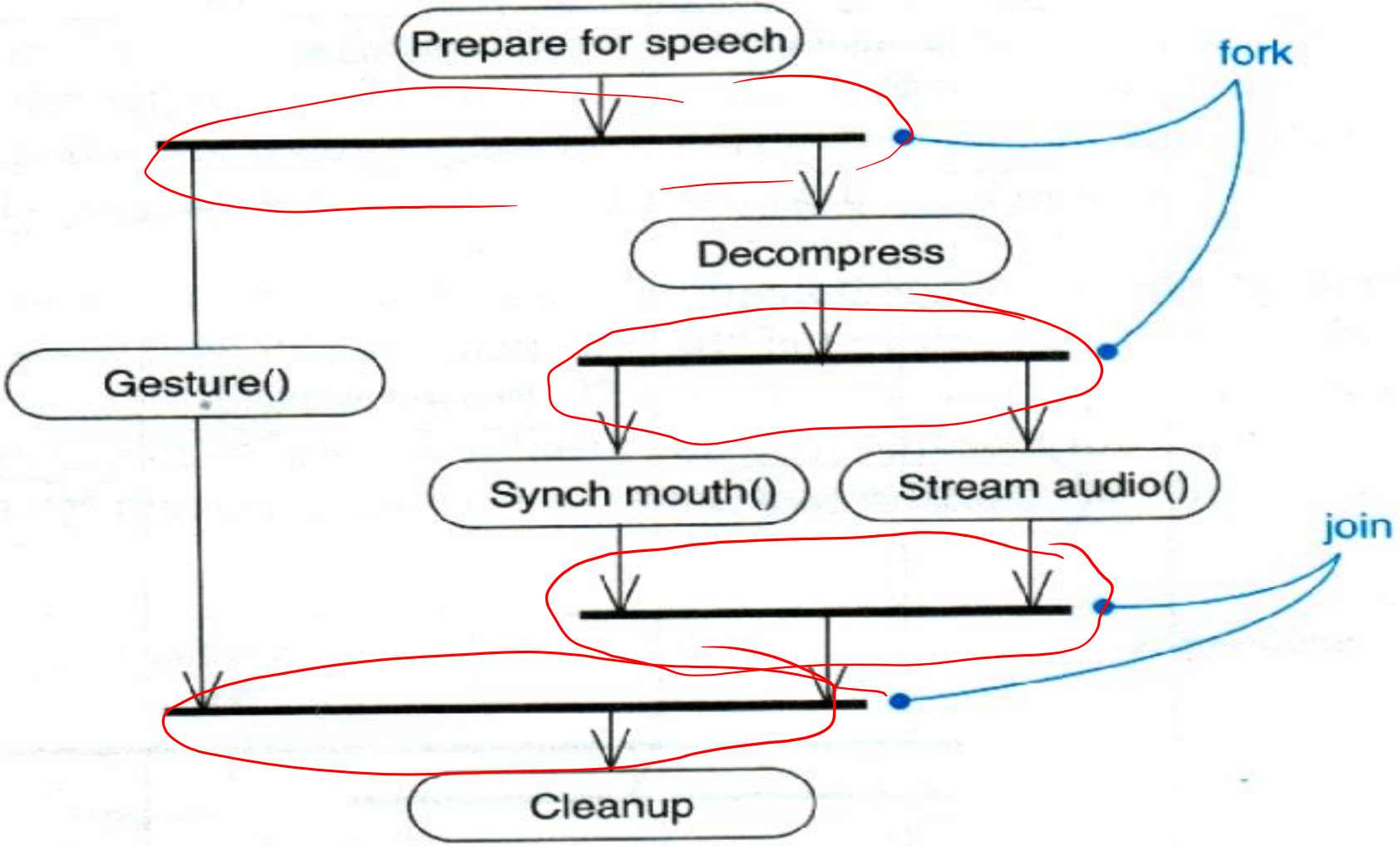
Fork

- A fork may have one incoming transitions and two or more outgoing transitions
  - each transition represents an independent flow of control
  - conceptually, the activities of each of outgoing transitions are concurrent
    - either truly concurrent (multiple nodes)
    - or sequential yet interleaved (one node)

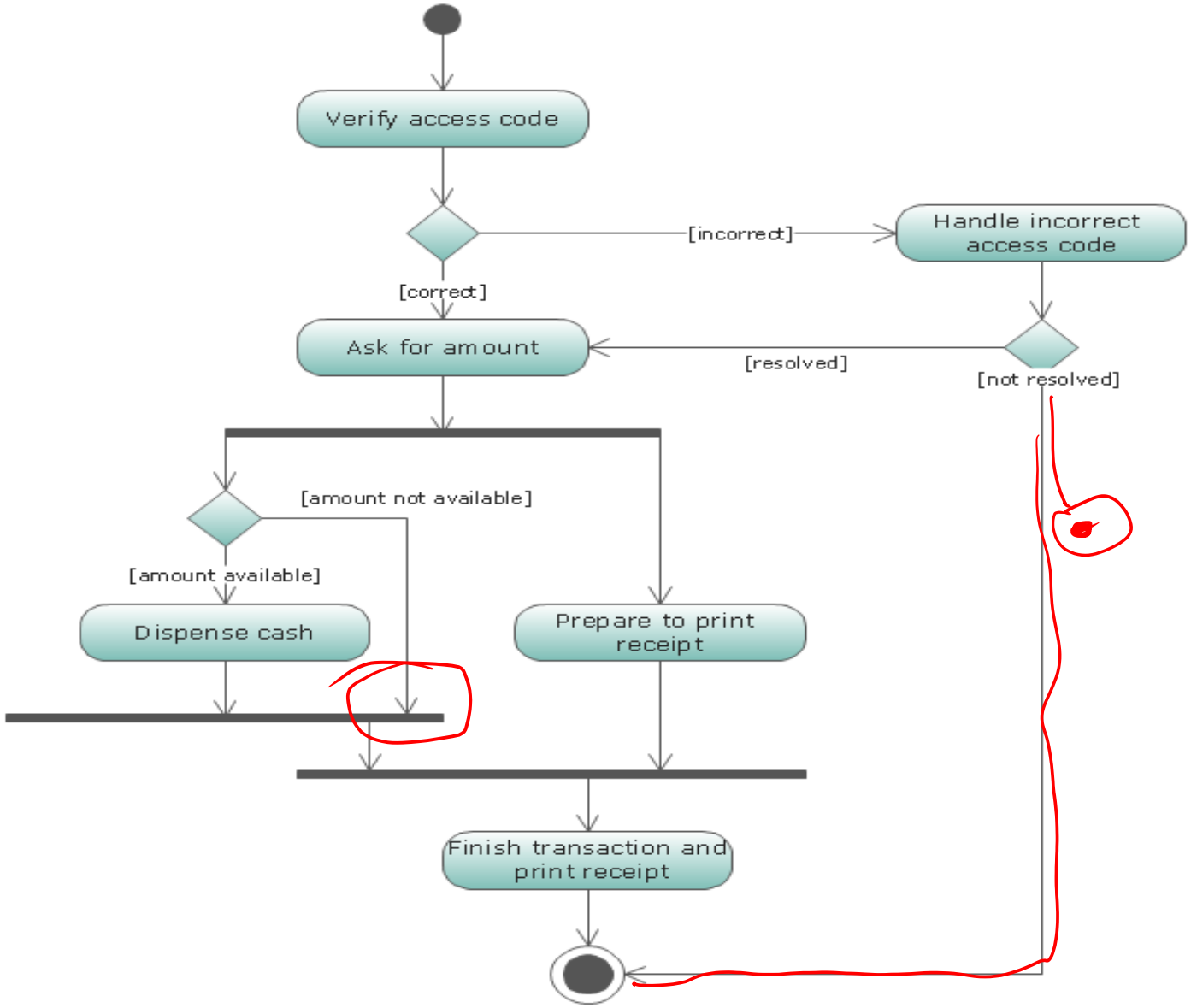
Join

- A join may have two or more incoming transitions and one outgoing transition
  - above the join, the activities associated with each of these paths continues in parallel
  - at the join, the concurrent flows synchronize
    - each waits until all incoming flows have reached the join, at which point one flow of control continues on below the join

Fork and Join Example



UML Activity Diagram



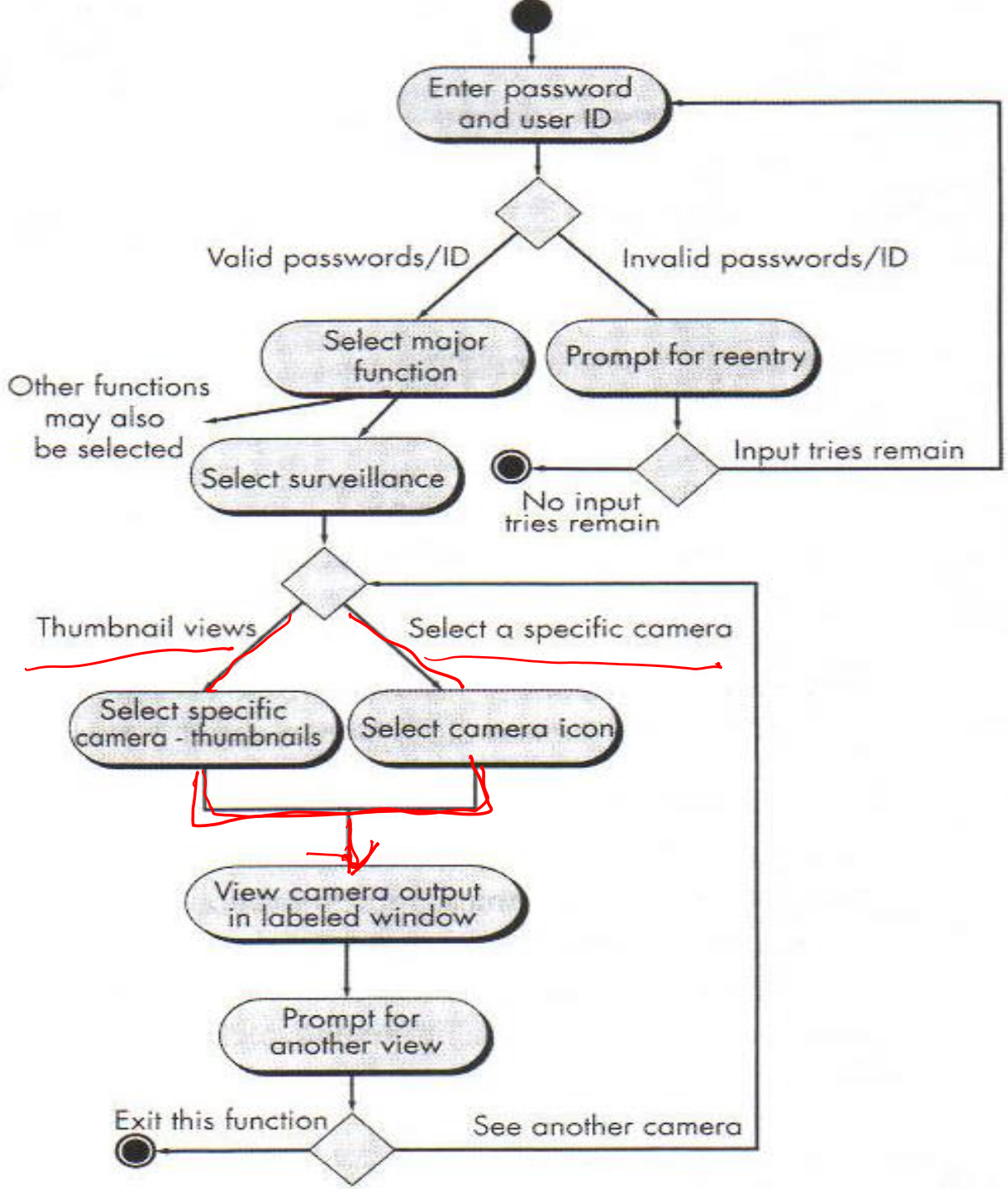
Revision: Activity Diagram

- **Rounded rectangle** → actions.
- **Diamond** → decisions/merge
- **Bars** → The start (fork) or end (join) of concurrent activities.
- **Black circle** → start (initial state)
- **Encircled black** →end (final state)
- **Arrow** → flow of control

Scenario:

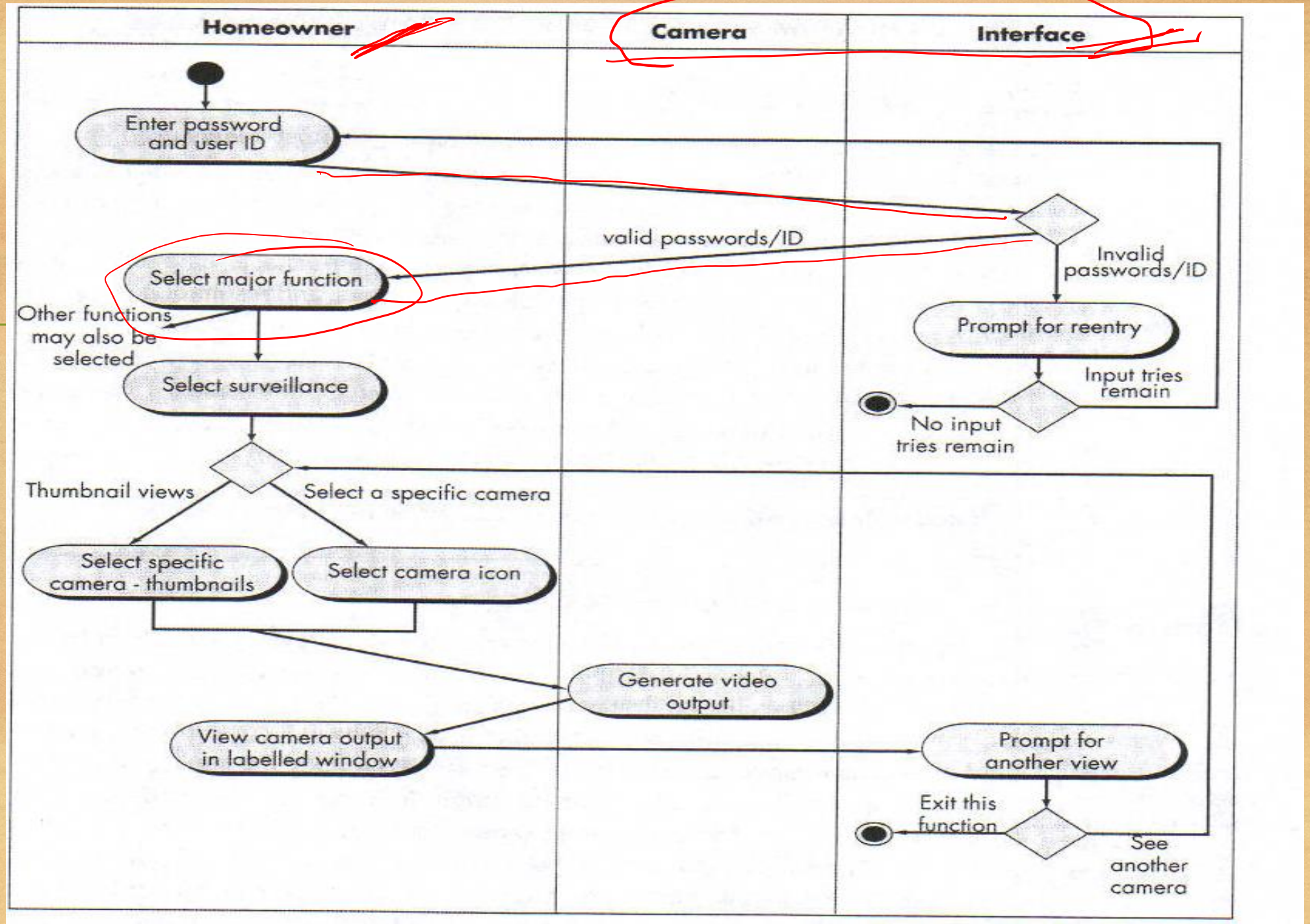
1. The homeowner logs onto the *SafeHome Products* Web site.
2. The homeowner enters his or her user ID.
3. The homeowner enters two passwords (each at least eight characters in length).
4. The system displays all major function buttons.
5. The homeowner selects "surveillance" from the major function buttons.
6. The homeowner selects "pick a camera."
7. The system displays the floor plan of the house.
8. The homeowner selects a camera icon from the floor plan.
9. The homeowner selects the "view" button.
10. The system displays a viewing window that is identified by the camera ID.
11. The system displays video output within the viewing window at one frame per second.

Activity diagram for Access camera surveillance—display camera views function

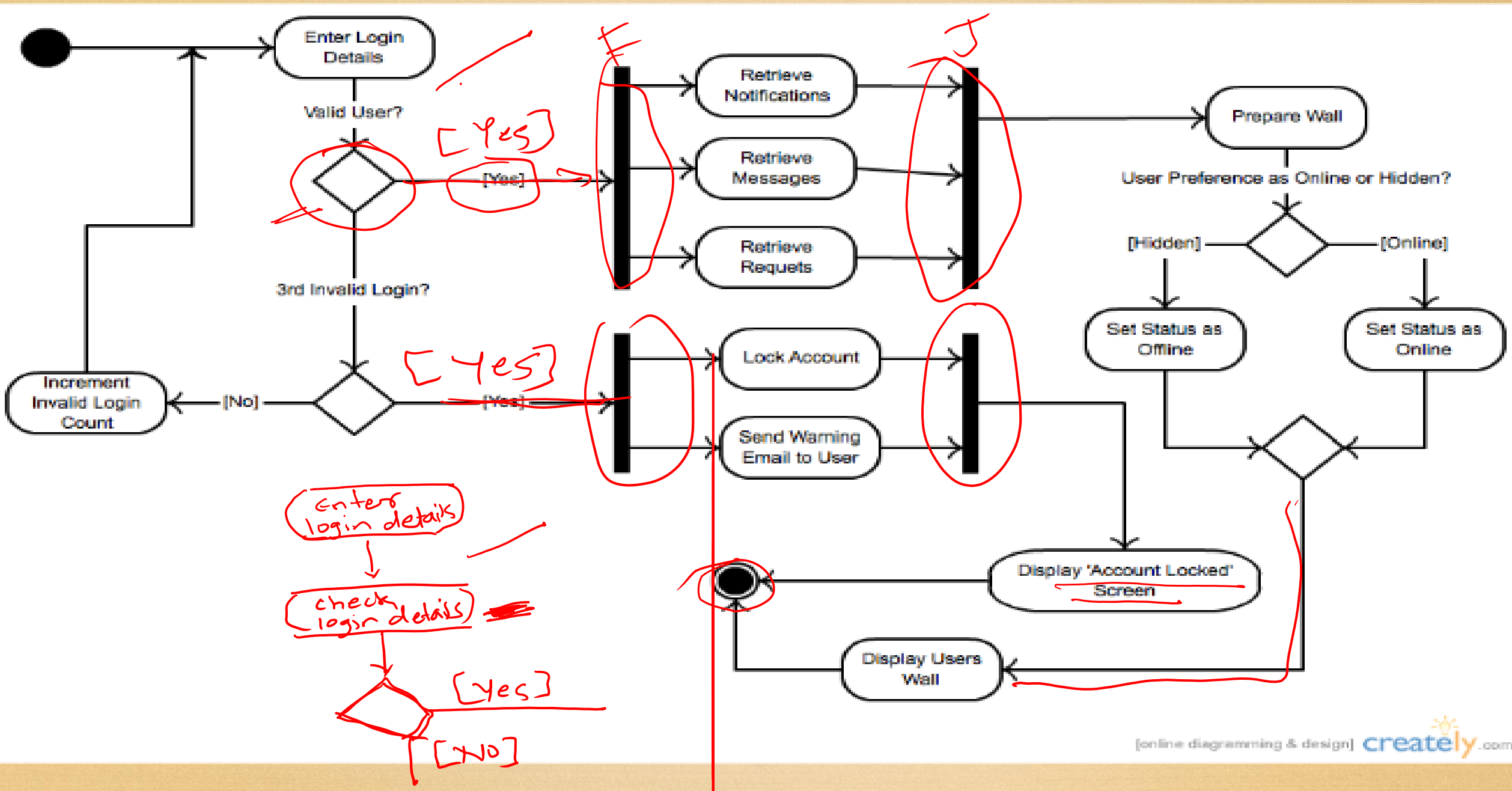


Swimlane Diagrams

- A variation of activity diagram
- Indicate which actor has responsibility for the action described by an activity rectangle
- A UML swimlane diagram represents the flow of actions, decisions and indicates which actors perform each


Homeowner
Camera
Interface
Enter password and user ID
valid passwords/ID
Invalid passwords/ID
Select major function
Other functions may also be selected
Prompt for reentry
Input tries remain
Select surveillance
No input tries remain
Thumbnail views
Select a specific camera
Select specific camera - thumbnails
Select camera icon
Generate video output
View camera output in labelled window
Prompt for another view
Exit this function
See another camera

Facebook is an online social media website, where the users must register before using the it. Once the user is authenticated by site, he can retrieve notifications, messages and requests. Further, in the next step the user can change the user preferences as online or hidden. Finally the status will be displayed in user's wall. User has option to login by entering credentials for three times. Third unsuccessful login will lock the account, send warning email of the user and display 'account locker' message on the screen. Else the system will prompt for login credentials again.

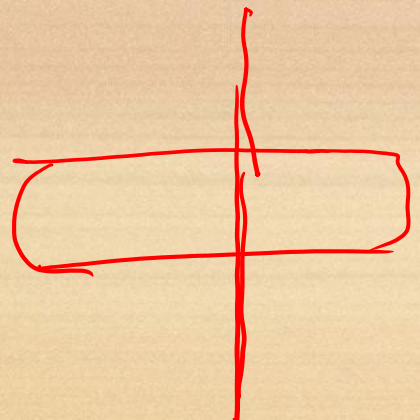

Enter Login Details
Valid User?
[Yes]
[Yes]
Retrieve Notifications
Retrieve Messages
Retrieve Requets
Prepare Wall
User Preference as Online or Hidden?
[Hidden]
[Online]
Set Status as Offline
Set Status as Online
3rd Invalid Login?
[Yes]
[No]
Increment Invalid Login Count
Lock Account
Send Warning Email to User
Display 'Account Locked' Screen
Display Users Wall
enter login details
check login details
[Yes]
[NO]
[online diagramming & design] creately.com

Swimlanes (1)

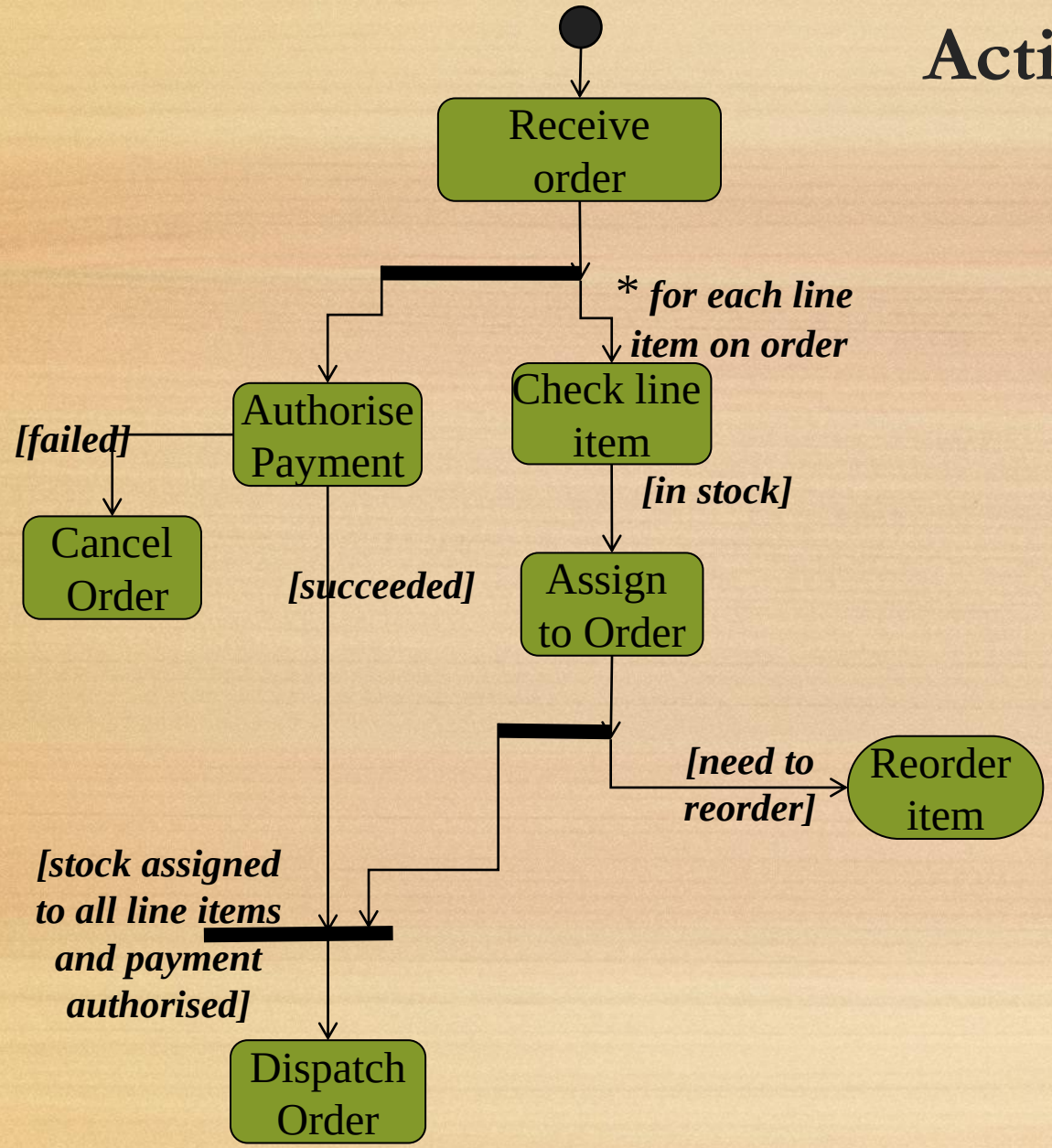
- A swimlane specifies a locus of activities
- To partition the activity states on an activity diagram into groups
  - each group representing the business organization responsible for those activities
  - each group is called a swimlane
- Each swimlane is divided from its neighbor by a vertical solid line

Swimlanes (2)

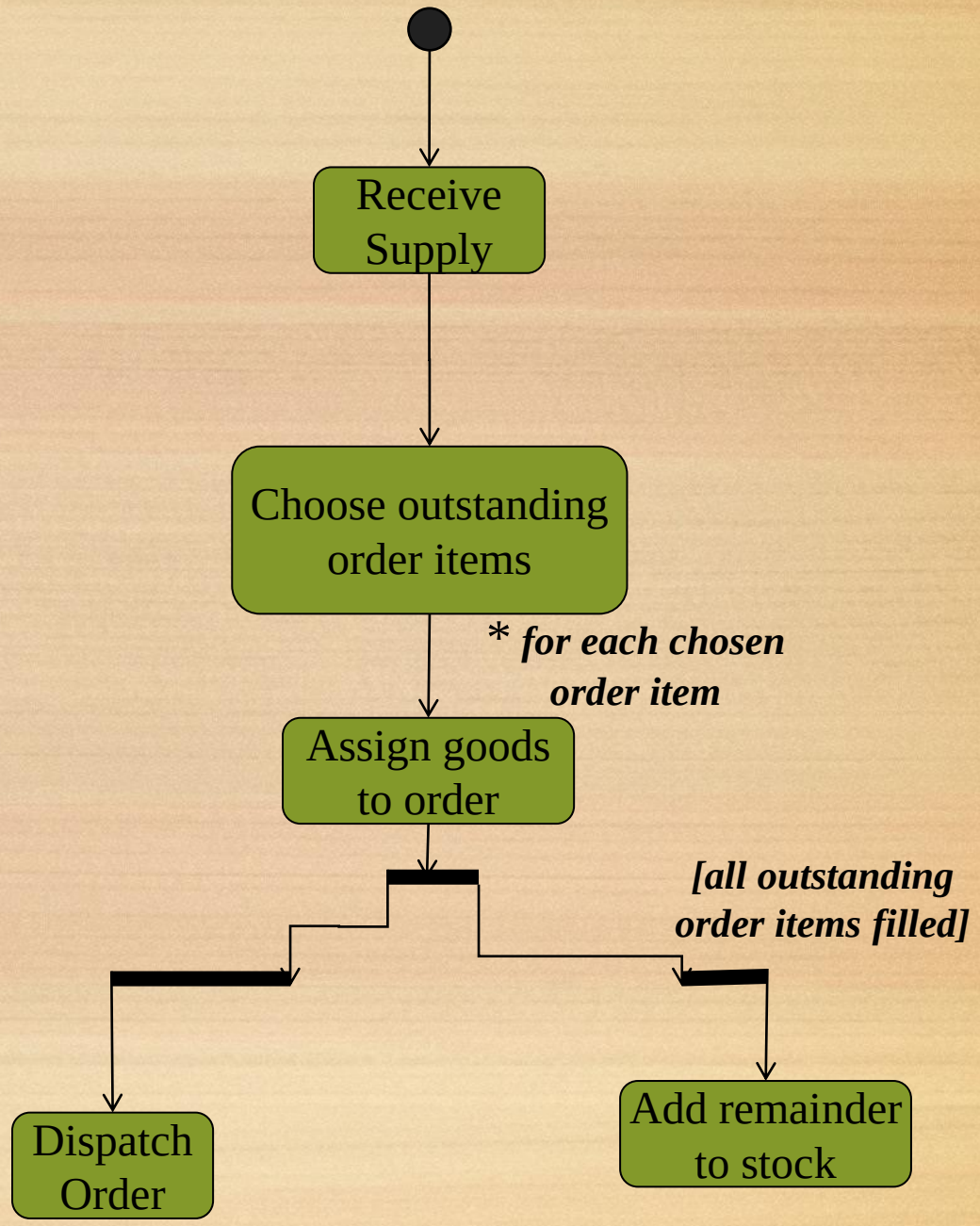
- Each swimlane has a name unique within its diagram
- Each swimlane may represent some real-world entity
- Each swimlane may be implemented by one or more classes
- Every activity belongs to exactly one swimlane, but transitions may cross lanes



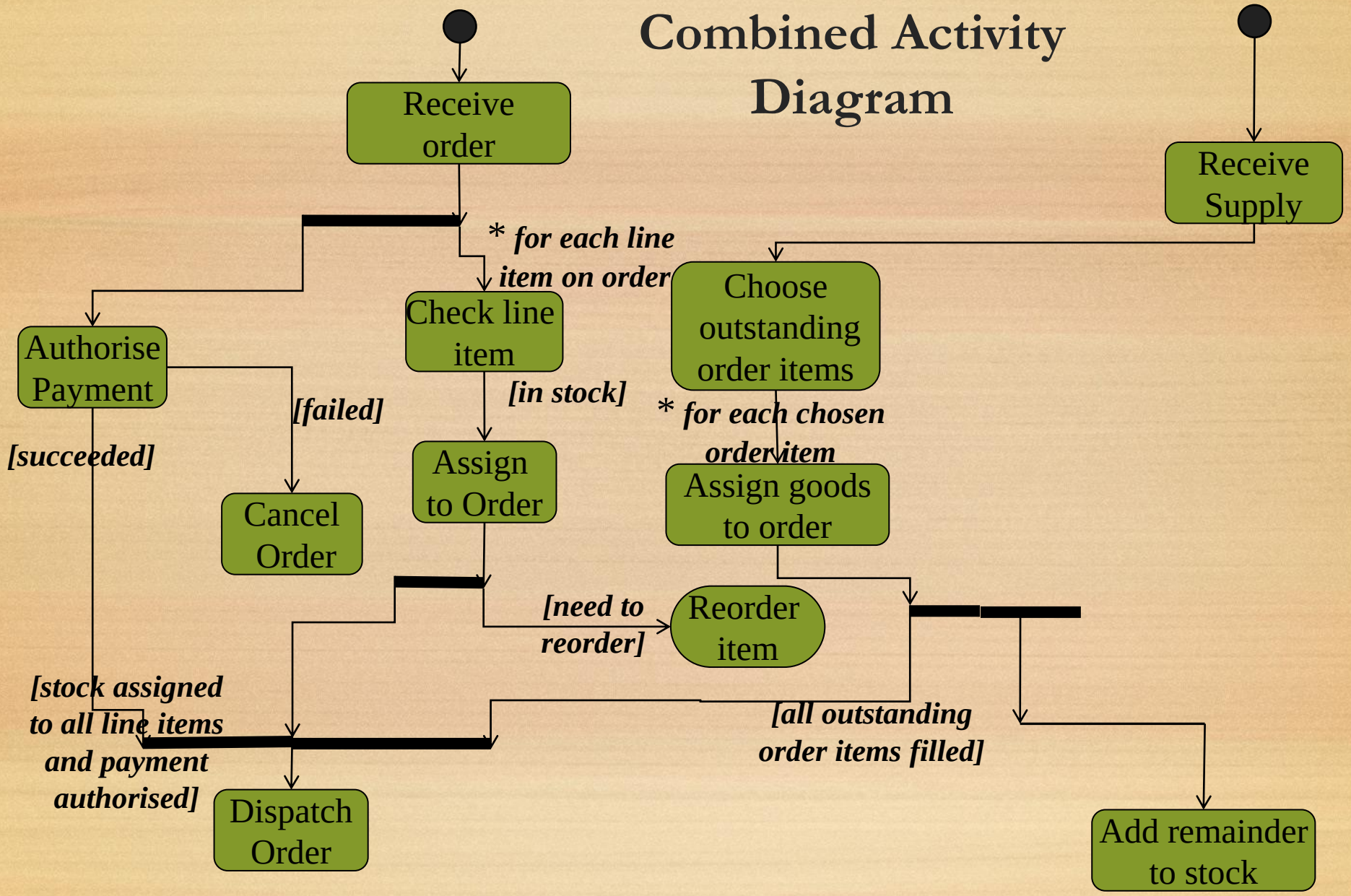
Activity Diagram for Receiving an Order

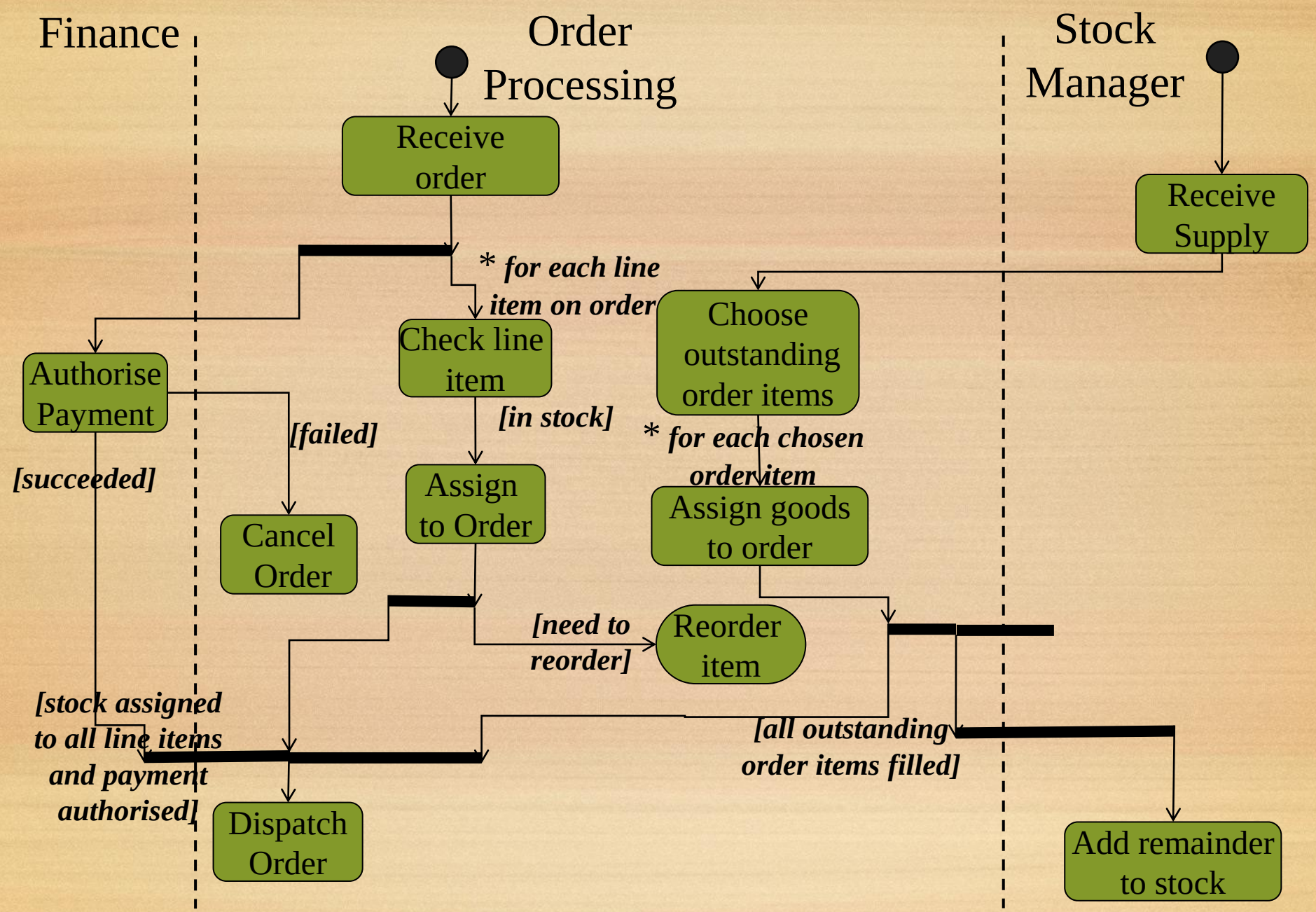


Activity Diagram for receiving Supply

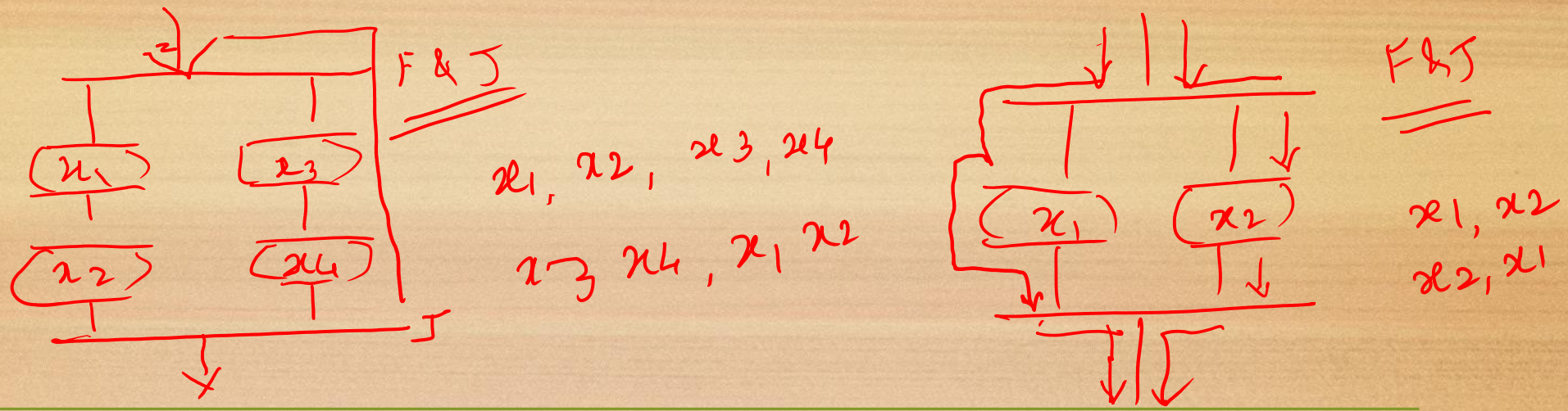


Combined Activity Diagram





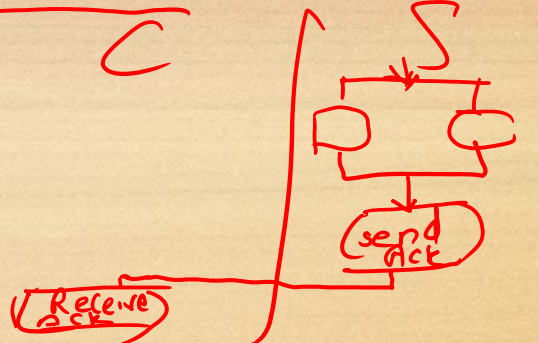
With Swimlanes

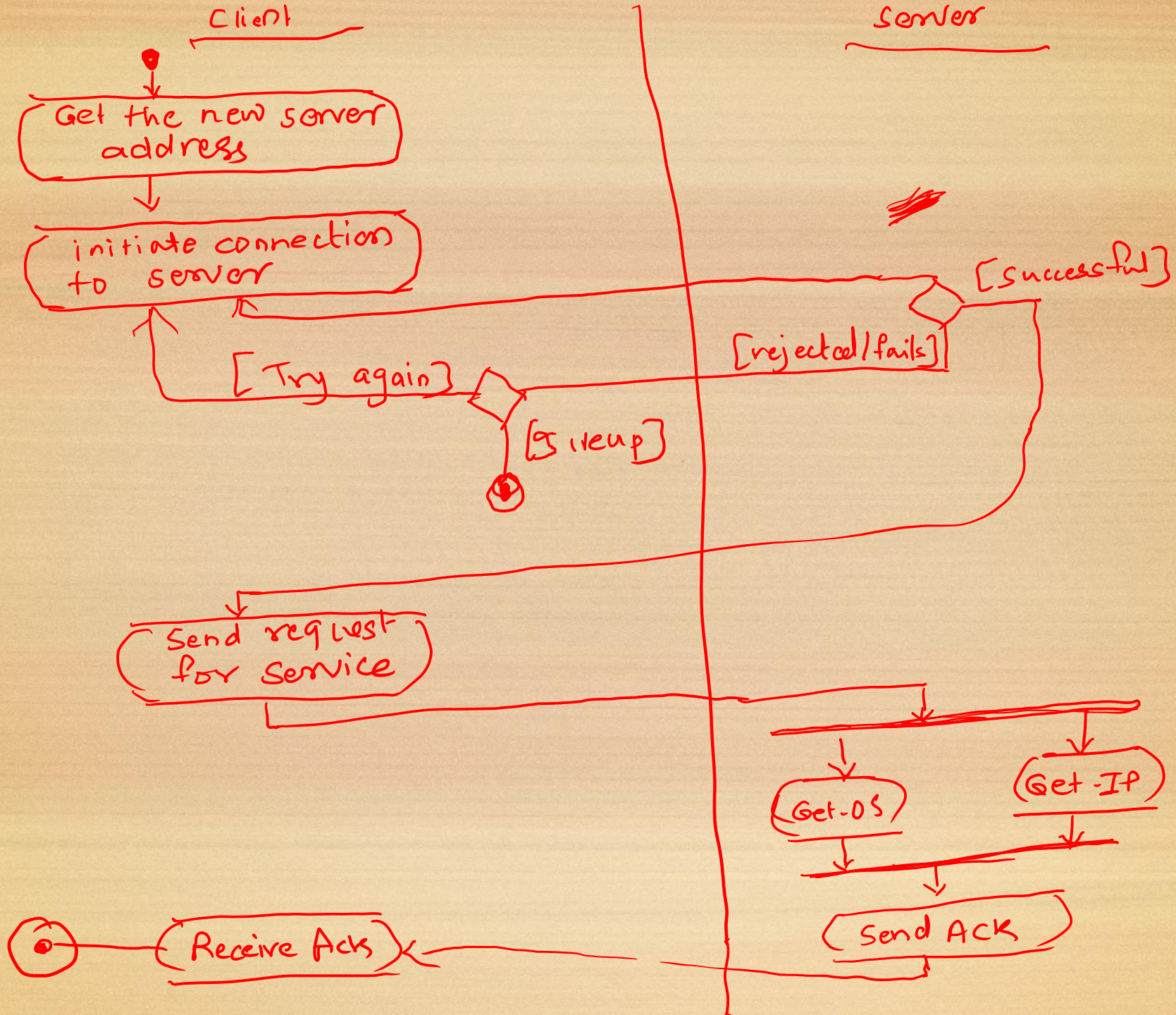


- The client needs to know the whole address of a server for the communication. Client performs some work which includes, making a decision to initiate a connection to a server. If connection to the server fails, or the server rejects the connection, the client may try again or may give up. On successful connection, the client reads the request to execute the services. GET_OS and GET_IP are the services which will execute parallel and acknowledgement will be sent to the client by the server.

Draw an activity diagram. C

Swimlane




Client
Server
Get the new server address
initiate connection to server
[successful]
[rejected/fails]
[Try again]
[give up]
Send request for service
Get-OS
Get-IP
Send ACK
Receive Acks

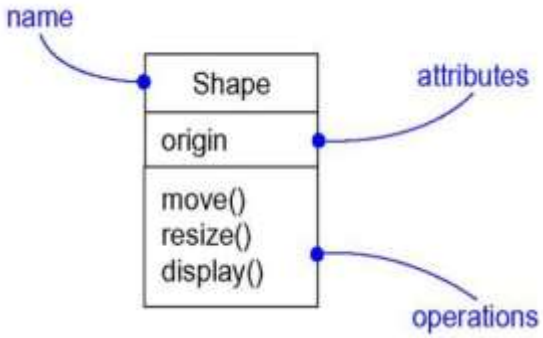
- The ATM will service one customer at a time. A customer will be required to insert an ATM card and enter a personal identification number (PIN) - both of which will be sent to the bank for validation as part of each transaction. The customer will then be able to perform one or more transactions.
- The card will be retained in the machine until the customer indicates that he/she desires no further transactions, at which point it will be returned. The ATM will communicate each transaction to the bank and obtain verification that it was allowed by the bank. If the bank determines that the customer's PIN is invalid, the customer will be required to re-enter the PIN before a transaction can proceed.
- If the customer is unable to successfully enter the PIN after three tries, the card will be permanently retained by the machine, and the customer will have to contact the bank to get it back. If a transaction fails for any reason other than an invalid PIN, the ATM will display an explanation of the problem, and will then ask the customer whether he/she wants to do another transaction.

- The ATM will also maintain an internal log of transactions to facilitate resolving ambiguities arising from a hardware failure in the middle of a transaction. Entries will be made in the log when the ATM is started up and shut down, for each message sent to the Bank (along with the response back, if one is expected), for the dispensing of cash, and for the receiving of an envelope.
- Log entries may contain card numbers and amounts, but for security will never contain a PIN. To avail ATM facility, a customer is required to open/have an account in the bank and apply for the ATM card. A customer can have one or more accounts and for each account, only one ATM card will be provided.
- The bank also provides SMS updates for every transaction of customer’s account. To obtain SMS updates, customer is required to register his/her mobile number against his account in the bank.

CLASS DIAGRAM

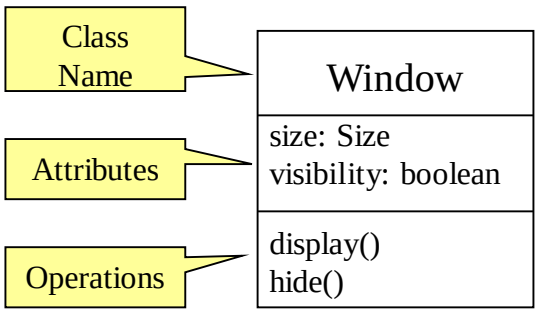
Introduction

- Most important building block
- Description of a set of objects that share the same attributes, operations, relationships and semantics
- Used to capture the vocabulary of the system
  - Include abstractions of the problem domain
  - Classes that make up the implementation
- Form a part of a balanced distribution of responsibilities across the system



Class

- Describes a set of objects having similar:
  - Attributes (status)
  - Operations (behavior)
  - Relationships with other classes
- Attributes and operations may
  - have their visibility marked:
  - "+" for *public*
  - "#" for *protected*
  - "–" for *private*
  - "~" for *package*

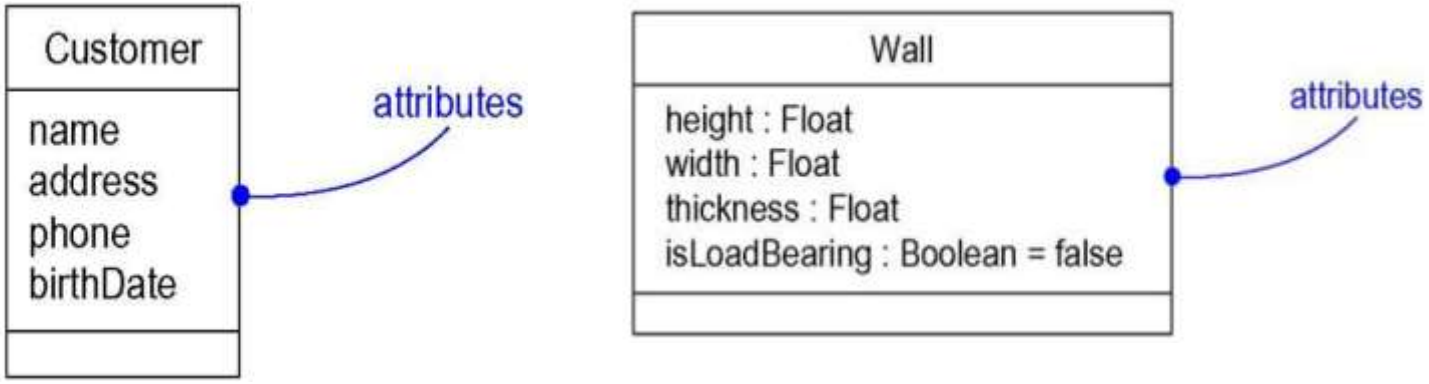


Class Names

- Name must be unique within its enclosing package
- Simple name and Path name
- Class names are noun phrases from the vocabulary of the system

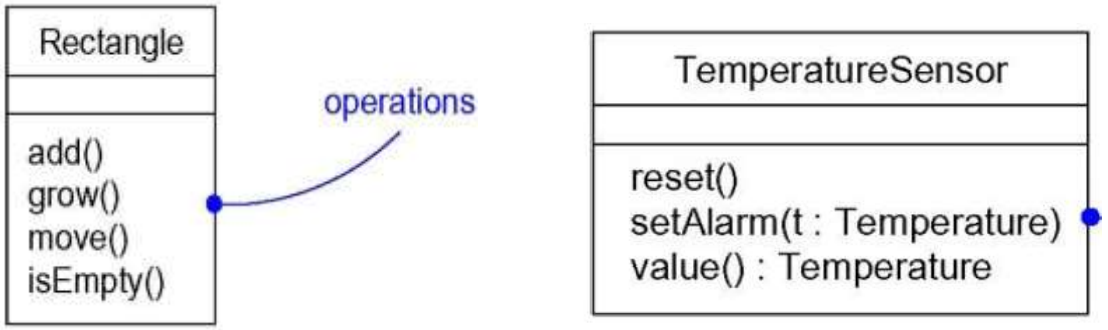
Attributes

- Named property of a class
  - Describes a range of values that instances of the property hold
  - Shared by all objects of that class
- An abstraction of the kind of data or state of an object
  - At a given moment, an object will have specific values for each of its attributes
- Noun phrase that represents some property



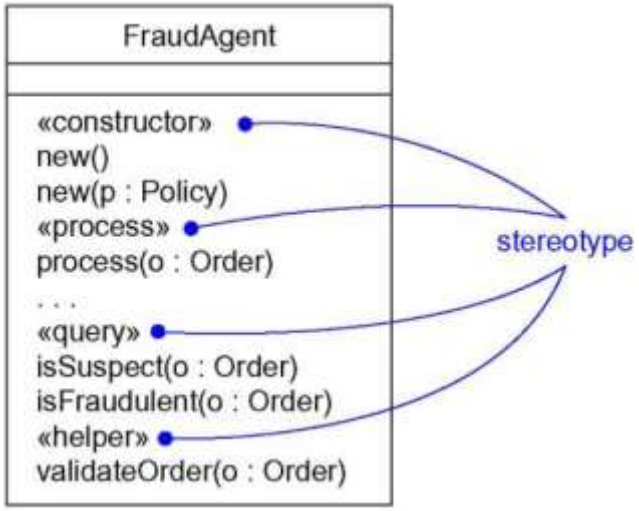
Operations

- A service that can be requested from any object of the class to affect behavior
- Invoking an operation changes the object's data or state
- Verb phrase representing some behavior of the class



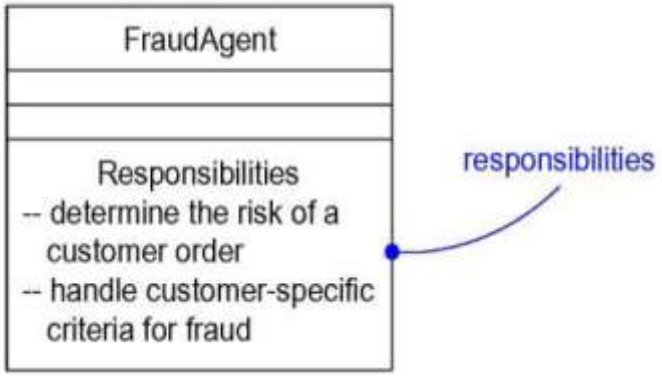
Organizing Attributes and Operations

- Don’t have to show every attribute and operation at once
- End each list with ellipsis (…)
- Can also prefix each group with a descriptive category using stereotypes



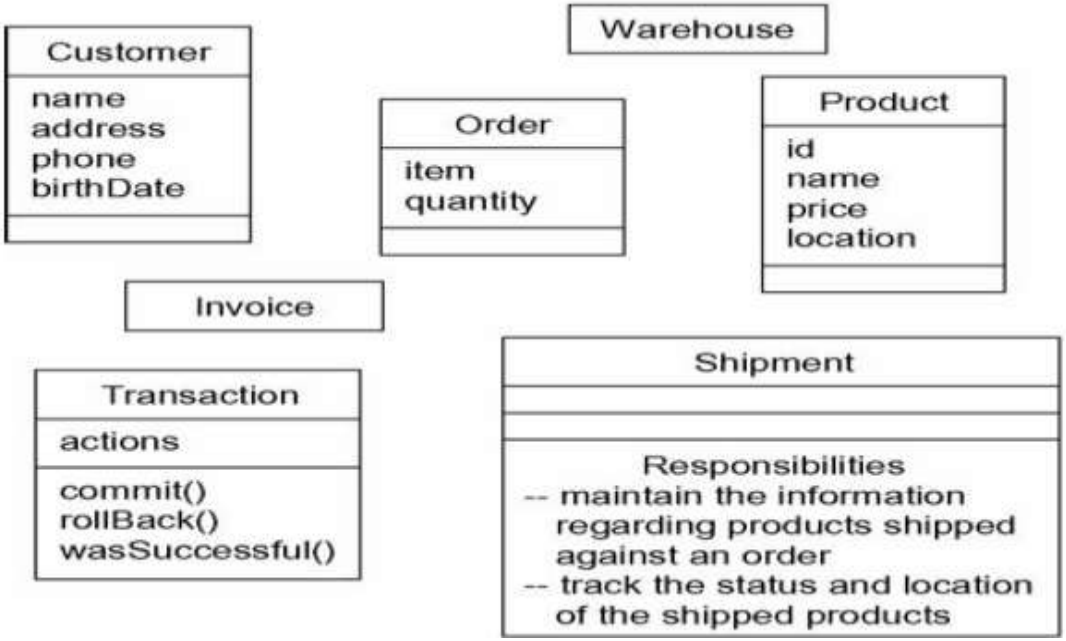
Responsibilities

- Is a contract or an obligation of a class
  - All objects of the class have the same kind of state and behavior
- Attributes and operations are the features that carry out the class's responsibility
  - `TemperatureSensor` class responsible for measuring temperature and raising an alarm
- Techniques used - CRC cards and use case based-analysis



Modeling the Vocabulary of a System

- Identify the things that describe the problem
- For each abstraction, identify a set of responsibilities
- Provide the attributes and operations needed to carry out those responsibilities



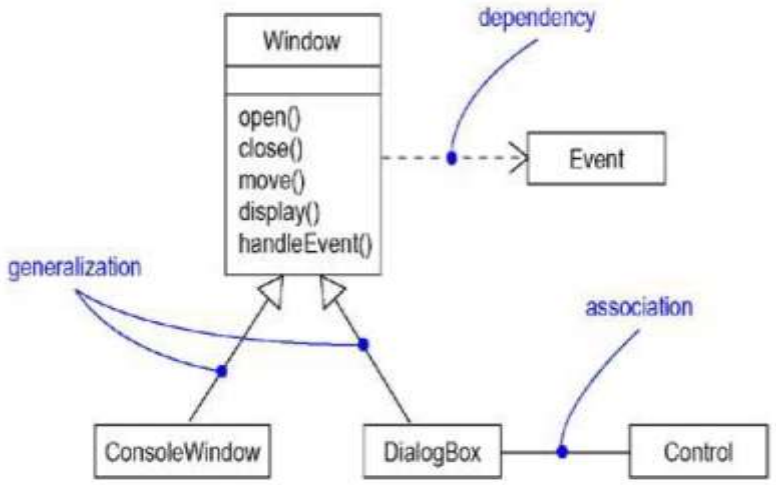
CLASS RELATIONSHIPS

Introduction

- Classes collaborate with other classes in a number of ways
- Three kinds of relationships in OO modeling
  - Dependencies
  - Generalizations
  - Associations
- Provide a way for combining your abstractions
- Building relationships is not like creating a balanced set of responsibilities

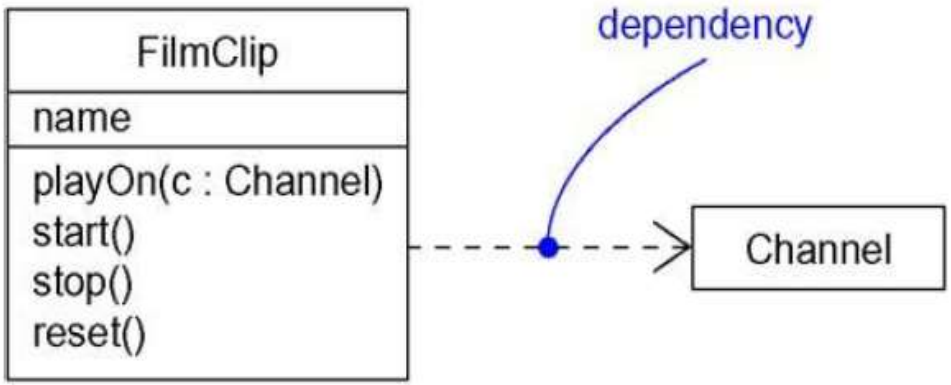
Introduction

- Relationship is a connection among things
- UML graphical notation
  - Permits to visualize relationships apart from any specific programming language
  - Emphasize the most important parts of a relationship
    - Name
    - Things it connects
    - Properties



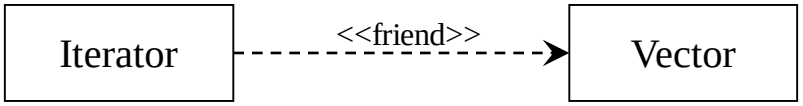
Dependency

- *Using* relationship
  - States that a change in specification of one thing may affect another thing that uses it
- Used in the context of classes
  - Uses operations or variables/arguments typed by the other class
  - Shown as an argument in the signature of an operation
  - If used class changes, the operation of the other class may be affected



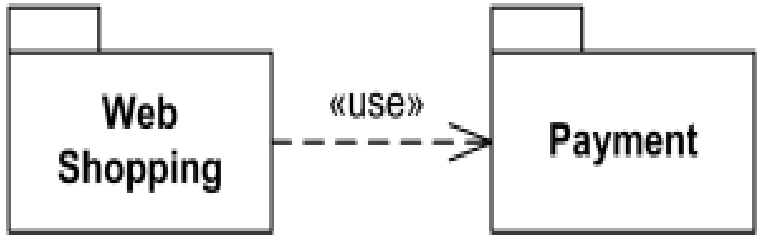
Dependency

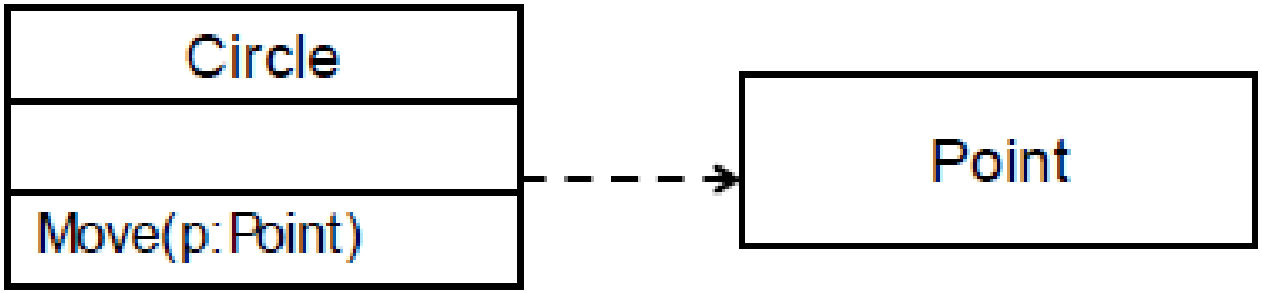
- Dependency is a weaker form of relationship which indicates that one class depends on another because it uses it at some point in time.
- One class depends on another if the independent class is a parameter variable or local variable of a method of the dependent class.
- This is different from an association, where an attribute of the dependent class is an instance of the independent class.



Dependency

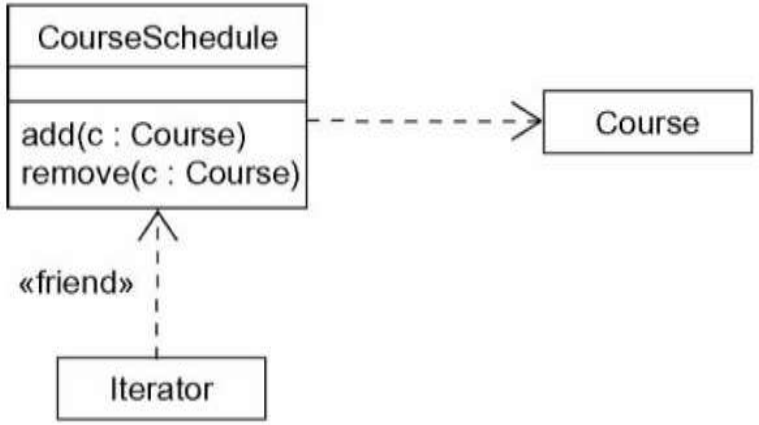

CarFactory
«instantiate»
Car


Web
Shopping
«use»
Payment


Circle
Move(p:Point)
Point

Modeling Simple Dependencies

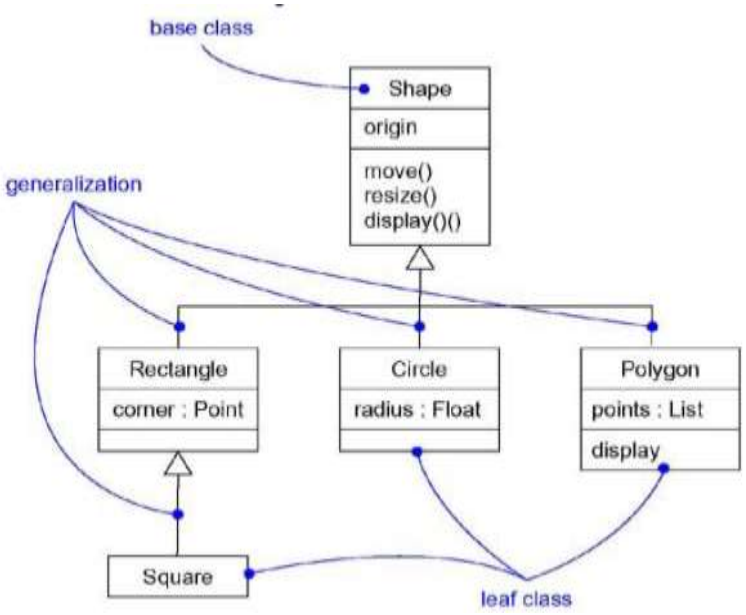
- To model the using dependency
  - Create a dependency pointing from the class with the operation to the class used as a parameter in the operation



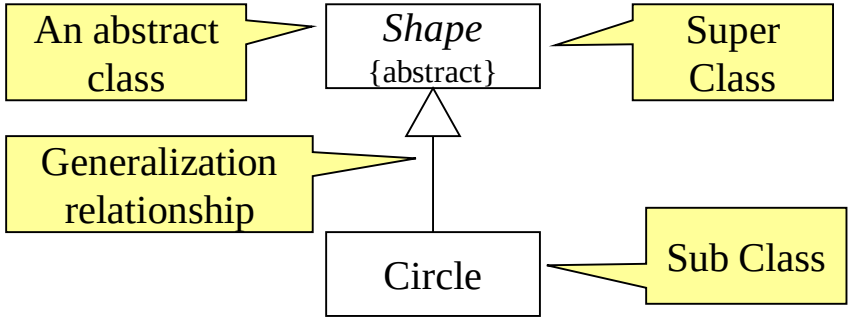
Generalization

- Relationship between a general thing (superclass) and a more specific kind of that thing (subclass)
- “Is-a-kind-of” relationship
- Child is substitutable for the parent
- Polymorphism
  - Operation of child has the same signature as an operation in the parent but overrides it

Generalization



{abstract} is a tagged value that indicates that the class is abstract. The name of an abstract class should be italicized

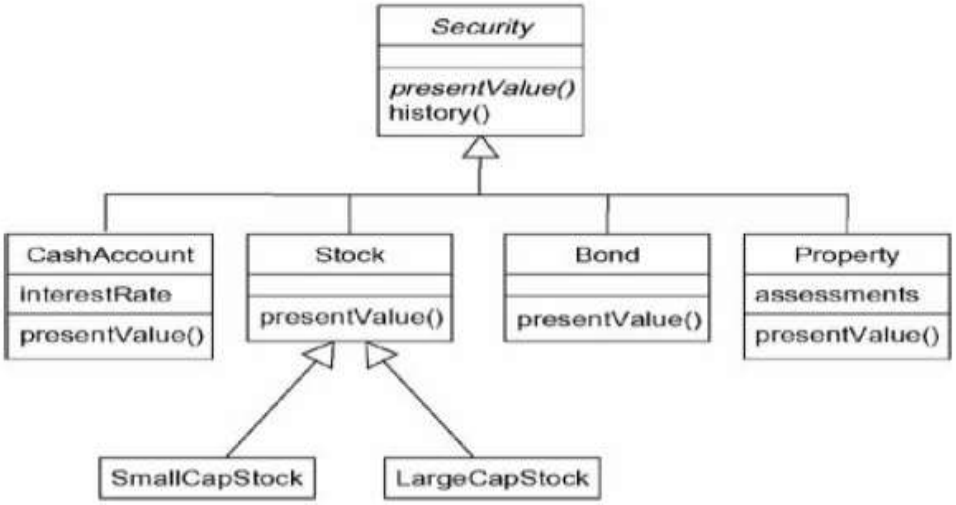


Generalization

- A sub-class inherits from its super-class
  - Attributes
  - Operations
  - Relationships
- A sub-class may
  - Add attributes and operations
  - Add relationships
  - Refine (override) inherited operations
- A generalization relationship **may not** be used to model interface implementation.

Modeling Single Inheritance

- Find classes that are structurally or behaviorally similar while modeling the vocabulary
- To model inheritance
  - Look for common responsibilities
  - Elevate these to a more general class
  - Specify that the more specific class inherits from the more general class

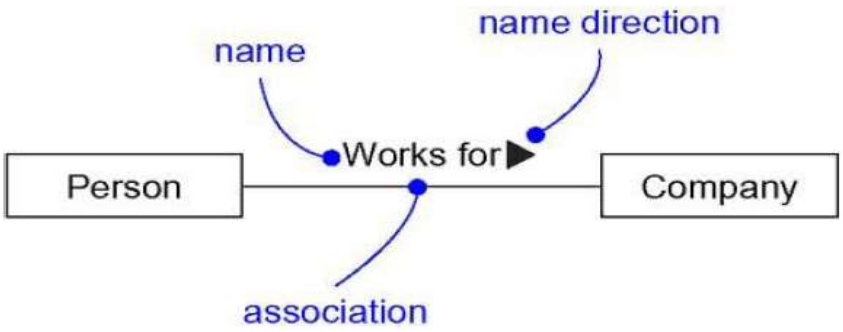


Association

- Structural relationship
  - Specifies that objects of one thing are connected to objects of another
- Can navigate from objects of one class to the other
- Self Association
  - Connects a class to itself
- Binary Association
  - Connects exactly two classes

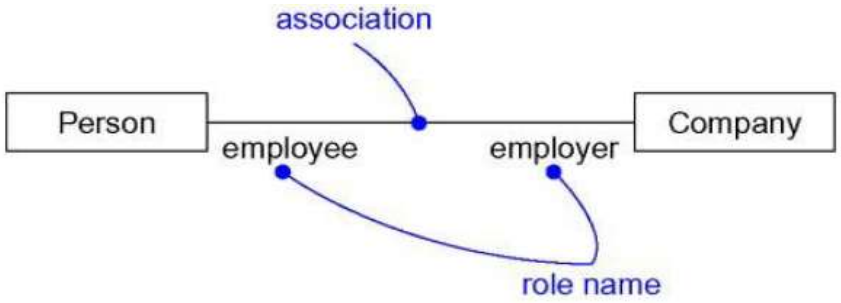
Association : Adornments

- Name
  - Use a name to describe the nature of the relationship
  - Can give a direction to the name



Association : Adornments

- Role
  - The face that the class at the far end of the association presents to the class at the near end
  - Explicitly name the role a class plays (*End name* or *Role name*)
  - Same class can play the same or different roles in other associations



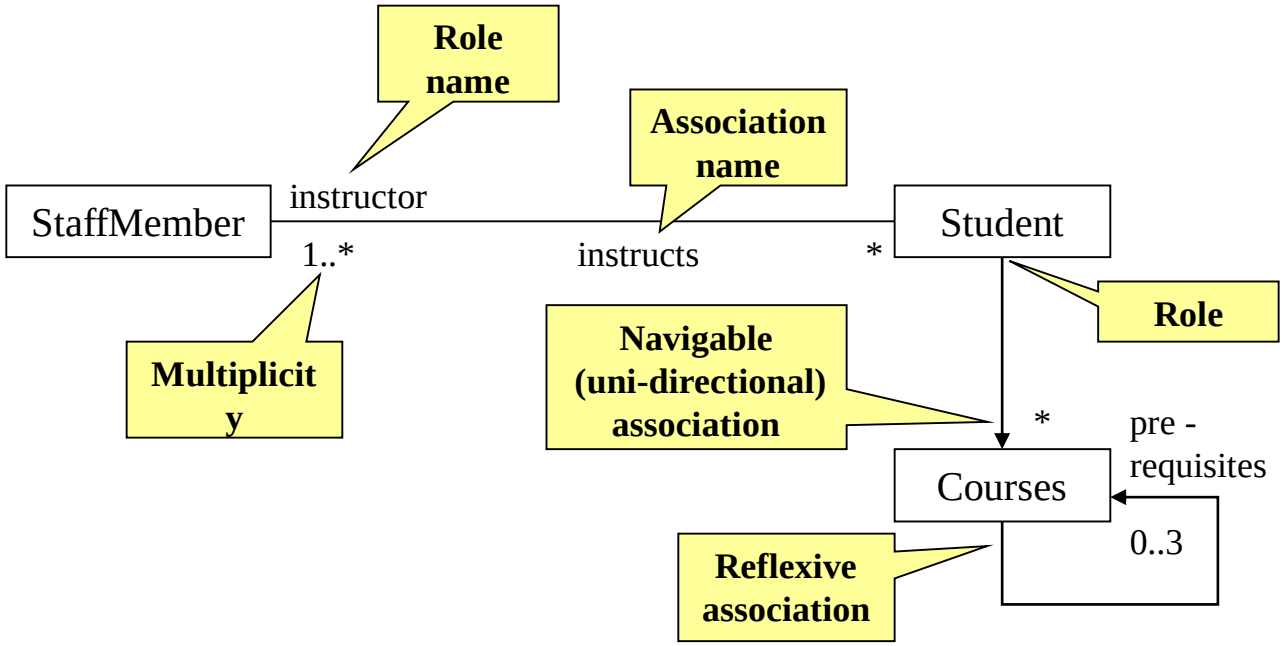
Association : Adornments

- Multiplicity
  - States how many objects may be connected across an instance of an association
  - An expression that evaluates to a range of values or an explicit value
  - Multiplicity at one end of an association means
    - For each object of the class at the opposite end, there must be that many objects at the near end
  - Show a multiplicity of
    - Exactly one (1)
    - Zero or one (0 . . 1)
    - Many (0 . . *)
    - One or more (1 . . *)
    - State an exact number (ex: 3)

Multiplicity

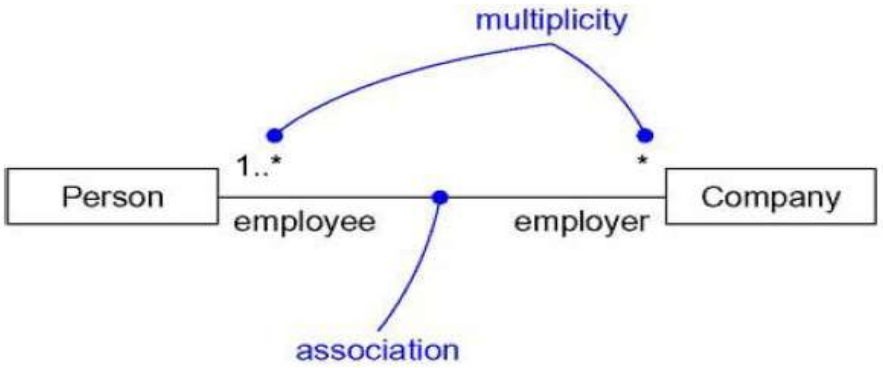
- 1:1
- 1:N
- N:1
- N:M
- Meta character – “*” and “+”:

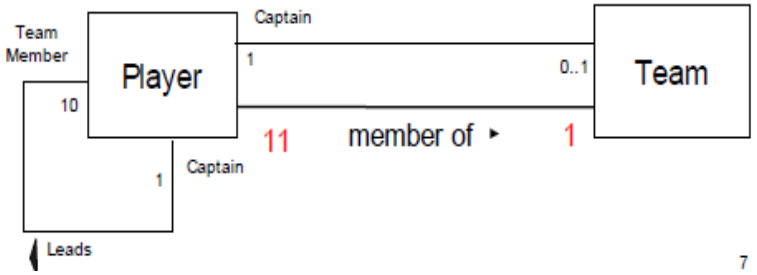
Associations (cont.)

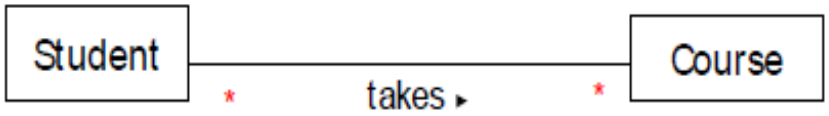


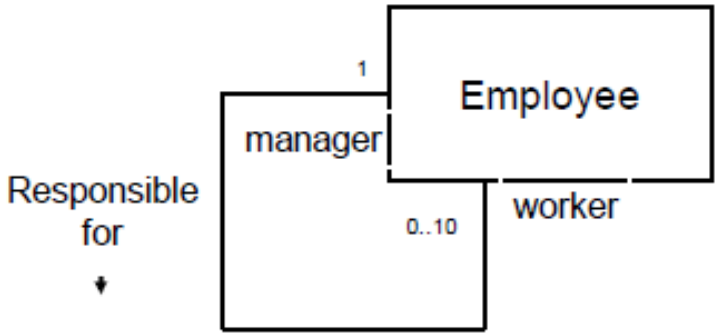
Association : Adornments

- Multiplicity
  - Specify complex multiplicities by using a list, 0..1, 3..4, 6..* (any number of objects other than 2 or 5)



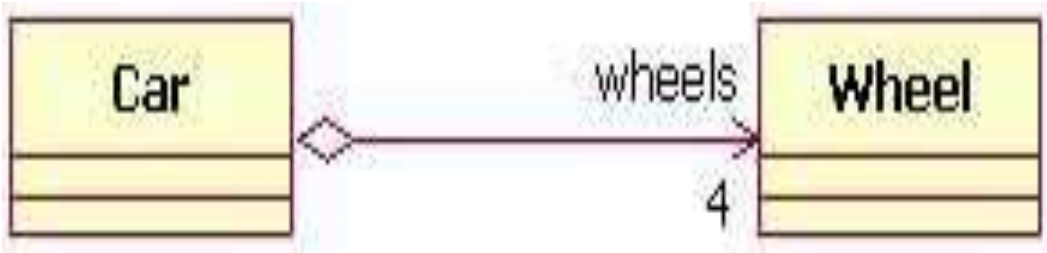






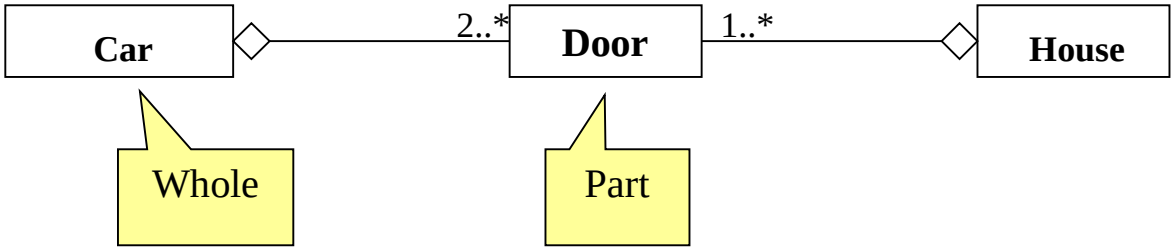
Association : Adornments

- Aggregation
  - Model a whole/part relationship
  - Represents a “has-a” relationship



Aggregation

- A special form of association that models a whole-part relationship between an aggregate (the whole) and its parts.
  - Models a “is a part-part of” relationship.

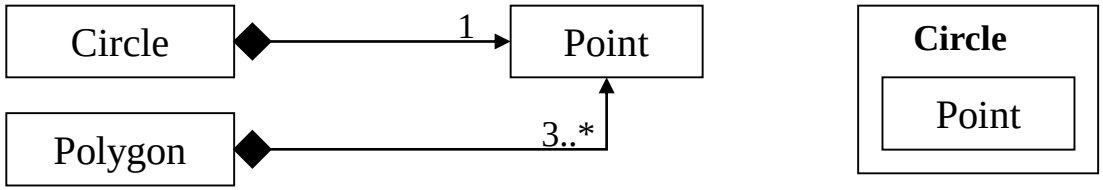


Aggregation (cont.)

- Aggregation tests:
  - Is the phrase “part of” used to describe the relationship?
    - A door is “part of” a car
  - Are some operations on the whole automatically applied to its parts?
    - Move the car, move the door.
  - Are some attribute values propagated from the whole to all or some of its parts?
    - The car is blue, therefore the door is blue.
  - Is there an intrinsic asymmetry to the relationship where one class is subordinate to the other?
    - A door **is** part of a car. A car **is not** part of a door.

Composition

- A strong form of aggregation
  - The whole is the sole owner of its part.
    - The part object may belong to only one whole
  - Multiplicity on the whole side must be zero or one.
  - The life time of the part is dependent upon the whole.
    - The composite must manage the creation and destruction of its parts.

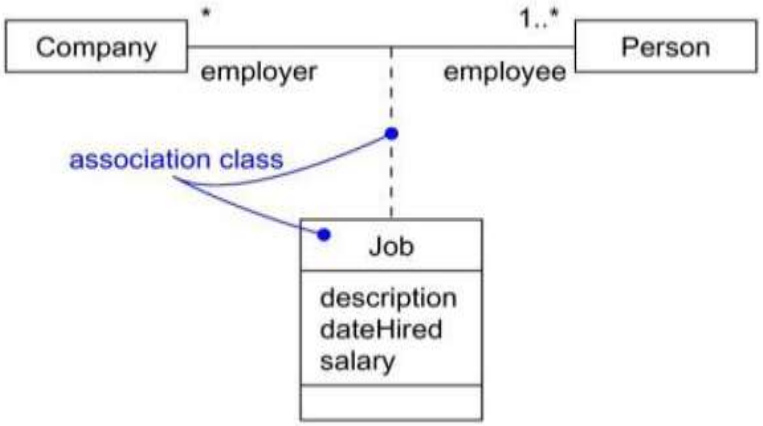


Realization

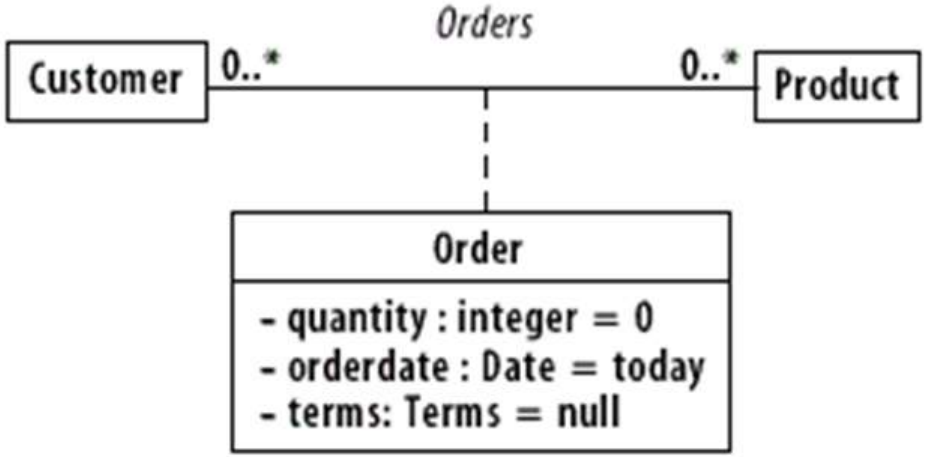
- A realization relationship indicates that one class implements a behavior specified by another class (an interface or protocol).
- An interface can be realized by many classes.
- A class may realize many interfaces.

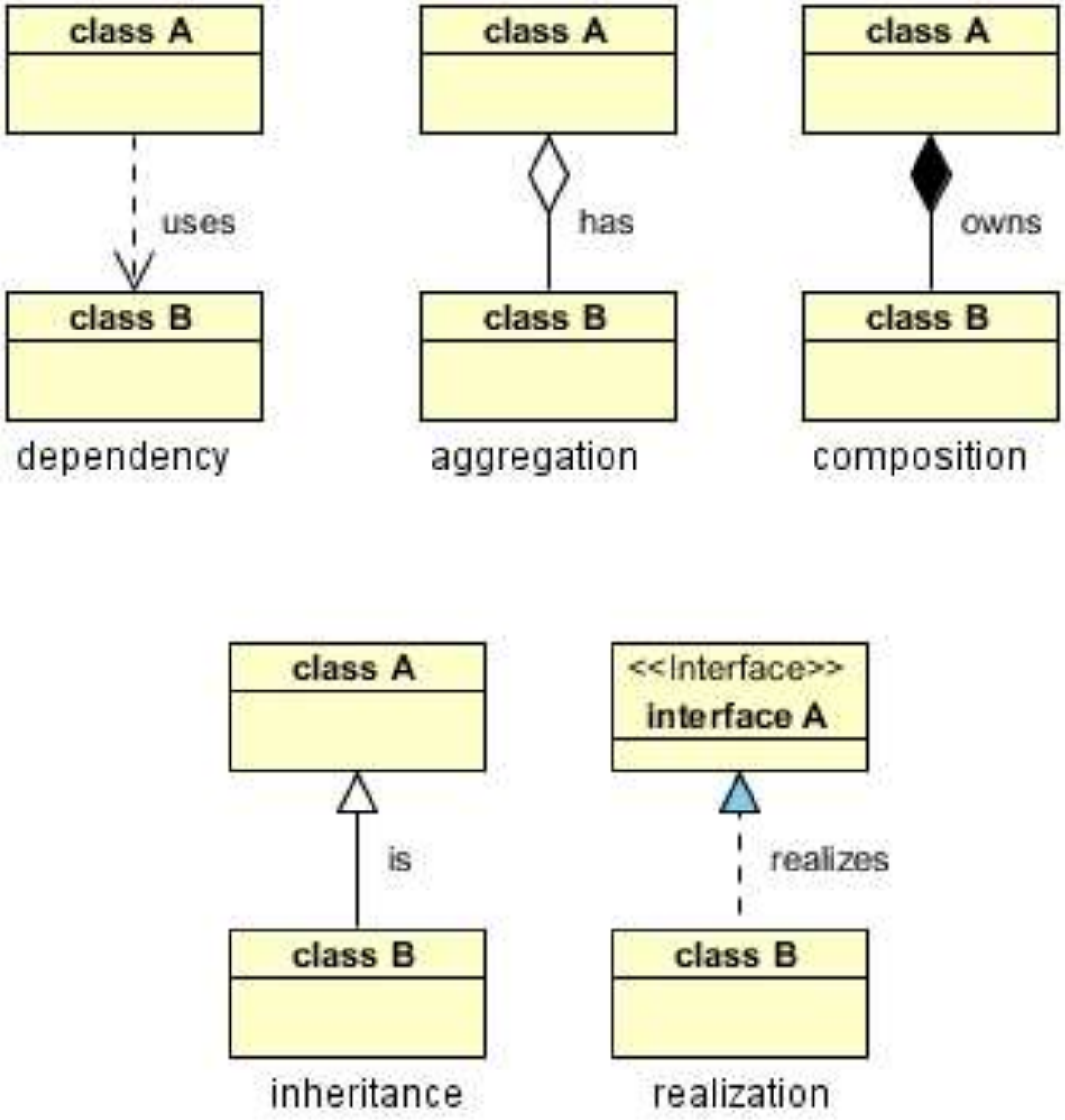
Association

- **Association Classes**
  - Association itself may have properties
  - Ex: In the relationship between a Company and a Person, there is Job that represent the properties of that relationship
    - Represents exactly one pairing of Person and Company
  - Modeling element that has both association and class properties
  - Can’t attach an association class to more than one association



Association Class




class A
uses
class B
dependency
class A
has
class B
aggregation
class A
owns
class B
composition
class A
is
class B
inheritance
<<Interface>>
interface A
realizes
class B
realization

Guidelines for Identifying Association

Class A and B are associated if

- An object of class A sends a message to an object of class B
- An object of class A creates an object of class B
- An object of class A has an attribute whose values are objects of class B
- An object of class A receives a message with an object of class B as an argument

Guidelines for Identifying a Super-sub Relationship

- Top-down
  - Look for noun phrases composed of adjectives in a class name.
- Bottom up
  - Look for classes with similar attributes or methods

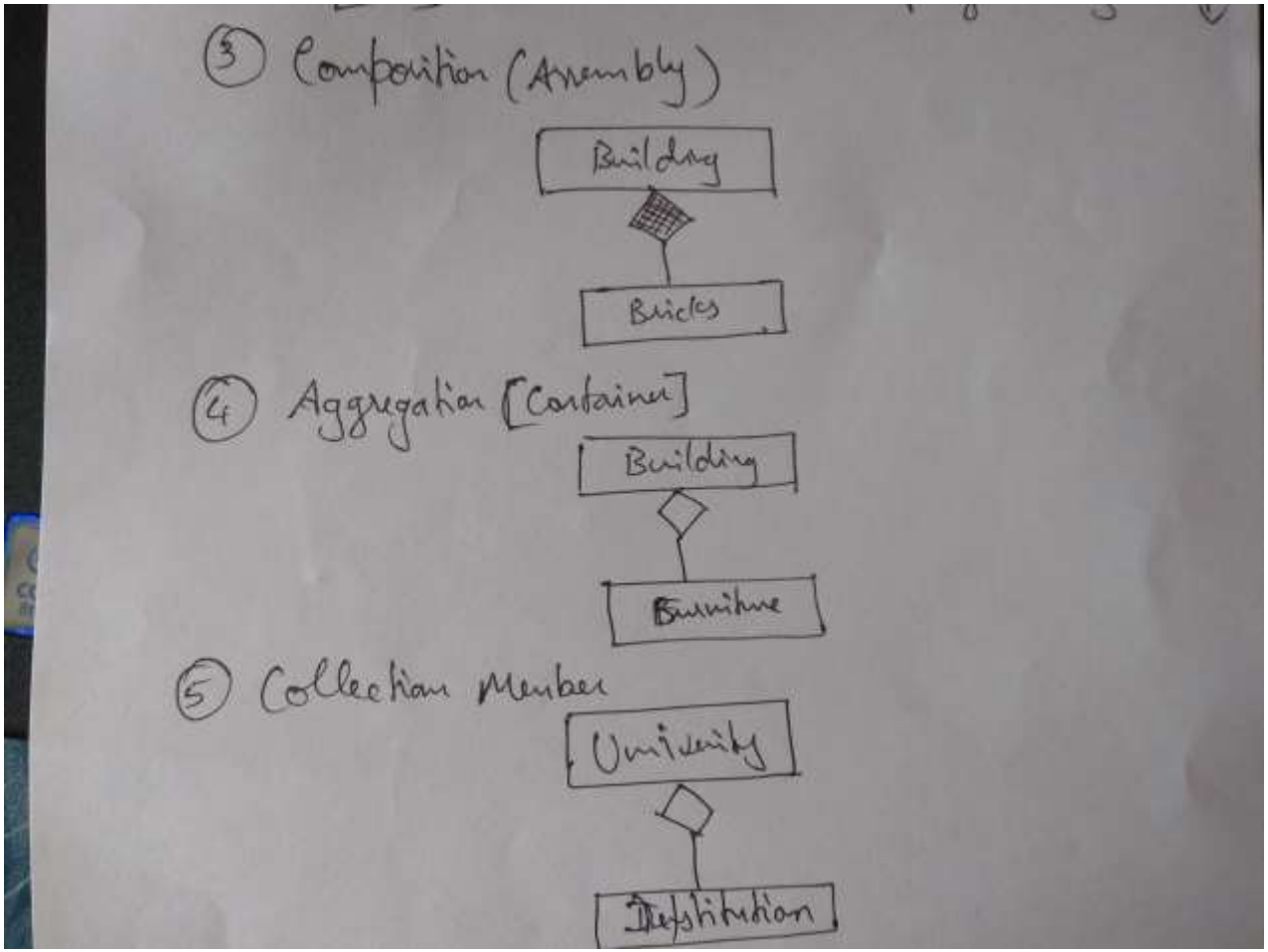
Identifying the Composition & Aggregation/a-part-of Relationship

- **Composition** - a physical whole is constructed from physical parts (Assembly)
  - Eg1: Building constructed by bricks, stones
  - Eg2: ATM with Card Reader, Console, Printer, Key Pad

- **Aggregation** - a physical whole encompasses but is not constructed from physical parts (Container)
  - Eg1: Building with Furniture, Appliances
  - Eg2: Car with AC and Radio

- **Collection-member** – a conceptual whole encompasses parts that may be physical or conceptual
  - Eg: Employer, employees

Examples on Relationships



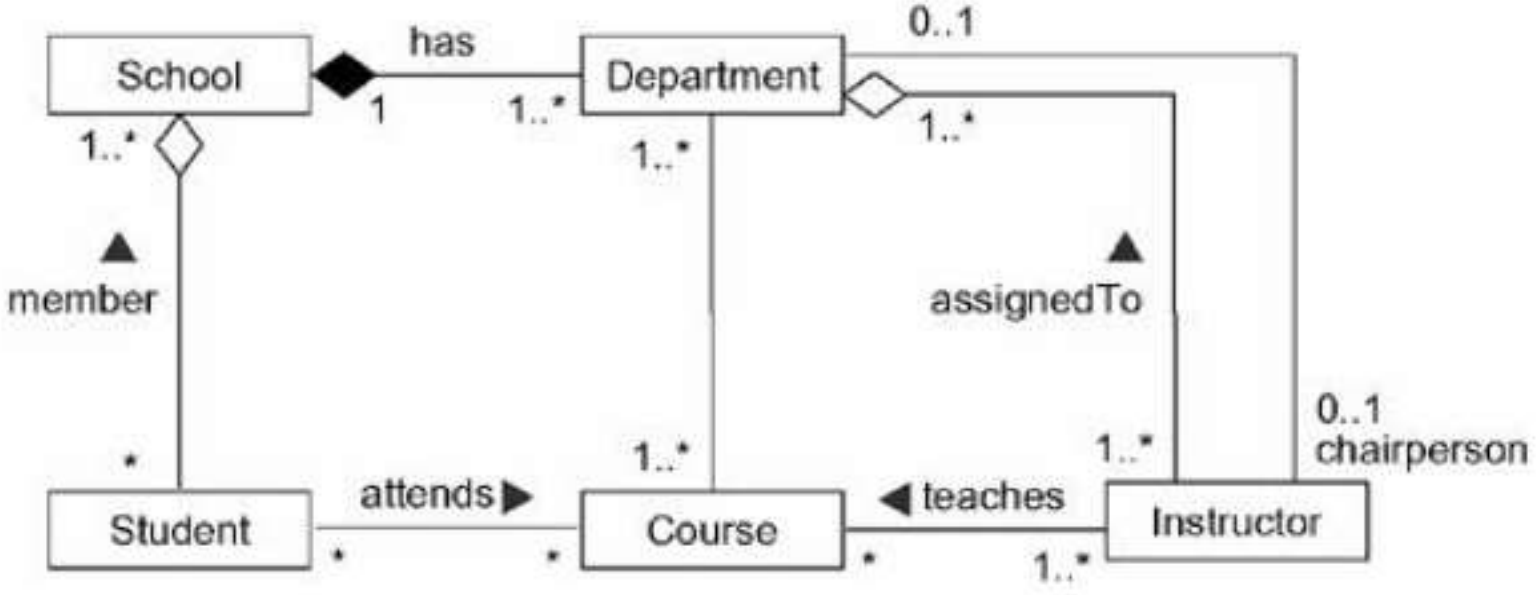
Modeling Structural Relationships

- Given an association between two classes
  - Both rely on each other in some way
  - Can navigate in either direction
  - Specifies a structural path across which objects can interact
- To model structural relationships
  - Navigation from object of one class to object of other – data-driven
  - Specify multiplicity and role names (helps to explain the model)
  - Mark as an aggregation if one class is structurally a whole compared with classes at the other end that look like parts

Modeling Structural Relationships

- Identify the association relationships among classes drawn from an information system for a school
  - School has one or more departments
  - Department offers one or more Courses
  - A particular Course will be offered by only one Department
  - Department has Instructors and Instructors can work for one or more Departments
  - Student can attend any number of Courses in a School
  - Every Course may have any number of Students
  - Instructors can teach zero or more Courses
  - Same Course can be taught by different Instructors
  - Students can be enrolled in more than one School
  - For every Department, there is exactly one Instructor who is the chairperson

Modeling Structural Relationships



Aggregation vs Composition

1. Dependency: Aggregation implies a relationship where the child **can exist independently** of the parent.

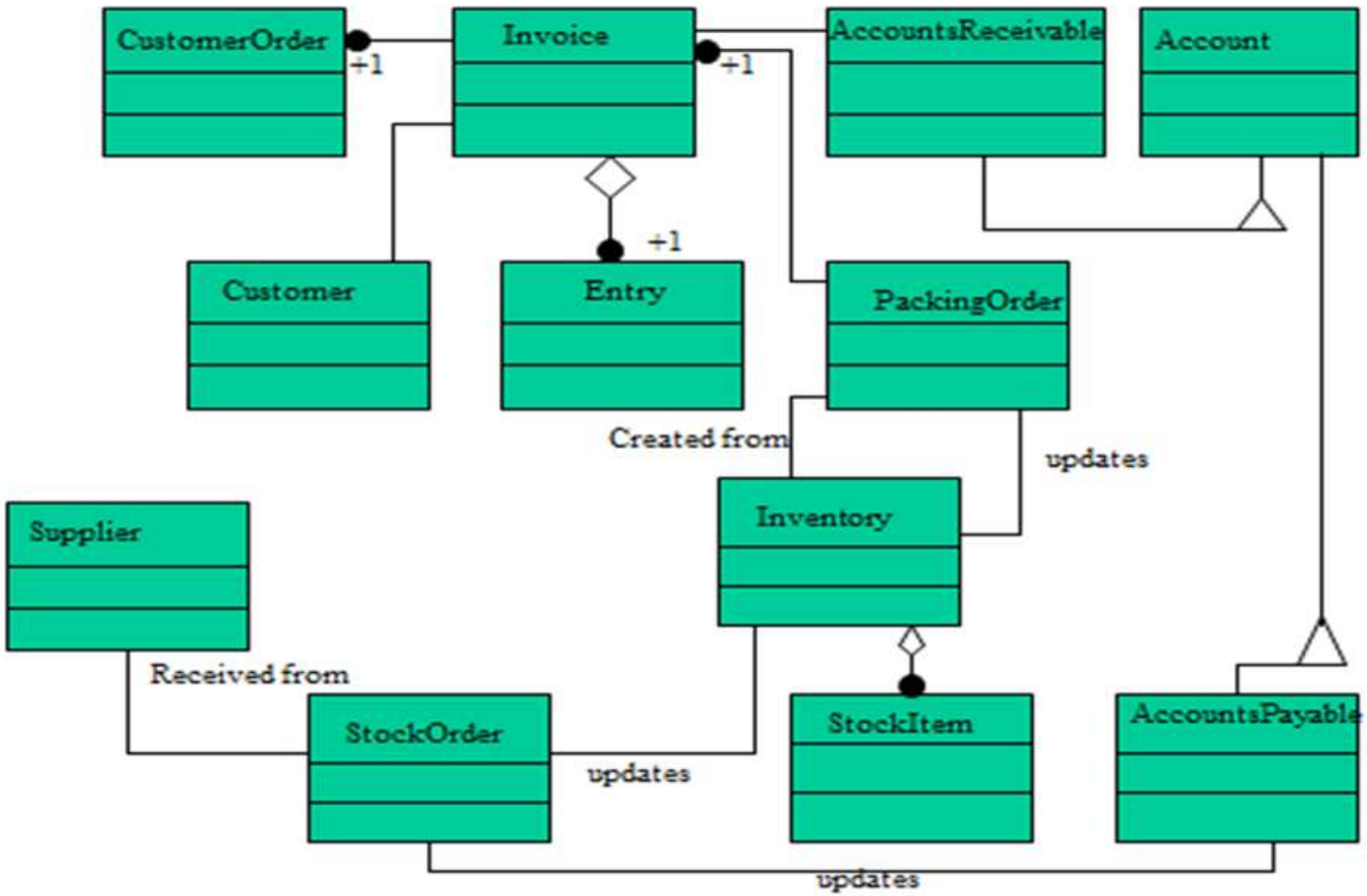
For example, Bank and Employee, delete the Bank and the Employee still exist. whereas Composition implies a relationship where the child **cannot exist independent** of the parent.

Example: Human and heart, heart don’t exist separate to a Human

2.Type of Relationship: Aggregation relation is **“has-a”,** and composition is **“part-of”** relation.

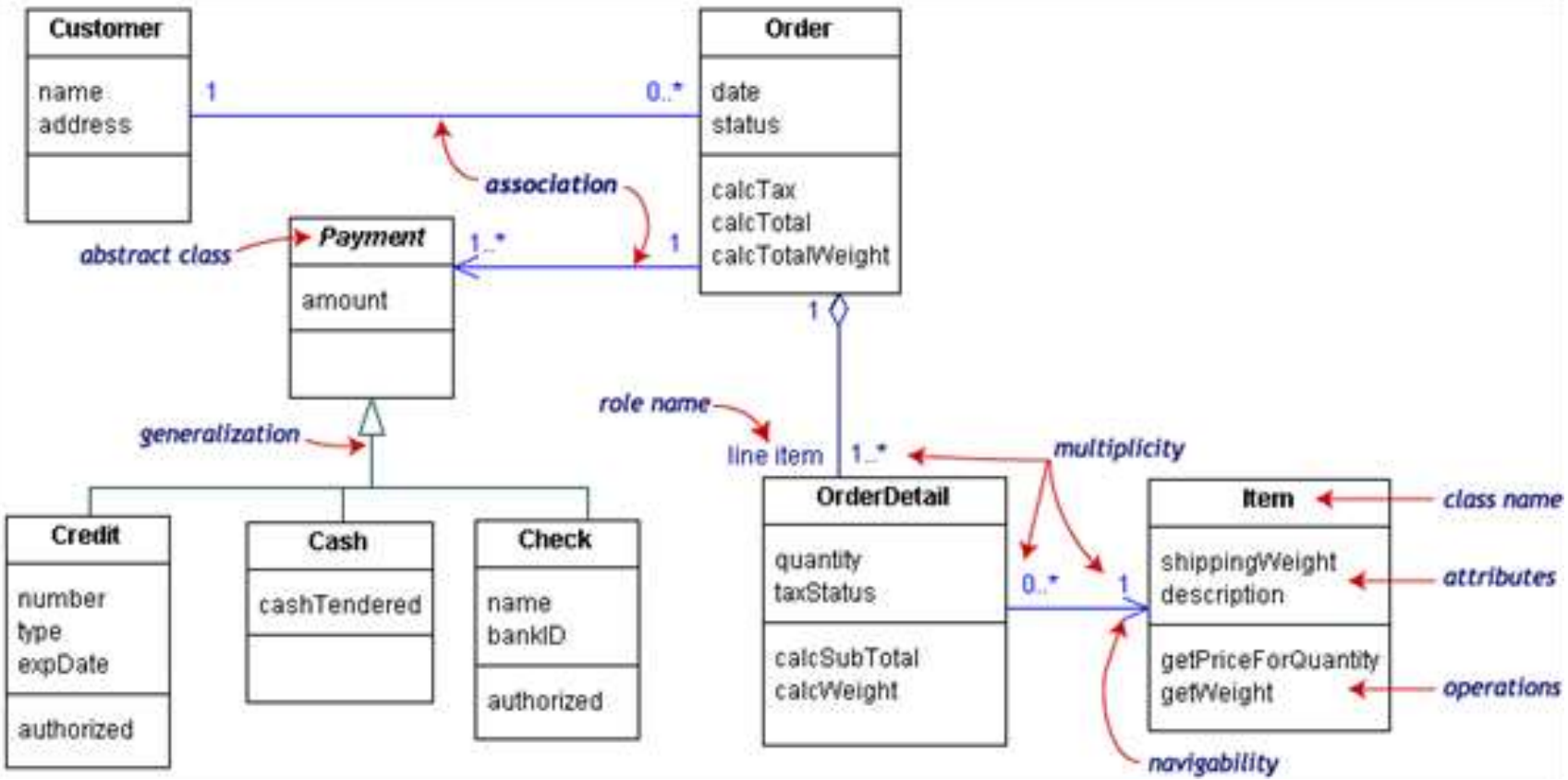
3. Type of association: Composition is a **strong** Association whereas Aggregation is a **weak** Association.

- An Invoice is composed of one or more Entries
- An Invoice is created for exactly one Customer
- An Invoice is created from one or more CustomerOrders
- An Invoice is used to update the AccountsReceivable account
- An Inventory is composed of zero or more StockItems
- A PackingOrder is initiated from one or more Invoices
- A PackingOrder is created from Inventory items
- A PackingOrder is used to update the Inventory
- A StockOrder is received from a Supplier
- A StockOrder is used to update the Inventory
- A StockOrder is used to update the AccountsPayable account


CustomerOrder
+1
Invoice
+1
AccountsReceivable
Account
+1
Customer
Entry
PackingOrder
Created from
updates
Supplier
Inventory
Received from
StockOrder
updates
StockItem
AccountsPayable
updates

Models a customer order from a retail catalog.

A customer order from a retail catalog. The central class is the **Order**. Associated with it are the **Customer** making the purchase and the **Payment**. A **Payment** is one of three kinds: **Cash**, **Check**, or **Credit**. The order contains **OrderDetails** (line items), each with its associated **Item**.

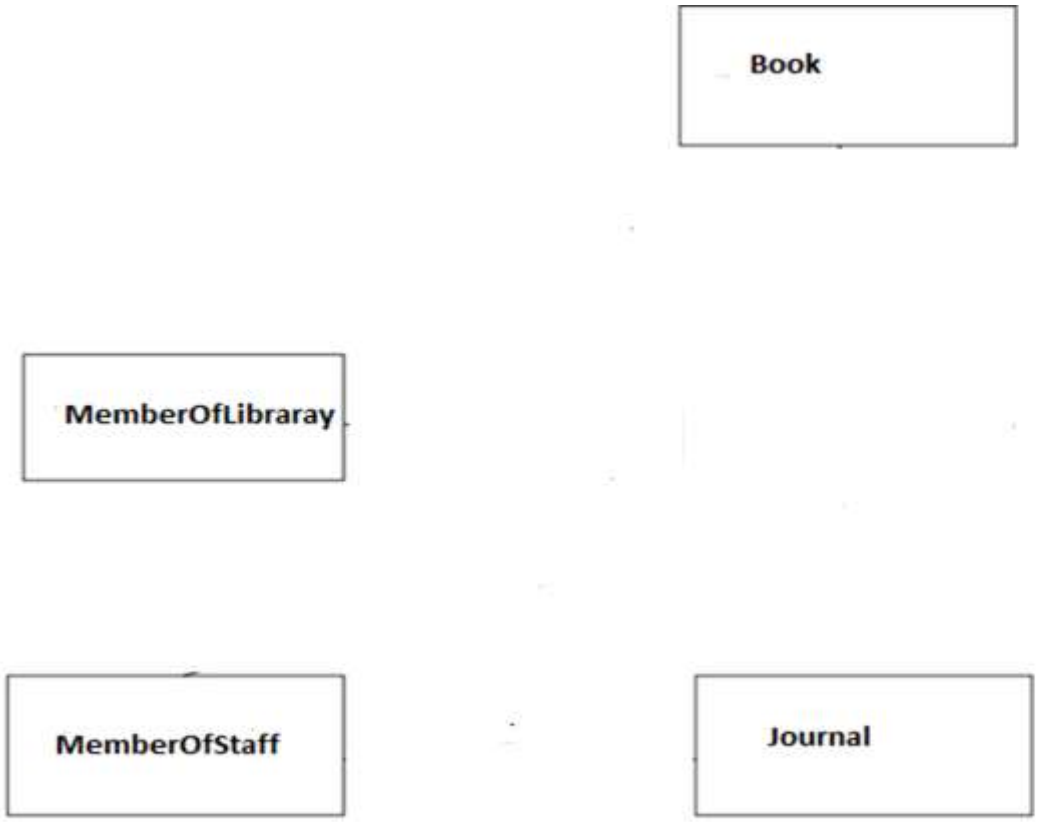

Customer
name
address
1
0..*
Order
date
status
calcTax
calcTotal
calcTotalWeight
association
abstract class
Payment
amount
1..*
1
1
role name
line item
1..*
multiplicity
generalization
Credit
number
type
expDate
authorized
Cash
cashTendered
Check
name
bankID
authorized
OrderDetail
quantity
taxStatus
calcSubTotal
calcWeight
0..*
1
Item
shippingWeight
description
getPriceForQuantity
getWeight
class name
attributes
operations
navigability

Identifying a list of candidate classes: Example

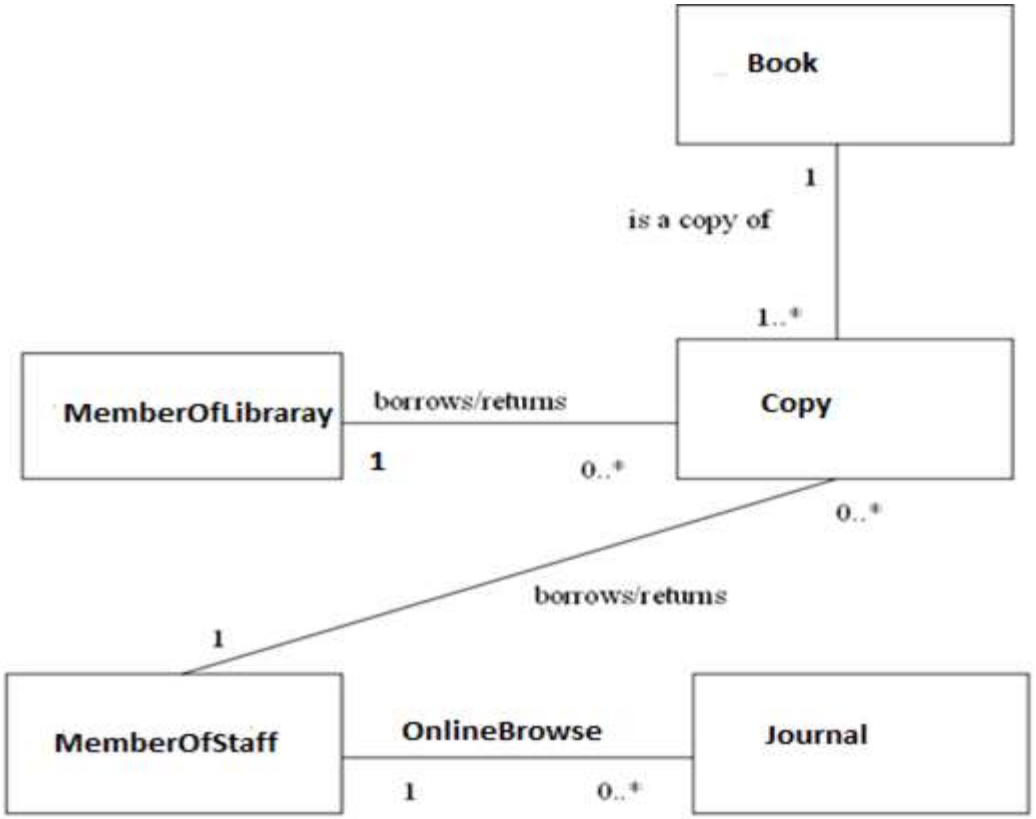
The library contains books and journals. It may have several copies of a given book. Some of the books are for short term loans only. All other books may be borrowed by any library members for 3 weeks. Member of the library can normally borrow up to 6 items at a time, but members of staff may borrow up to 12 items at one time. Only members of staff may borrow journals.

The system must keep track of when books and journals are borrowed and returned by the user, enforcing the rules described above.

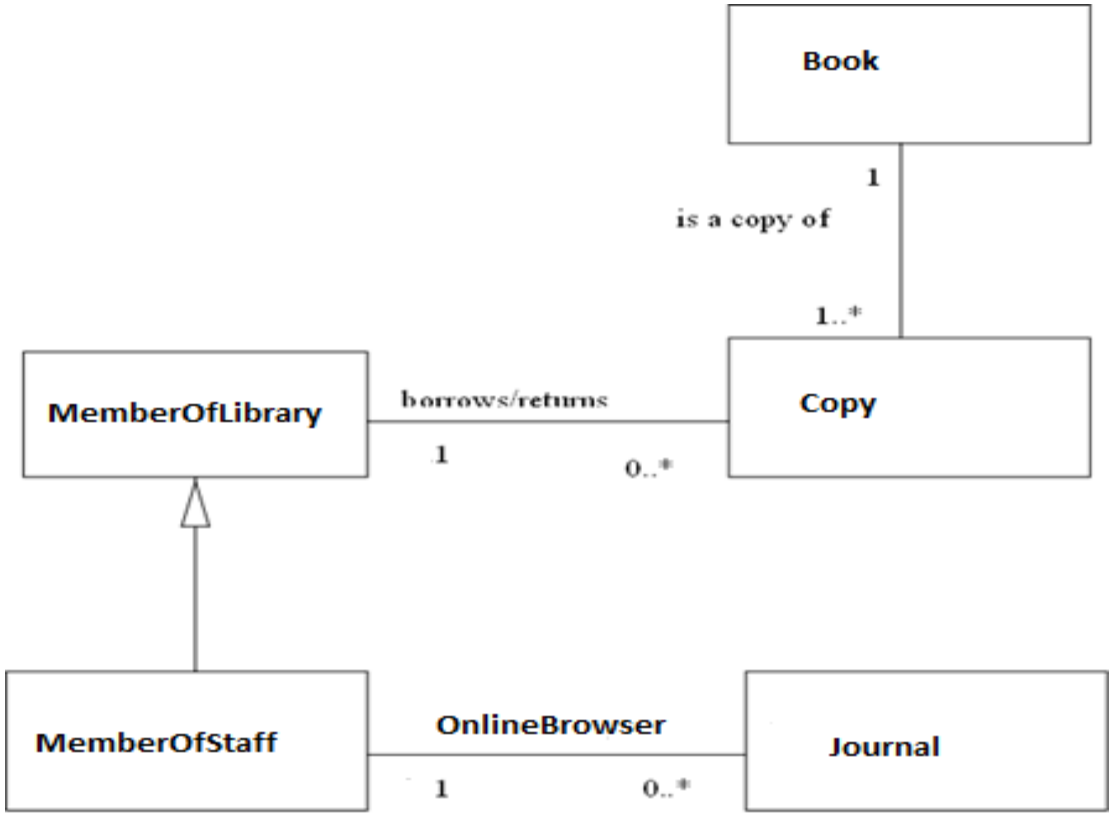
Identified Classes for Library System



Initial Class model for Library System

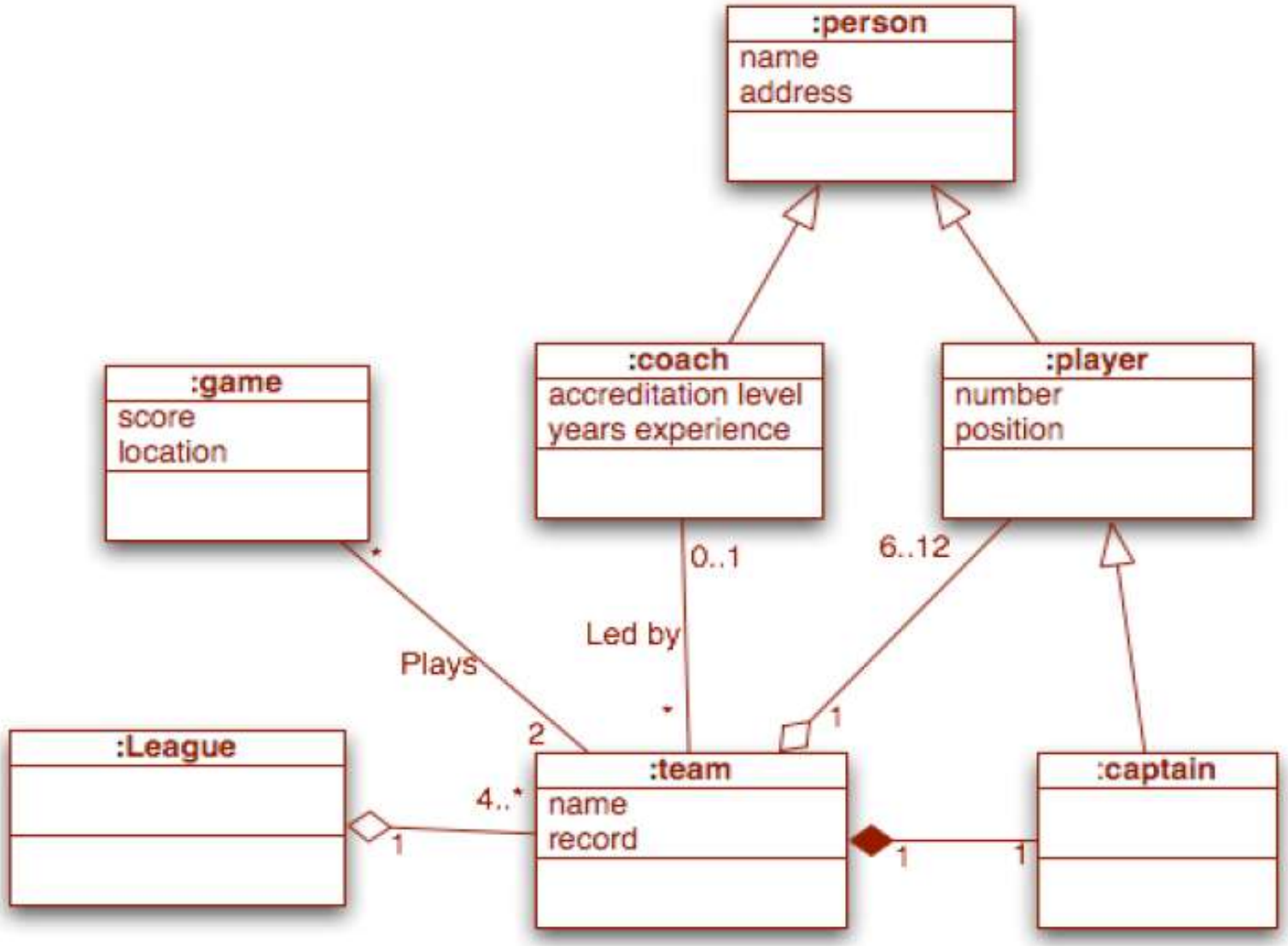


Revised class model for Library System



Class Diagram Example

A hockey league is made up of at least four hockey teams. Each hockey team is composed of six to twelve players, and one player captains the team. A team has a name and a record. Players have a number and a position. Hockey teams play games against each other. Each game has a score and a location. Teams are sometimes lead by a coach. A coach has a level of accreditation and a number of years of experience, and can coach multiple teams. Coaches and players are people, and people have names and addresses.


:person
name
address
:coach
accreditation level
years experience
:player
number
position
:game
score
location
0..1
6..12
Led by
Plays
*
*
2
1
:League
:team
name
record
:captain
4..*
1
1
1

Design concepts

Functional Independence

- Cohesion – is an indication of the relative functional strength of a module
- Coupling – is an indication of the relative interdependence among modules

State Diagram

State Chart Diagram

State Transition Diagram

Introduction

- Two interaction diagrams with objects of the same class receiving the same messages may respond differently

- This is because an object's behavior is affected by the values of its attributes

- UML State Machine Diagram records these dependencies

State Transition Diagram

- **A state transition diagram is a technique to depict:**

  1. **The states of an entity**
  2. **The transitions of states of the entity**
  3. **The trigger or the event that caused the transition of state of the entity**

- **The *entity* may be a physical device such as a light switch or a vending machine; it may be a software system or component such as a word processor or an operating system; it may be a biological system such as a cell or a human; or - - - -**

This modeling technique came from a more formal area called automata theory. State transition diagram depicted a *Finite State Machine*.

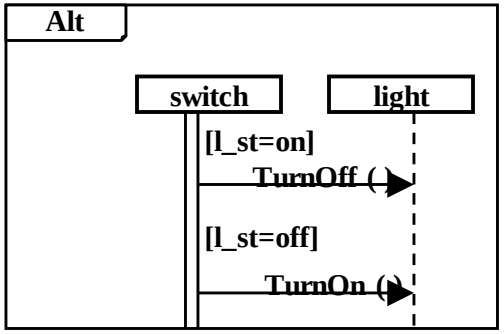
Software Program View

- **The end product of a software is a program which executes. In depicting the program (or an object) we can consider:**
  - **Variables which take on different values**
  - **Control structure and assignment statements (events) in the program that change the values of the variables**

1. ***Combination of values of the data (variables & constants) at any point of the program represent the program state at that point.***
2. ***The change made to the values of the variables through assignment statements represent a transition of state***

A very simple example

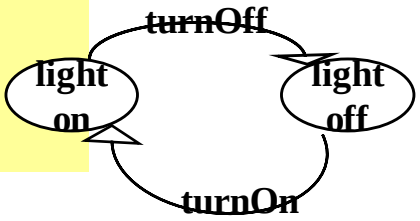
light switch



1. "Sequence diagram" (alternative fragment) for switch and light interaction

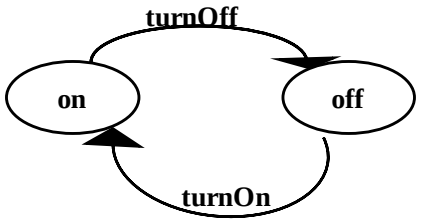
From State (light)	Event (switch)	To State (light)
on	turnOff	off
off	turnOn	on

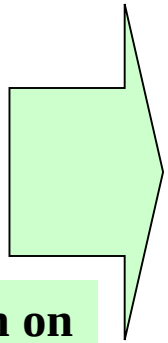
2. "State transition table" for light with switch events

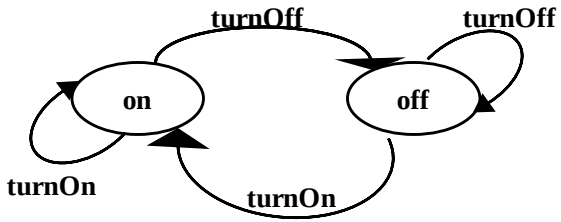


3. "State transition diagram" for light with switch events

A little “more” on the light switch







What happens if we turn on a light that is already on?

state can “transition” to its current state

Using State Transition Diagram

- **Model the entity at the "abstraction" level where the *number of states is* "*manageable.*"**

1. **List (design) the *states* (should not be large)**
2. **List *events* that will trigger the state transition (should not be big)**
3. **There must be a *starting state***
4. **There must be a *terminating state or states***
5. **Design the *transition rules* (*the bulk of your design work is thinking through the transition rules*)**

1.The above is not necessarily performed in sequence; iterate through these.

2. Even with a modest number of states and events, the *state transition* diagram, which really depicts the transition rules, can be enormous.

State Diagrams

- State diagrams are used to show possible states a single object can get into
  - shows states of an object

- How object changes state in response to events
  - shows transitions between states

Elements of State Diagram

- Start marker
- Stop marker
- States – box with rounded corners
- Transitions – shown as arrows between states
- Events – that cause transition between states
- Action – an object's reaction to an event
- Guard

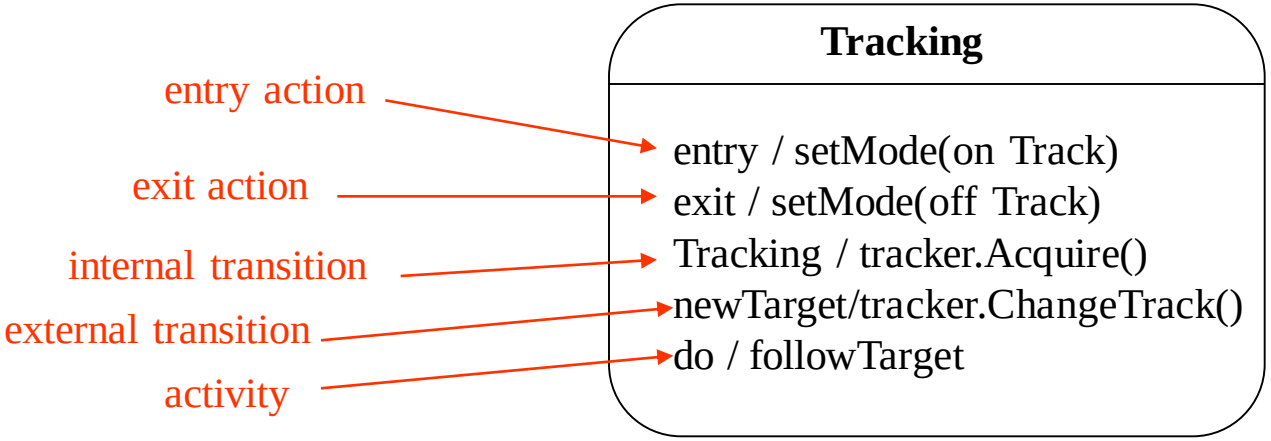
Naming Conventions for a state

- Each unique state has a unique name
- State Names are **verb** phrases
- A noun to name a state – **incorrect**

State Diagrams: States

- States are represented as rounded boxes which contain:
  - the **state name**
  - and the following optional fields
    - **entry and exit actions**: entry and exit actions are executed whenever the state is entered or exited, respectively
    - **Internal transitions**: A response to an event that causes the execution of an action but does not cause a change of state or execution of exit or entry actions.
    - **External transition**: A response to an event that causes a change of state or a self-transition, together with a specified action.
    - **Activities**: Typically, once the system enters a state it sits idle until an event triggers a transition. Activities help you to model situations where while in a state, the object does some work that will continue until it is interrupted by an event

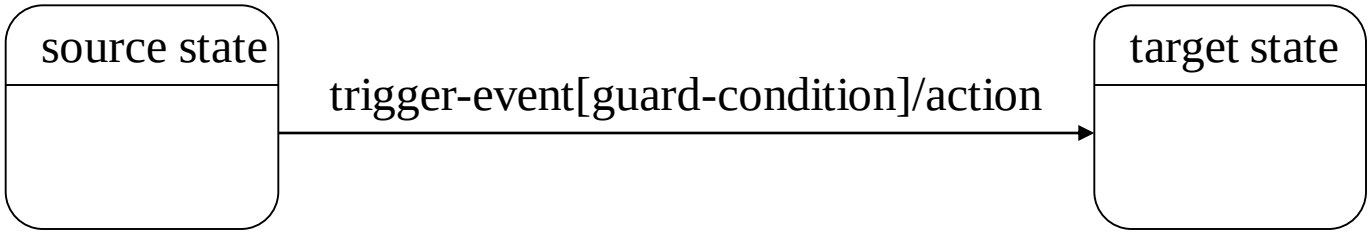
State Diagrams: States



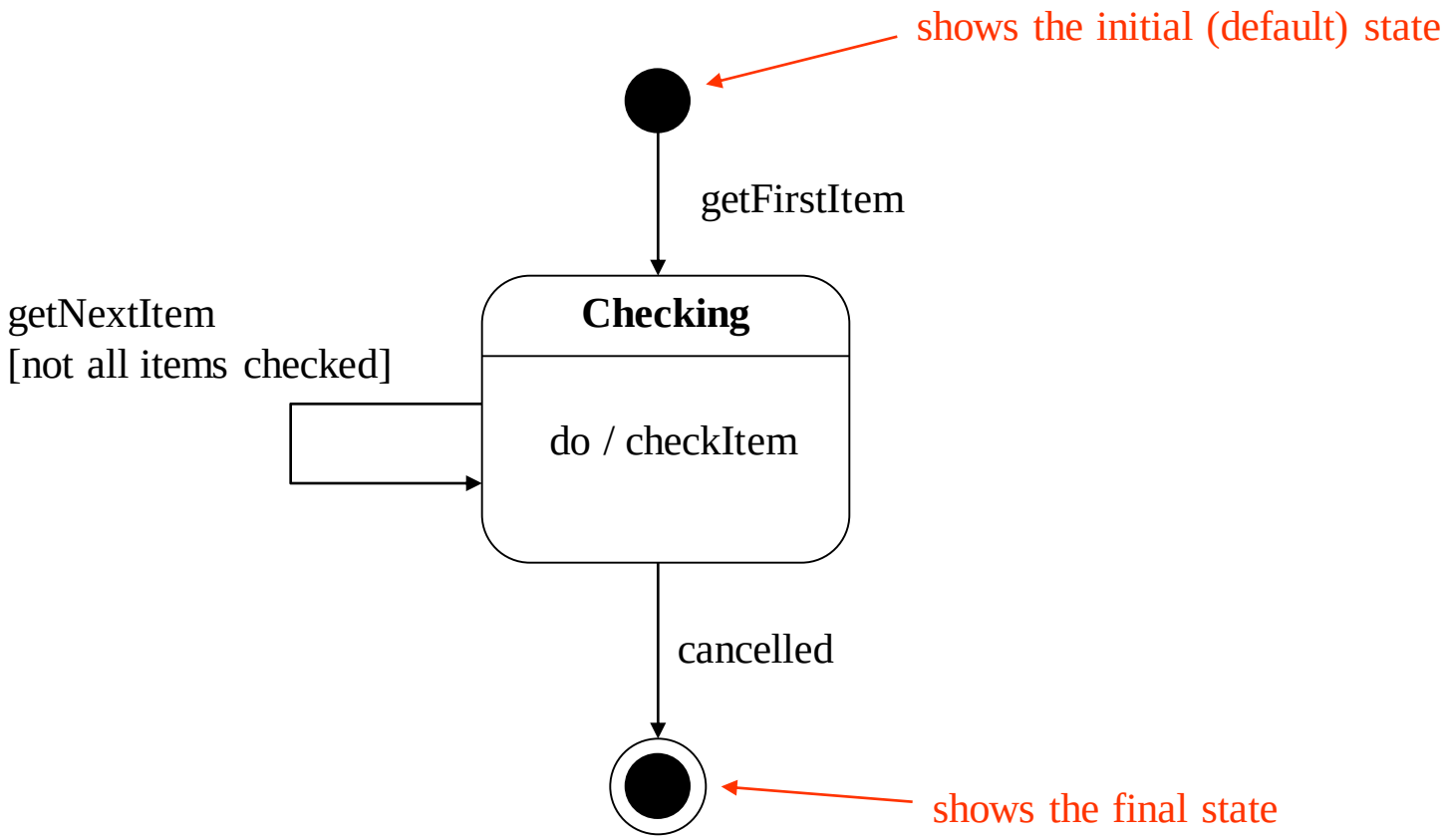
Note that, “entry”, “exit”, “do” are keywords

State Diagrams: Transitions

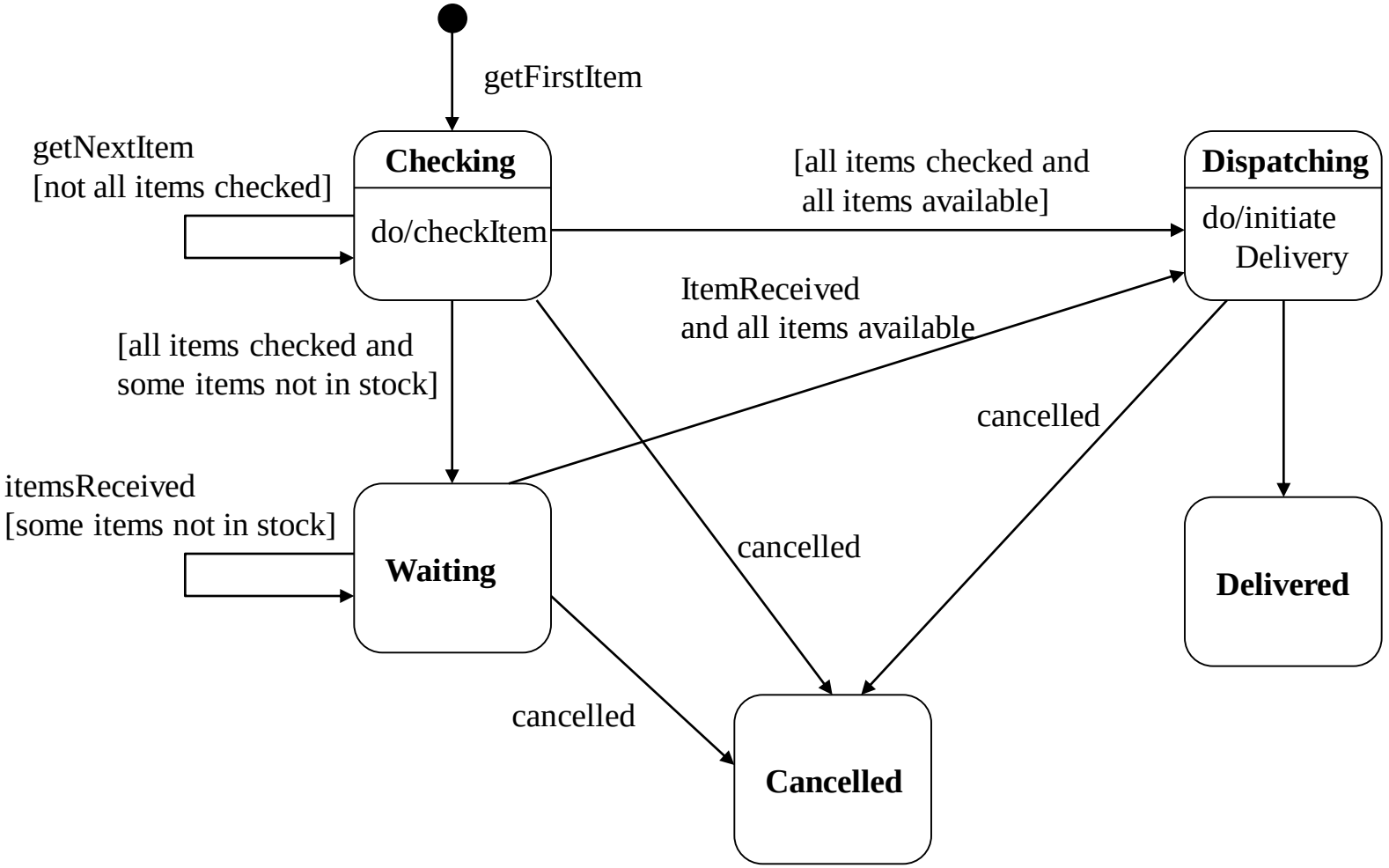
- Transitions
  - **source state** and **target state**: shown by the arrow representing the transition
  - **trigger event**: the event that makes the transition fire
  - **guard condition**: a Boolean expression that is evaluated when the trigger event occurs, the transition can fire only if the guard condition evaluates to true
  - **action**: an executable atomic computation that can directly act on the object that owns the state machine or indirectly on other objects that are visible to the object
  - **initial and final states**: shown as filled black circle and a filled black circle surrounded by an unfilled circle, respectively



State Diagrams

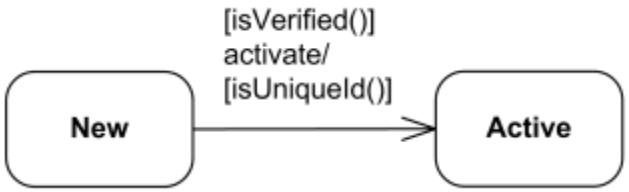


State Diagram Example: States of an Order object



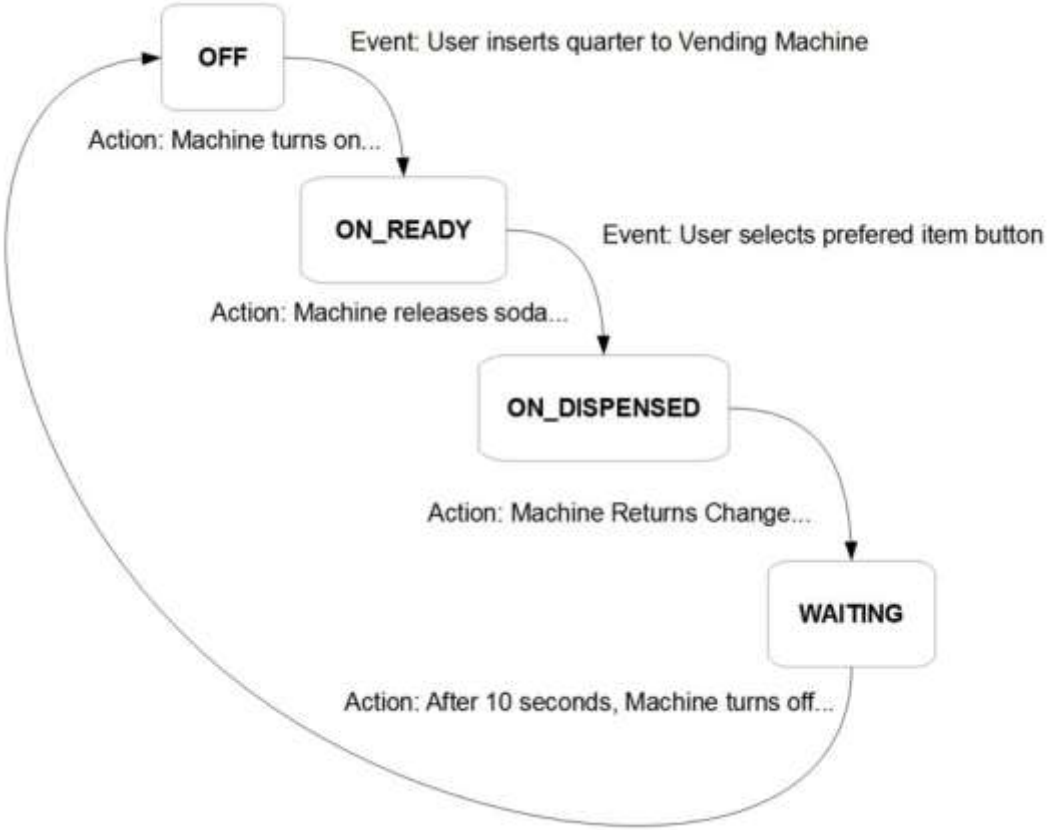
Types of State Machine Diagram

- Protocol State Machines: These are used to express a **usage protocol** or a **lifecycle** of some **classifier**. It shows which operations of the classifier may be called in each state of the classifier, under which specific conditions, and satisfying some optional postconditions after the classifier transitions to a target state.
- Behavioral State Machines: These are specialization of behavior and is used to specify discrete behavior of a part of designed system through finite state transitions.



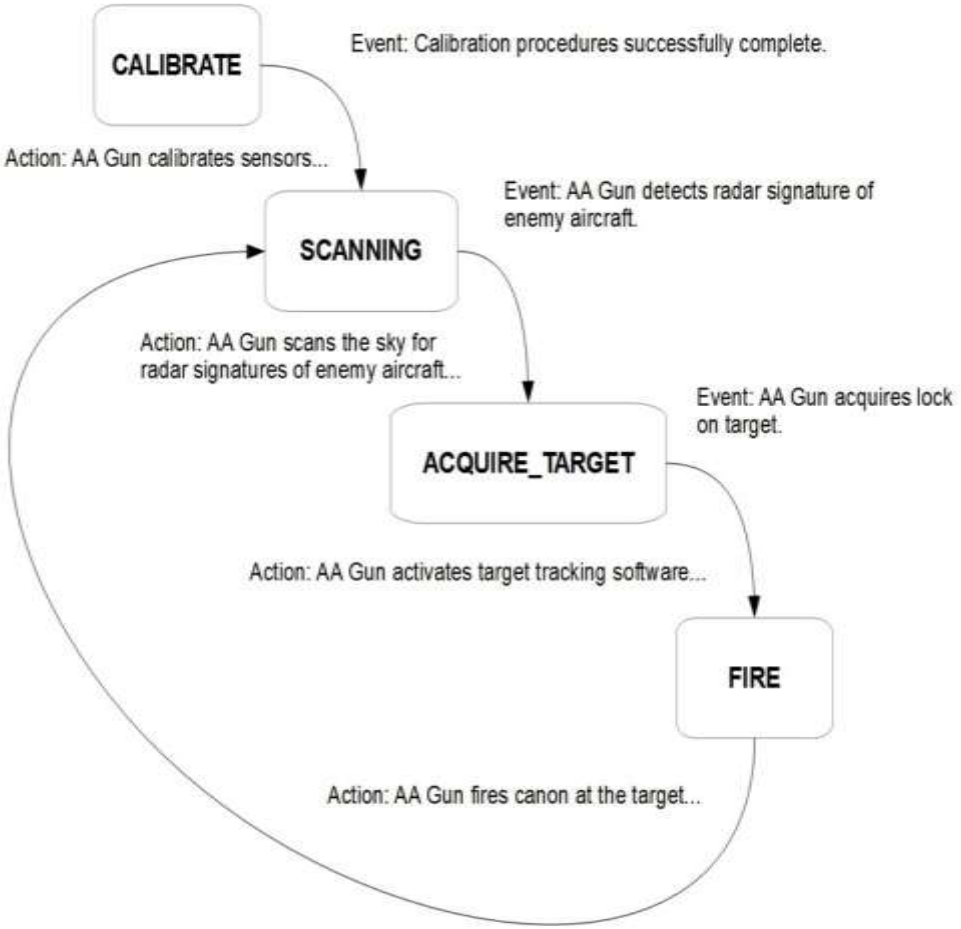
Example of Protocol State Machines

Soda Vending Machine



Example of Behavioral State Machines

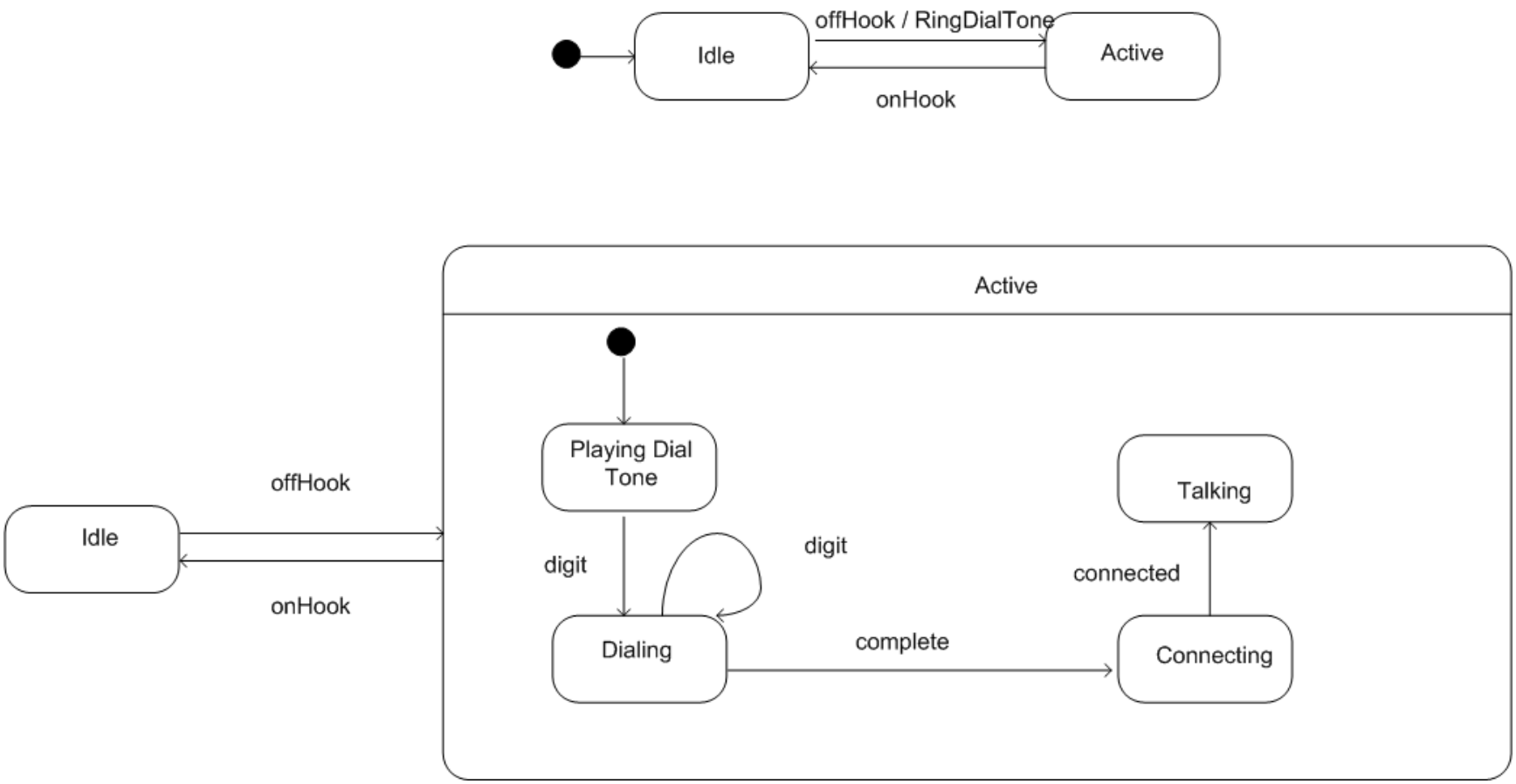
Anti-Aircraft (AA) Gun



Advanced State Machine Modeling

- Nested states
- Concurrent states

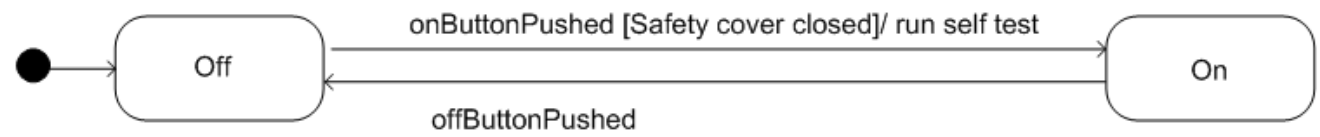
Nested State Machine- Telephone

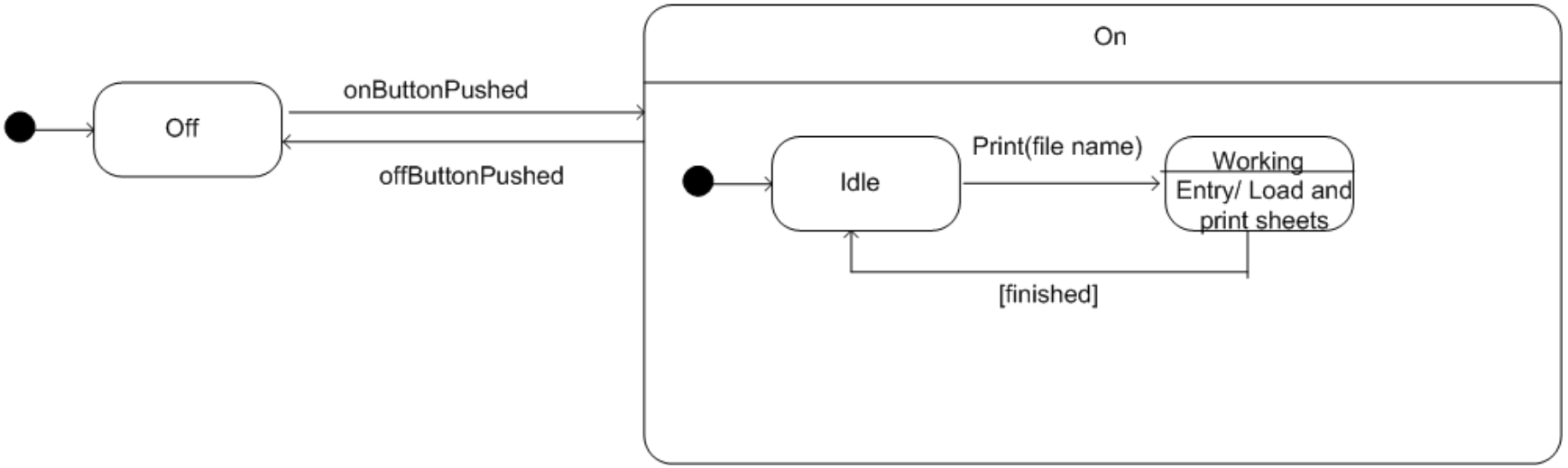


Nested State Machine- Printer

- Example: When the printer is in On state, it may also be in Idle or Working
- To show these two states, draw lower level state diagrams within On state
- When the printer is on, it begins at Idle state, so printer is in both On and Idle state
- When print message is received, it moves to working state and also remains in On state

Nested State Machine- Printer

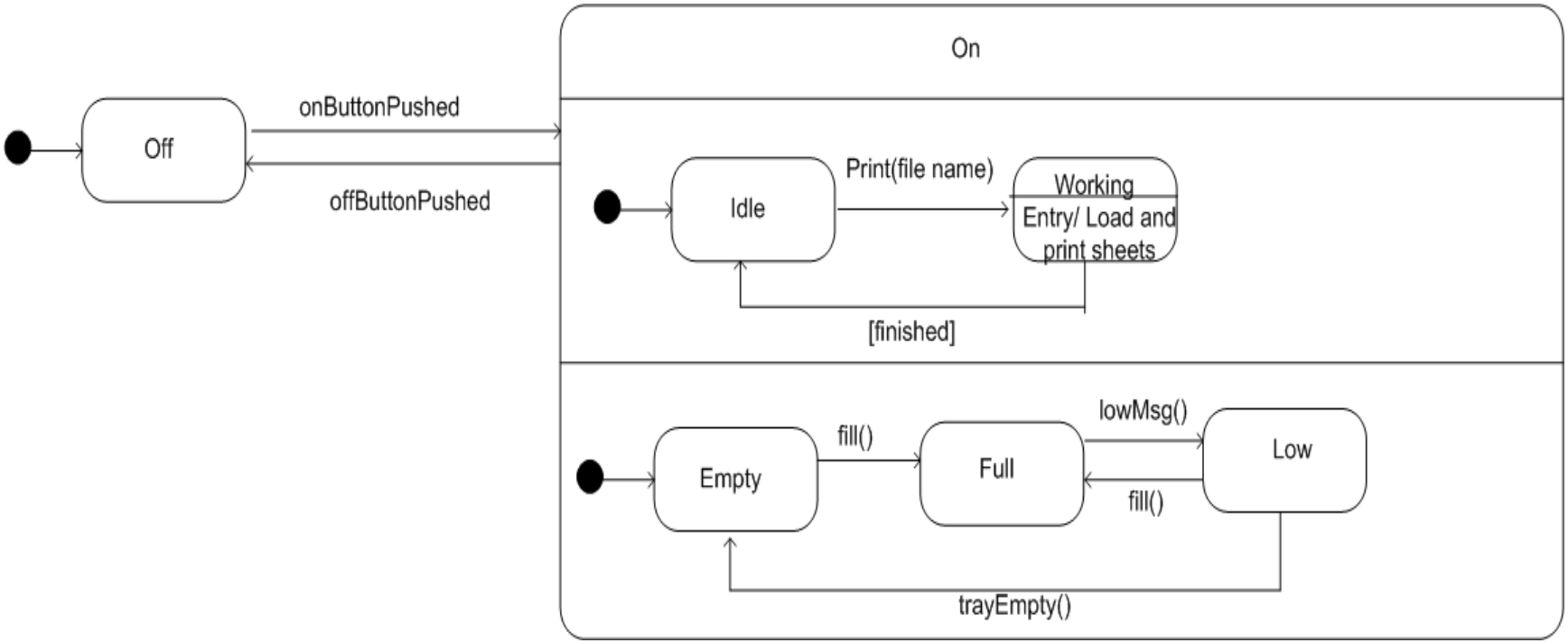




Concurrent states

- A Printer object cycles between two separate paths. The two independent paths are;
  - Representing states of the work cycle.
  - Representing states of the input paper tray

Concurrent State Machine- Printer



State Diagrams Importants

- Use them to show the behavior of a single object not many objects
  - for many objects use interaction diagrams

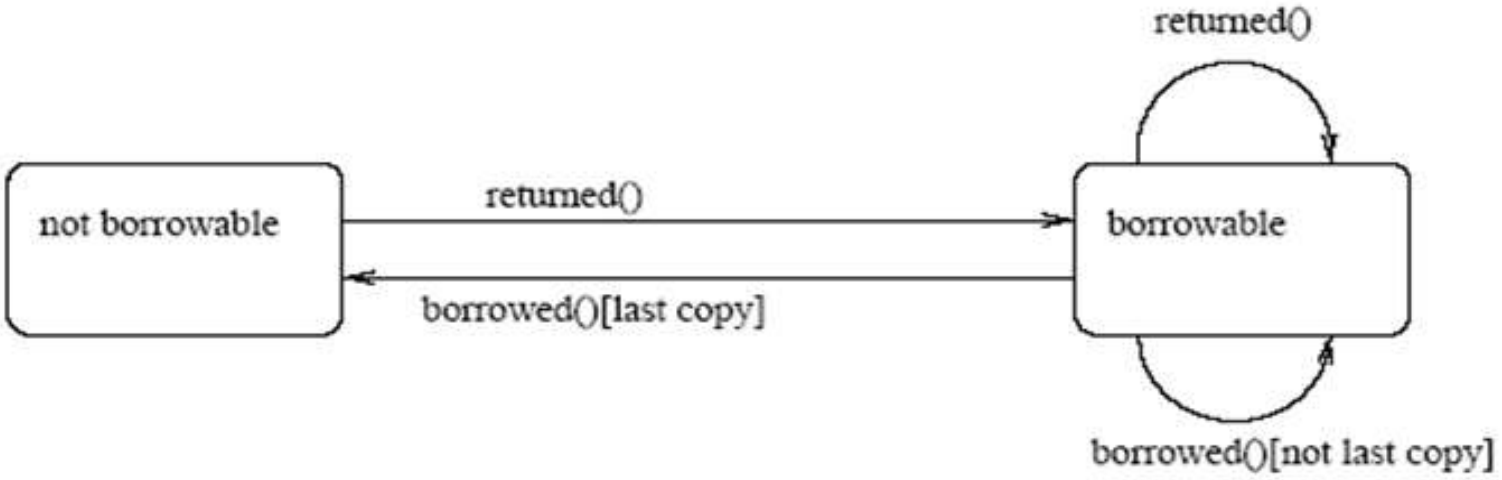
- Do not try to draw state diagrams for every class in the system, use them to show interesting behavior and increase understanding

CRC for Copy & Book

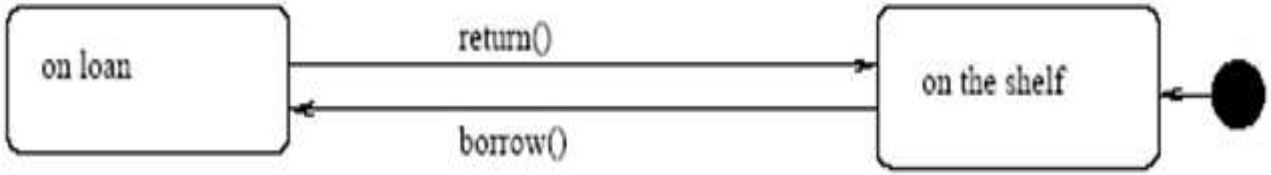
Copy	
Responsibilities	**Collaborators**
Maintain data about a particular copy of a book Inform corresponding `Book` when borrowed and returned	 `Book`

Book	
Responsibilities	**Collaborators**
Maintain data about one book Know whether there are borrowable copies	

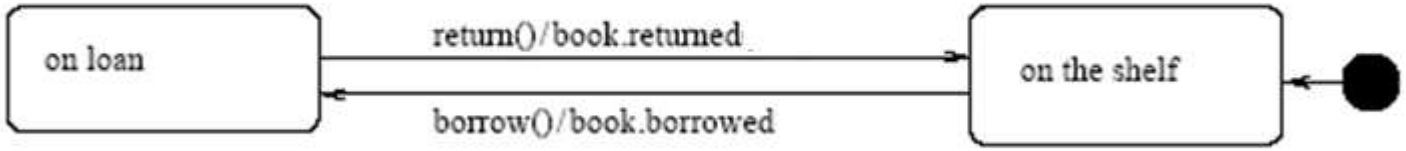
State Machine Diagram for class *Book*



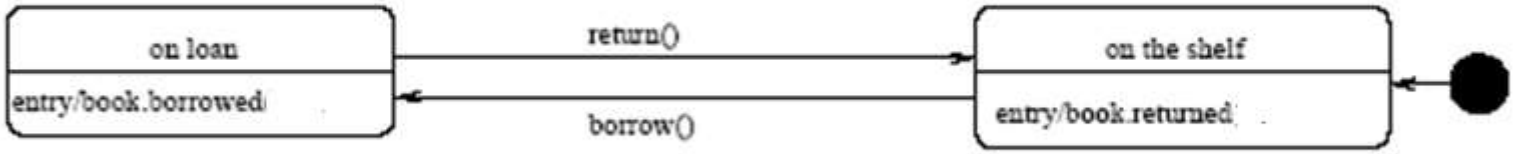
State Machine Diagram for class *Copy*



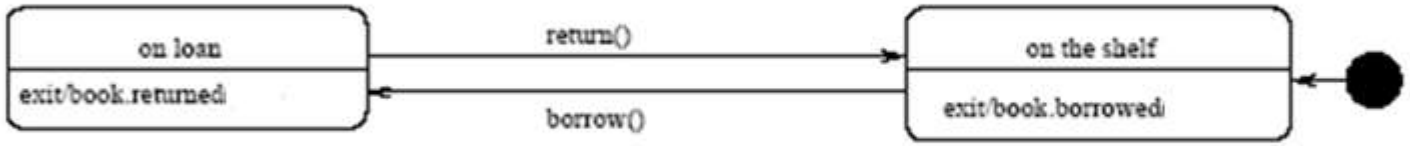
State Machine Diagram for class *Copy* with actions



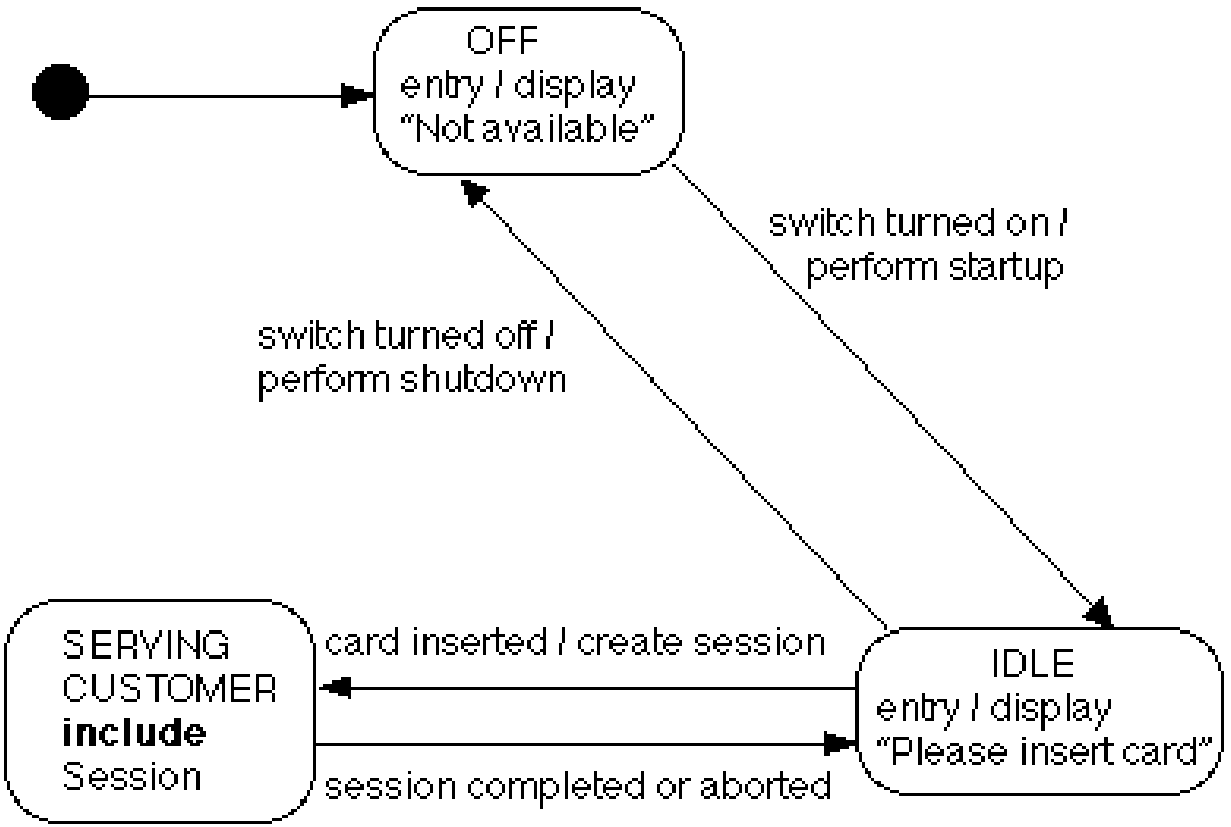
State Machine Diagram for class *Copy* with entry actions



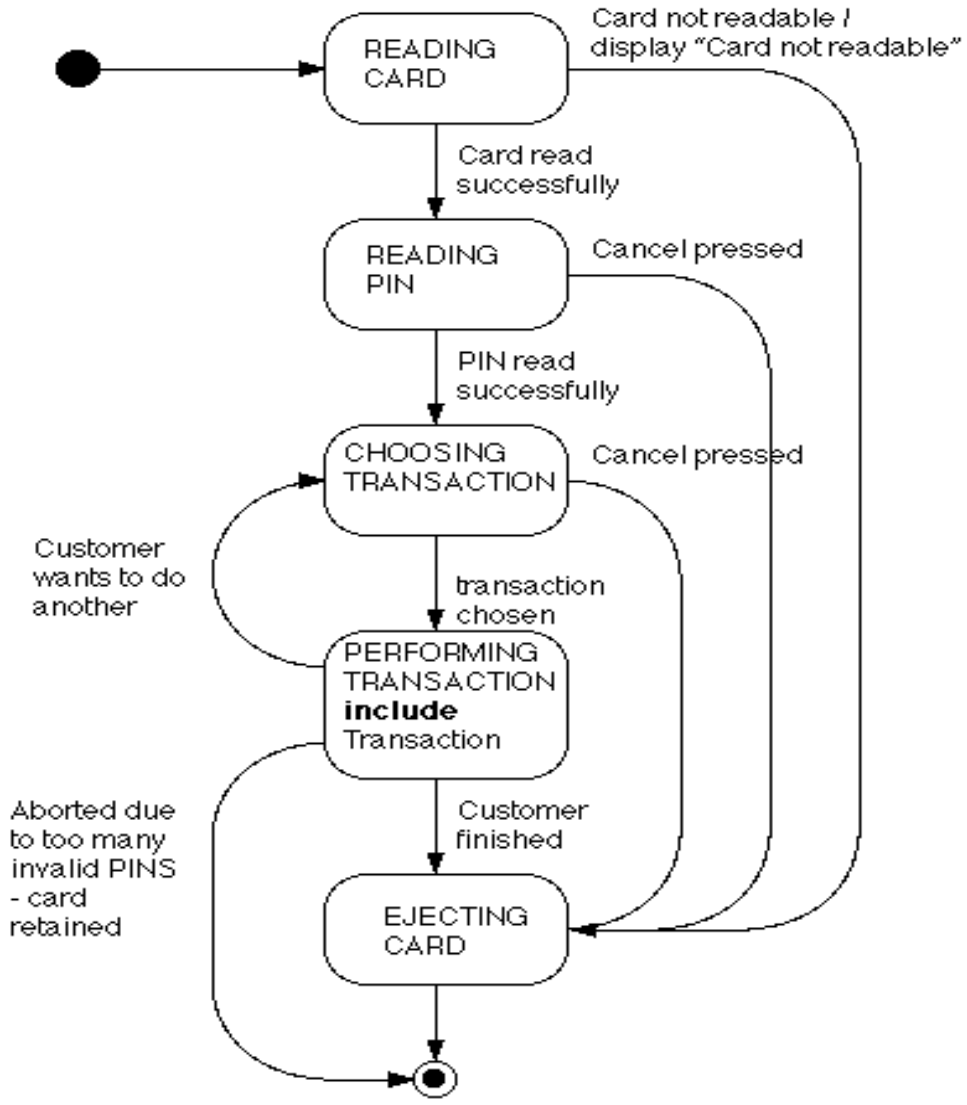
State Machine Diagram for class *Copy* with exit actions



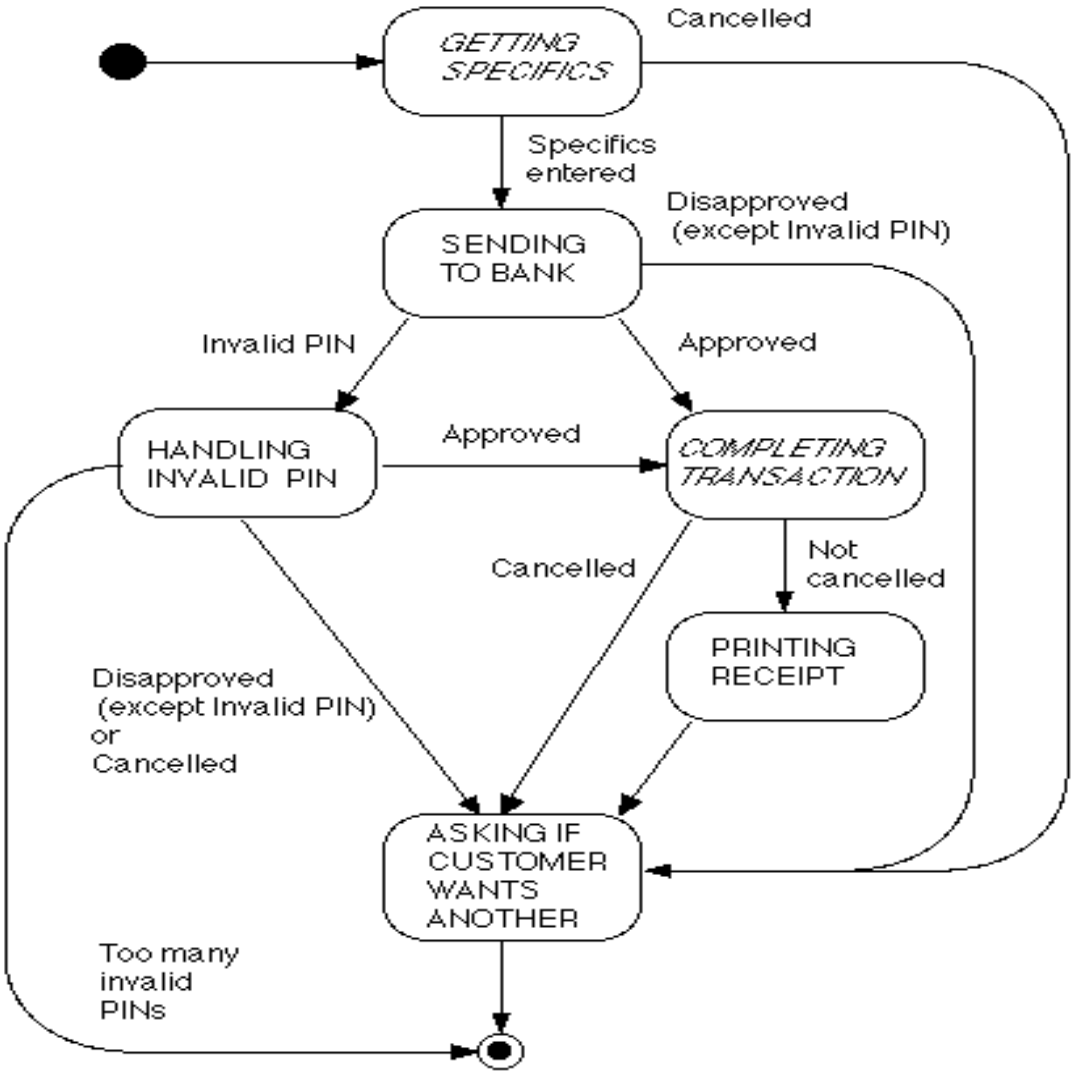
State-Chart for Overall ATM (includes System Startup and System Shutdown Use Cases)



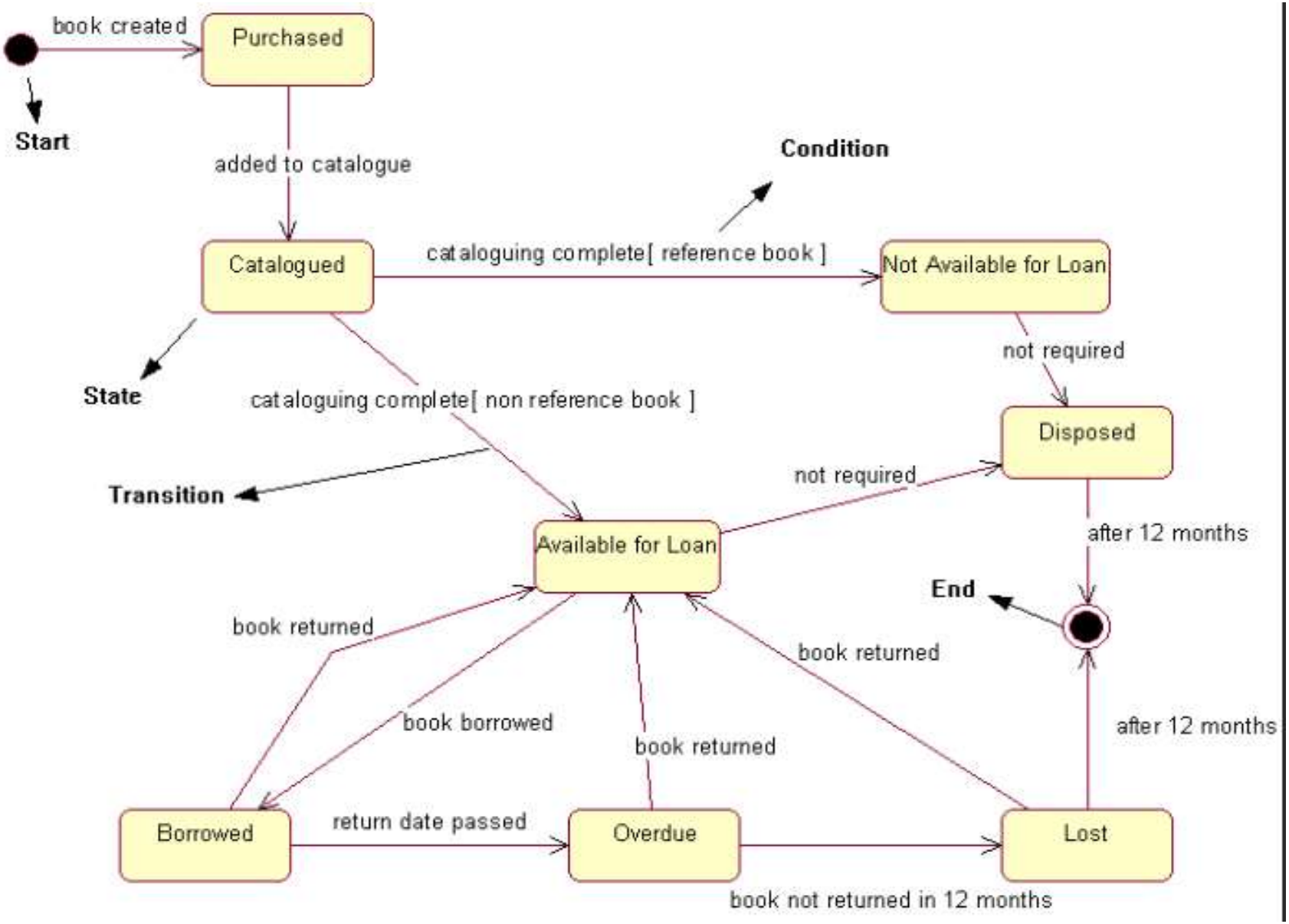
State-Chart for One Session



State-Chart for One Transaction
(italicized operations are unique to each particular type of transaction)



Librarians categorize the library books into loanable and non-loanable books. The non-loanable books are the reference books. However, the loanable books are the non-reference books. After cataloguing the books, the books are available for loan. Students who borrow the library books should return them back before the due date. Books that are 12 months over the due date would be considered as a lost state. However, if those books are found in the future, they must be returned back to the library. When the books are found not required in the library or have been damaged, the book would be disposed.


book created
Purchased
Start
added to catalogue
Condition
Catalogued
cataloguing complete[reference book]
Not Available for Loan
State
not required
cataloguing complete[non reference book]
Disposed
Transition
not required
Available for Loan
after 12 months
End
book returned
book returned
book borrowed
book returned
after 12 months
Borrowed
return date passed
Overdue
Lost
book not returned in 12 months

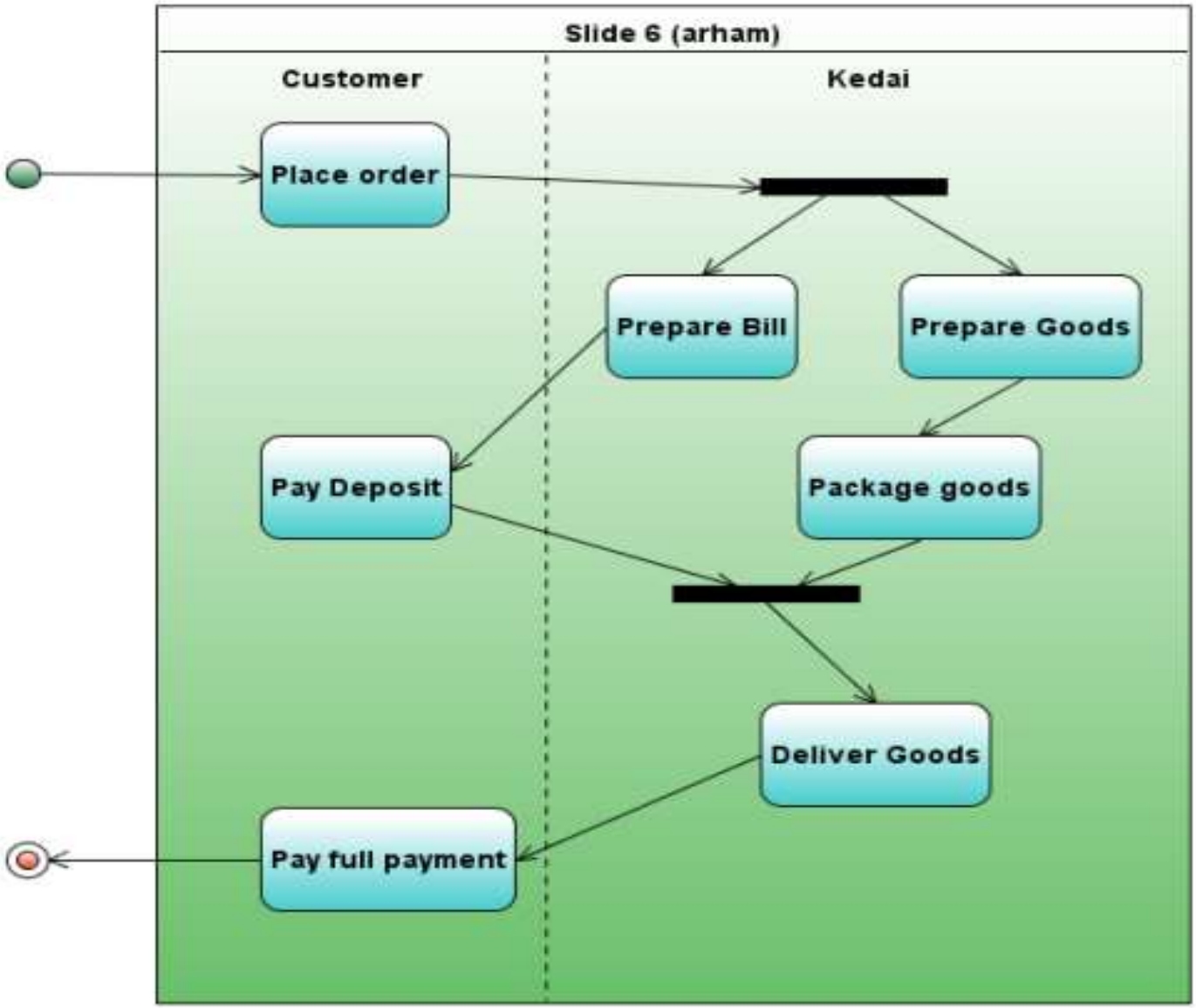
Difference between Activity and State Chart Diagram

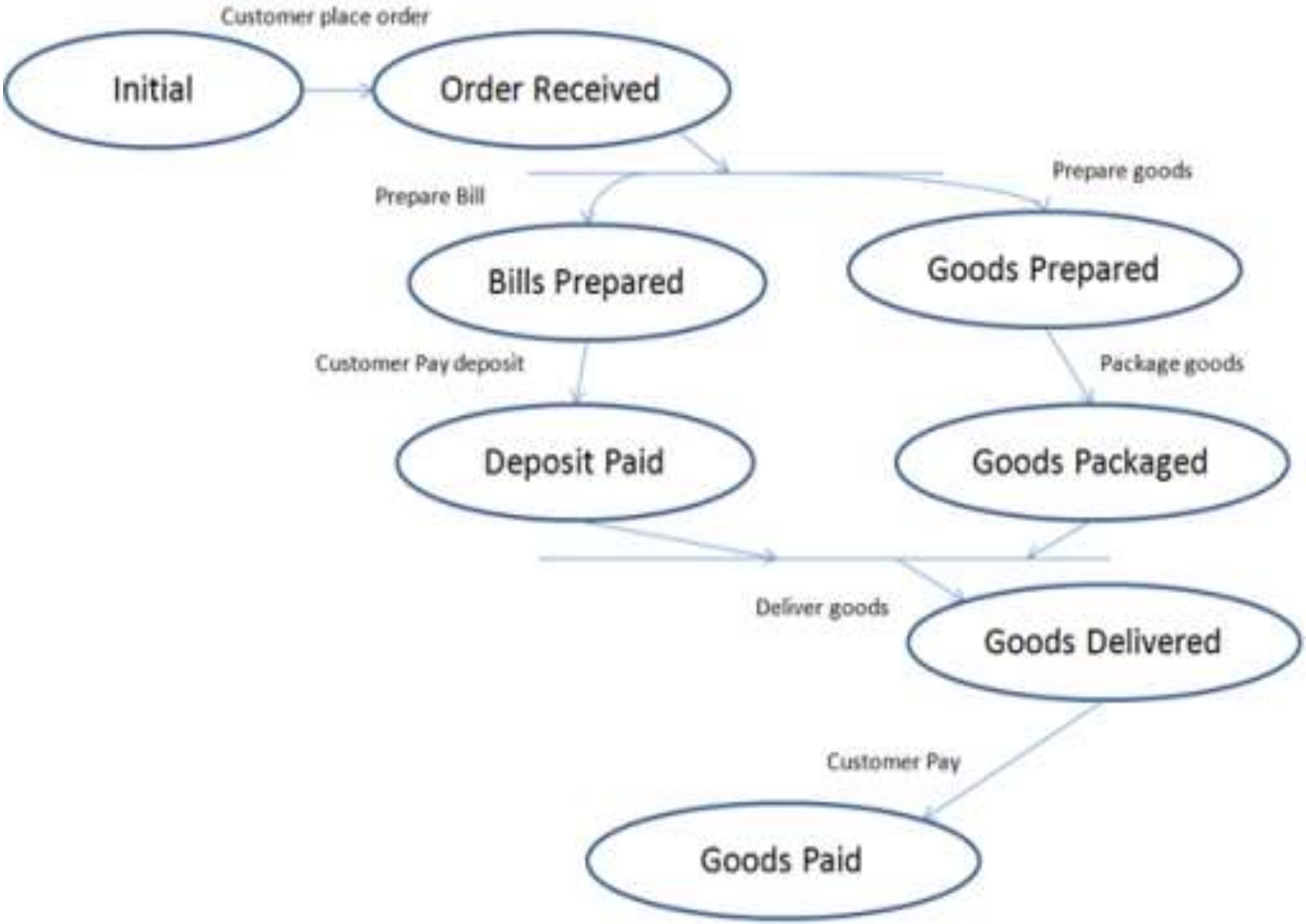
State diagram shows the object undergoing a process. It gives a clear picture of the changes in the object's state in this process.
e.g: ATM withdraw
Card object state: Checking, Approving, Rejecting

Activity diagram is a fancy flow chart which shows the flow of activity of a process.

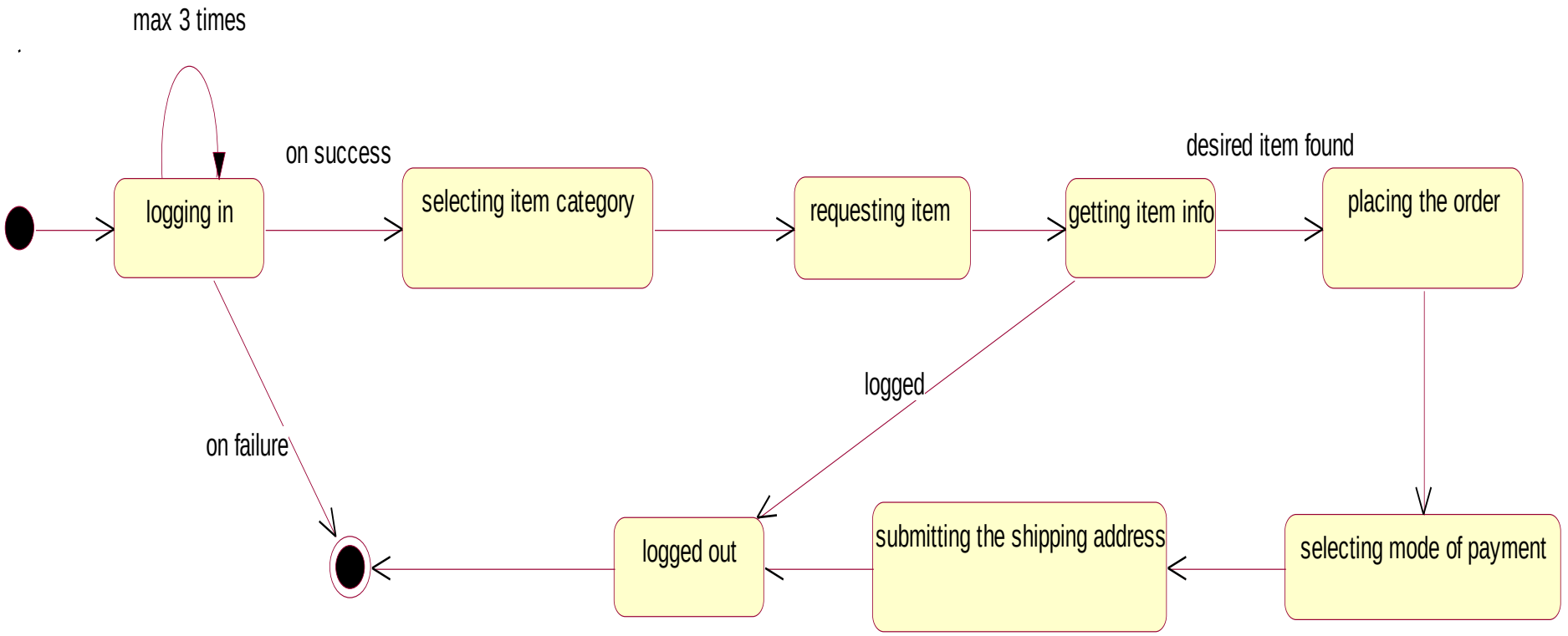
e.g: ATM withdraw
Withdraw activity: Insert Card, Enter PIN, Check balance, with draw money, get card

State chart shows the dynamic behavior of an object.
Activity diagram shows the workflow behavior of an operation as set of actions

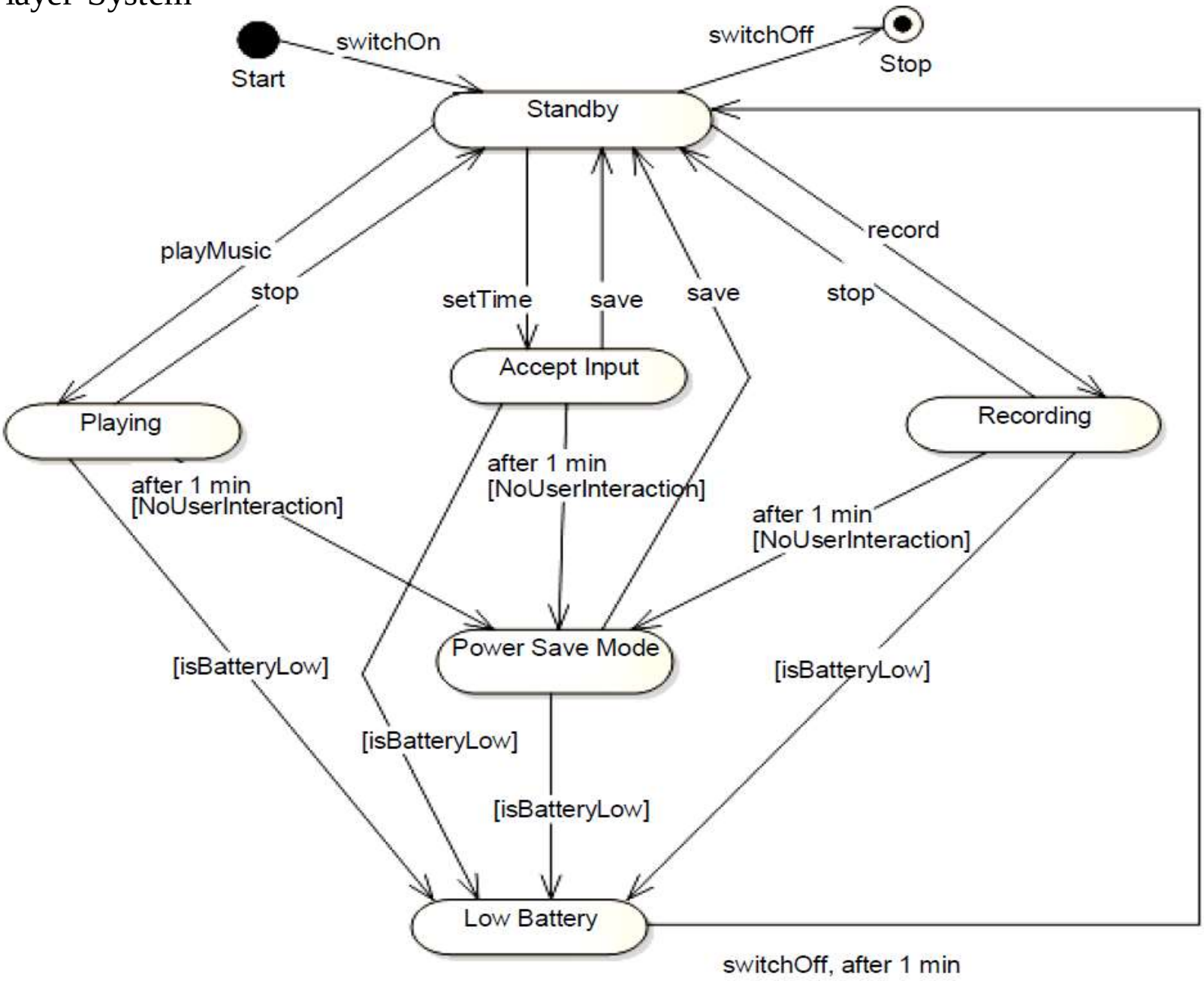

Slide 6 (arham)
Customer
Kedai
Place order
Prepare Bill
Prepare Goods
Pay Deposit
Package goods
Deliver Goods
Pay full payment


Customer place order
Initial
Order Received
Prepare Bill
Prepare goods
Bills Prepared
Goods Prepared
Customer Pay deposit
Package goods
Deposit Paid
Goods Packaged
Deliver goods
Goods Delivered
Customer Pay
Goods Paid

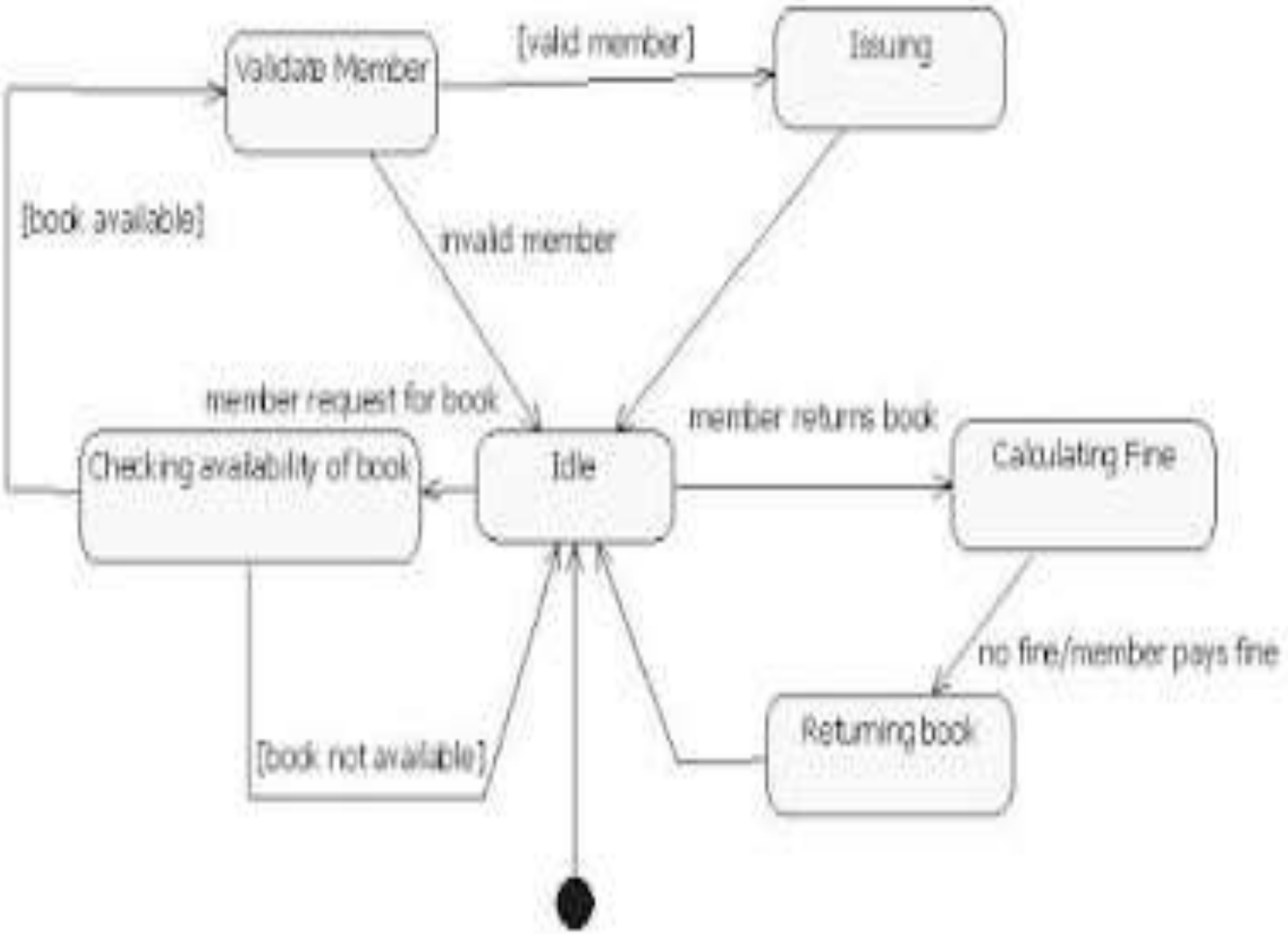
Online Shopping System



Music Player System



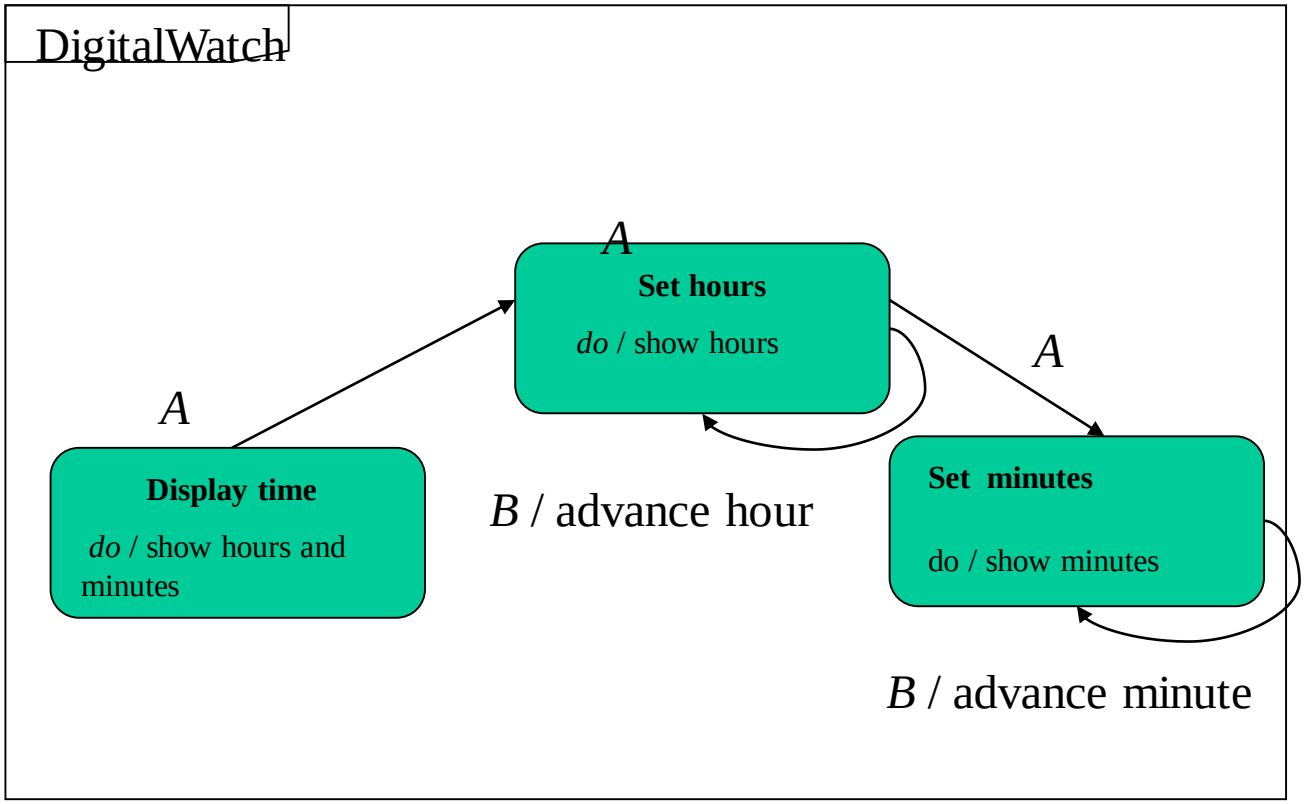
Library Management System



Problem # 1

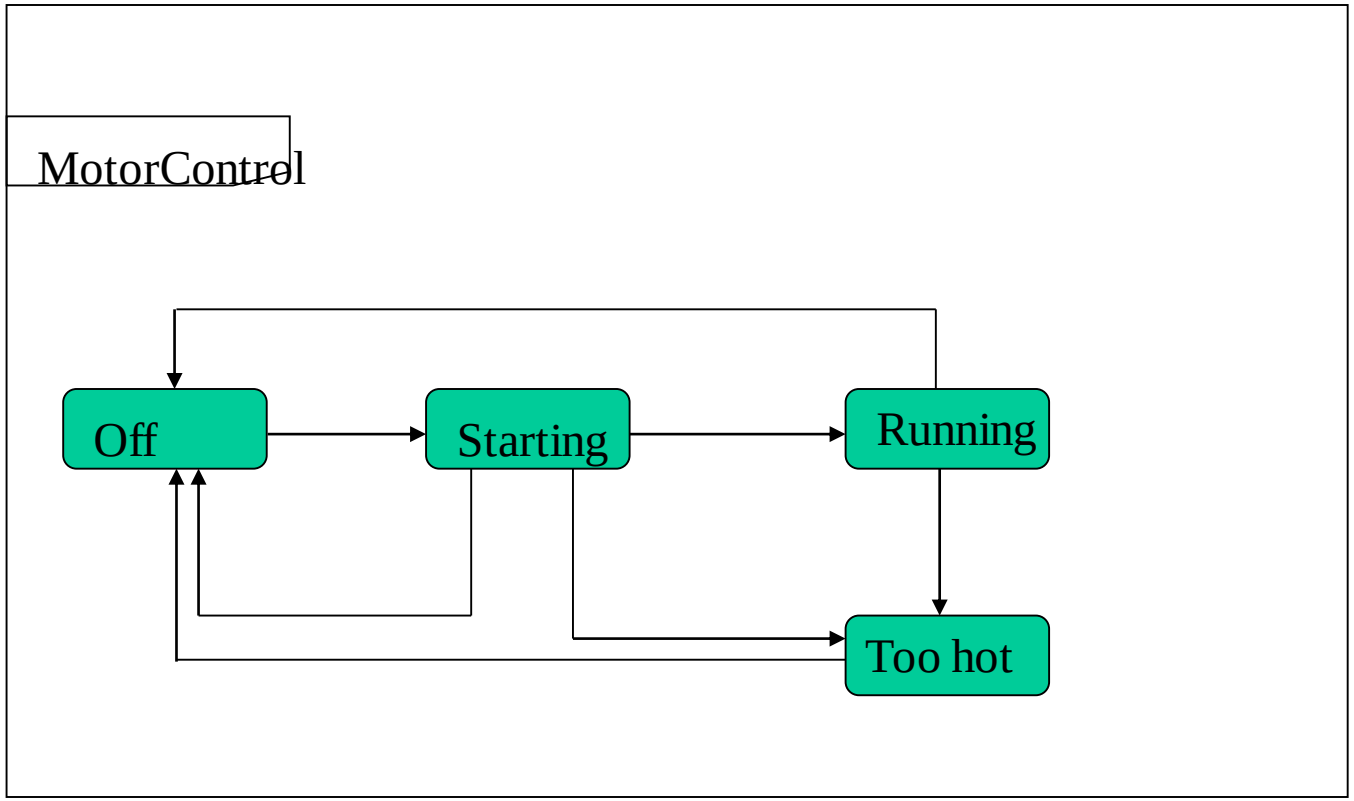
- A simple digital watch has a display and two buttons to set it, the A button and the B button. The watch has two modes of operation, display time and set time.
- In the display time mode, the watch displays hours and minutes, separated by a flashing colon.
- The set time mode has two submodes, set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

Solution of Problem # 1

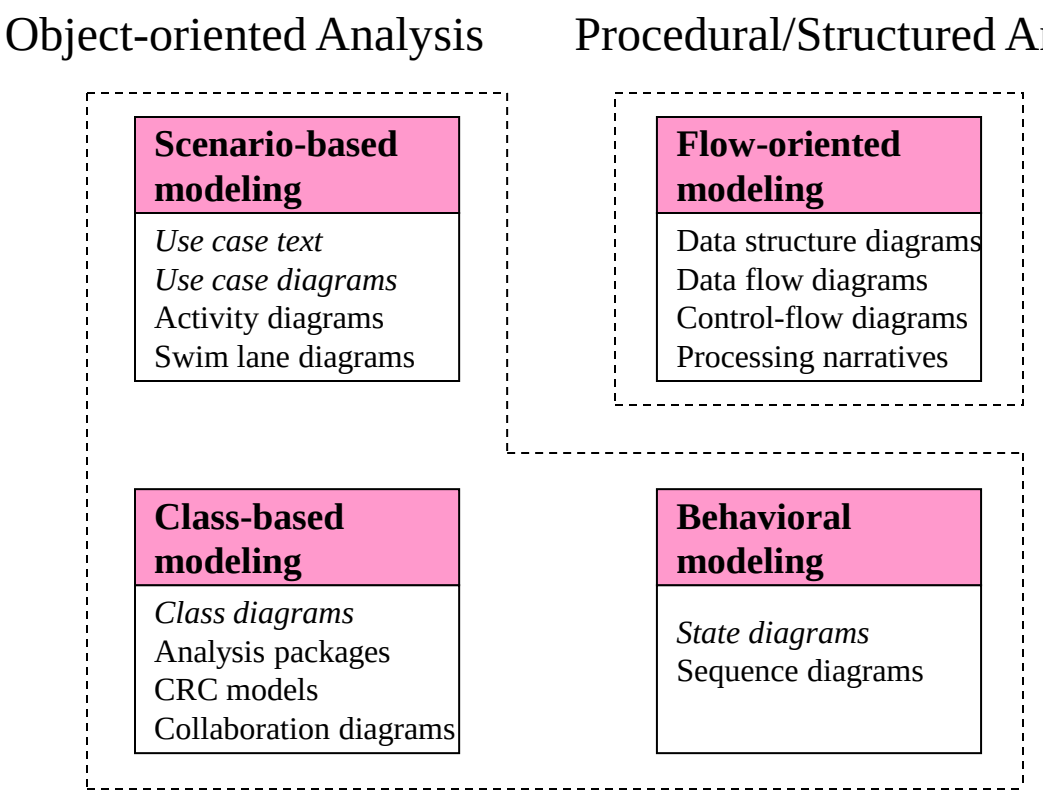


Problem # 2

- A separate *appliance control* determines when the motor should be on and continuously asserts on as an input to the motor control when the motor should be running.
- When on is asserted, the motor control should start and run the motor.
- The motor starts by applying power to both the start and the run windings. A sensor, called a starting relay, determines when the motor has started, at which point the start winding is turned off, leaving only the run winding powered. Both winding are shut off when on is not asserted.
- Appliance motors could be damaged by overheating if they are overloaded or fail to start. To protect against thermal damage, the motor control often includes an over-temperature sensor. If the motor becomes too hot, the motor control removes power from both windings and ignores any on assertion until a reset button is pressed and the motor has cooled off.
- **Add the following to the diagram**.
  - *Activities*: apply power to run winding, apply power to start winding.
  - *Events*: motor is overheated, on is asserted, on is no longer asserted, motor is running, reset.
  - *Condition*: motor is not overheated.


MotorControl
Off
Starting
Running
Too hot

Elements of the Analysis Model



Sequence Diagrams

Interaction Diagrams

- A series of diagrams describing the *dynamic behavior* of an object-oriented system.
  - A set of messages exchanged among a set of objects within a context to accomplish a purpose.
- Often used to model the way a use case is realized through a sequence of messages between objects.

Interaction Diagrams (Cont.)

- The purpose of Interaction diagrams is to:
  - Model interactions between objects
  - Assist in understanding how a system (a use case) actually works
  - Verify that a use case description can be supported by the existing classes
  - Identify responsibilities/operations and assign them to classes

Interaction Diagrams (Cont.)

- UML
  - Collaboration Diagrams
    - Emphasizes structural relations between objects
  - Sequence Diagram

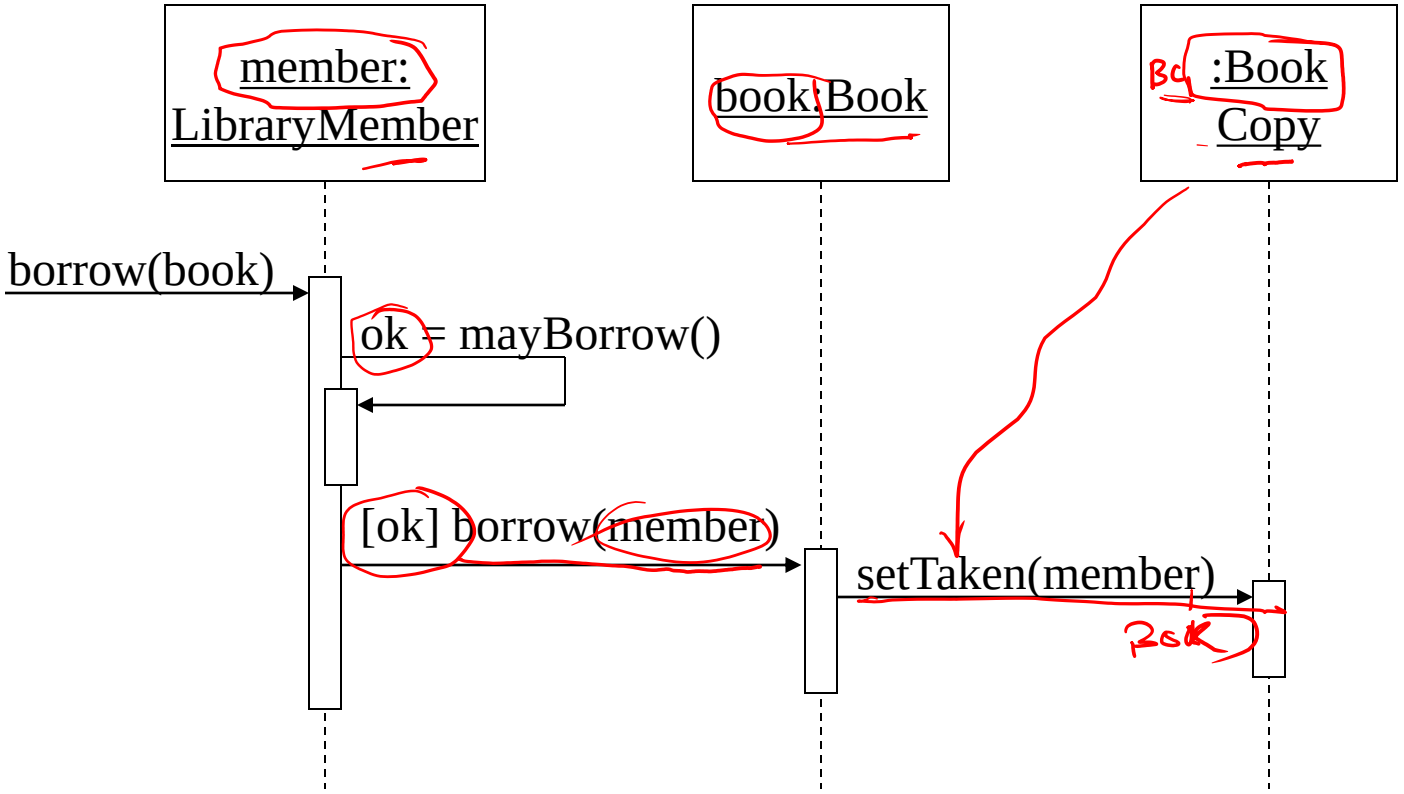
Importance of Sequence Diagrams

- Depict object interactions in a given scenario identified for a given Use Case
- Specify the messages passed between objects using horizontal arrows including messages to/from external actors
- Time increases from Top to bottom

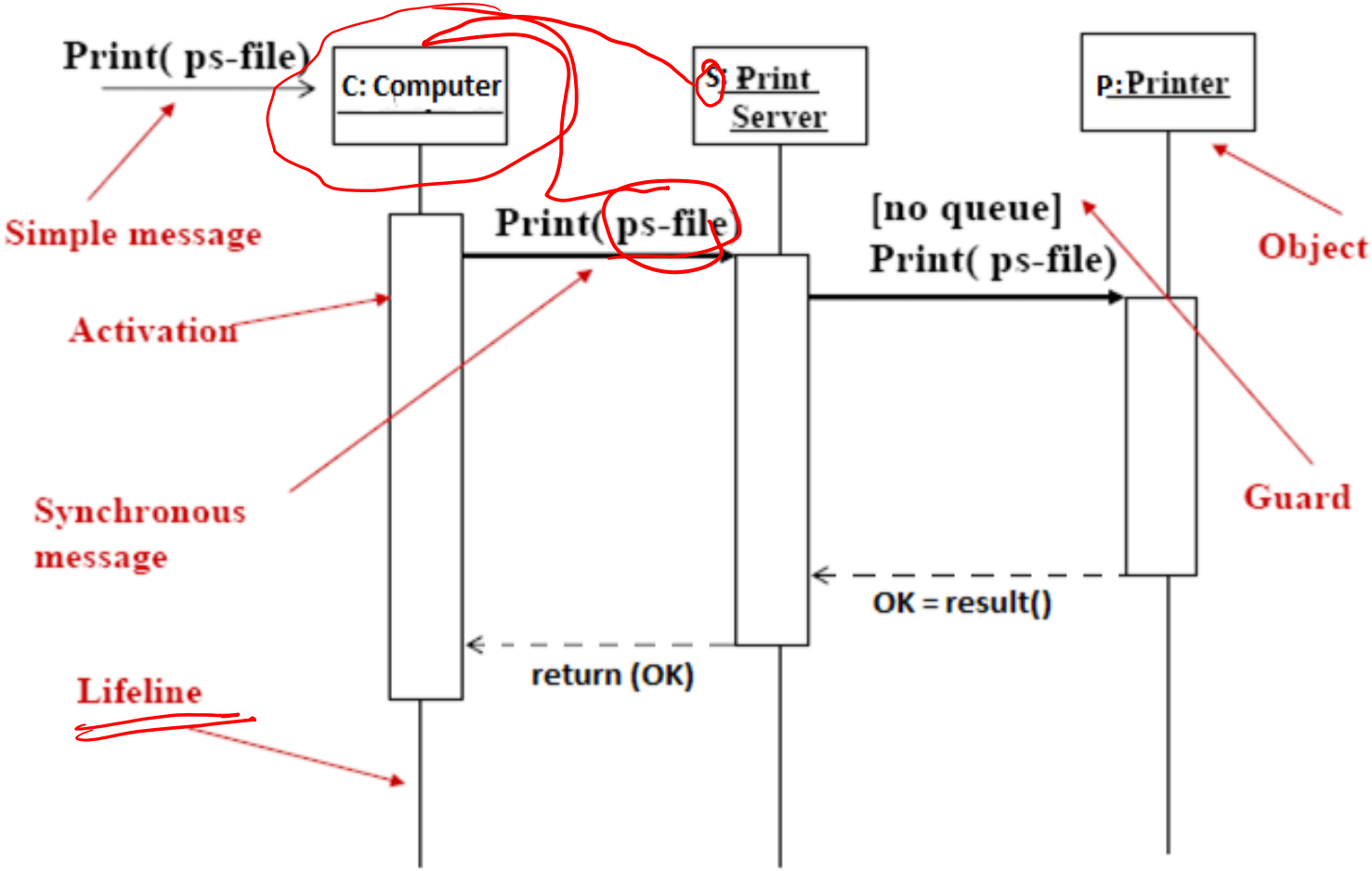
Sequence Diagrams

- Sequence diagrams illustrate how objects interact with each other.
- They focus on message sequences, that is, how messages are sent and received between a number of objects.
- The Branches, Conditions, and Loops may be included.
- Emphasizes time ordering of messages.
- Can model simple sequential flow, branching, iteration, recursion and concurrency.

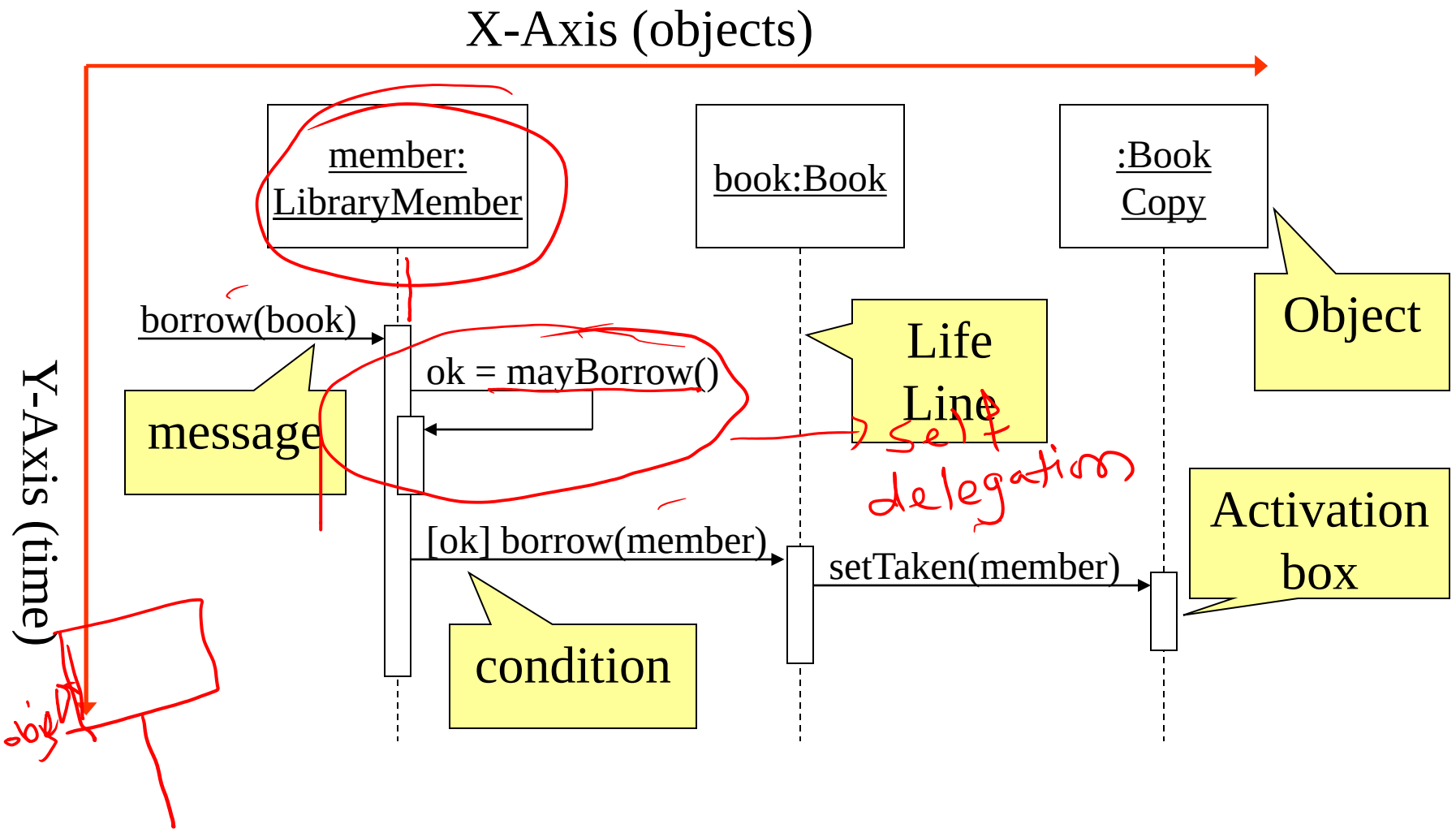
A Sequence Diagram



Sequence Diagram - Syntax

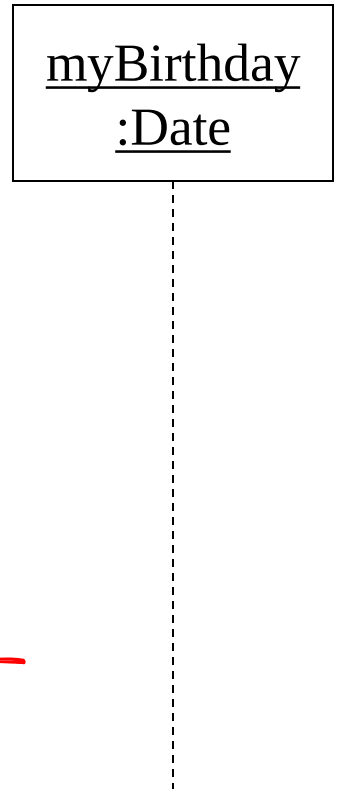


A Sequence Diagram



Object

- Object naming:
  - syntax: *[instanceName][:className]*
  - Name classes consistently with your class diagram (same classes).
  - Include instance names when objects are referred to in messages or when several objects of the same type exist in the diagram.
- The *Life-Line* represents the object's life during the interaction



Messages

- An interaction between two objects is performed as a message sent from one object to another (simple operation call, Signaling, RPC)
- If object obj_1 sends a message to another object obj_2 some link must exist between those two objects (dependency, same objects)

Message types

- Synchronous messages
- Asynchronous messages

Messages (Cont.)

- A message is represented by an arrow between the lifelines of two objects.
  - Self calls are also allowed
  - The time required by the receiver object to process the message is denoted by an *activation-box.*
- A message is labeled at minimum with the message name.
  - Arguments and control information (conditions, iteration) may be included.

Return Values

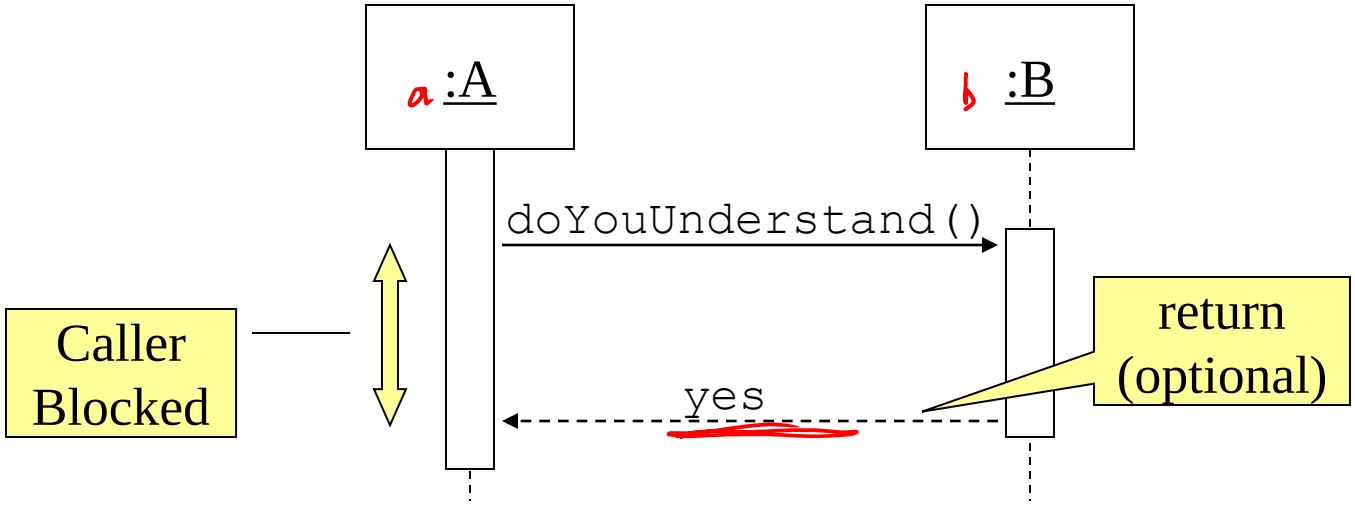
- Optionally indicated using a dashed arrow with a label indicating the return value.
  - Don't model a return value when it is obvious what is being returned, e.g. getTotal()
  - Model a return value only when you need to refer to it elsewhere, e.g. as a parameter passed in another message.
  - Prefer modeling return values as part of a method invocation, e.g. `ok = isValid()`

Asynchronous messages

- Asynchronous calls, which are associated with operations, typically have only a send message, but can also have a reply message.
- In contrast to a synchronous message, the source lifeline is not blocked from receiving or sending other messages.
- You can also move the send and receive points individually to delay the time between the send and receive events.
- You might choose to do this if a response is not time sensitive or order sensitive.

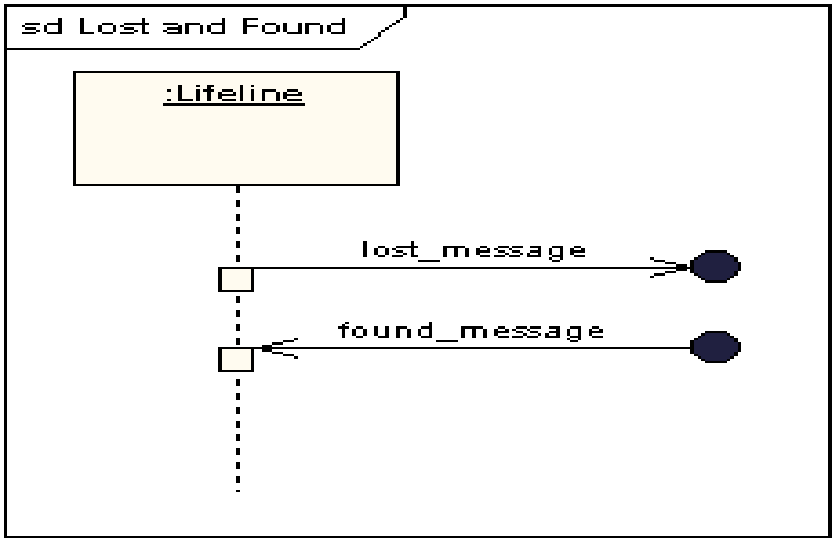
Synchronous Messages

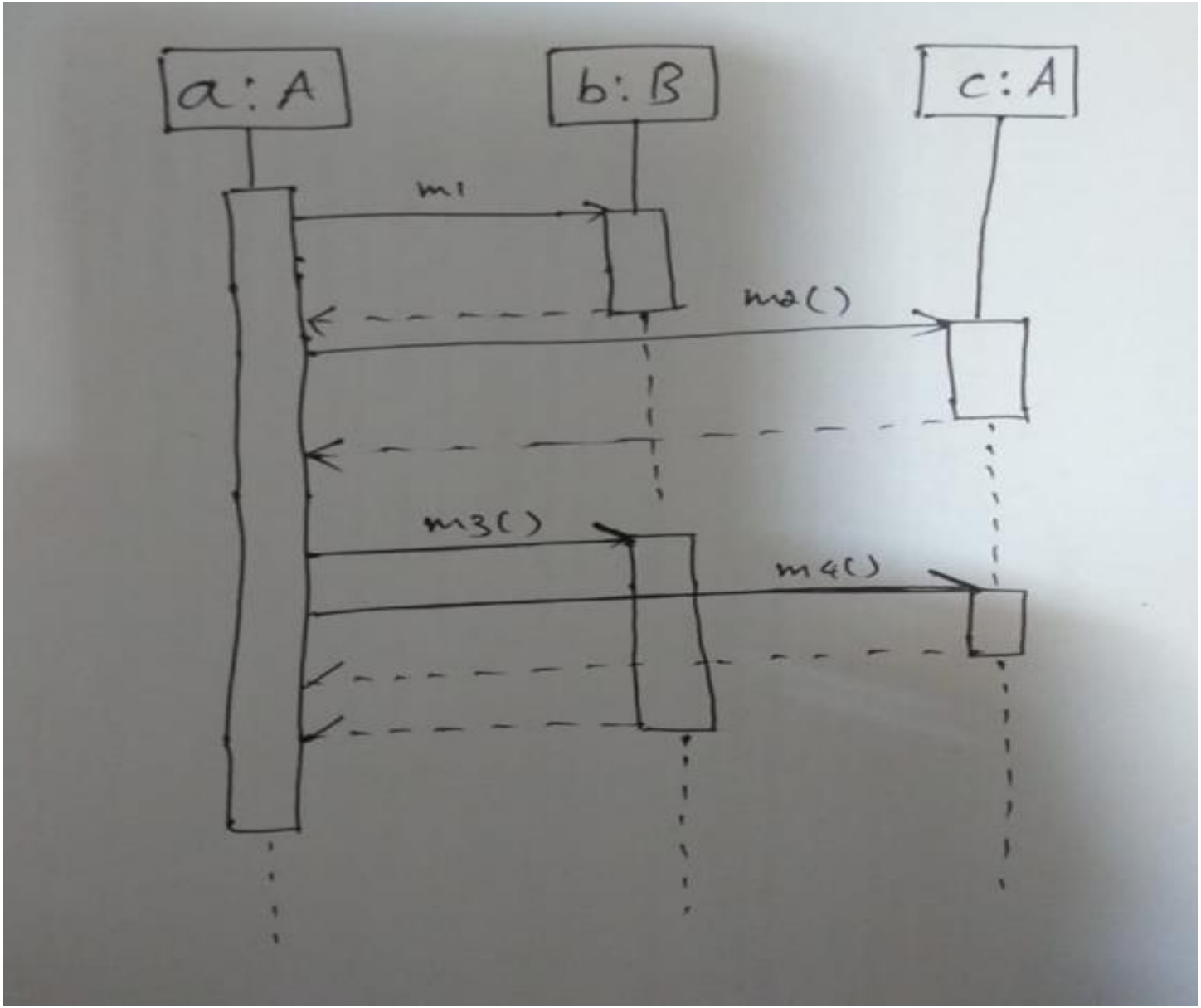
- Nested flow of control, typically implemented as an operation call.
  - The routine that handles the message is completed before the caller resumes execution.



Lost and Found Messages

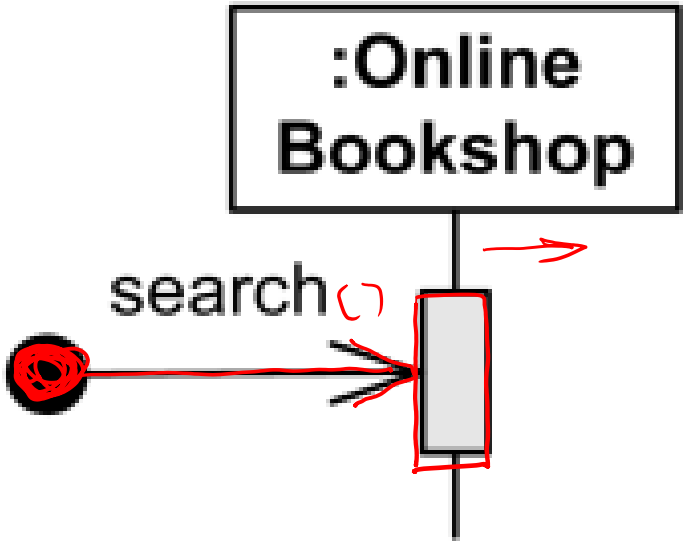
- Lost messages are those that are either sent but do not arrive at the intended recipient, or which go to a recipient not shown on the current diagram.
- Found messages are those that arrive from an unknown sender, or from a sender not shown on the current diagram. They are denoted going to or coming from an endpoint element.

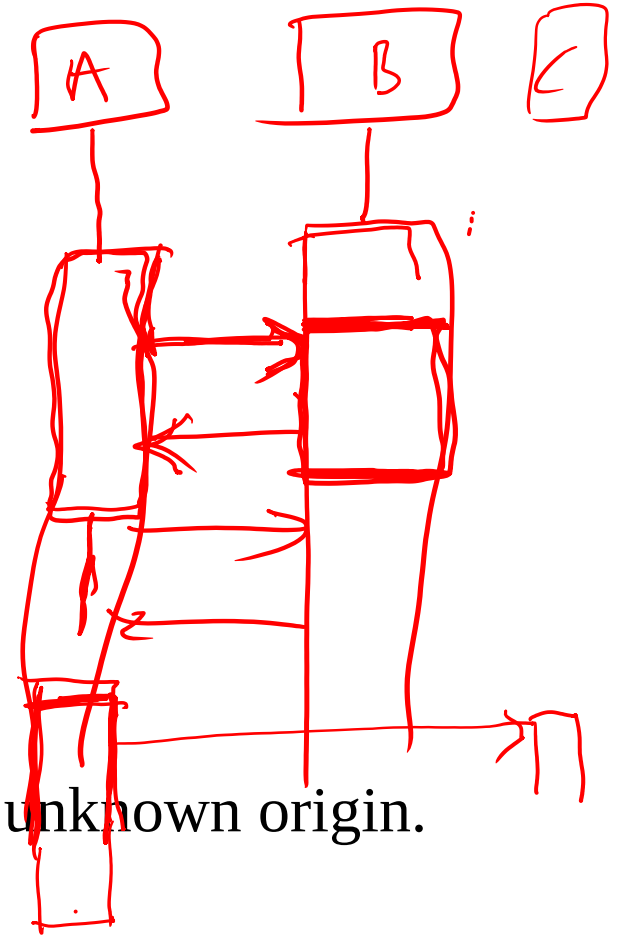



a:A
b:B
c:A
m1
m2()
m3()
m4()

Found & Lost Message

- **Found Message** is a message where the receiving event is known, but there is no (known) sending event

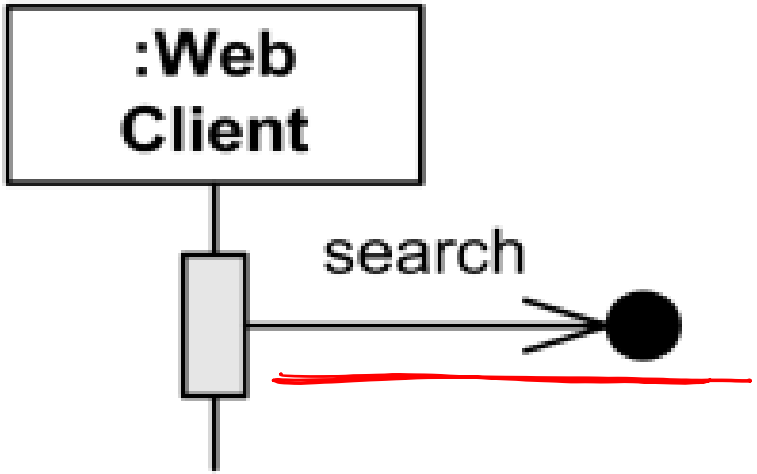




Online Bookshop gets search message of unknown origin.

Found & Lost Message

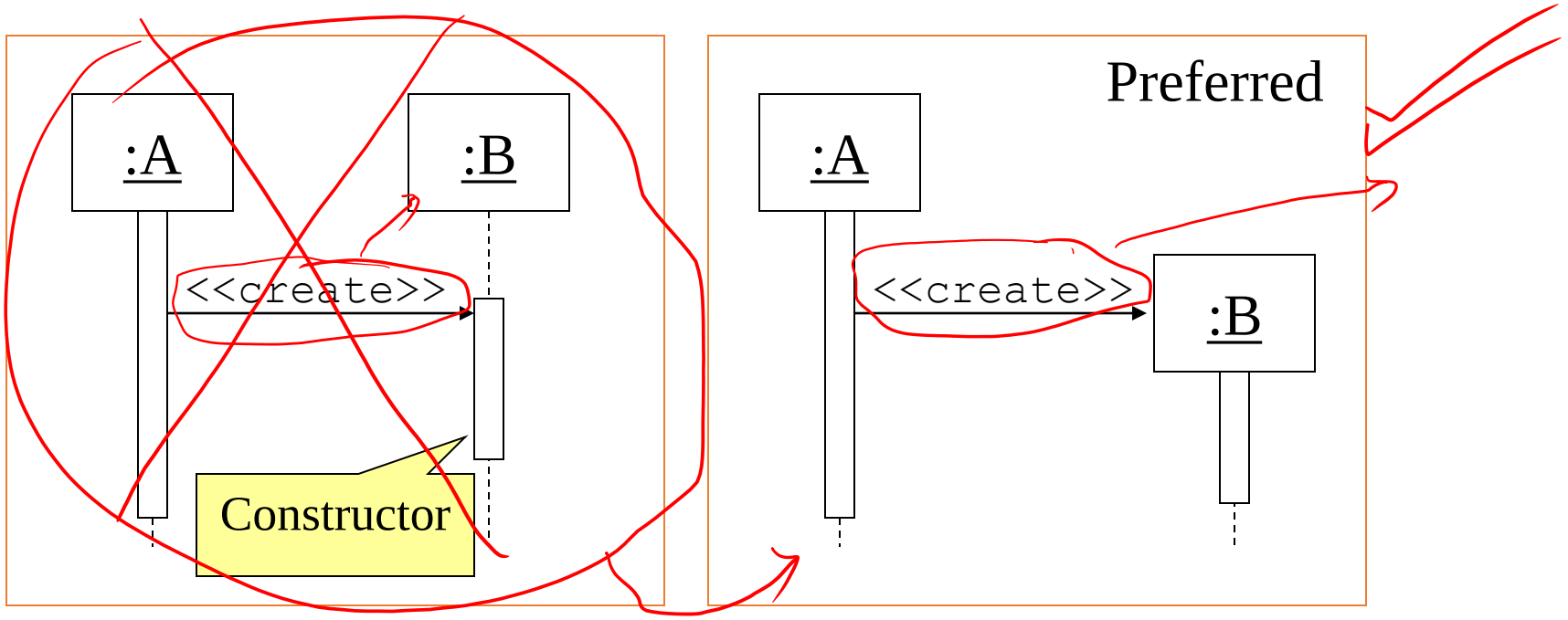
- **Lost Message** is a message where the sending event is known, but there is no receiving event



Web Client sent search message which was lost.

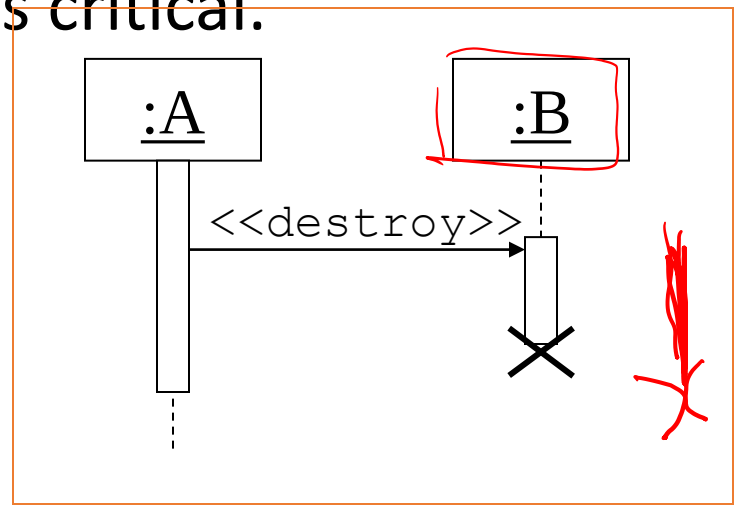
Object Creation

- An object may create another object via a `<<create>>` message.

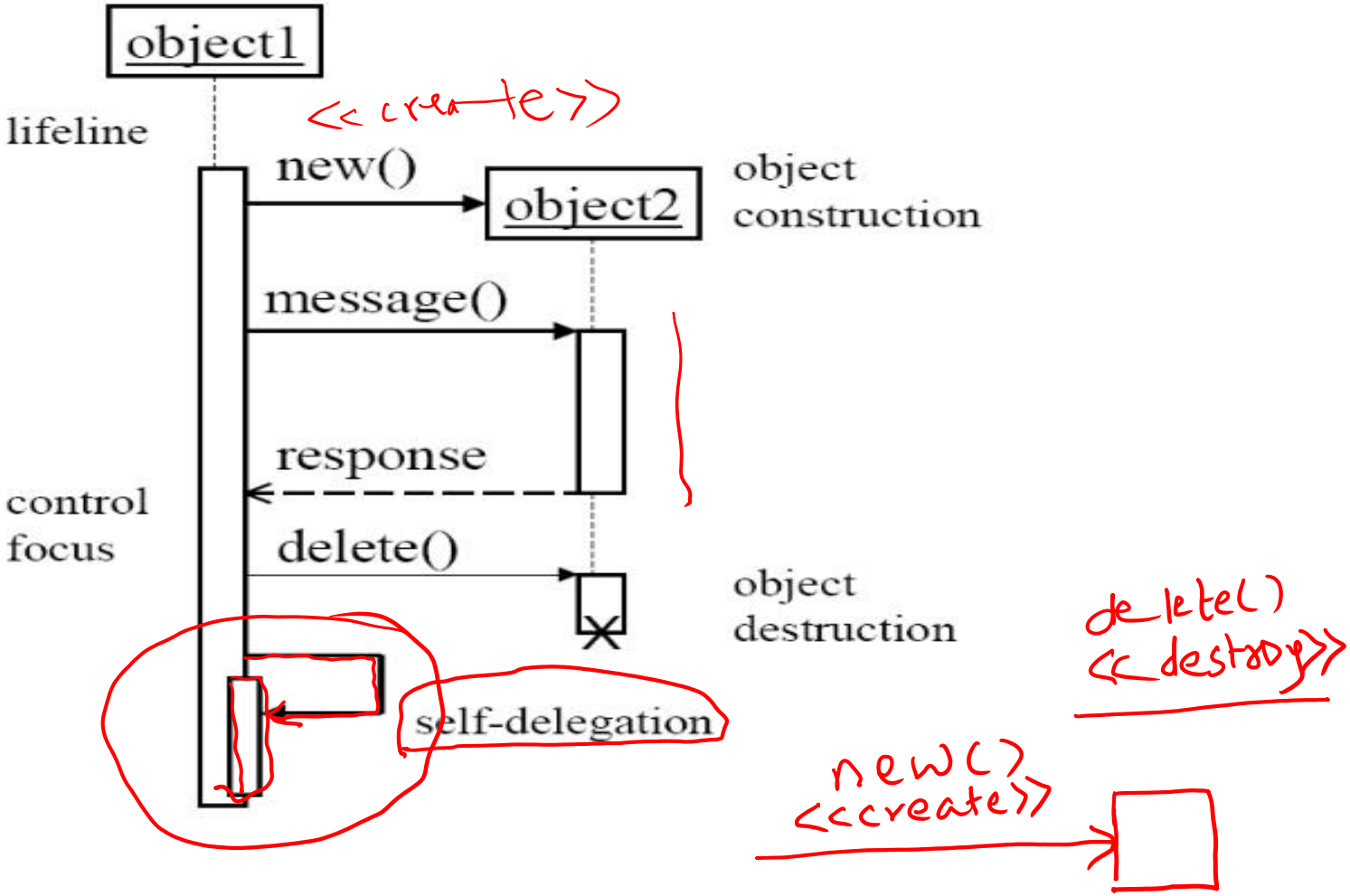


Object Destruction

- An object may destroy another object via a `<<destroy>>` message.
  - An object may destroy itself.
  - Avoid modeling object destruction unless memory management is critical.



Sequence Diagram - 2

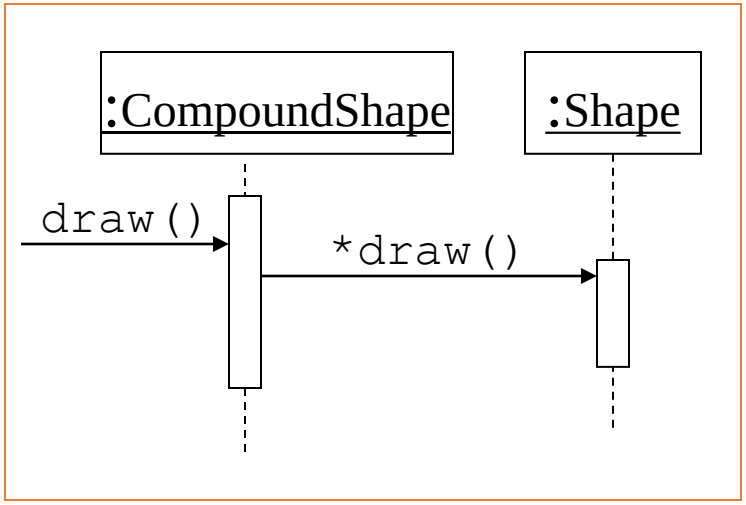


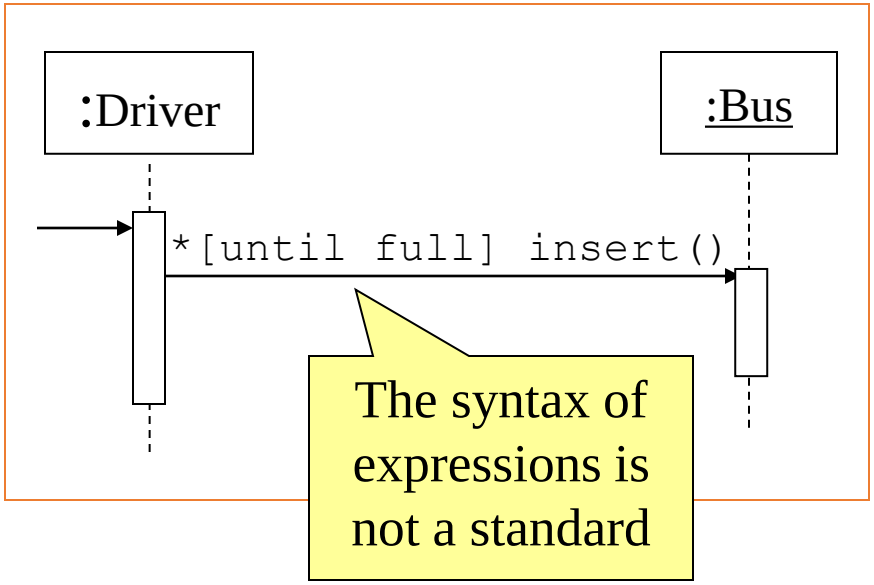
Control information

- Condition
  - syntax: ‘[‘ expression ’]’ message-label
  - The message is sent only if the condition is true
  - example:
- Iteration
  - syntax: * [‘[‘ expression ‘]’] message-label
  - The message is sent many times to possibly multiple receiver objects.

Control Information (Cont.)

- **Iteration examples:**

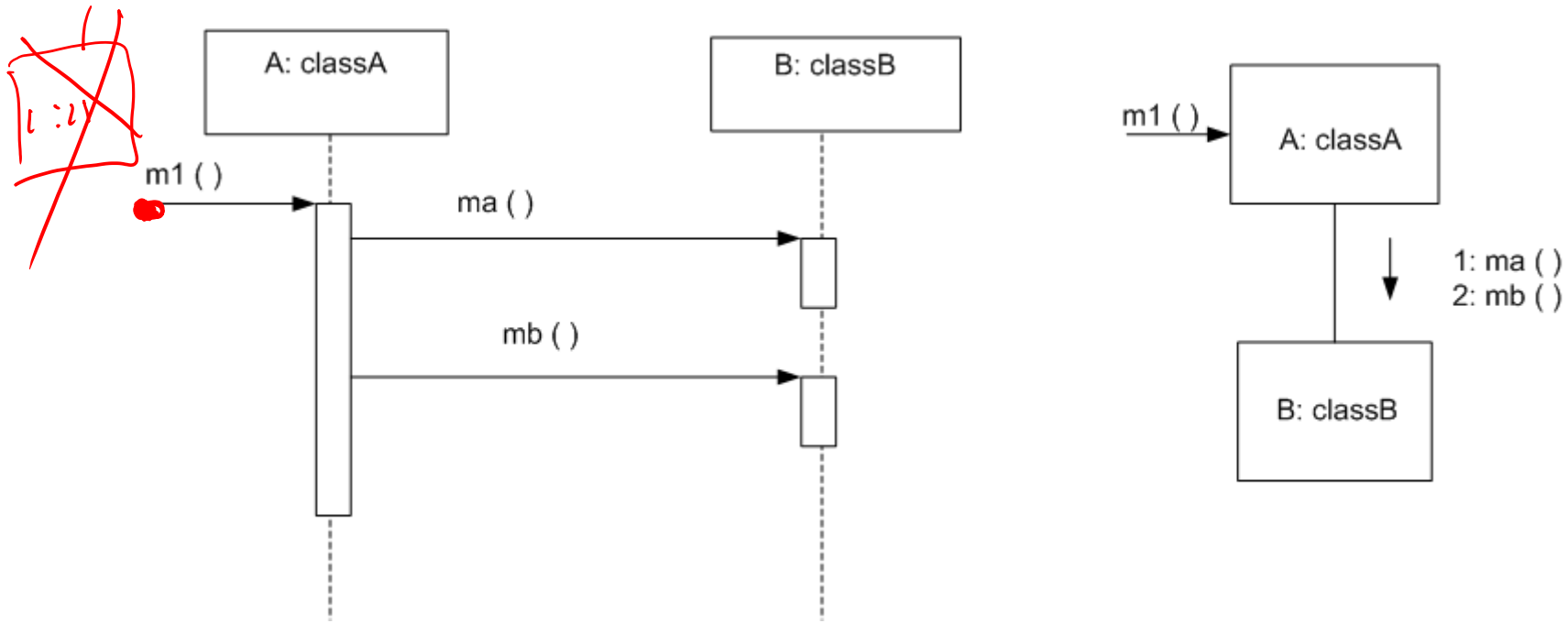




Control Information (Cont.)

- The control mechanisms of sequence diagrams suffice only for modeling simple alternatives.
  - Consider drawing several diagrams for modeling complex scenarios.
  - Don’t use sequence diagrams for detailed modeling of algorithms (this is better done using *activity diagrams, pseudo-code* or *state-charts*).

Sequence Diagram basic example

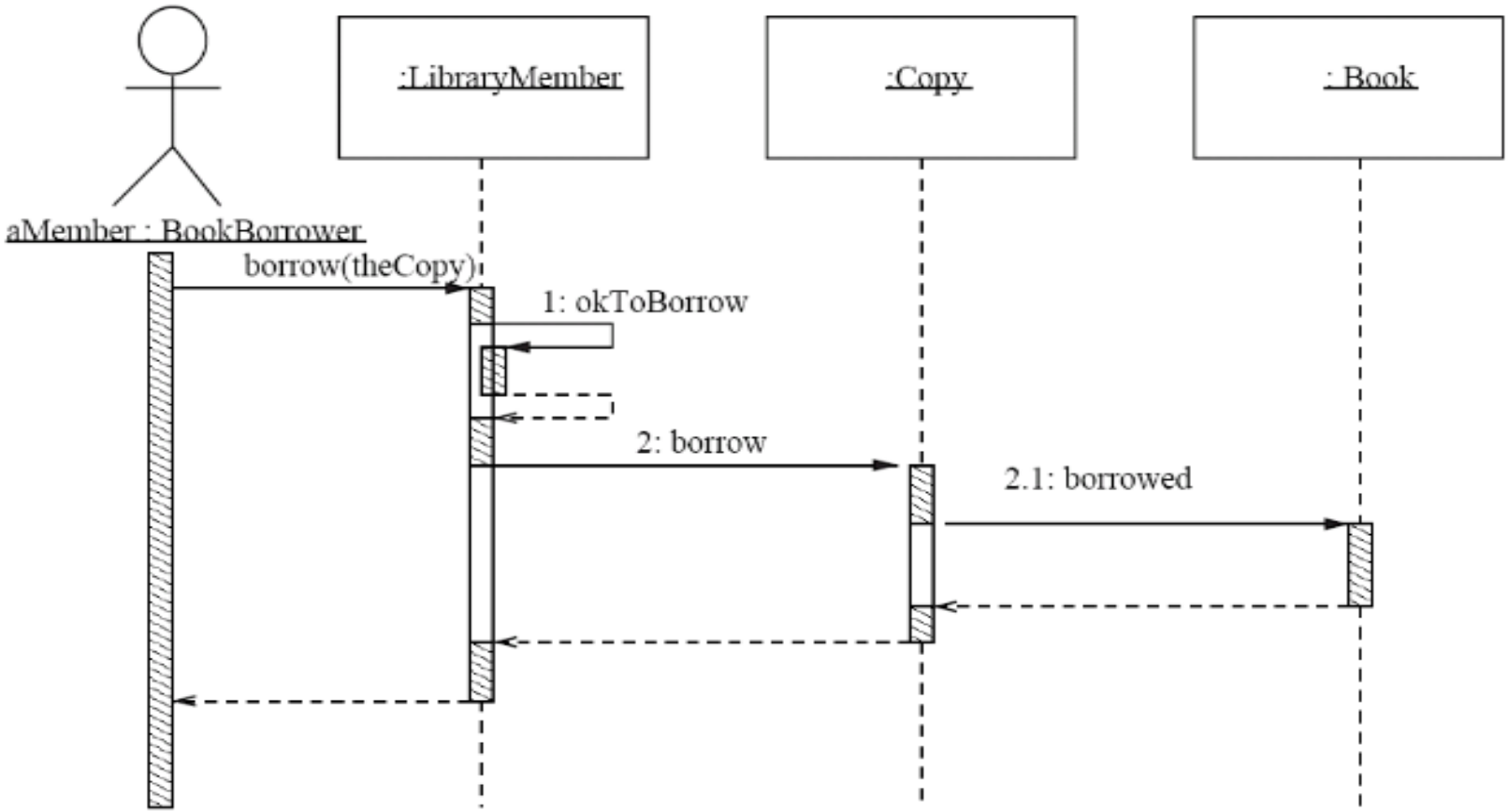



```
public class classA
private classB B = new ClassB();
public void m1(){
B. ma();
B.mb();
}
```

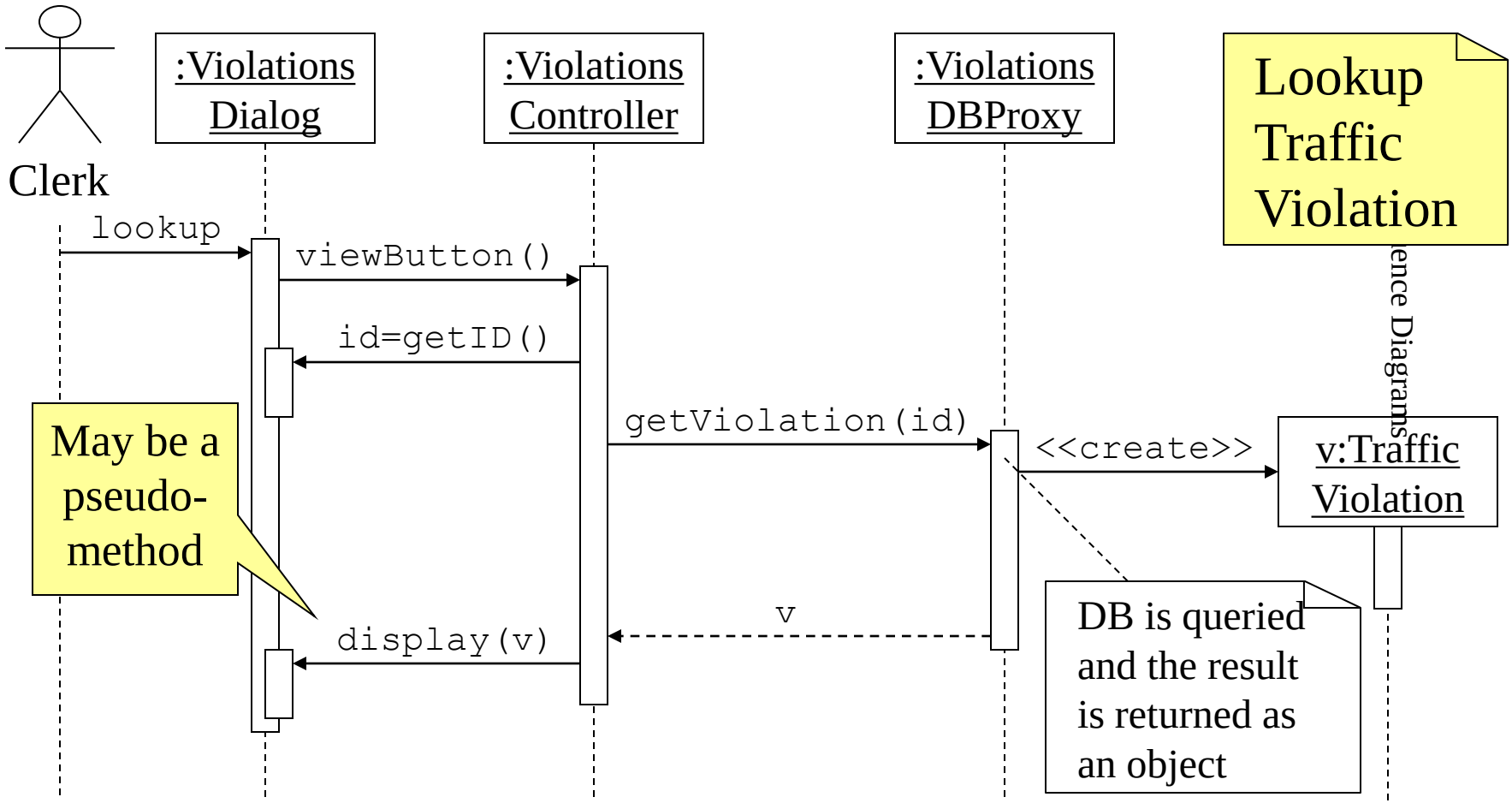
status = m1 ( )

[OK]/OK

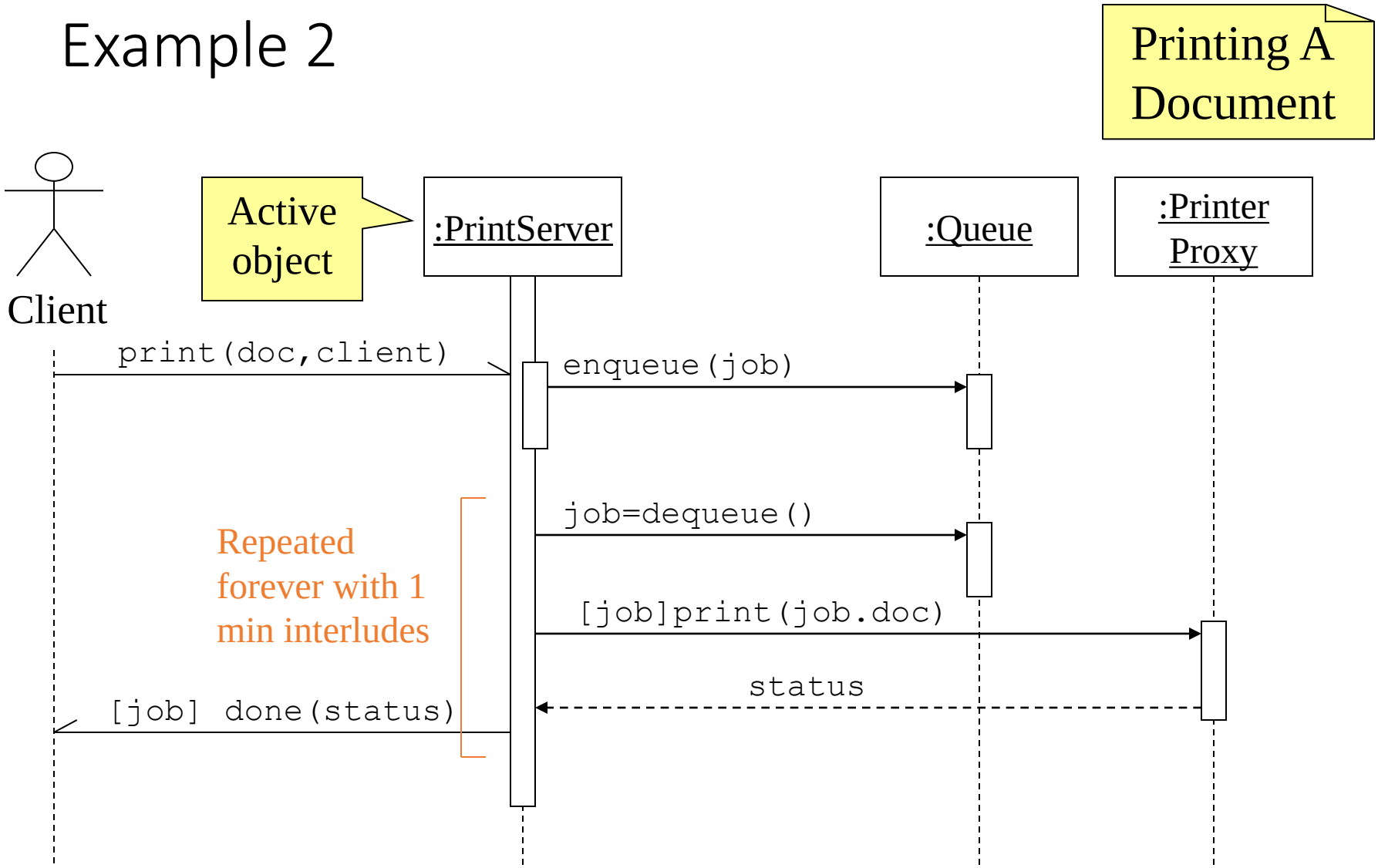
Nested activation

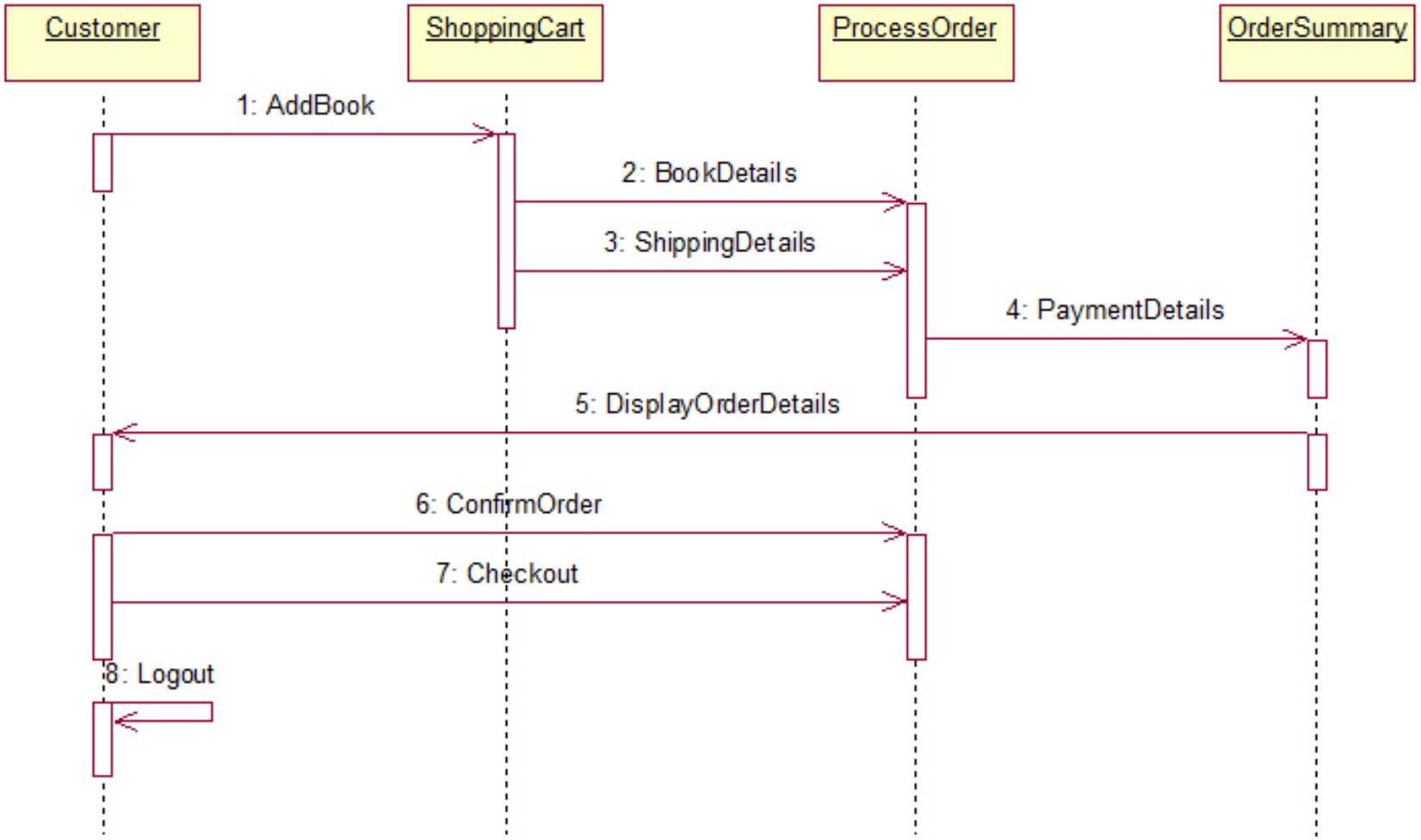


Example 1

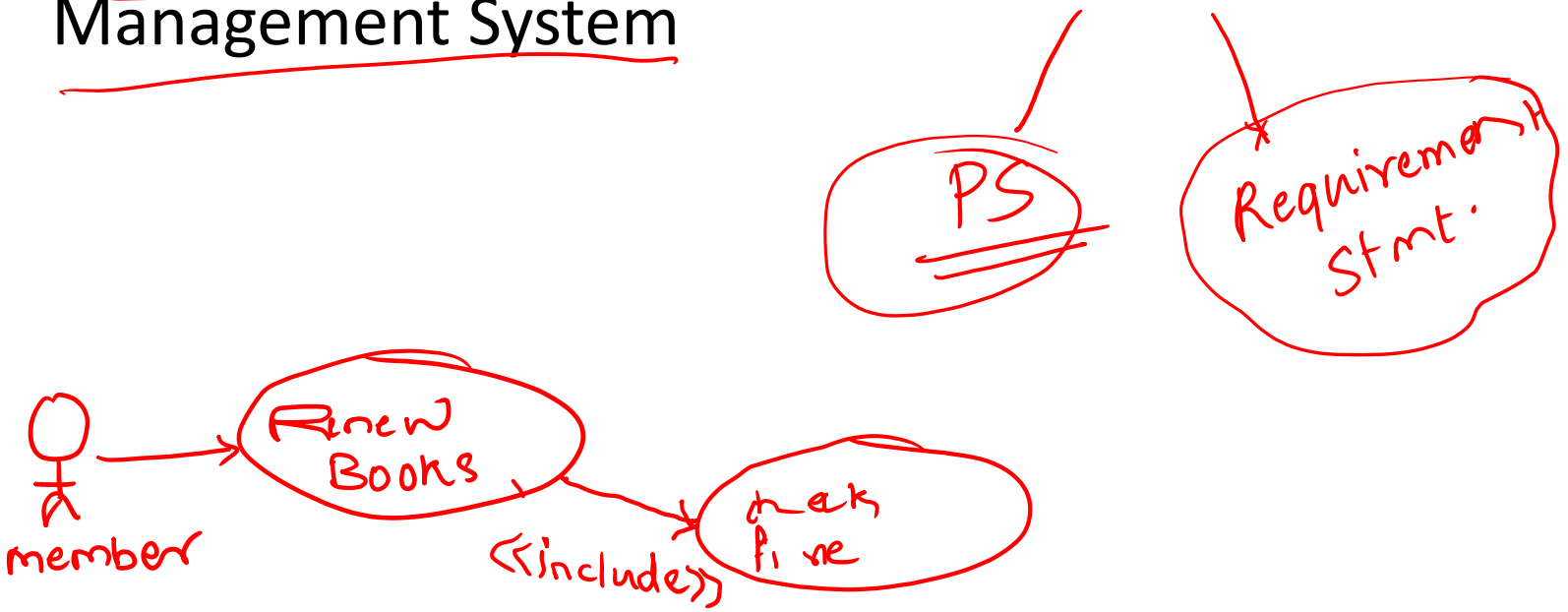

Clerk
:Violations Dialog
:Violations Controller
:Violations DBProxy
Lookup Traffic Violation
lookup
viewButton()
id=getID()
getViolation(id)
<<create>>
v:Traffic Violation
May be a pseudo-method
v
display(v)
DB is queried and the result is returned as an object

Example 2

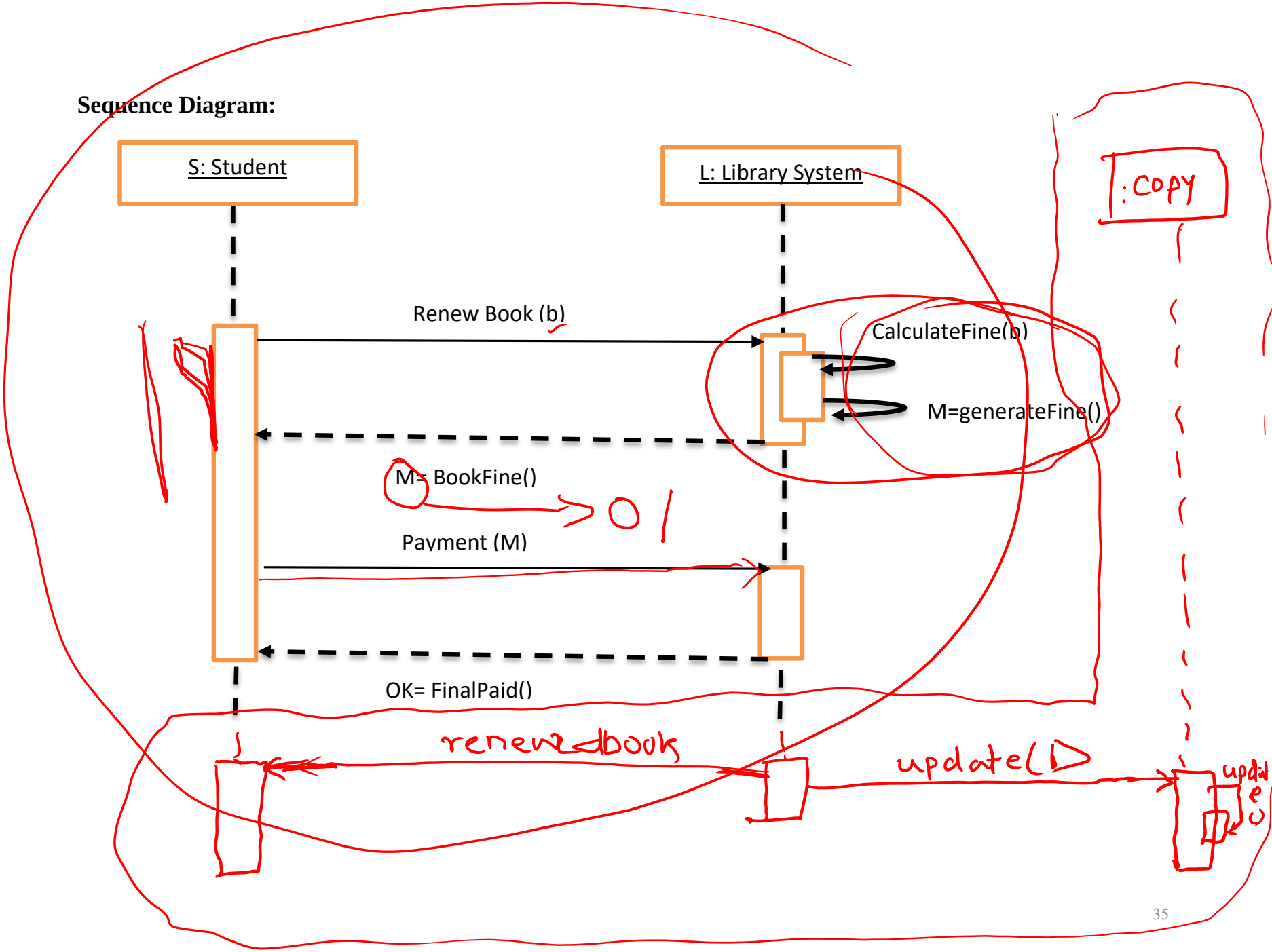

Printing A Document
Client
Active object
:PrintServer
:Queue
:Printer Proxy
print(doc,client)
enqueue(job)
job=dequeue()
Repeated forever with 1 min interludes
[job]print(job.doc)
status
[job] done(status)


Customer
ShoppingCart
ProcessOrder
OrderSummary
1: AddBook
2: BookDetails
3: ShippingDetails
4: PaymentDetails
5: DisplayOrderDetails
6: ConfirmOrder
7: Checkout
8: Logout

Draw sequence diagram for "Renew books with fine generation for late renewal" scenario in Library Management System



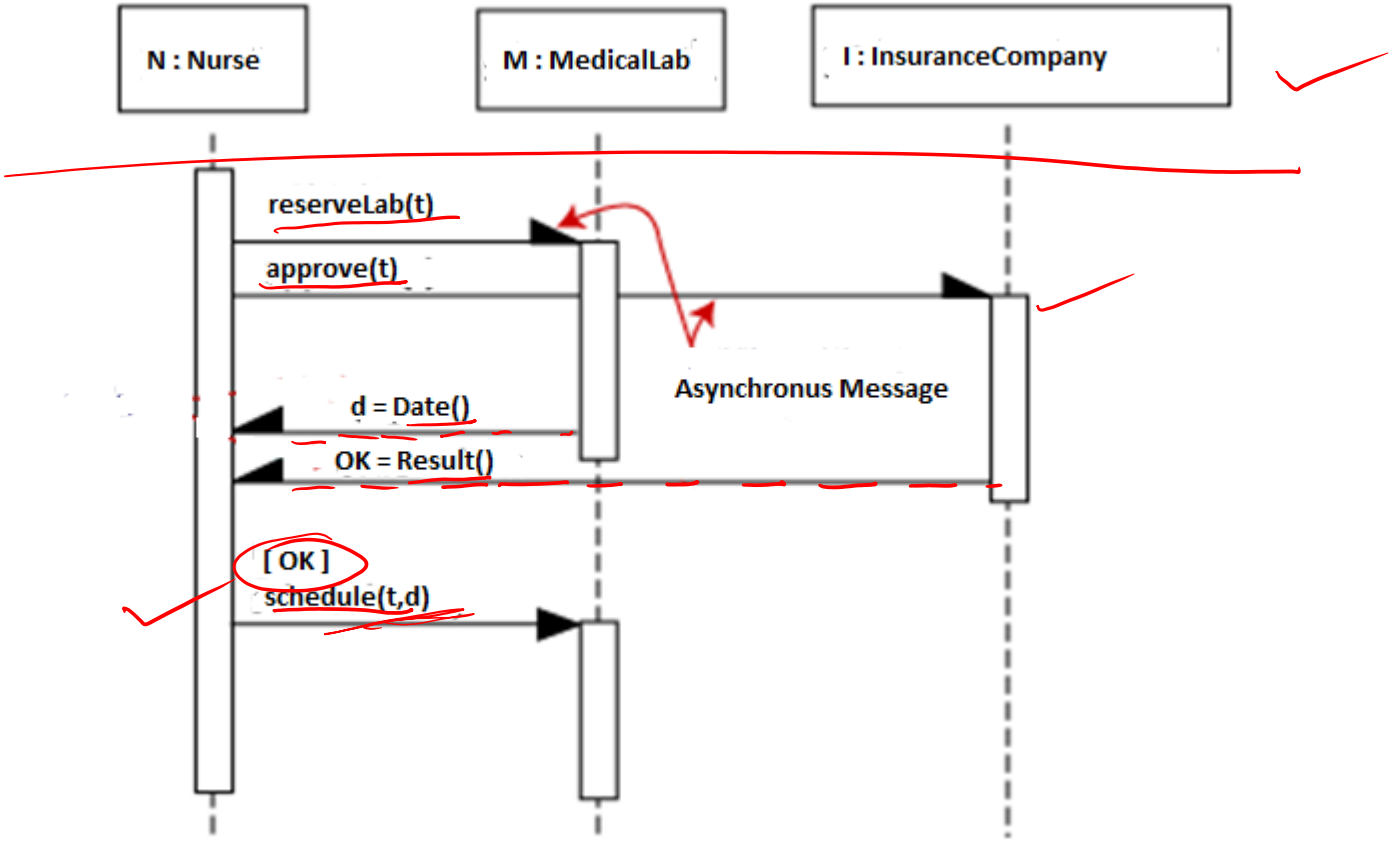
Sequence Diagram:



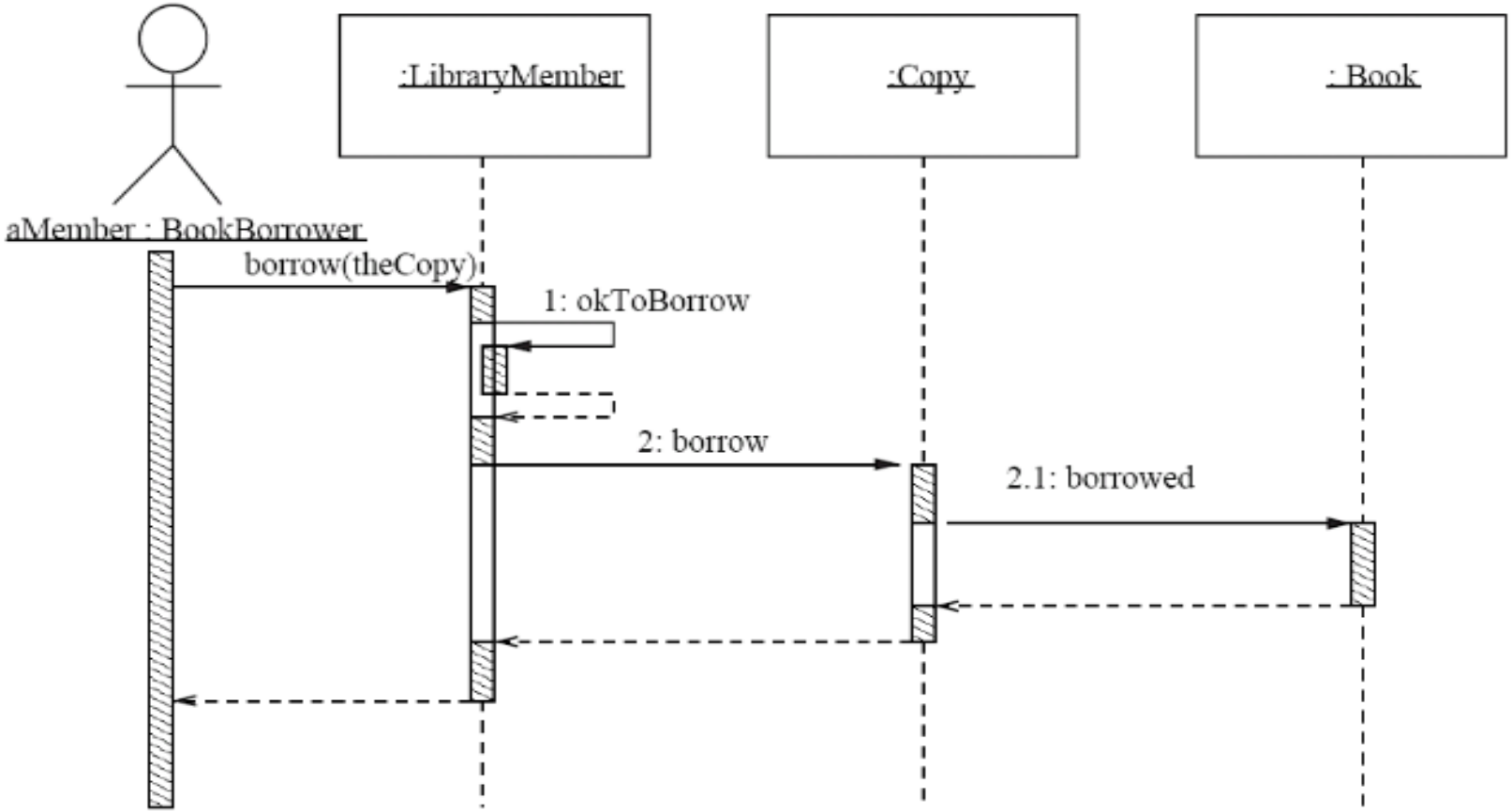
Example

In a hospital, nurse simultaneously requests the medical lab to reserve a date for the patient's diagnostic test (t) and the insurance company to approve the test. If the Insurance Company approves the test, then the Nurse will schedule the test on the date supplied by the Medical lab.

Asynchronous messages



Nested activation

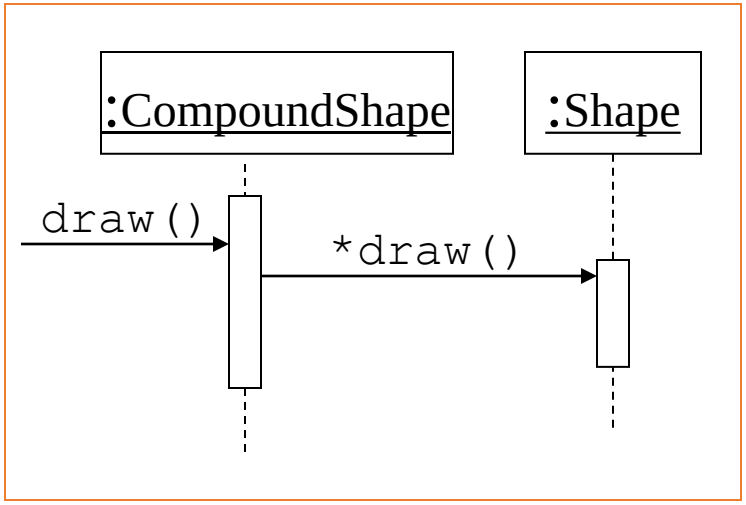


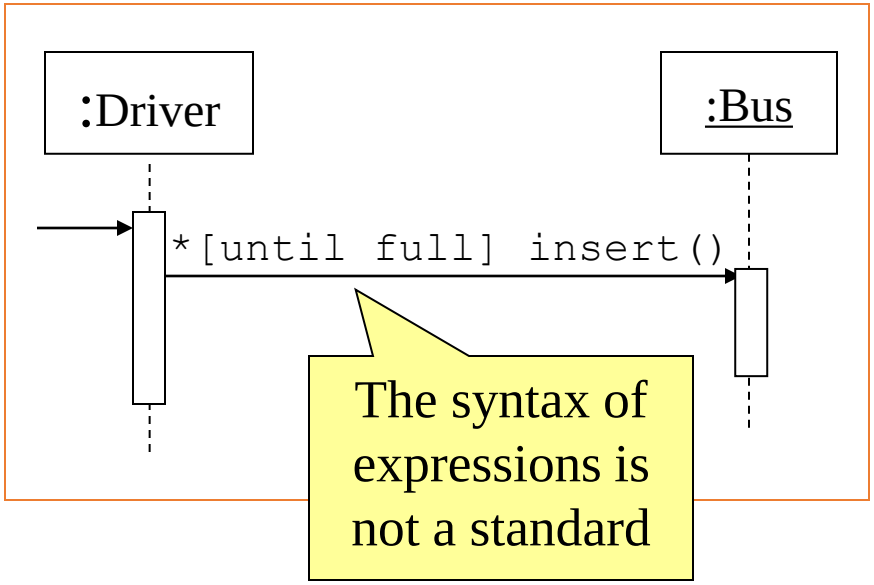
Control information

- Condition
  - syntax: ‘[‘ expression ’]’ message-label
  - The message is sent only if the condition is true
  - example:
- Iteration
  - syntax: * [‘[‘ expression ‘]’] message-label
  - The message is sent many times to possibly multiple receiver objects.

Control Information (Cont.)

- **Iteration examples:**





Control Information (Cont.)

- The control mechanisms of sequence diagrams suffice only for modeling simple alternatives.
  - Consider drawing several diagrams for modeling complex scenarios.
  - Don’t use sequence diagrams for detailed modeling of algorithms (this is better done using *activity diagrams, pseudo-code* or *state-charts*).

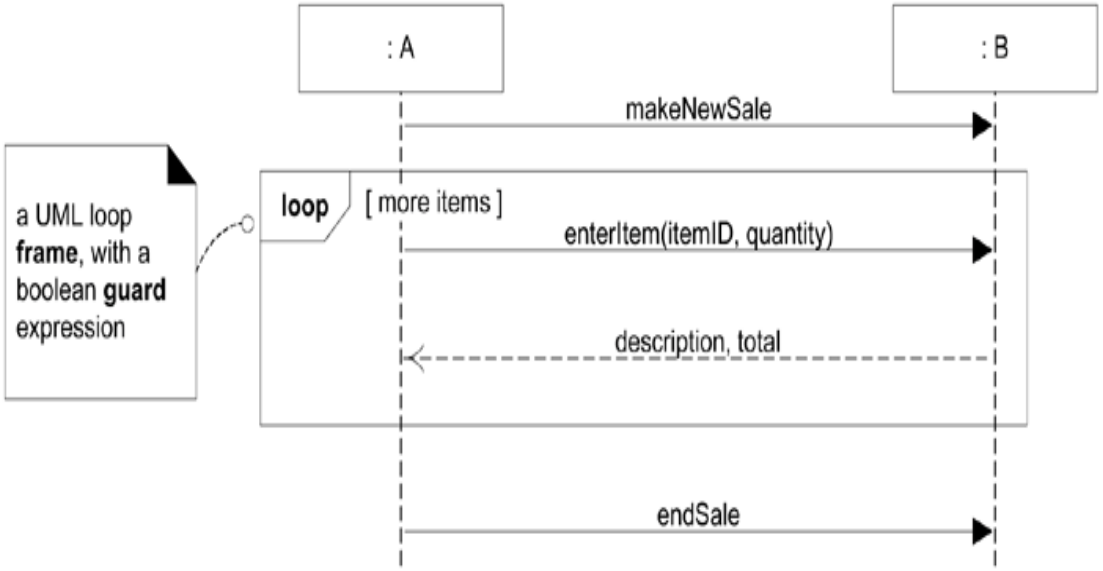
Interaction Fragment in Sequence Diagram

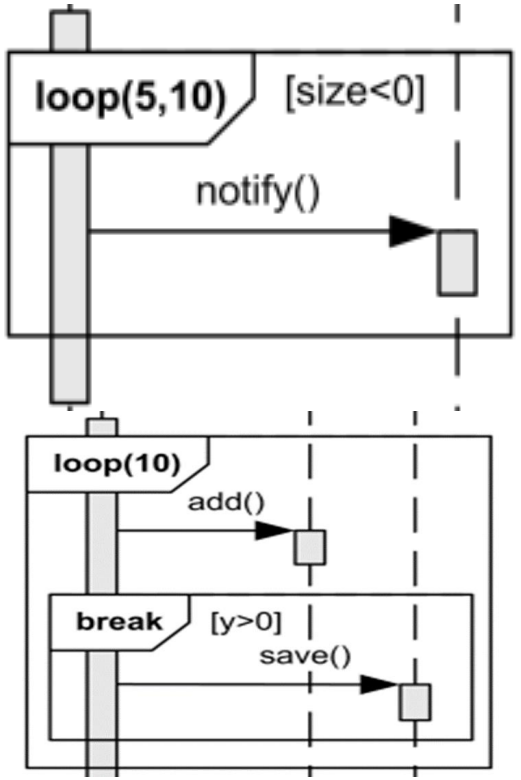
- An Interaction fragment which defines an expression of interaction fragments.
- An interaction fragment is defined by an interaction operator and corresponding interaction operands. Through the use of interaction fragments the user will be able to describe a number of traces in a compact and concise manner.
- interaction fragment may have constraints also called guards in UML

- Typical frame operators are:

Frame Operator	Semantics
alt	Alternative fragment for mutual exclusion conditional logic expressed in the guards.
loop	Loop fragment while guard is true. Can also write loop(n) to indicate looping n times.
opt	Optional fragment that executes if guard is true.
par	Parallel fragments that execute in parallel.
region	Critical region within which only one thread can run.

- Frames in UML Sequence Diagrams :
  - To allow the visualization of complex algorithms, sequence diagrams support the notion of frames;
  - Frames are regions of the diagrams that have an operator and a guard,. Figure.





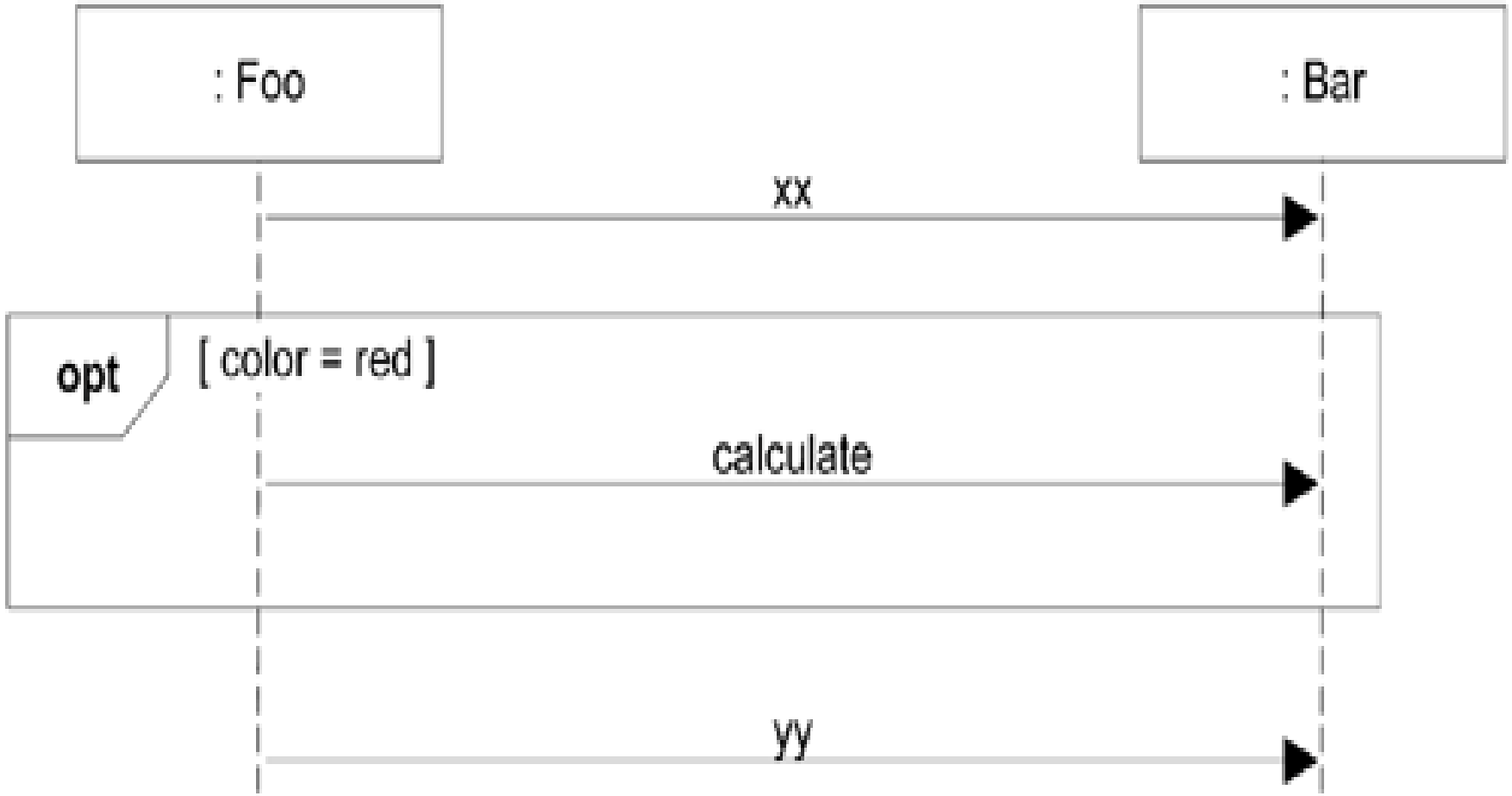


Figure Using the opt Frame

- Next Figure illustrate the use of the alt frame for mutually exclusive alternatives

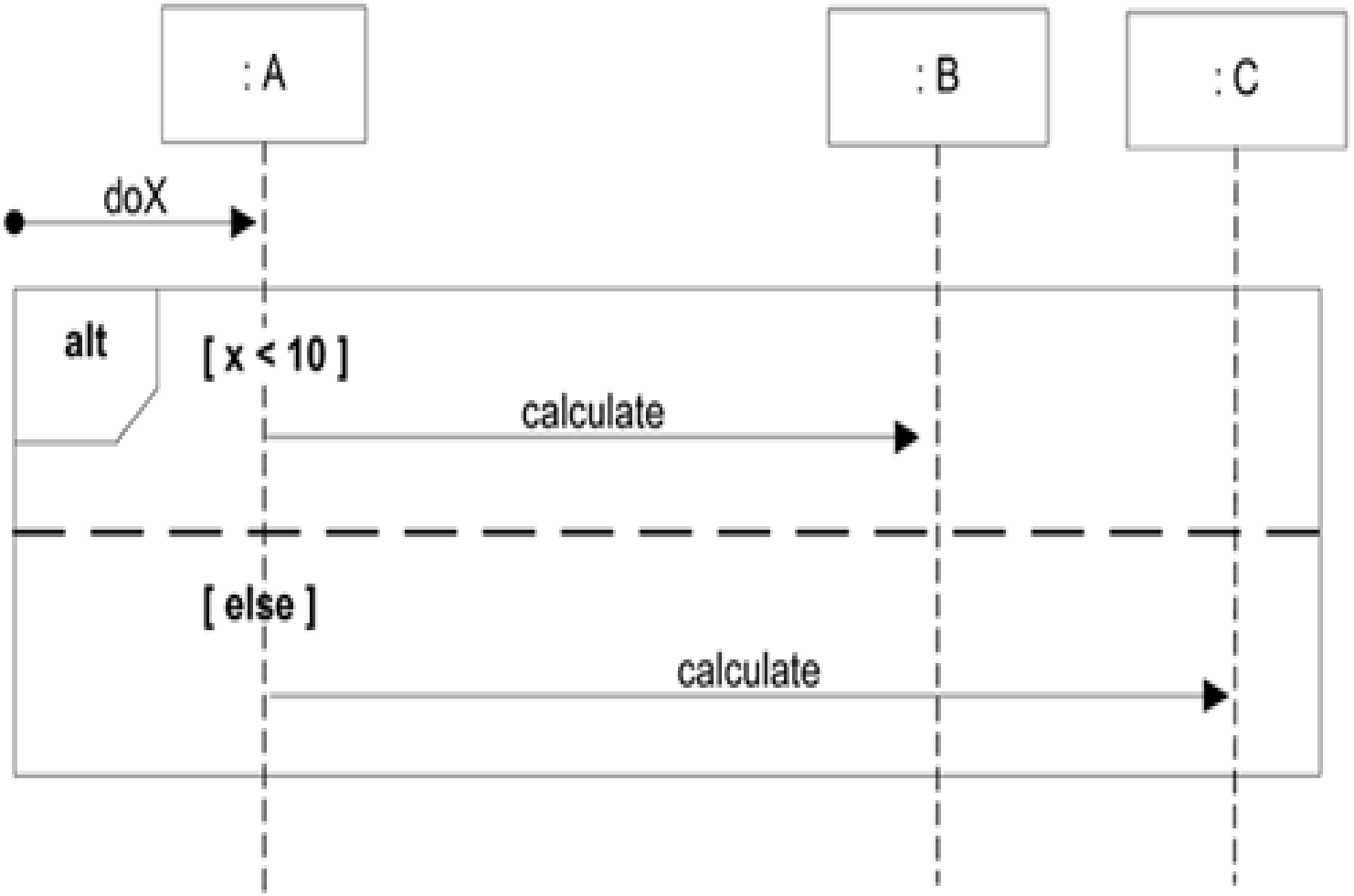
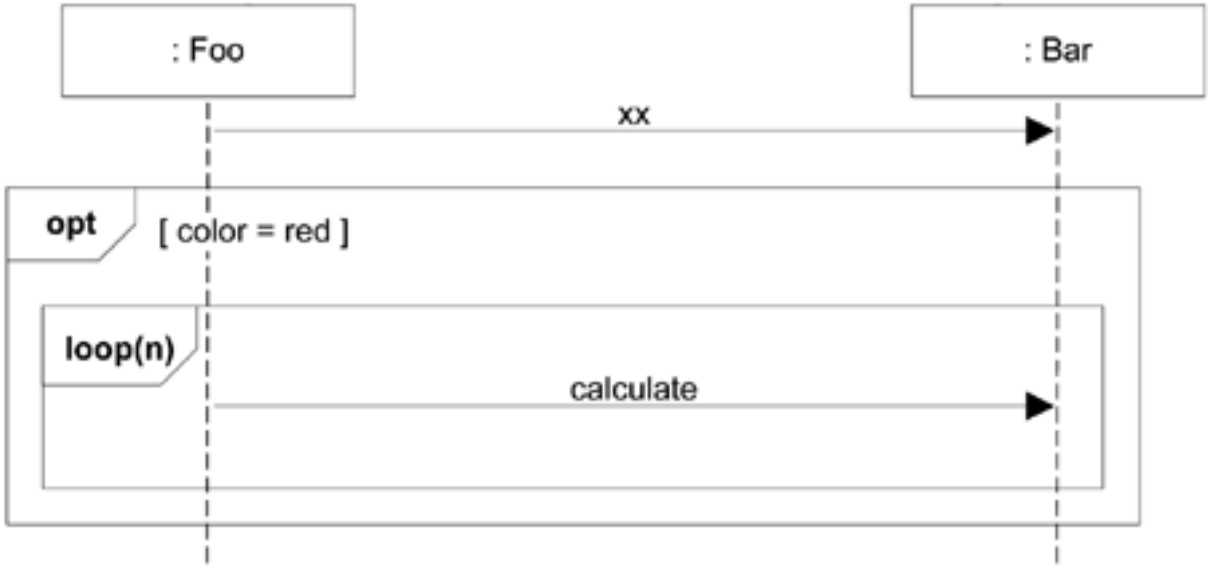
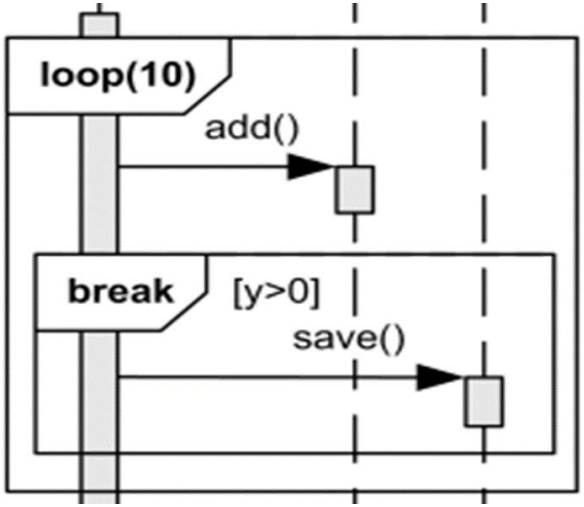


Figure Using the alt Frame

- Nesting of Frames :
  - Frames can be nested:





- MyManipal is an online social networking service only for MIT students and staff. The users must register before using the site. Once the user is authenticated he can retrieve notification, updates and messages. Further, in the next step the user can change the user preferences as online/ hidden. Finally, the status will be displayed in users' wall. Unsuccessful login will lock the account for 1 minute to prevent from brute force attack.

- The user wants to boot a server with Linux operating system (OS). So whenever the user sends a signal *start* the operating system will load RAM and the booting process will be initiated. After a time interval, the OS will get its current date and time. Moreover, the user will be acknowledged with current date *d* and time *t* by the OS within a stipulated time period. Based on the user request the OS will spawn two processes simultaneously namely P_i and P_j. The order of creation of above two processes is random. Each process will return its *pid* to OS. Due to the scarcity of resources the OS is required to kill any one of the processes. The built-in functionality in OS called *Preemptive Algorithm* chooses P_i as victim process. OS sends a signal *kill* to kill the process P_i. If the OS is unsuccessful in killing the process P_i, then the OS will send *kill* message to kill the process P_j.