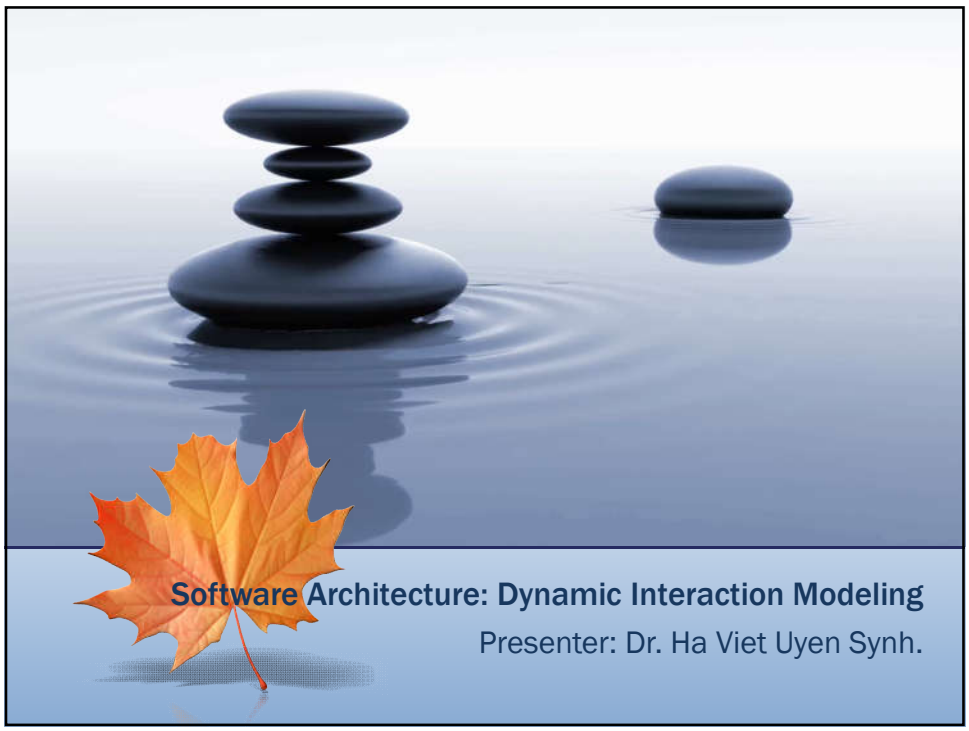
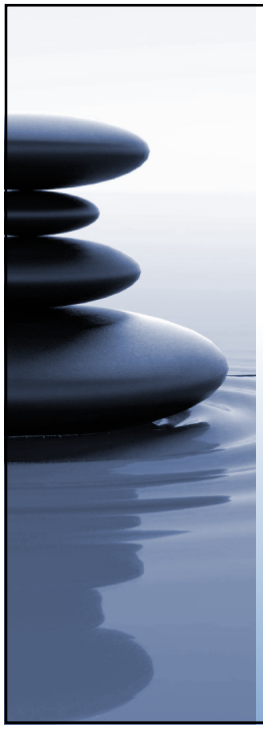


# Software Architecture: Dynamic Interaction Modeling

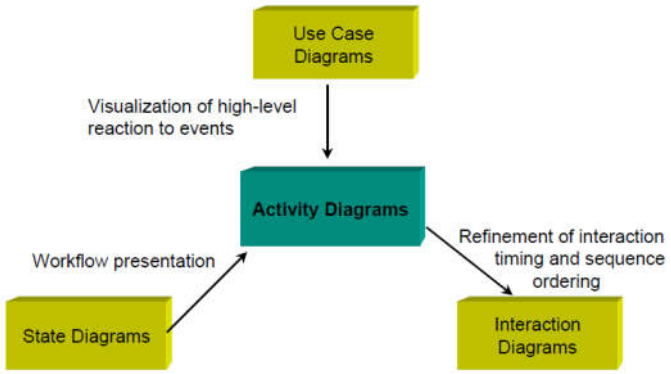
Presenter: Dr. Ha Viet Uyen Synh.



PART #1

# ACTIVITY DIAGRAM

## Role of Activity Diagrams in UML



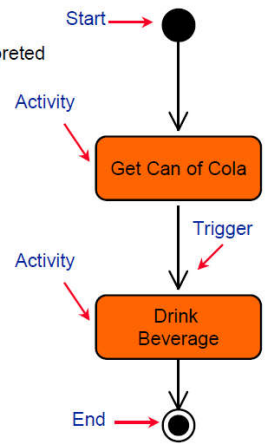

## Activity Diagrams

The core symbol is the **activity** box.

Depending on the perspective, the activity is interpreted differently.

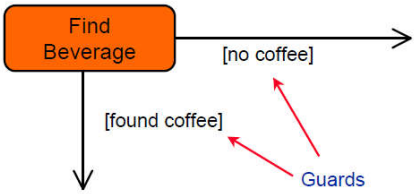
- **Conceptual** perspective:
  - An activity is some **task** that needs to be done whether by a human or a computer.
- **Specification** perspective:
  - An activity is a **method** on a class.

The links between the activities are the **triggers**.

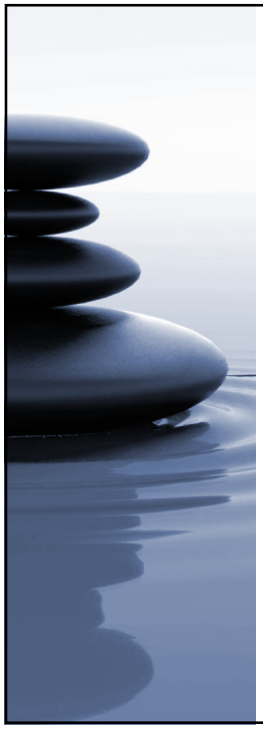


# Guards

Just like in state diagrams, guards determine which trigger is used.

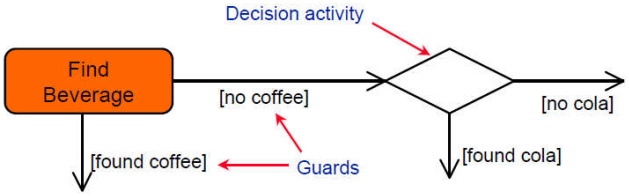



At a single point in time, only one trigger can occur.
Therefore **guards** essentially need to be **exclusive** !



# Decision Activity

To describe nested decisions, UML activity diagrams offer the decision-diamond activity symbol.




Coding:

```
if (found_coffee()) { ... }
else
  if (found_cola()) {  ... }
  else { ... }
```

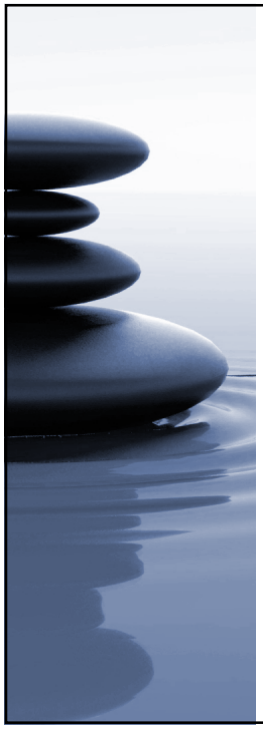
# Concurrent Activities

The difference between **flowcharts** and **activity diagrams** is that in activity diagrams **parallel behavior** can be expressed.

This is important for business modeling, where unnecessary sequential processes can be designed for parallel execution.

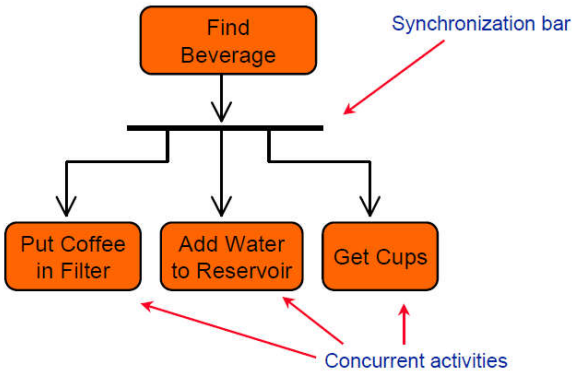
This improves the efficiency and responsiveness of business processes.

Activity diagrams are also useful for concurrent programs, since you can graphically lay out what threads you have and when they need to synchronize



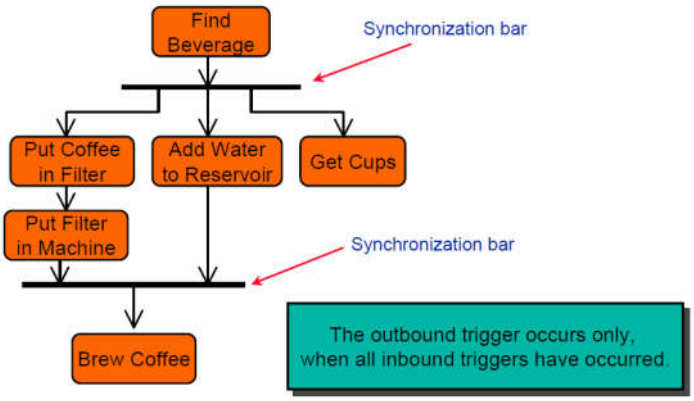
# Synchronization Bars

Synchronization bars initiate concurrent sections in an activity diagram. In these concurrent sections, triggers can occur in parallel and no sequential order is established.



# Synchronization Bars

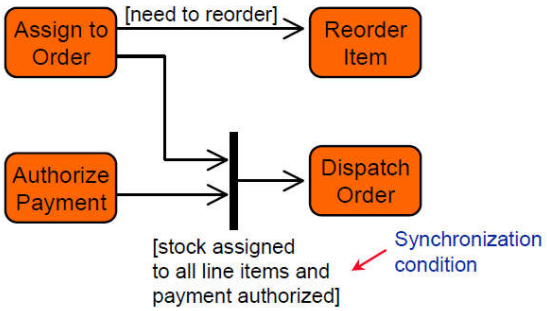
Synchronization bars synchronize concurrent activities.




# Synchronization Conditions

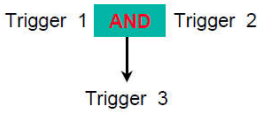
The default behavior of synchronization bars is that the outbound trigger occurs as soon as all inbound triggers have occurred.

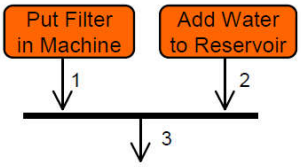
In addition to this condition, you can specify an **extra synchronization condition** which is checked every time an inbound trigger occurs.



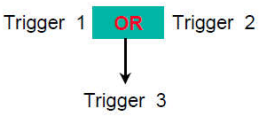
# Multiple Incoming Triggers

For synchronization bars, the outbound trigger occurs only if all inbound triggers have occurred.




For activities the trigger semantics is different. Activities occur as soon as one (of many possible) inbound trigger occurs.



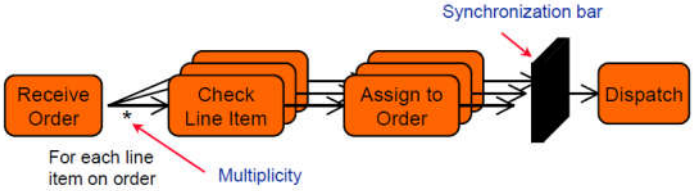


# Multiple Triggers

The second source of **parallelism** in activity diagrams are **multiple triggers**.

The multiple trigger with the multiplicity marker (*) is used just like it is done in class diagrams.

- Although it is not part of the UML, you should state what the basis of the multiple trigger is, to improve readability and understandability.
- To synchronize multiple threads initiated by a multiplicity marker again a synchronization bar is used.



# Swimlanes

Activity diagrams tell you what happens, but they do not tell you **who** does what.

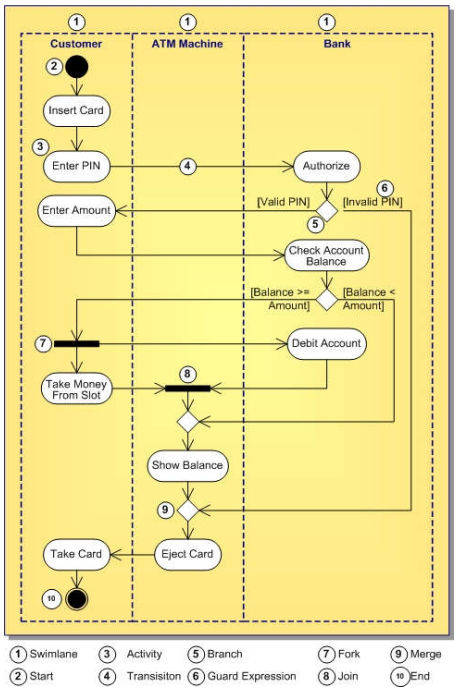
From the **implementation** perspective, this means, that the diagram does not convey which **class** is **responsible** for which activity.

From the **domain** view point, this means that the diagram does not show which **people** or departments are responsible for which activity.

**Swimlanes** are a way around this.

Swimlanes are indicated by vertical dashed lines which separate the diagram into **zones**.

Each zone represents a particular class, person or department, etc.



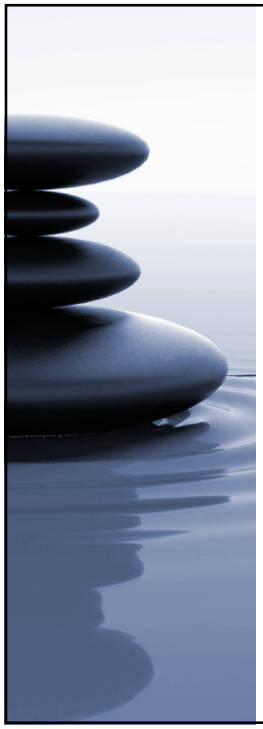
# When to Use Activity Diagrams

Activity diagrams show **behavior** that **spans** over **multiple use cases** to describe the **workflow** of the overall process.

For multiple **objects** and their high-level **interaction**, activity diagrams are particularly helpful for representing an overview of **concurrent processes**.

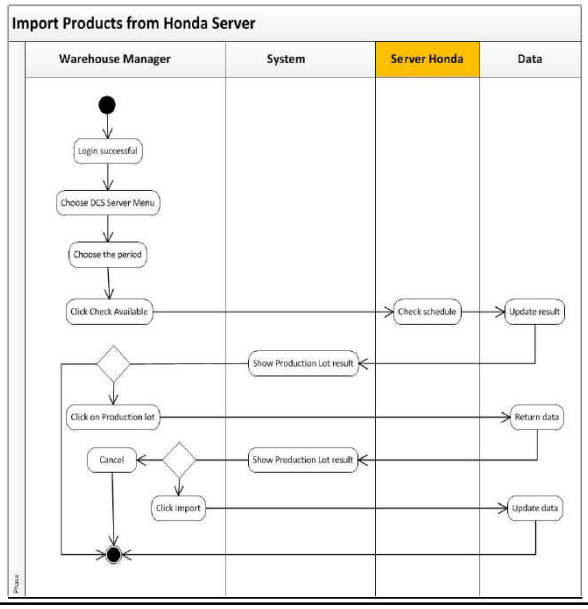
Do **not** use activity diagrams to see how objects **collaborate**. An interaction diagram is simpler and gives you a clearer picture of collaborations.

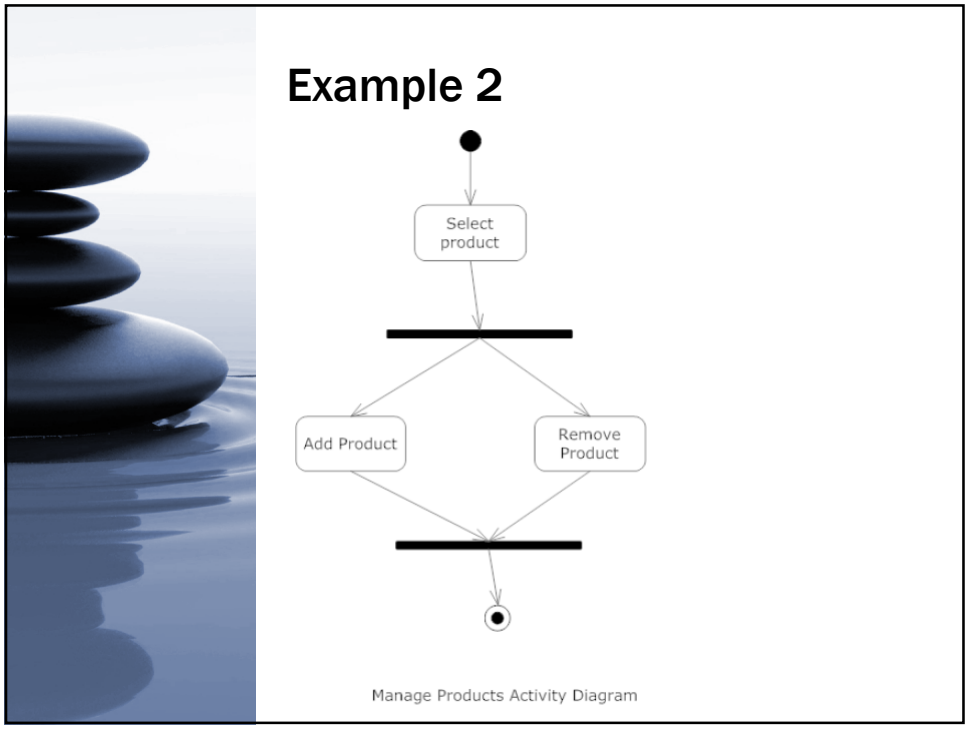
Activity diagrams are **not** accurate for describing how an **object behaves** over its lifetime. Use a state diagram instead.

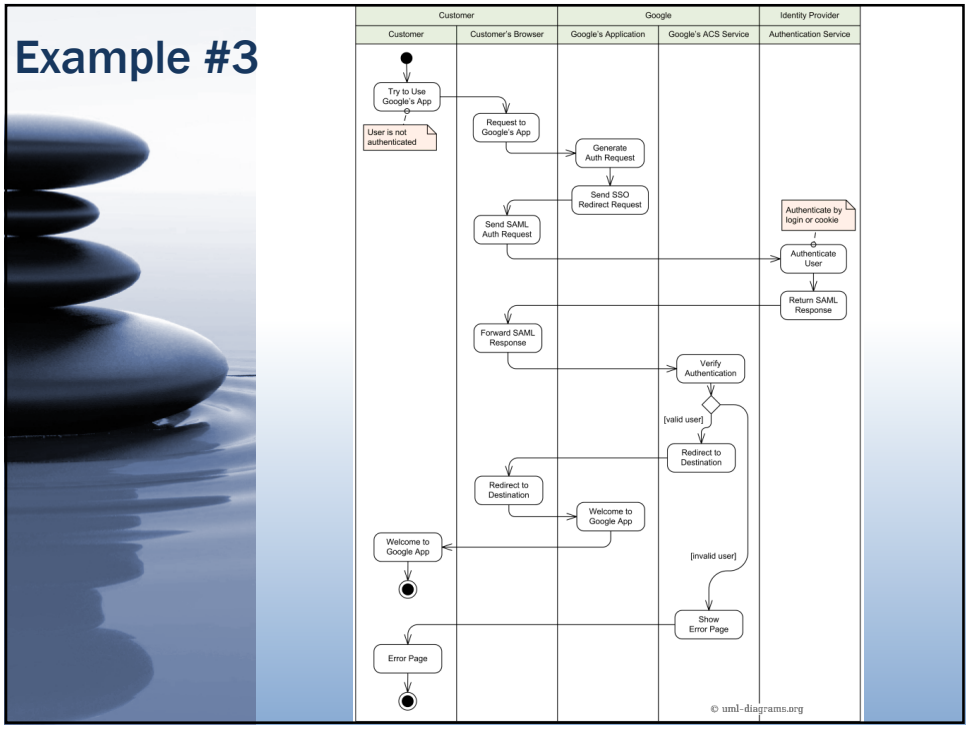


# Example

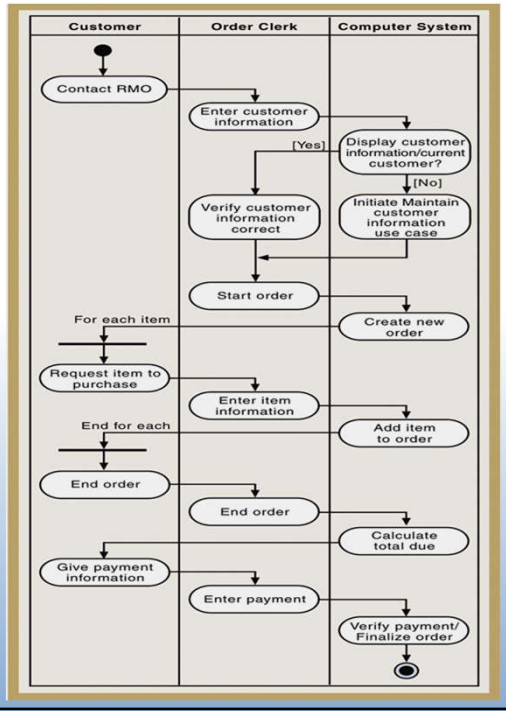
What's wrong here?




Example 2
Select product
Add Product
Remove Product
Manage Products Activity Diagram


Example #3
Customer
Google
Identity Provider
Customer
Customer's Browser
Google's Application
Google's ACS Service
Authentication Service
Try to Use Google's App
User is not authenticated
Request to Google's App
Generate Auth Request
Send SSO Redirect Request
Send SAML Auth Request
Authenticate by login or cookie
Authenticate User
Return SAML Response
Forward SAML Response
Verify Authentication
[valid user]
Redirect to Destination
Redirect to Destination
Welcome to Google App
Welcome to Google App
[invalid user]
Show Error Page
Error Page
© uml-diagrams.org

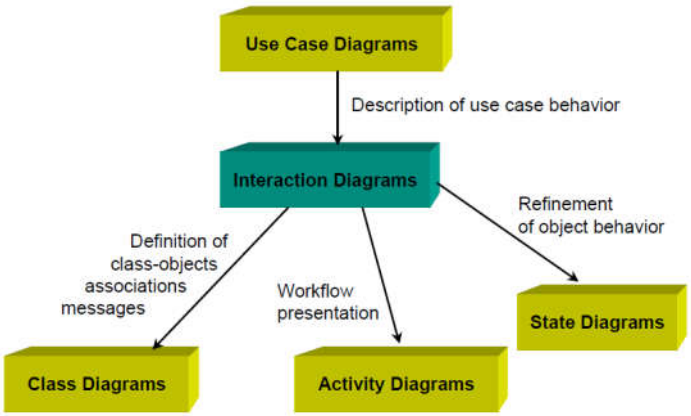
## Example #4




PART #2

# INTERACTION DIAGRAM

## Role of Interaction Diagrams in UML




## Interaction Diagram

Interaction diagrams describe **exemplary** how groups of objects **collaborate** in some **behavior**.

An interaction diagram typically captures the behavior of a **user goal use case**.

Interaction diagrams do **not** capture the **complete** behavior, **only** typical **scenarios**

There are two types of interaction diagrams:

**Sequence diagrams** emphasize the order or concurrency of the interactions.

**Collaboration diagrams** emphasize the interacting objects

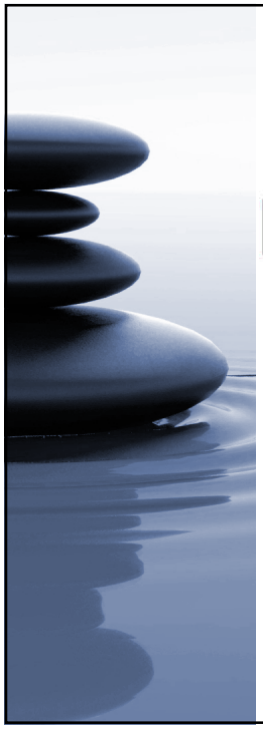
## Interaction Diagrams and Use Cases

Behavior of the “order” use case:

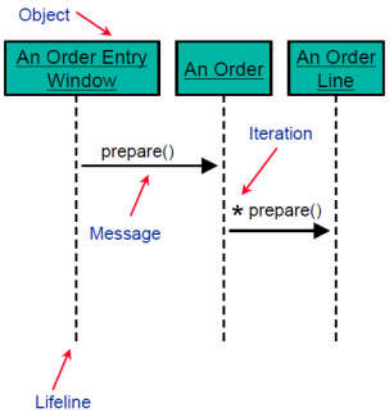
- A customer orders several products.
- The (sub-)orders (“order lines”) for each product are prepared separately.
- For each product check the stock.

• If the product is in stock, remove requested amount from stock.

• If the product stock falls below a predefined level, reorder it.

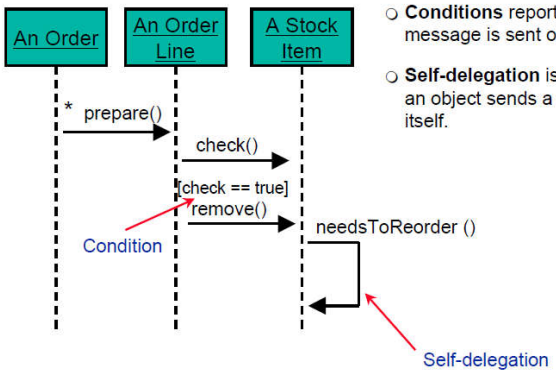


## Sequence Diagrams



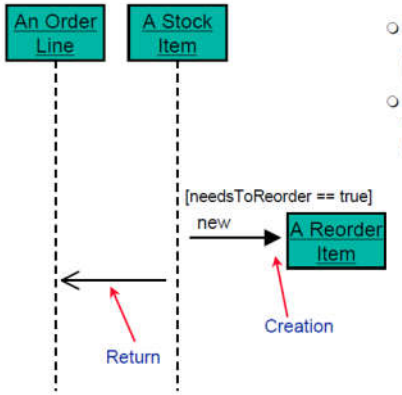

- The **objects** and **messages** are taken from the class diagram.
- The **lifeline** represents the object´s life during the interaction.
- **Messages** are represented by arrows between lifelines, labeled at minimum with the message name.
- To send a message between objects, there has to be an **association** between the classes in the class diagram.
- The **sequence** in which messages occur is shown top to bottom.
- The **iteration** marker indicates that a message is sent many times to multiple receivers.

# Sequence Diagrams



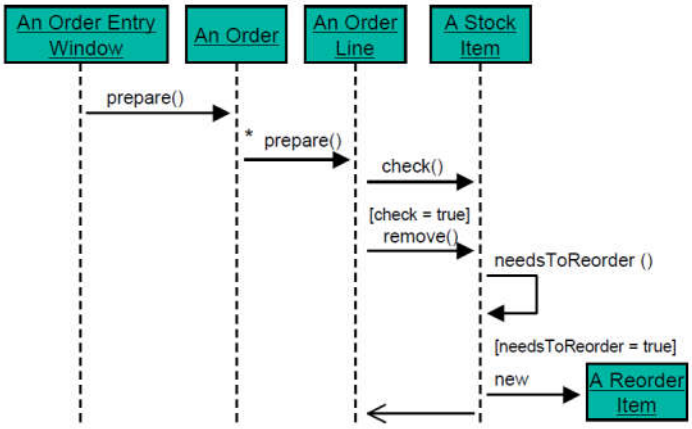

- **Conditions** report whether a message is sent or not.
- **Self-delegation** is used whenever an object sends a message to itself.

# Sequence Diagrams




- **Creation** is whenever a new object appears below the starting level.
- **Returns** indicate the return of control after a message, **not** a new message.

# Sequence Diagrams




## Concurrent Processes and Activations

UML offers diagram elements to show concurrent processes explicitly within sequence and collaboration diagrams.

Activation:
- indicates whenever a method is active,
- shows methods which are waiting for returns.

Asynchronous message:
- allows the creation of new threads or objects,
- communicates with already running threads or objects.

Deletion:
- occurs by self destruction of threads or objects,
- can be initiated by external messages.

## Concurrency in Sequence Diagrams

Example:

- A bank transaction transfers $100 from bank account #123 to #456.

• Withdraw $100 from bank account #123.

• Deposit $100 at bank account #456.

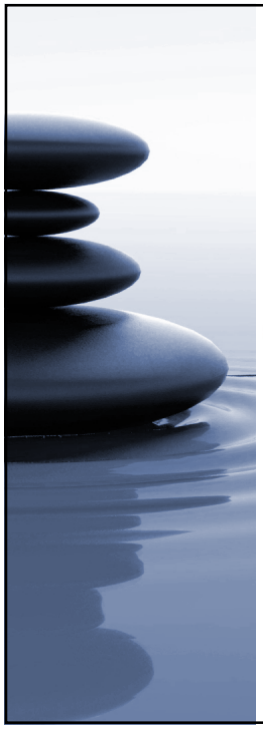
- Withdraw and deposit may be executed in parallel (concurrently).

- The transaction is only successful if both sub-transactions (withdraw and deposit) are successful.

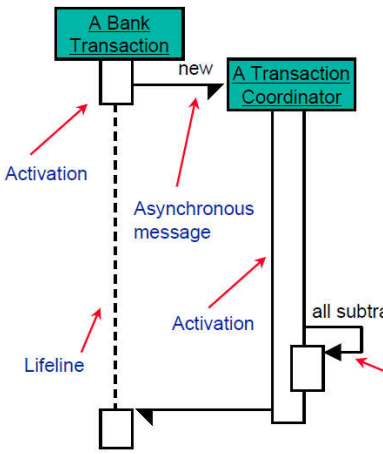
- Therefore a transaction coordinator checks the successful completion of the concurrent sub-transactions.

• Each time a sub-transaction completes successfully it checks if all other sub-transactions have been completed successfully. If this is the case, it reports success to its client.

• If any sub-transaction fails, it cancels the other sub-transactions and reports failure to its client.

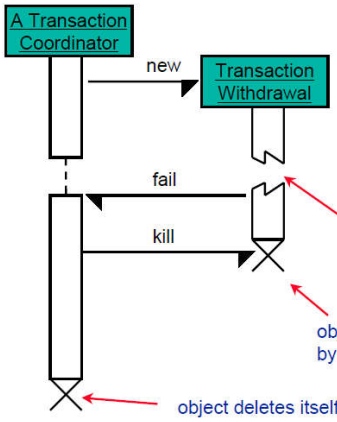


## Concurrency in Sequence Diagrams



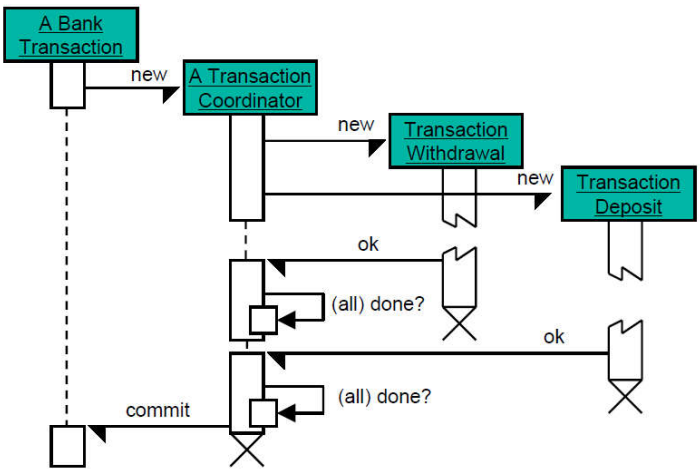

- An **activation** is a method that is either executing or waiting for a return (active method).
- Activations are also useful for describing **self-delegation**.
- An **asynchronous message** is a message that does not block the caller, i.e., caller and receiver execute concurrently.

## Concurrency in Sequence Diagrams

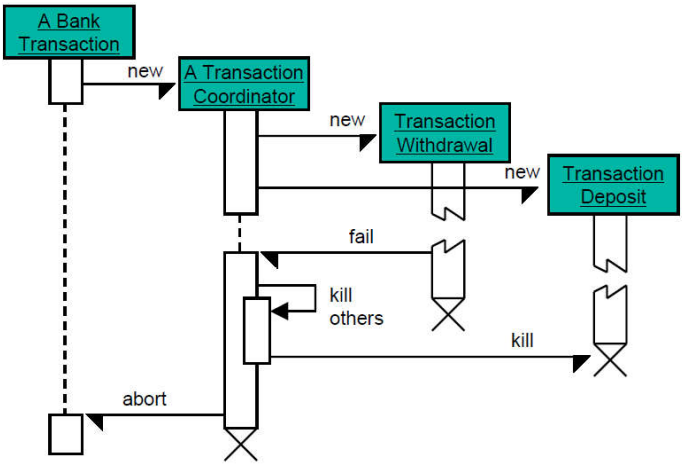



- Unnecessary **details** may be **suppressed**. Remember that sequence diagrams are only scenarios and not a complete description of the behavior.
- Objects may be **deleted explicitly** by other objects or by themselves (self-destruction).

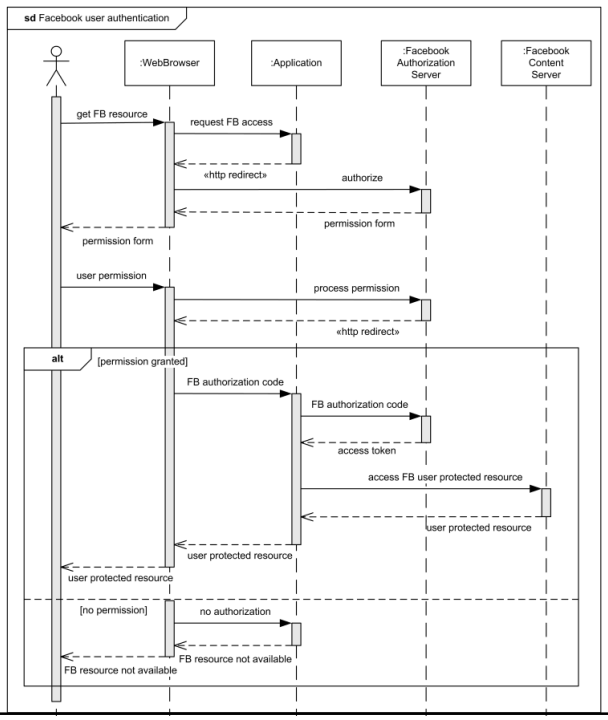
## A Successful Bank Transaction



# A Failing Bank Transaction

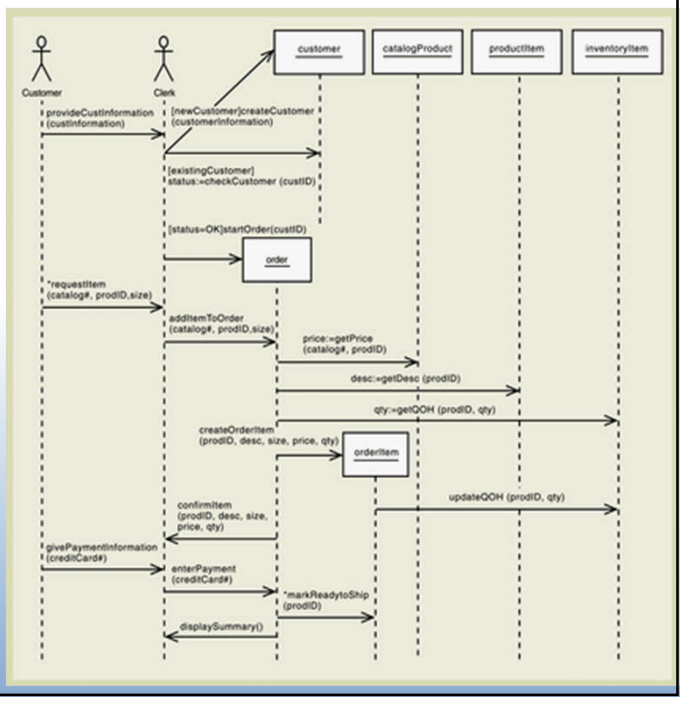



# Example #1

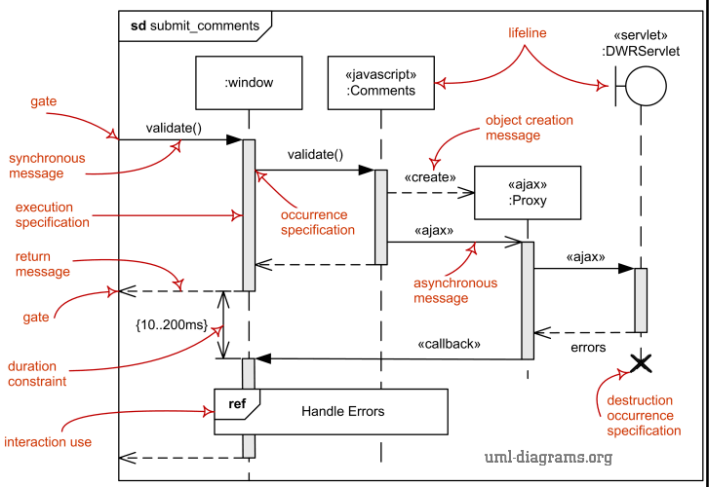


# Example #2

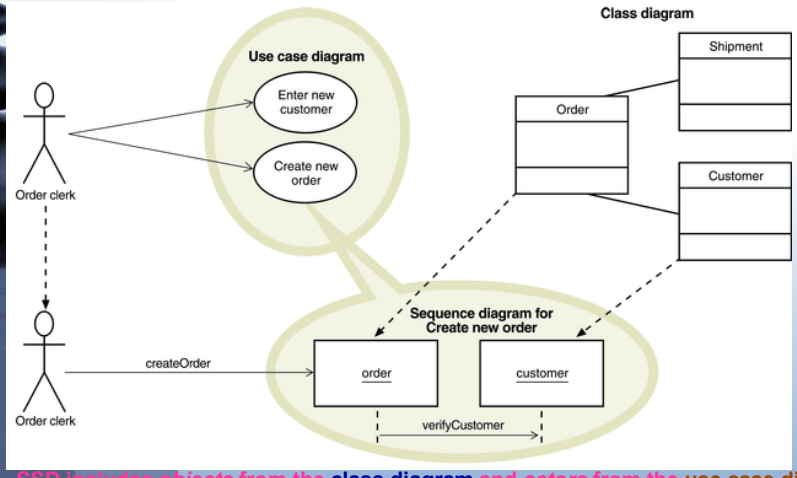
A complete SSD for the RMO use case *Create new order*




# Summary



# Relationship between use case diagram, class diagram and sequence diagrams




SSD includes *objects* from the class diagram and *actors* from the use case diagram



# Collaboration Diagrams

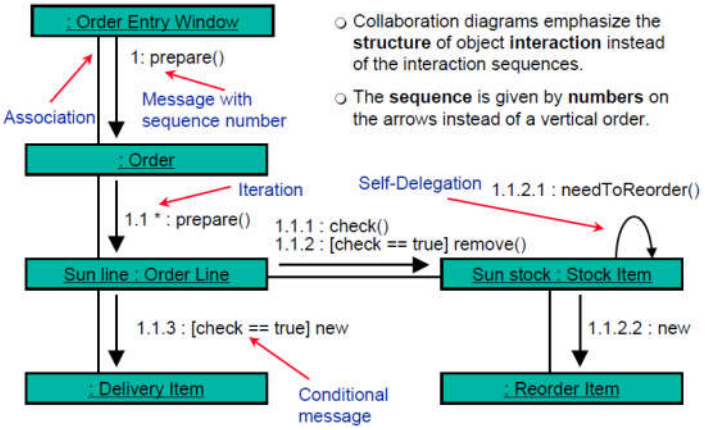
Dynamic behavior of objects can, in addition to sequence diagrams, also be represented by collaboration diagrams.

The transformation from a sequence diagram into a collaboration diagram is a bidirectional function.

The difference between sequence diagrams and collaboration diagrams is that ***collaboration diagrams emphasize more the structure than the sequence of interactions***.

Within sequence diagrams the order of interactions is established by vertical positioning whereas in collaboration diagrams the sequence is given by numbering the interactions.

# Collaboration Diagrams




- Collaboration diagrams emphasize the **structure** of object **interaction** instead of the interaction sequences.
- The **sequence** is given by **numbers** on the arrows instead of a vertical order.

# When to Use Interaction Diagrams

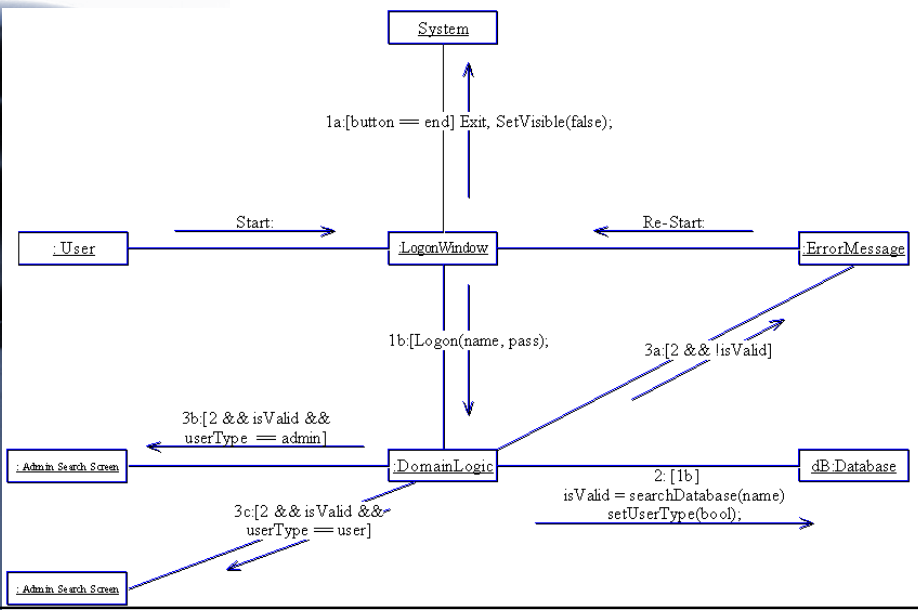
**Use Interaction Diagrams**

- When catching user requirements:
  - Describe the behavior of several objects within a single use case.
  - Show collaborations among objects.
- After having described the object behavior completely with state and activity diagrams:
  - Test the state and activity diagrams against the scenarios

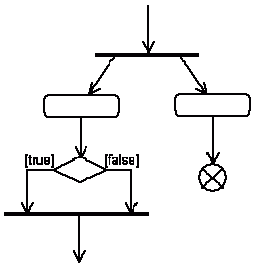
**Do not Use Interaction Diagrams**

- For precise definition of a single class behavior (use state diagrams).
- If you want to describe the behavior across many use cases or many threads (consider an activity diagram).

# Exercise #1




What, if anything, is wrong with the following activity diagram (other than missing action node names)?



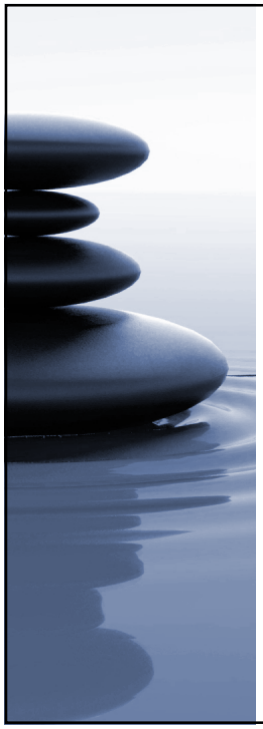
# Exercise #2

Create an activity diagram based on the following narrative.

The purchasing department handles purchase requests from other departments in the company.

People in the company who initiate the original purchase request are the "customers" of the purchasing department. A case worker within the purchasing department receives that request and monitors it until it is ordered and received.

Case workers process the requests for purchasing products under Rs. 1,500, write a purchase order, and then send it to the approved vendor. Purchase requests over Rs. 1,500 must first be sent out for a bid from the vendor that supplies the product. When the bids return, the case worker selects one bid. Then, the case worker writes a purchase order and sends it to the approved vendor.



# Exercise #3_ An online stock broker

Write an activity diagram for the processing of a stock trade order that has been received by an online stock broker.

The online stock broker first verifies the order against the customer’s account, then executes it with the stock exchange.

If the order executes successfully, the system does three things concurrently: mails trade confirmation to the customer, updates the online portfolio to reflect the results of the trade, and settles the trade with the other party by debiting the account and transferring cash or securities.

When all three concurrent threads have been completed, the system merges control into a single thread and closes the order. If the order execution fails, then the system sends a failure notice to the customer and closes the order.

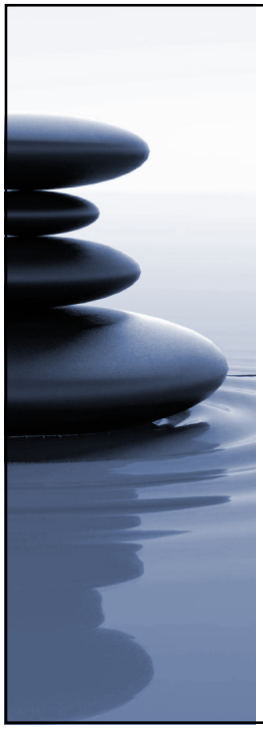
## Exercise #4_ Start New Game Round

Let's do a sequence diagram for the following poker casual use case, :

The scenario begins when the player chooses to start a new round in the UI. The UI asks whether any new players want to join the round; if so, the new players are added using the UI.

All players' hands are emptied into the deck, which is then shuffled. The player left of the dealer supplies an ante bet of the proper amount. Next each player is dealt a hand of two cards from the deck in a round-robin fashion; one card to each player, then the second card.

If the player left of the dealer doesn't have enough money to ante, he/she is removed from the game, and the next player supplies the ante. If that player also cannot afford the ante, this cycle continues until such a player is found or all players are removed.



## Exercise #5_ Betting Round

The scenario begins after the Start New Round case has completed. The UI asks the first player for a bet. That player chooses to either bet a given amount, or check (no bet).

The next player is asked what to do. If the prior player placed a bet, the next player must either match ("see") it, or match it plus add an additional bet ("raise"), or choose not to match and exit the round ("fold"). This continues around the table until an entire pass is made in which all players have either matched all other players' bets or folded.

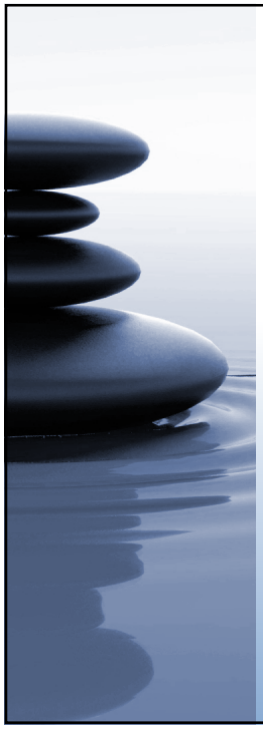
If the next player doesn't have enough money to match the current bet, the player is allowed to bet all of their money. But they can then win only up to the amount they bet; the rest is a "side pot" among the more wealthy players remaining in the round.

# Exercise #6

The scenario begins when the user chooses to add a new appointment in the UI. The UI notices which part of the calendar is active and pops up an Add Appointment window for that date and time.

The user enters the necessary information about the appointment's name, location, start and end times. The UI will prevent the user from entering an appointment that has invalid information such as an entering an appointment that has invalid information, such as an empty name or negative duration. The calendar records the new appointment in the user's list of appointments. Any reminder selected by the user is added to the list of reminders.

If the user already has an appointment at that time, the user is shown a warning message and asked to choose an available time or replace the previous appointment. If the user enters an appointment with the same name and duration as an existing group meeting the calendar asks the user whether he/she intended to join that group meeting instead. If so, the user is added to that group meeting's list of participants.



# Any Questions?

✉ hvusynh@hcmiu.edu.vn