Unified Modeling Language (UML)

Instructor:

Agenda

After the course, student will:

- **Understand about UML concepts**
- **Be able to read simple analysis/design UML diagrams**
  - Can analysis overview of the system through symbols of UML such as: actor, events, use case scenario, diagram
  - Can analysis the activities/processing/ relationship of object in system through symbols

What is UML ?

- UML - **U**nified **M**odeling **L**anguage
- Standard & graphical language
- Used for software as well as non software
  - UML is a language (***notation***) for modeling Object Oriented system
- Used for making software blue print
  - ***specifying, visualizing, constructing, documenting*** the artifacts of software system

Why UML?

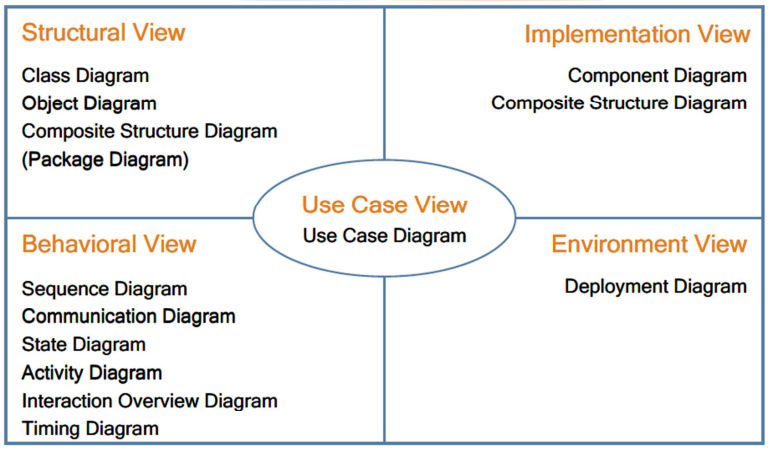
- Graphical notation
  - A picture is worth a thousand words
- ***Standard communication*** language
- Provides multiple diagrams for capturing different ***Architectural Views***
- Promotes component reusability

UML is a standard language for ***visualizing, specifying, constructing, and documenting*** software systems

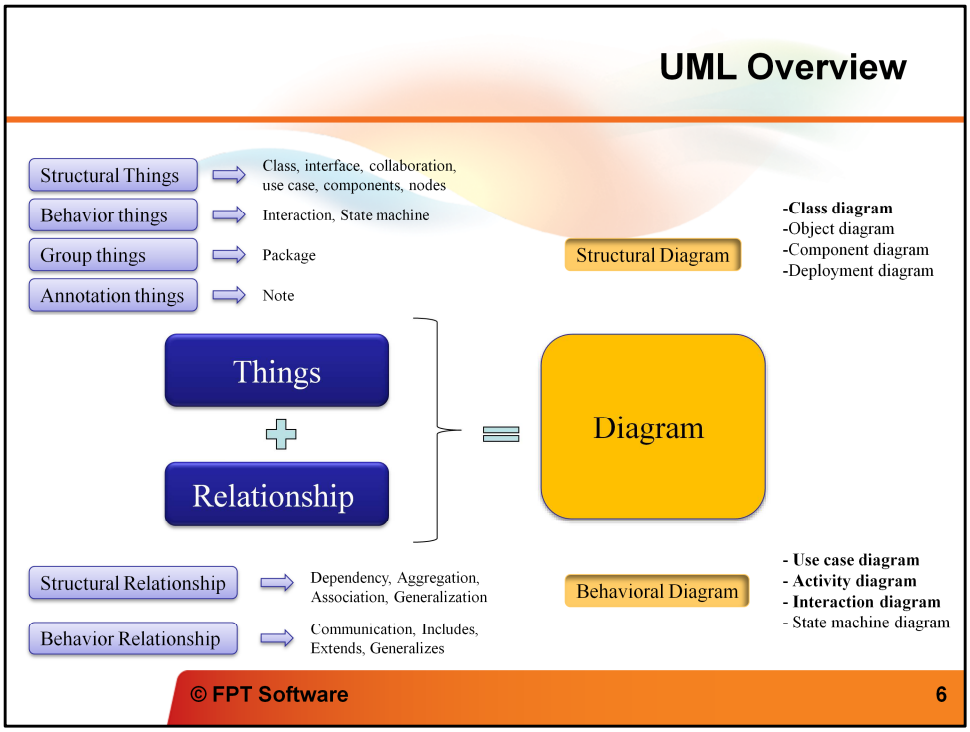
UML Architectural Views and Diagrams

- UML defines 13 diagrams that describe 4+1 architectural views

  4+1 architectural views model was proposed by Philippe Kruchten, IBM



Emphasis that this slide will focus on 4 common used diagrams including Use case diagram, Class diagram, Activity diagram and Sequence diagram.



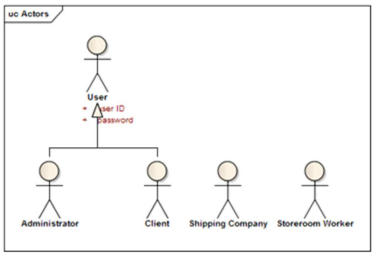
•Structural things allow the user to describe relationships.
•Behavioral describe how things work
•Group things are used to define boundaries.
•Annotational things are used to add notes to the diagrams

Structural relationships are used to tie the things together in the structural diagrams
Behavioral relationships are used in the behavioral diagram

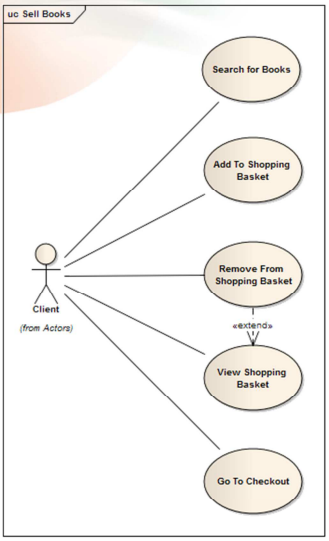
For example, structural diagram used to describe the relationships between classes. Behavioral diagrams can be used to describe the interaction between people (called actors in UML) and the things we refer as a use case, or how the actors use the system.
Bold ones are the most commonly used UML diagrams.

UML – Use case diagram

- Describes the functionality provided by system
- A Use Case represents a discrete unit of interaction between a user (human or machine) and the system.
- Contains ***actors, use cases***, and ***relationships***



BookShop Sample: Actors + Sell Book Use Cases

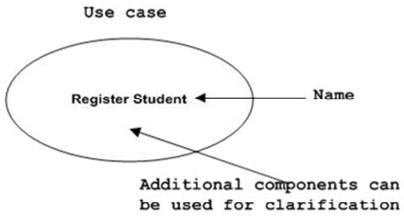


Use case digaram là để mô tả một cách tổng quát chức năng của hệ thống theo actor (một người hoặc một đối tượng nào mà có tương tác với hệ thống) và theo những gì mà actor đó muốn. Hay nói một cách đơn giản là use case diagram cho biết hệ thống có những chức năng, actor nào và chúng liên quan đến nhau như thế nào

A use case diagram, describing how the system is used.
A use case scenario (although technically it is not a diagram) is a verbal articulation of exception to the main behavior described by the primary use case.

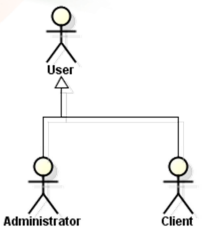
Use case diagram – Structural things

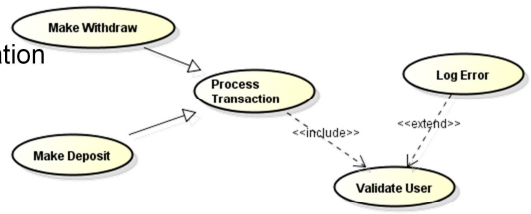
- Actor:
  - Can be human user, internal application external application
- Use case
  - High level functionalities of a system



Use case diagram – Relationship

- Actor Relationship: Generalization
  - Used to define overlapping roles between actors
- Actor – Use case relationship
  - Association
- Use case Relationship
  - Include
  - Extend
  - Generalization/Specification

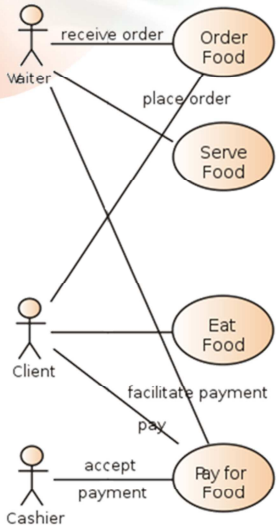




Actor relationship: generalization: tức là tổng quát hoá: như hình bên: khách hàng và người quản trị hê thống sẽ được tổng quát hoá = 1 actor là người sử dụng. Việc sử dụng relationship này rất hữu dụng khi bạn muốn định nghĩa những chức năng trùng nhau của actor, ví dụ: cả khách hàng và người quản trị hê thống đều phải login và có account/password

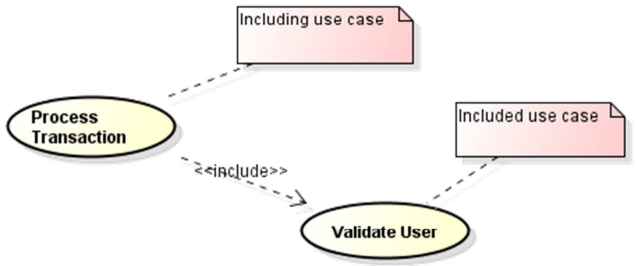
Use case diagram – Association Relationship

- Association
  - Relationship between Actors & use cases
  - Actor is involved in interaction described by use case
  - If association line has Arrow head:
    - Indicate direction of invocation, primary actor
    - Indicate control flow (not data flow)



Use case diagram – Include Relationship

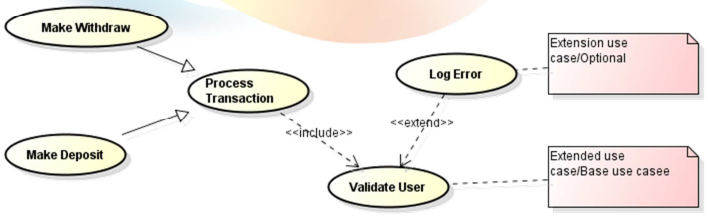
- Include:
  - Directed Relationship between 2 use cases
  - Behavior of the included use case is inserted into the behavior of the including use case



Use case relationship: include: use case đầu tiên sử dụng/ phụ thuộc vào kết quả của use case được bao gồm, ví dụ: việc thực hiện giao dịch (Process Transaction) tuỳ thuộc vào kết quả của việc kiểm định người sử dụng (benefit: use-case duoc included thuong duoc dung chung cho nhieu use-case khac)

Use case diagram – Extend Relationship

- Extends

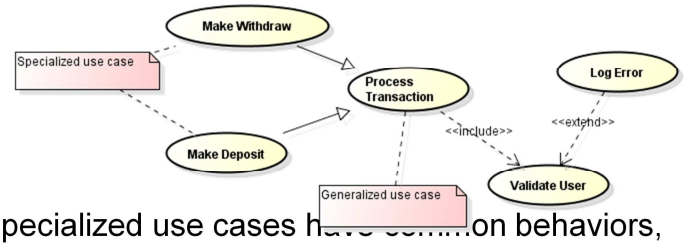


  - Behavior of extension use case may be inserted in extended use case ***under some conditions***
  - Extension use case：
    - *Optional ,*
    - *Potentially not executed with the base use case,*
    - *Not required to achieve the base use case goal.*

Use case relationship: extend: cho biết hành vi của một use case được extend (mở rộng) theo một use case khác theo một điều kiện nào đó, ví dụ: việc kiểm định người sử dụng có thể được mở rộng ra một use case Log error (ghi lỗi) nếu account/password không đúng

Use case diagram – Generalization Relationship

- Generalization/Specialization



  - Specialized use cases have common behaviors, requirements, constraints, assumptions
  - Commons are described once in general use case
  - Differences are described in specialized use case

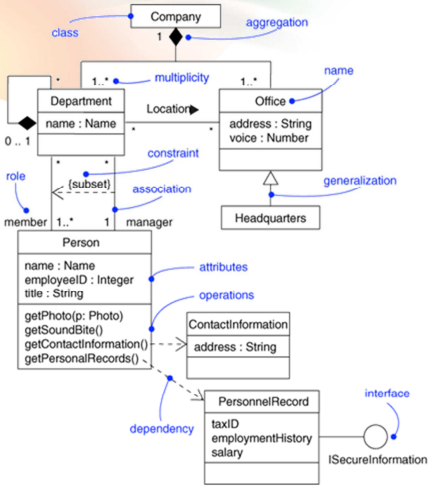
Use case relationship: generalization/specialization: cho biết sẽ có một số use case cụ thể hơn sẽ kế thừa hoặc thêm vào use case đầu tiên, ví dụ: Make withdrawl và Make Deposit là hai use case cụ thể hơn của use case thực hiện giao dịch (Process Transaction)

Use case diagram

- To draw use case diagram, identify
  - Functionalities to be presented as use case
  - Actors
  - Relationship among the use cases and actors
- Good use case diagram
  - **Name of use case** is very important.
    - Give a name that can identify the functionality to be performed
  - Give suitable **name for actor**
  - Show relationships and dependencies clearly in diagram
  - Do not include all types of relationship.
    - Main purpose of use case diagram is to capture requirement
  - Use note when ever required to clarify some important points

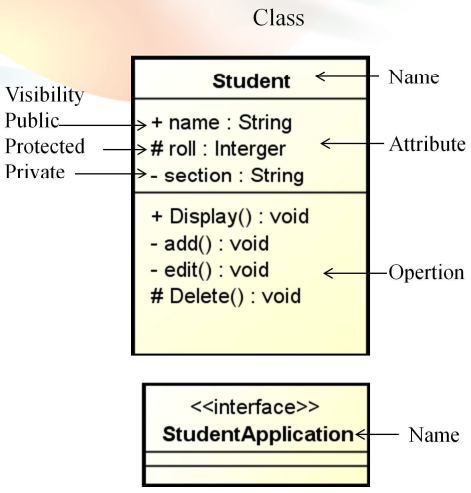
Class diagram

- Describe the structure of a system
  - Show the system's classes, their attributes, and the relationships between the classes.
- Purpose:
  - Analysis and design of the static view of an application
  - Show the collaboration among the element of a static view
  - Describe responsibilities of a system
  - Base for component and deployment diagrams



Class diagram – Structural Things

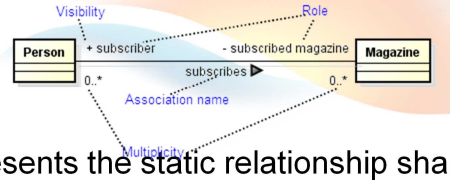
- Class
  - Represent object, having properties and responsibilities
- Interface
  - Used to describe functionalities without implementation
  - Just like a template
  - Class implement a template ➔ Implement the functionalities



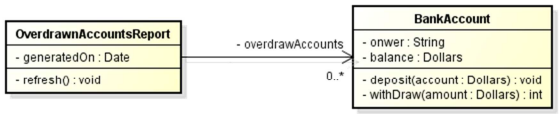
Class diagram - Relationship

- Instance level relationship
  - Association
  - Aggregation
  - Composition
- Class level relationship
  - Generalization
  - Realization
- General relationship
  - Dependency

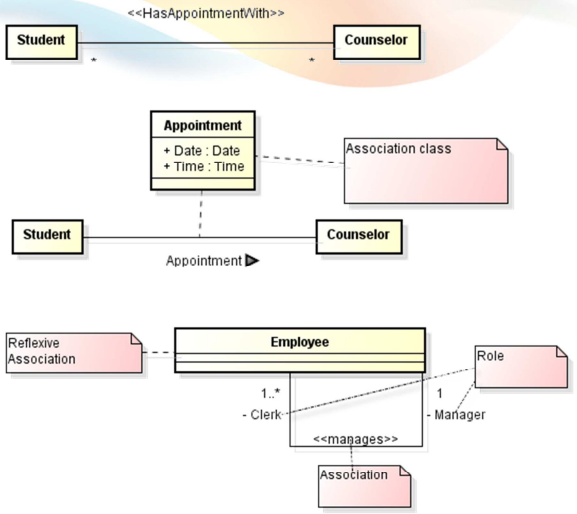
Class diagram – Association Relationship



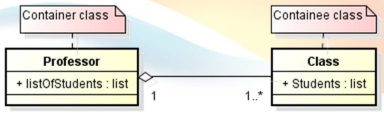
- Represents the static relationship shared among the objects of two classes
- Bi-directional:
  - both classes are aware of each other and their relationship
- Uni-directional:
  - two classes are related, but only one class knows that the relationship exists



Class diagram – Association Relationship



Class diagram – Aggregation Relationship



- A type of association
- Aggregation – “Has a” relationship
- One class is collection or container of another classes
- Contained classes do not have strong life cycle dependency on the container
  - If container is destroyed, contents are not.

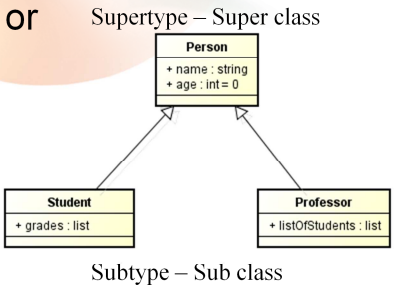
Class diagram – Composition Relationship

- Strong type of association
- Composition - “owns a” relationship
- One class is the collection of another class
- **Strong *life cycle dependency*** between container objects and contained objects
  - If container is destroyed, normally every instance that it contains is also destroyed

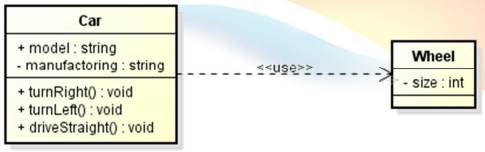
Car “has a” carburetor: composition: một động cơ có một bộ chế hoà khí. Xe mà lao xuống sông thì bộ chế hoà khí cũng tiêu luôn.

Class diagram – Generalization Relationship

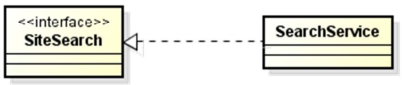
- Generalization – Inheritance or "Is a" relationship -
  - "Professor" **is a** subtype of Person
  - "Person" is a generalization of "Student" and "Professor"



Class diagram – Other Relationship



- Dependency Relationship
  - Weak relationship
  - One class depend on the other because it use the other
  - Dependency exists if a class is a parameter variable of a method of another class



- Realization Relationship
- One model implement/realize the other model
  - Ex: One class implement the interface defined by other class

Create Good Class Diagram

- **Name** of class, name of diagram should be **meaningful** to describe the aspect of the system
- Element and relationship should be identified in advance
- Responsibility (attributes and methods) should be identified in advance
- For a class: **Minimum of properties** should be specified.
  - Unnecessary properties make class diagram look complicated
- Use note when ever required
- Diagram may be drawn on plain paper and rework many times to make it correct

Activity diagram (1)

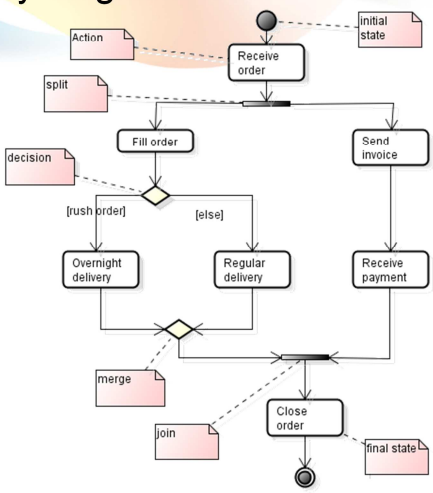
- **Activity diagrams** are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency.
- Activity diagrams show the overall flow of control.

Activity diagram (2)

- Activity diagrams are constructed from a limited number of shapes, connected with arrows. The most important shape types:
  - *rounded rectangles* represent *actions*
  - *diamonds* represent *decisions*
  - *bars* represent the start (*split*) or end (*join*) of concurrent activities
  - a *black circle* represents the start (*initial state*) of the workflow
  - an *encircled black circle* represents the end (*final state*).

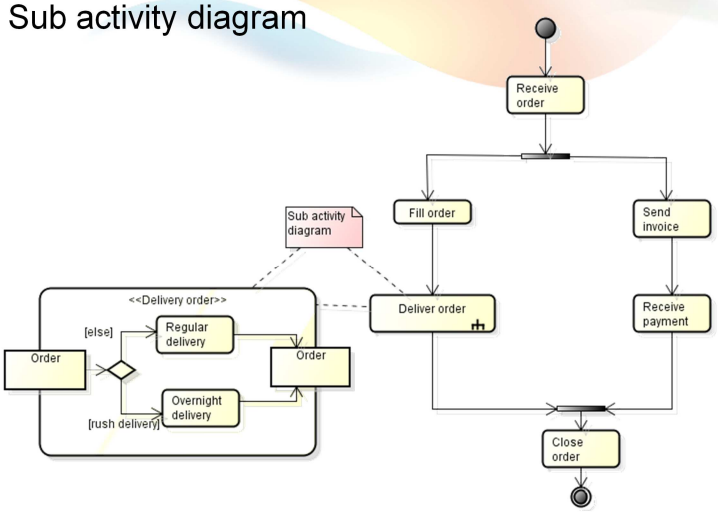
Activity diagram (3)

- Example of an activity diagram



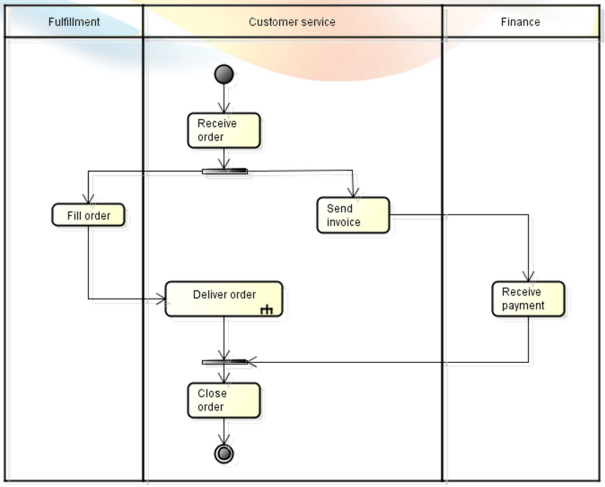
Activity diagram (4)

- Sub activity diagram



Activity diagram (5)

- Partitions



Partition help to show who do which action.

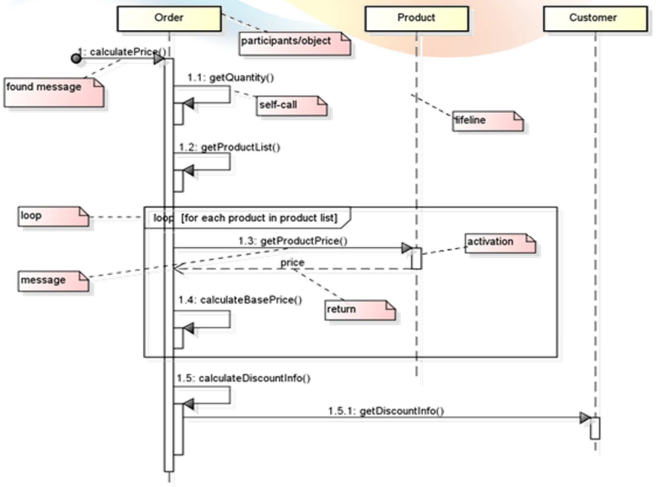
Sequence diagram(1)

- A **sequence diagram** is an interaction diagram that shows how processes operate with one another and in what order.
- A sequence diagram captures the behavior of a single scenario.
- Sequence diagram emphasis on time sequence of messages.

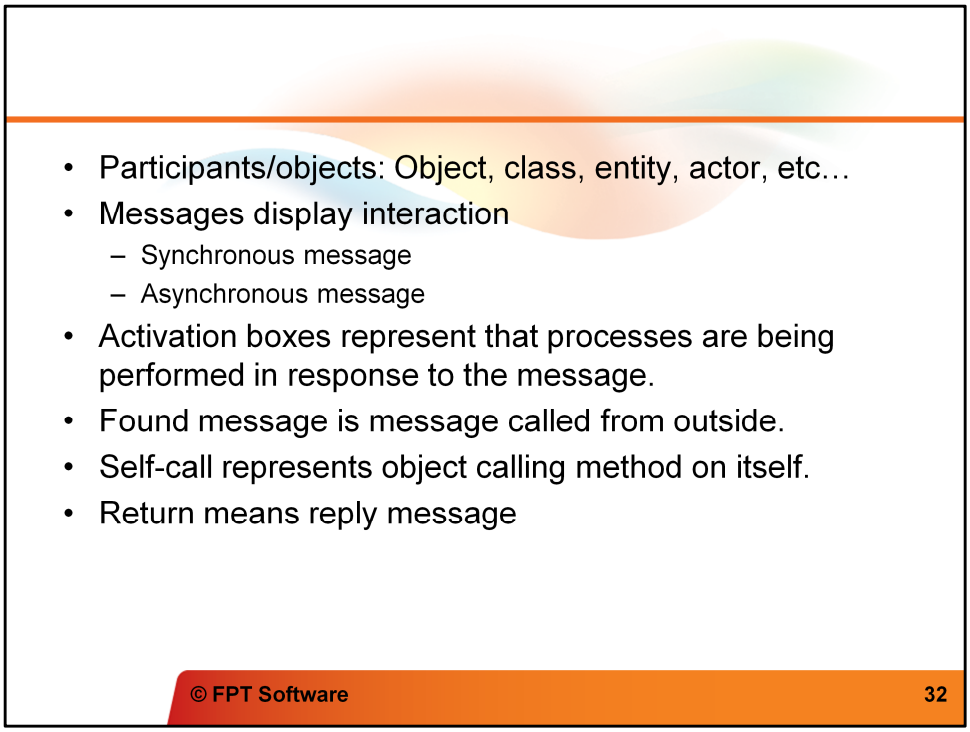
Sequence diagrams show the interaction by showing each participant with a lifeline that runs vertically down the page and the ordering of messages by reading down the page.

Sequence diagram(2)

- An example of sequence message



- Messages, written with horizontal arrows with the message name written above them, display interaction. Solid arrow heads represent synchronous calls, open arrow heads represent asynchronous messages, and dashed lines represent reply messages
- Found message is the first message that doesn't have a participant that sent it, as it comes from an undetermined source
- Participants/objects: Object, class, entity, actor, etc…
- Self-call: the object invokes the method ifself.
- Activation bar that shows when the participants is activated in the interaction.



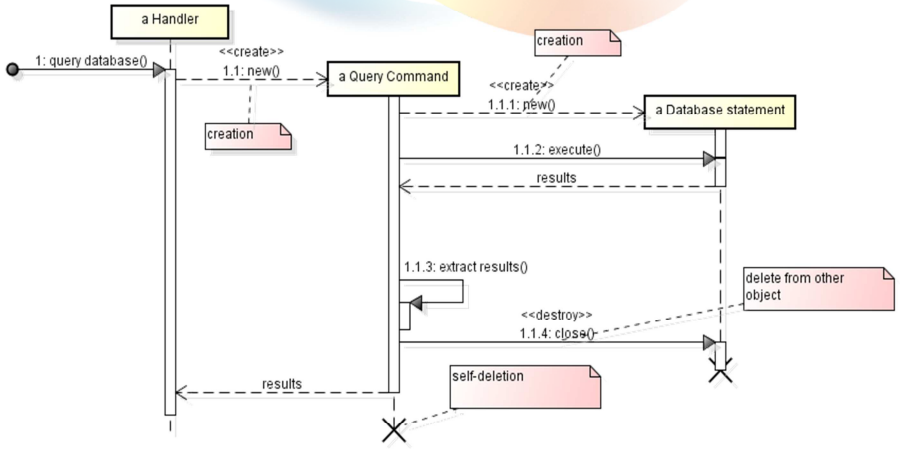
Synchronous message is represented by horizontal line with solid arrow -> If a caller sends a synchronous message, it must wait until the message is done, such as invoking a subroutine.

Asynchronous message is represented by horizontal line with open arrow -> If a caller sends an asynchronous message, it can continue processing and doesn't have to wait for a response.

Activation boxes, or method-call boxes, are opaque rectangles drawn on top of lifelines to represent that processes are being performed in response to the message

Sequence diagram(3)

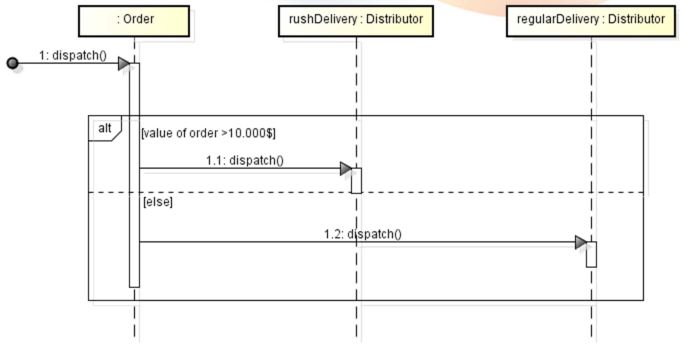
- Create and delete participants



To create a participant, you draw the message arrow directly into the participant box. A message name is optional here if you are using a constructor, but I usually mark it with "new" in any case. If the participant immediately does something once it's created, such as the query command, you start an activation right after the participant box.
Deletion of a participant is indicated by big X. A message arrow going into the X indicates one participant explicitly deleting another; an X at the end of a lifeline shows a participant deleting itself.

Sequence diagram(4)

- Combined Fragment



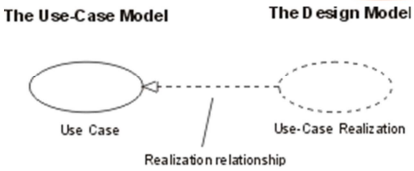
Loop: Refer to the first example

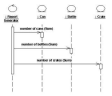
Sequence diagram(5)

- Combined Fragment
  - **alt**: Alternative multiple fragments; only the one whose condition is true will execute (if-else)
  - **opt** :Optional; the fragment executes only if the supplied condition is true. Equivalent to an alt with only one trace (if without else)
  - **loop**: Critical region; the fragment can have only one thread executing it at once.

Use case realization

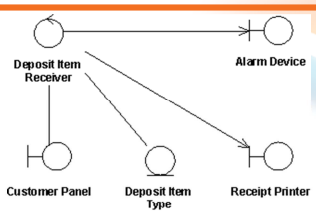
- Use-case realization represents how a use case will be implemented in terms of collaborating objects



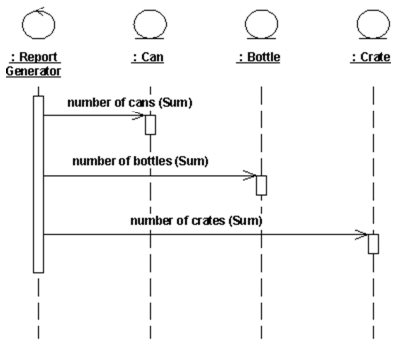


- For each use case in the use-case model, there can be a use-case realization in the analysis/design model with a realization relationship to the use case
- It may include:
  - A textual description (a document),
  - One or more class diagrams of participating classes and subsystems.
  - One or more Interaction diagrams (communication and sequence diagrams)

Use case realization Example

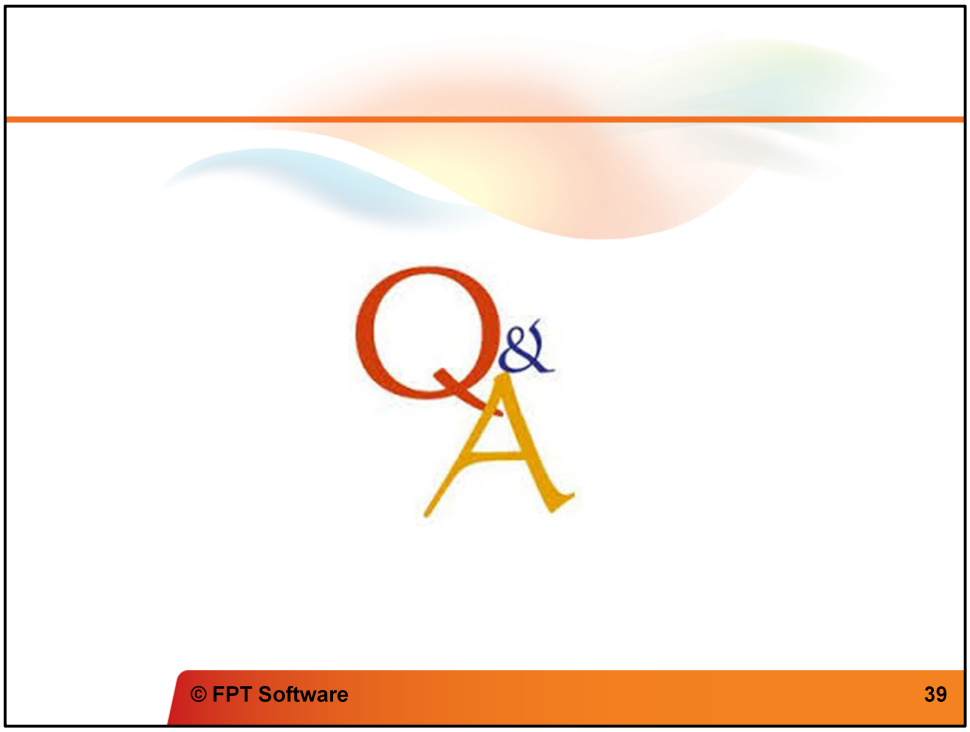


- Class diagram – Analysis stereotype
- At analysis stage
- Boundary class-
  - Handle the communication between actors and system internal components
- Entity class
  - Model for information handled by system and behavior associated with information
- Control class
  - Handle the flow of control for a use case



Resources & references

- Resources
  - http://en.wikipedia.org/wiki/Unified_Modeling_Language
  - Technical CD in Rational Rose software
  - EAExample.eap (EA 8.0)
- Recommended readings
  - The unified modeling language reference manual
- UML Tools:
  - Enterprise Architect (EA - http://www.sparxsystems.com.au/)
  - Visio
  - Rational Rose
  - Astah: http://astah.net/editions/community
  - StarUML


Q&A
© FPT Software
39