

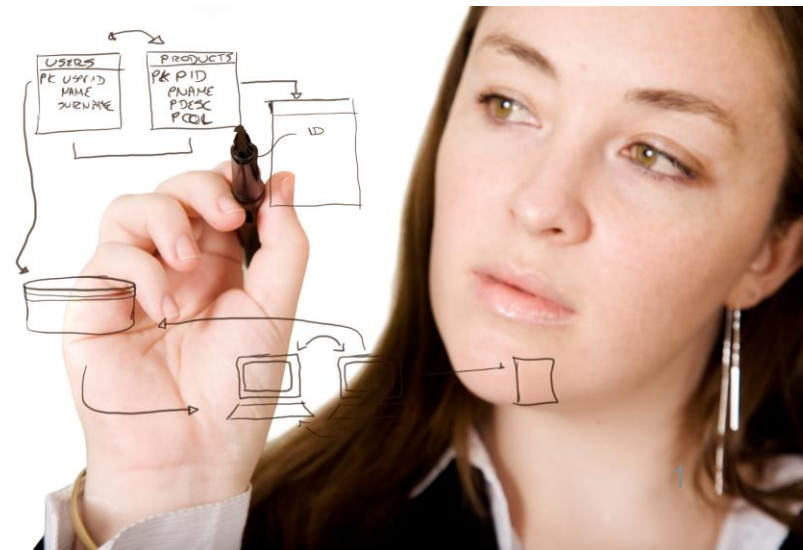
מודלים לפיתוח מערכות תוכנה

Software Systems Modeling

קורס 12003
סמסטר ב' תשע"ו

UML II .4

ד"ר ראובן יגל
robi@post.jce.ac.il



הפעם

- UML

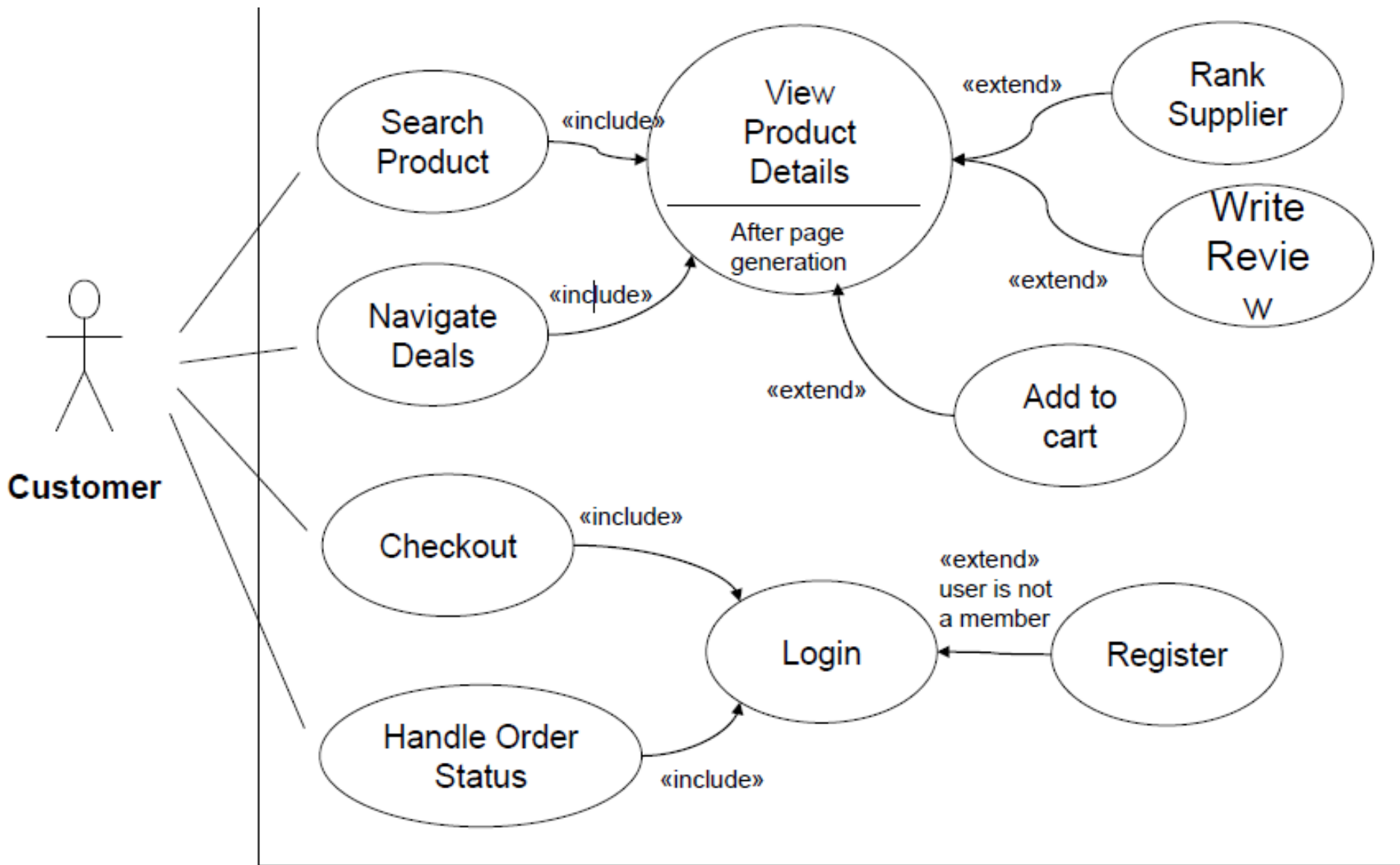
- חזרה: דיאגרמת תרחישי שימוש וכתיבת תרחיש שימוש פורמלי
- תרשימים נוספים: רצף, מחלקה, התנהגות, רכיבים והפצה
- תרגיל 1 מידול בסיסי
 - עבודה על חלק א'
 - חלק ב' + ג'
- טיוטת פרויקט הקורס

מקורות

RIT Class Wei Le •

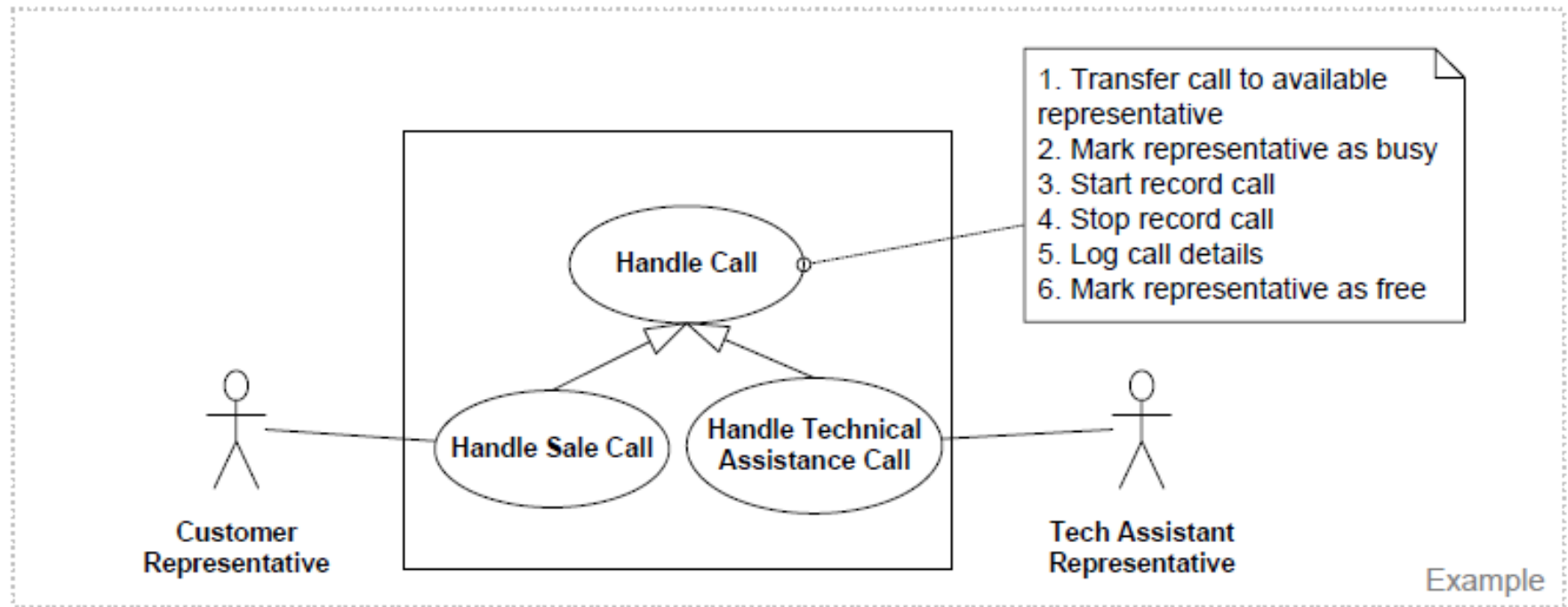
- UML
 - Fowler, "UML Distilled: A Brief Guide to the Standard Object Modeling Language"
 - Ambler, [Introduction to Object-Orientation and the UML](#)
 - Cockburn [Writing Effective Use Cases](#)
 - se-class requirement lecture
<http://jce-il.github.io/se-class/lecture/se03-requirements.pdf>

Review



Generalization

use case may have common behaviors, requirements, constraints, and assumptions with a more general use case.



Writing Use Cases

- Name:
- Actors:
- Descriptions:
 - Precondition
 - Main flow
 - Sub flow
 - Alternative flow

Precondition

- What the system needs to be true before running the use-case.
 - User account exists
 - User has enough money in her account
 - There is enough disk space

Main flow

The success scenario is the main story-line of the use-case

- Assumption: everything is okay, no errors or problems occur, and it leads directly to the desired outcome of the use-case
- It is composed of a sequence of subflows

Example:

Step 1: Administrator enters course name, code and description (interaction)

Step 2: System validates course code

Step 3: System adds the course to the db and shows a confirmation message (interaction)

Sub flow

Branches:

If the user has more than 10000\$ in her account, the system presents a list of commercials

Otherwise...

Repeats:

User enters the name of the item he wishes to buy

System presents the items

User selects items to buy

Systems adds the item to the shopping cart

User **repeats steps 1-4 until indicating he is done**

Alternative flows

Used to describe exceptional functionality

Examples:

1. Errors
2. Unusual or rare cases
3. Failures
4. Starting points
5. Endpoints
6. Shortcuts

Write Include in User Case

Reference

1. System presents homepage

2. User performs login to the system

OR

<include: login to the system>

Write Exclude in Use Case

Extension Point

User enters search string

System presents search results

Extension point: results presentations

OR

<extension point: results presentations>

Effective Use Cases

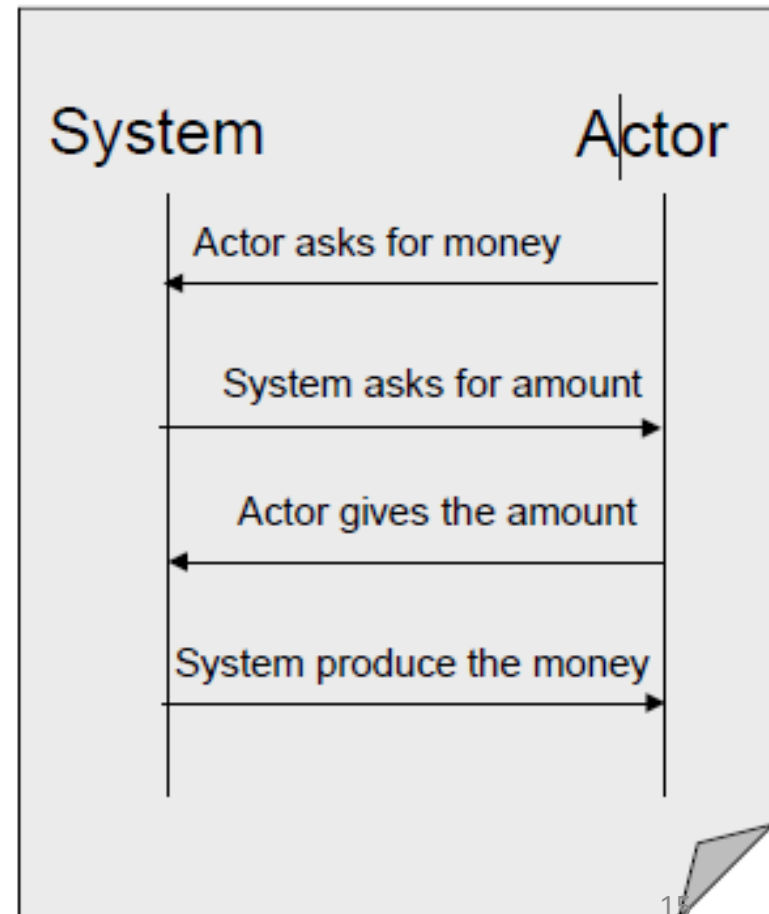
- Only one side (system or actor) is doing something in a single step
- Write from an “objective” point of view using active tense
- Any step should lead to some progress

Effective Use Cases

ATM

“Get the amount form the user and give him the money”

“User click the enter key”



Effective Use Cases – Common Mistakes

1. No actor
2. Too many user interface details “User types ID and password, clicks OK or hits Enter”
3. Very low goal details
 - User provides name
 - User provides address
 - User provides telephone number

From Use-Case to Use-Case Diagrams

- Top down ?
Starting with an overview of the system, and then splitting Use-cases
- Bottom up ?
Starting with throwing all scenarios on the page, and then combining them:

Most of the analysis process are actually
combined

Common Rules

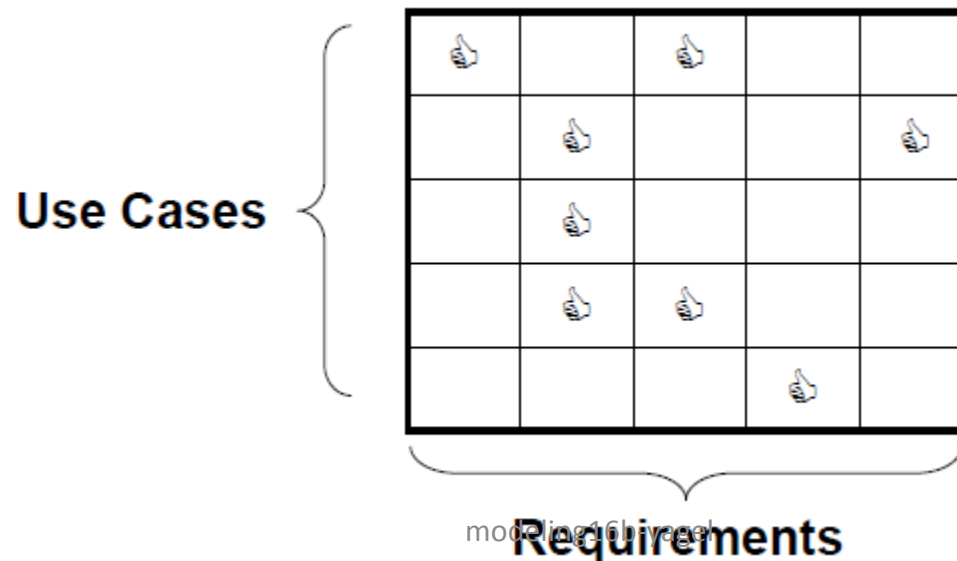
- Number Limit: The diagram should have between 3 to 10 base use-cases. No more than 15 use cases (base + included + extending).

---- If the dependency between two parts of a use-case is weak, they should be divided.

- Abstraction: All use-cases should be in similar abstraction levels.
- Size: Use cases should be described in half a page or more.
 - split using include/exclude

When we are done

- When every actor is specified.
- When every functional requirement has a use-case which satisfies it.
- A tractability matrix can help us determine it:



Use Case and Use Case Diagram

a Summary

When to use

What is Use Case and Use Case Diagram

How to Construct Use Case Diagram

How to Write Use Case

Questions?

Behavior Models

- Behavioral models at the requirement level
 1. Interaction models: describe interactions between actors and the system – sequence diagram
 2. Contracts: Specify operations invoked by actors.

Sequence Diagram

A sequence diagram shows the sequence of messages exchanged by the set of objects performing a certain task

Used during requirements analysis

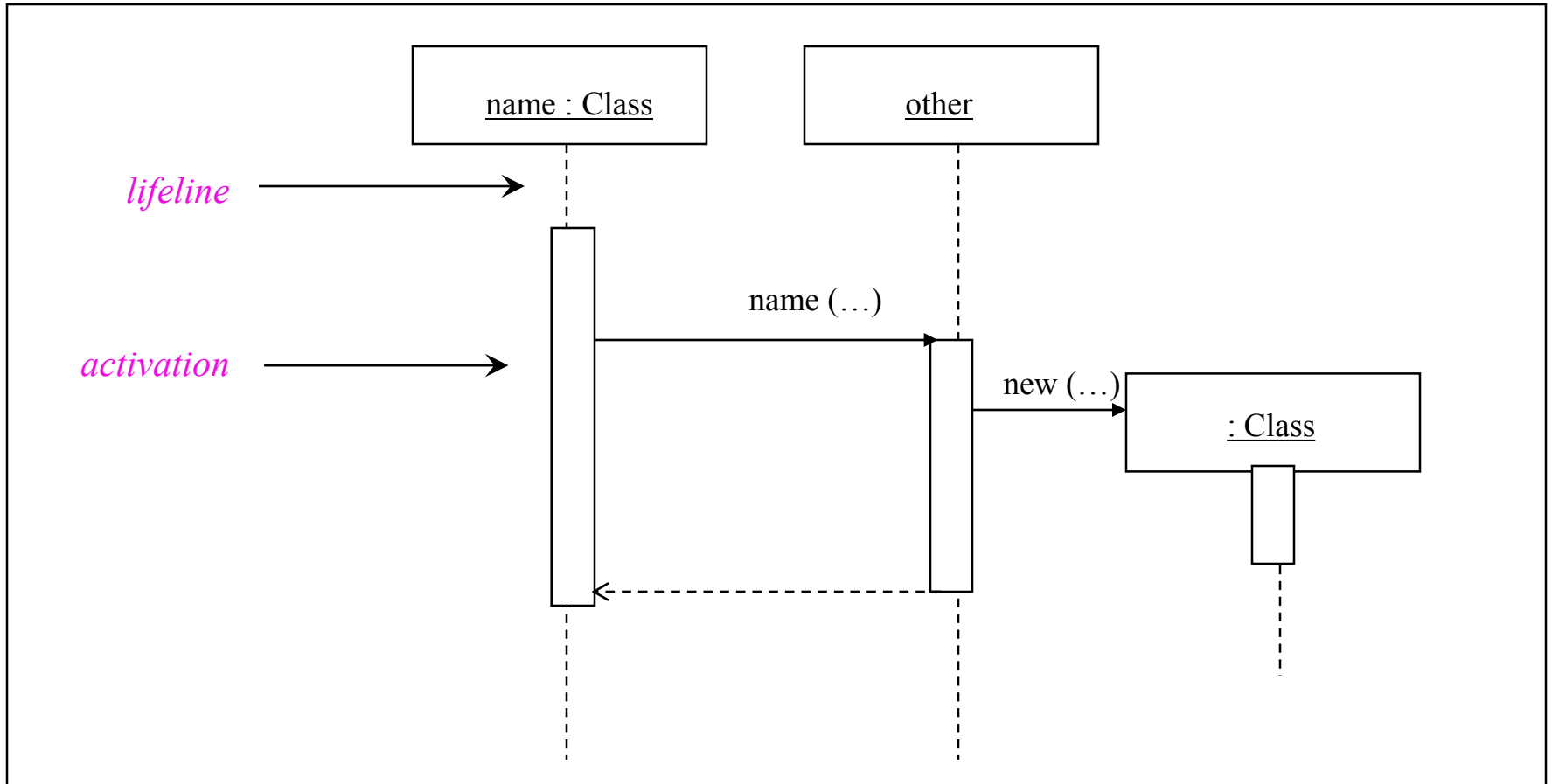
- To refine use case descriptions
- To find additional objects

Used during system design to refine subsystem interfaces

Sequence Diagram

1. **Objects** are represented by columns (first column is actor that initiates use case)
2. **Messages** are represented by arrows
3. **Activations** of an operation are represented by narrow rectangles

Sequence diagram



Sequence Diagram

- The **objects** are arranged horizontally across the diagram.
- An **actor** that initiates the interaction is often shown on the left.
- The **vertical dimension represents time**.
- A vertical line, called a *lifeline*, is attached to each object or actor.
- The lifeline becomes a broad box, called an *activation box* during the *live activation* period.
- A **message** is represented as an arrow between activation boxes of the sender and receiver.
 - A message is labelled and can have an argument list and a return value.

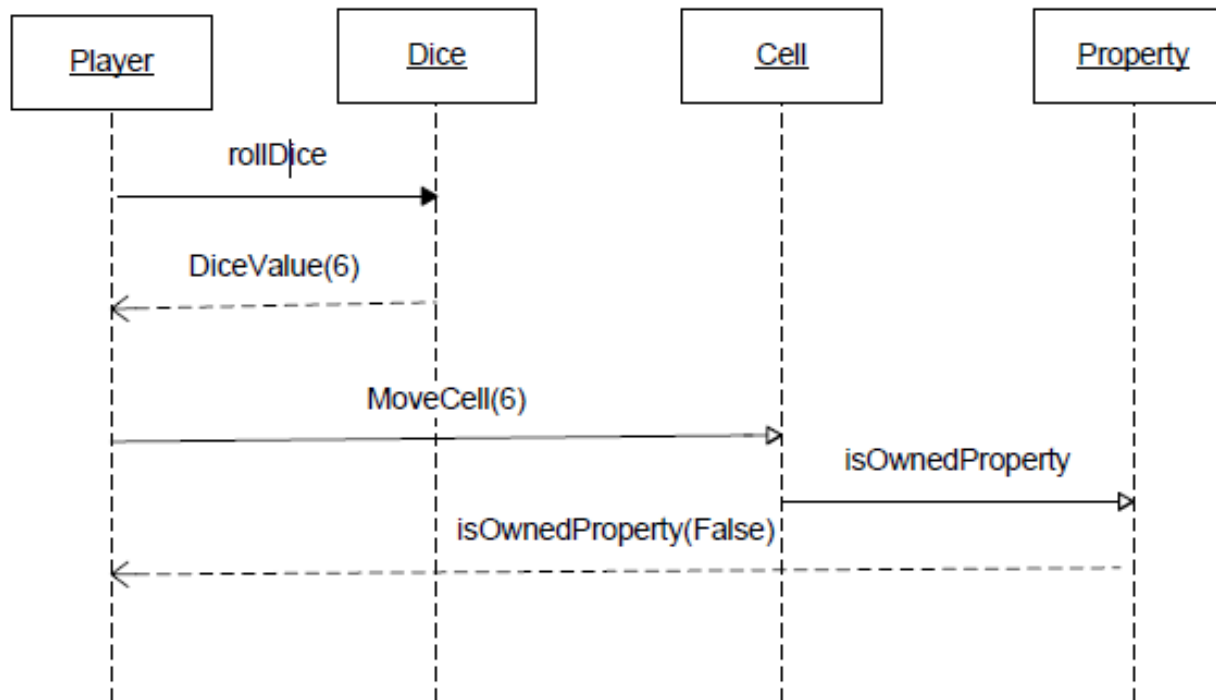
Example

A player rolls the dice and gets a 6. The player moves 6 cells. The player lands on a cell that is an un-owned property. The player's turn is over.

Example

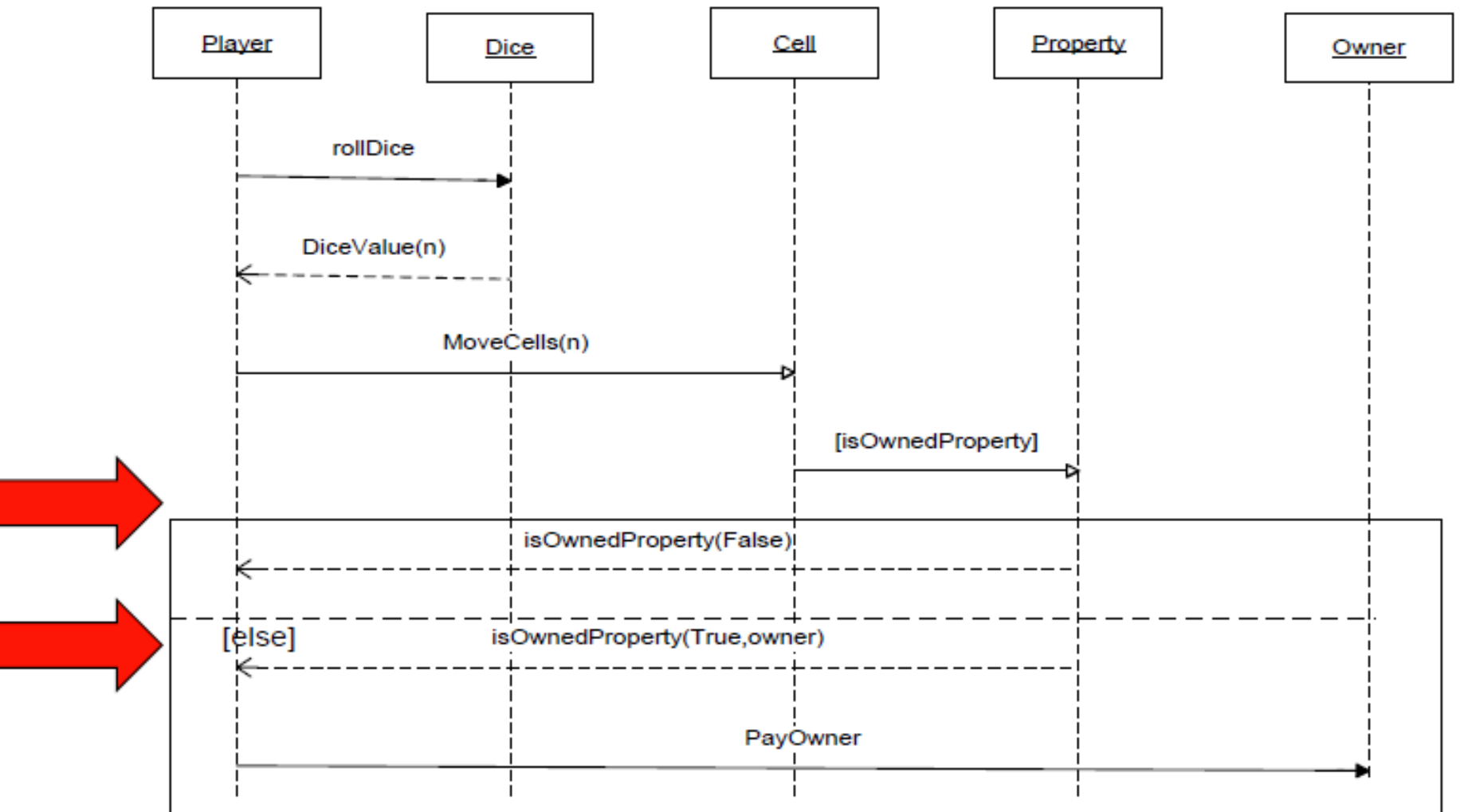
A **player** rolls the **dice** and gets a 6. The player moves 6 **cells**. The player lands on a cell that is an un-owned **property**. The player's turn is over.

Not all nouns become objects such as "turn"

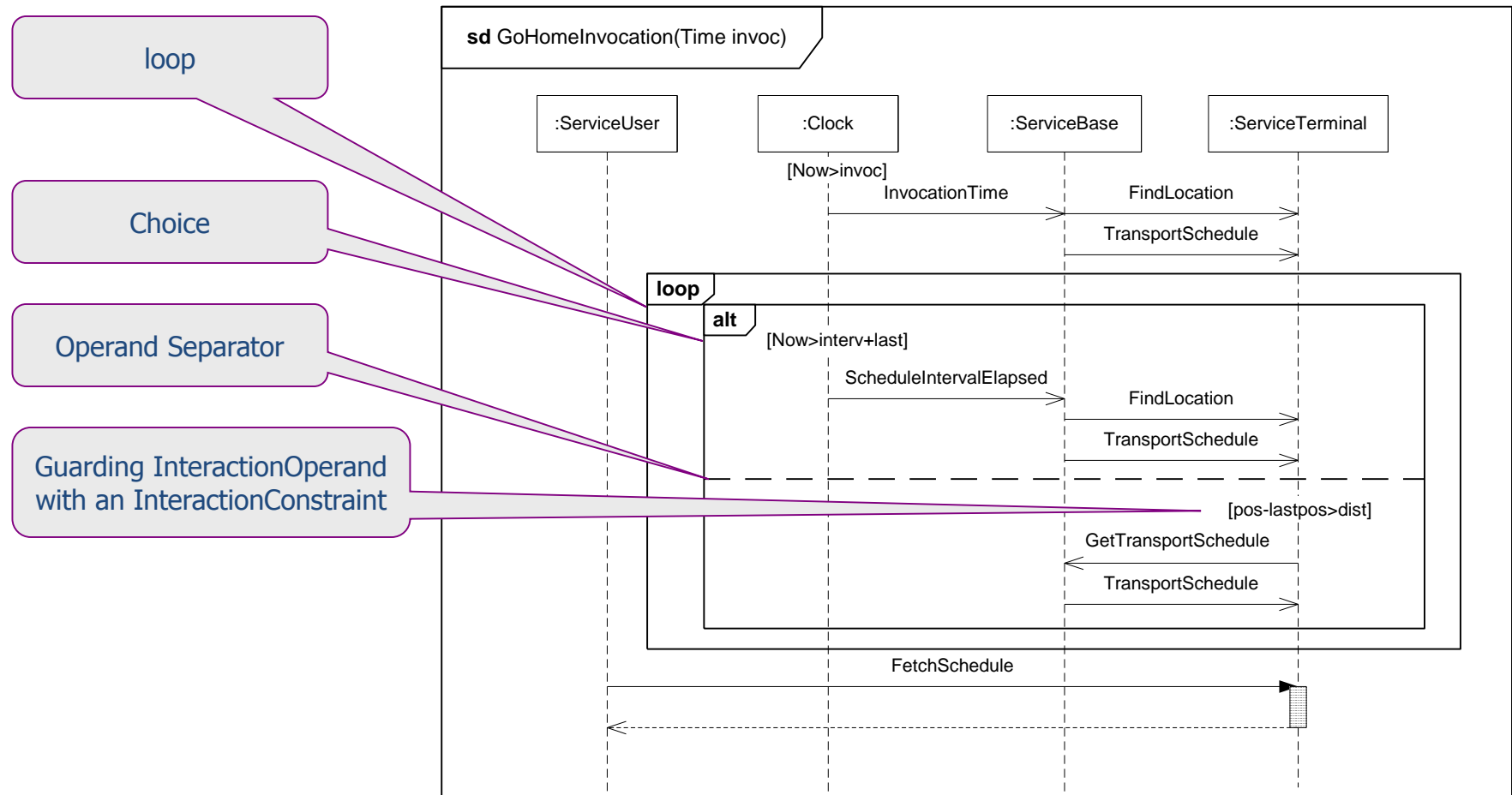


Conditional Logic

1. If the player lands on a cell that is an un-owned property, the player's turn is over
2. If the player lands on a cell that is owned, the player must pay rent to the owner of the property
3. Then, the player's turn is over.



More Complicated Sequence Diagram



UML class diagram

UML class diagram

When to use

- requirement analysis and architecture design
- represent classes and relationships of classes

Key elements

- class
- attributes
- operations
- relationships among classes

UML class diagram – class and object

“ A class is a description of a set of objects that share the same attributes, operations, relationships, and semantics.” (UML user guide, 1999)

OOD perspective: A class represents a solution concept. (Think Java/C++ classes)

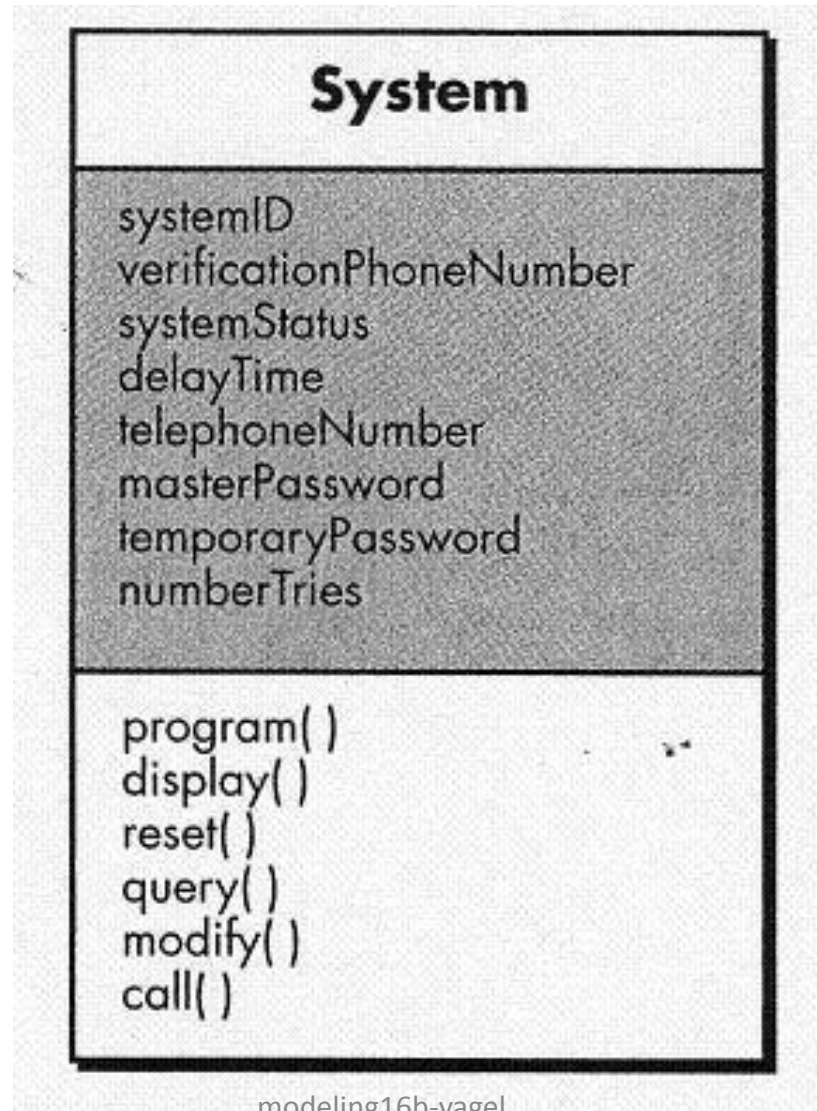
Object is an instance of the class

Class in UML

A class encapsulates state (*attribute*) and behavior (*operations*).

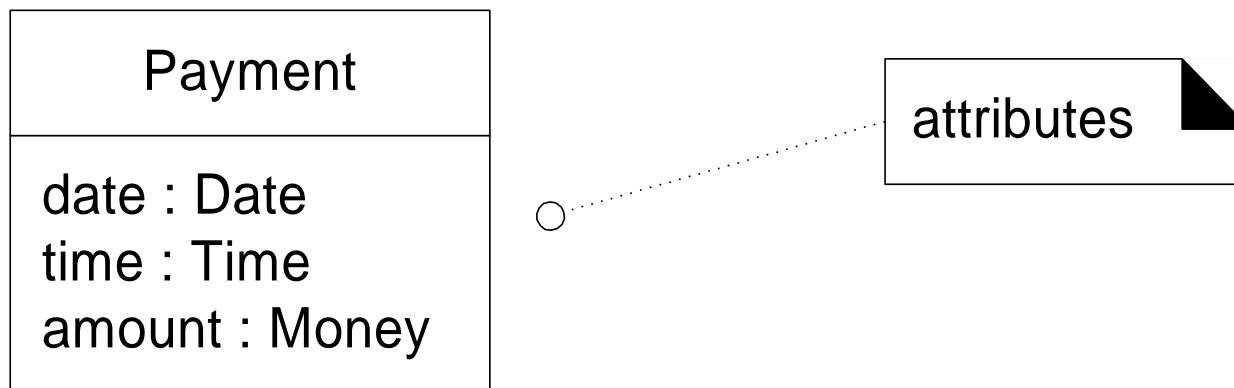
- Each attribute has a *type*.
- Each operation has a *signature*.
- The class name is the only mandatory information.

Class in UML



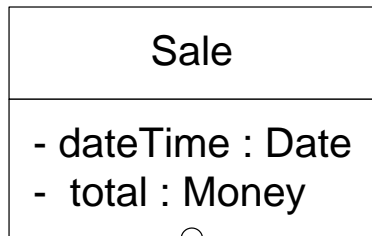
UML class diagram – attributes

- attribute: a named property
- each object has a value
- “simple” primitive types

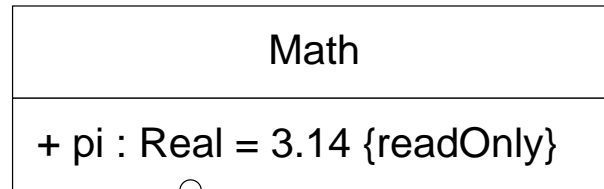


Syntax for Attributes in UML

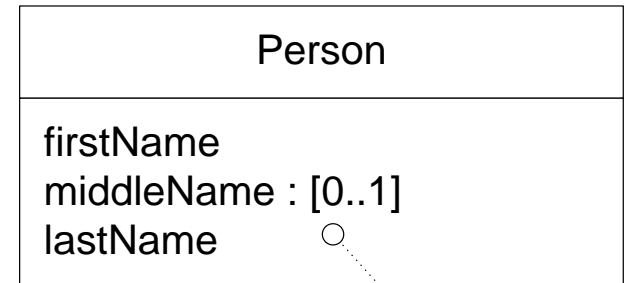
[visibility] name [multiplicity] [: type] [= initial-value]



Private visibility
attributes



Public visibility readonly
attribute with initialization

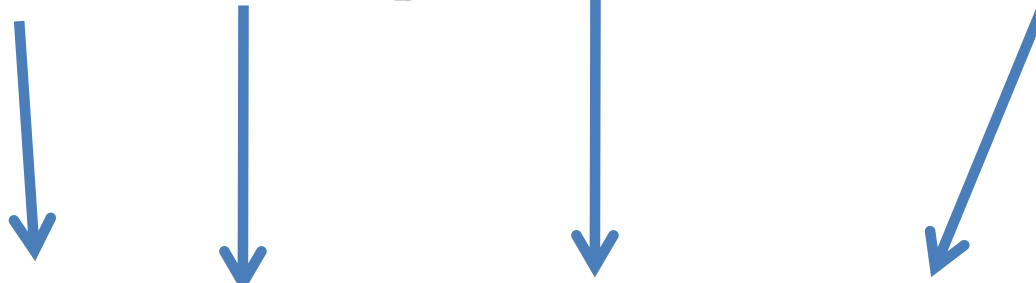


Optional value

Operations:

Syntax

[visibility] name [(parameter-list)] [: return type]



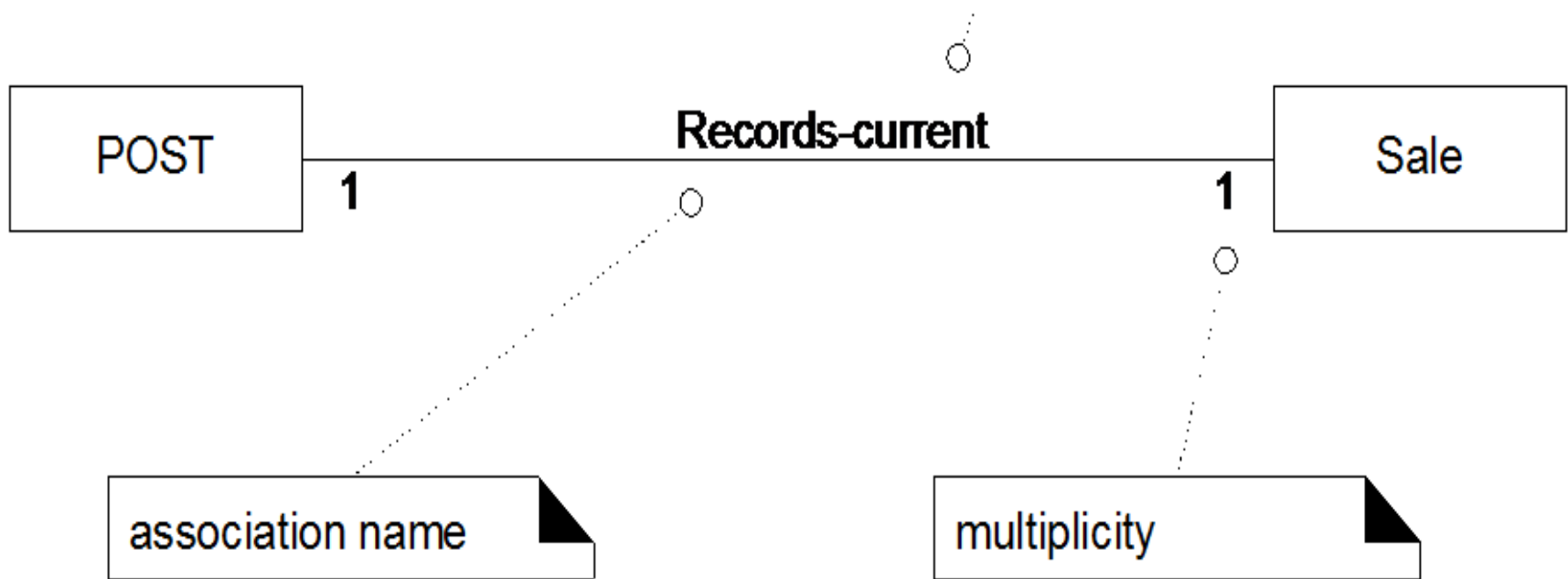
+ balanceOn (date: Date) : Money

Class Relationships

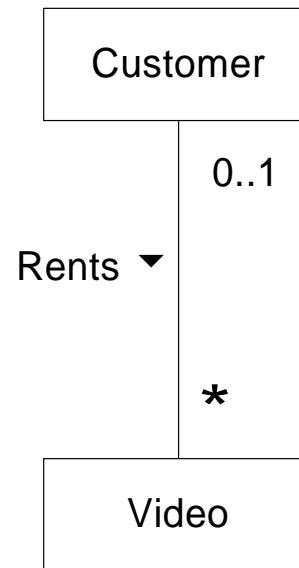
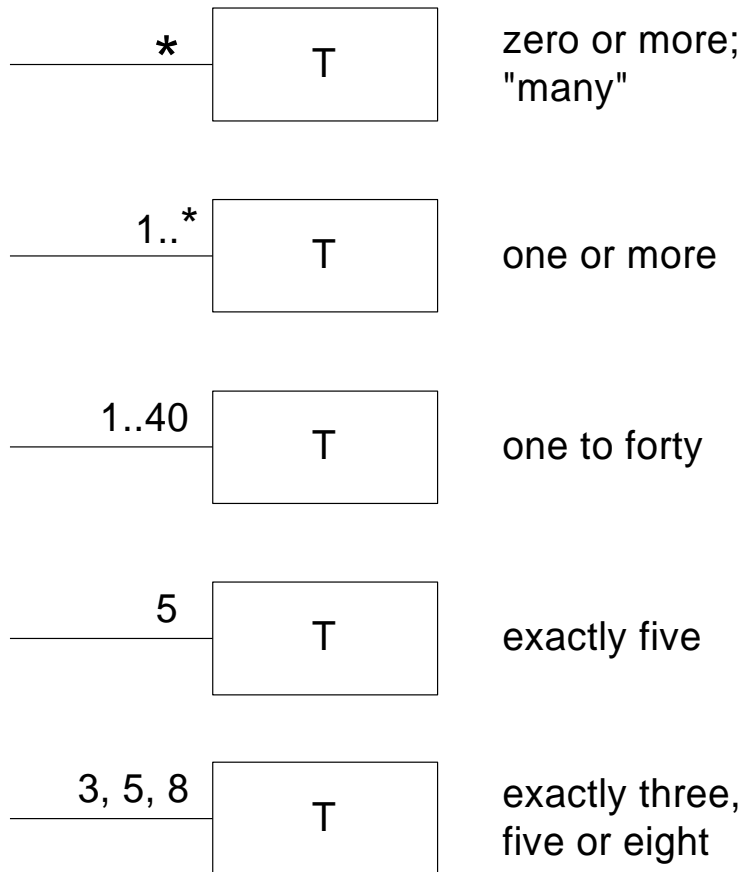
- Association: static relationship shared among the objects of two classes
 - Aggregation
 - Composition
- Generalization: generalization/specialization class level structures

Association

- Solid line from the source class to the target class (arrow is optional)



Multiplicity



One instance of a Customer may be renting zero or more Videos.

One instance of a Video may be being rented by zero or one Customers.

Modeling Associations

- If an object is part of another object and there is no existence dependency between the objects, model as an aggregation
- If the part is existentially dependent on the whole, model as a composition
- If there is a conceptual relationship between two peer objects, model as a general association

Aggregation

- If the association conveys the information that one object is part of another object (“has-a”), but their lifetimes are independent (they could exist independently).
- Aggregation is unidirectional, unlike associations in which classes have equal status
- There is a container and one or more contained objects.
 - For example, we may say that “a Department contains a set of Employees,” or that “a Faculty contains a set of Teachers.”

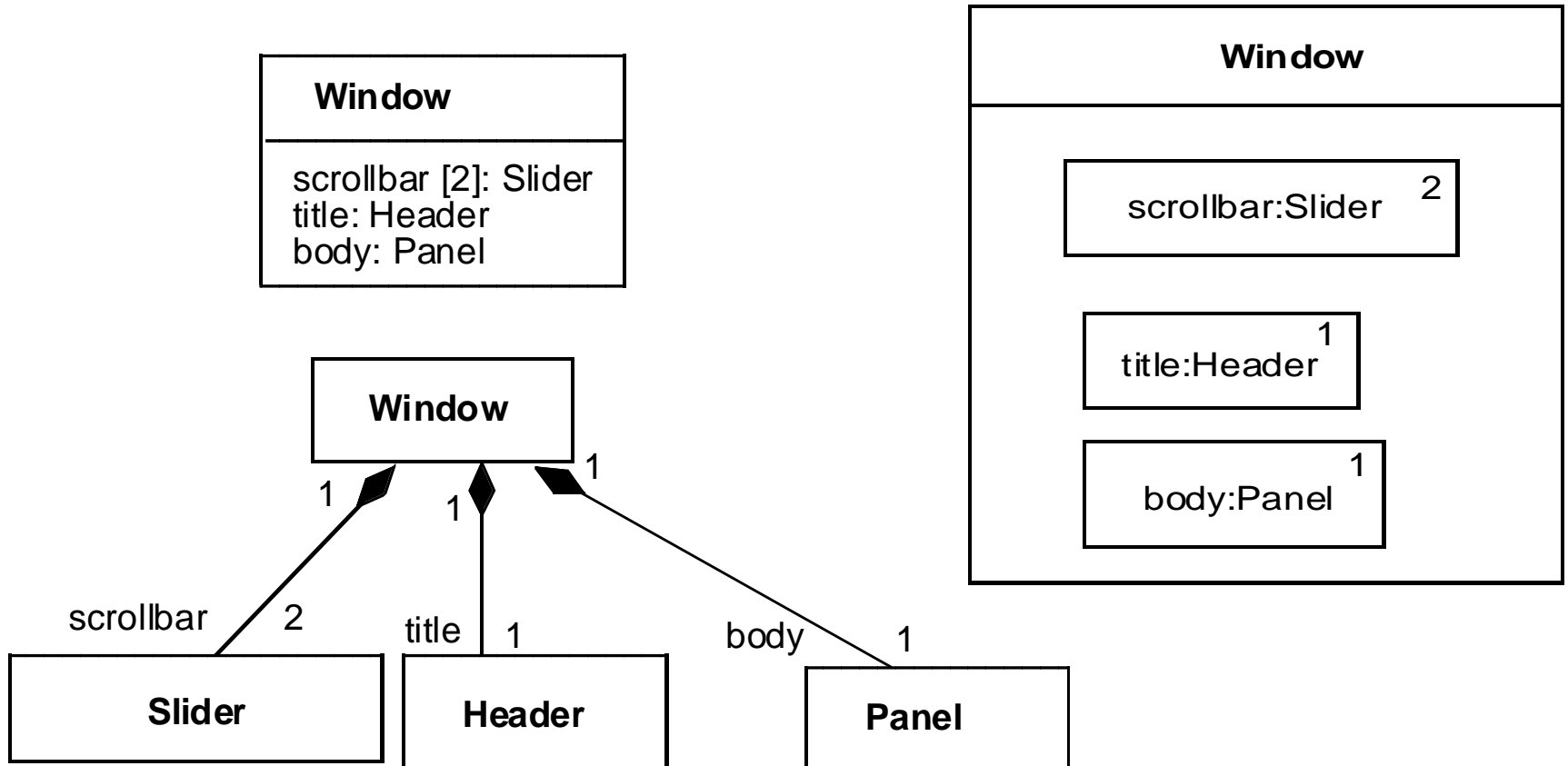


Composition

When an object is contained in another object, and it can exist only as long as the container exists and it only exists for the benefit of the container.

- An aggregation is a special form of association; composition is a stronger form of aggregation.
- multiplicity at whole end must be 1 or 0..1
- Both aggregation and composition are a part-whole hierarchy.

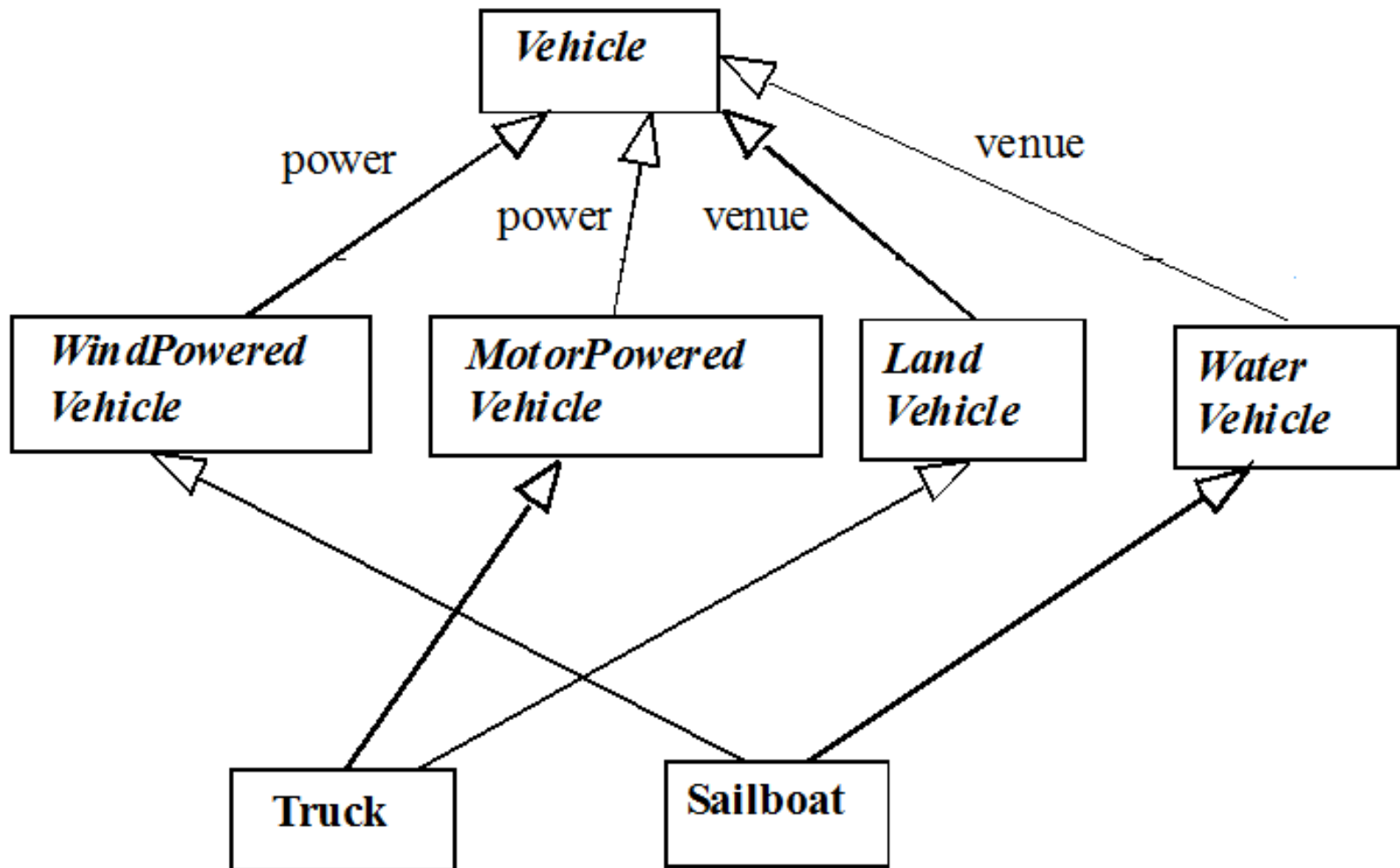
Composition



Generalization

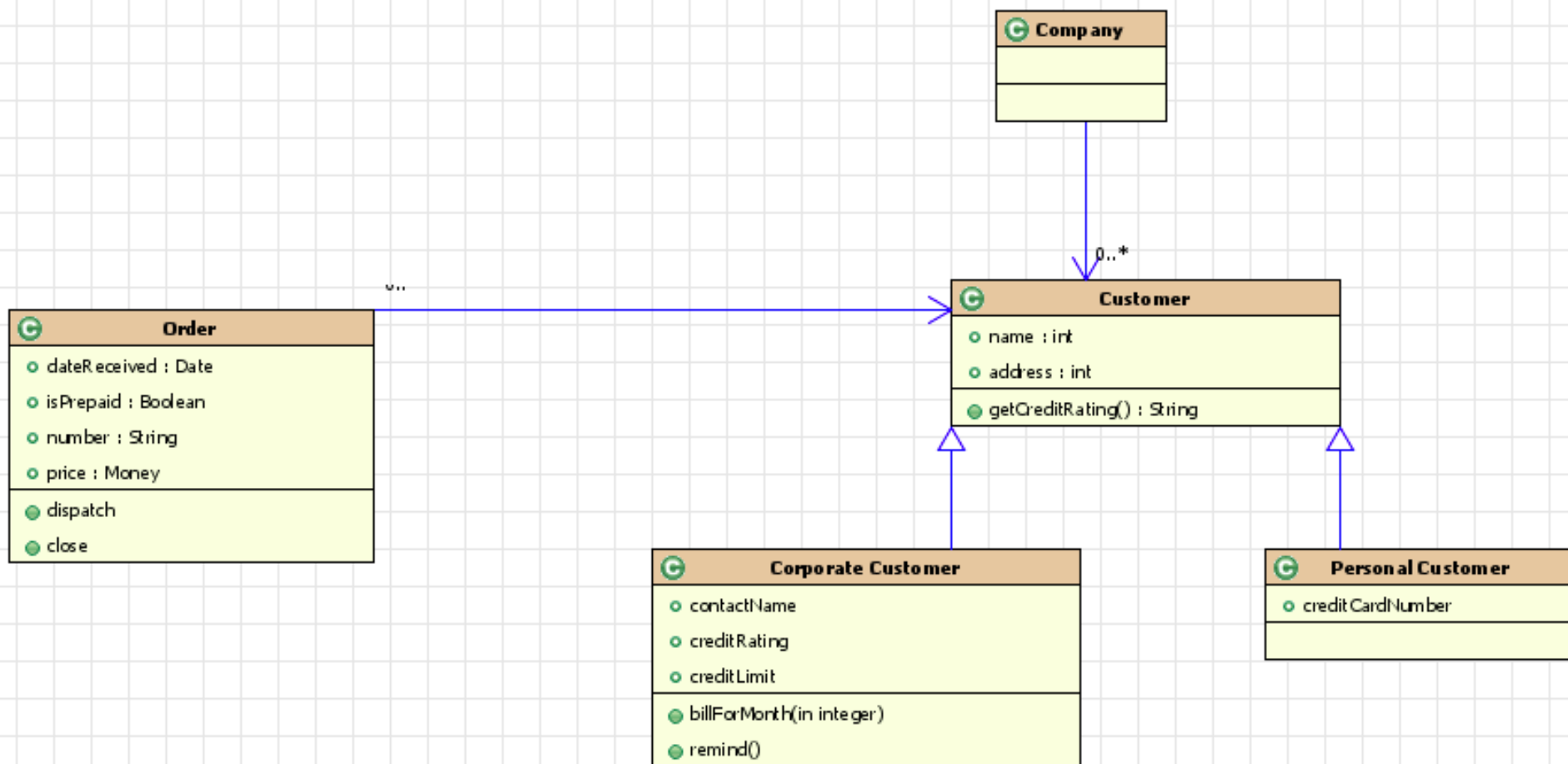
- A generalization (or specialization) is a relationship between a general concept and its specializations.
 - Example: *Mechanical Engineer* and *Aeronautical Engineer* are specializations of *Engineer*
- Generalization relationships denote inheritance between classes.
- The children classes inherit the attributes and operations of the parent class.

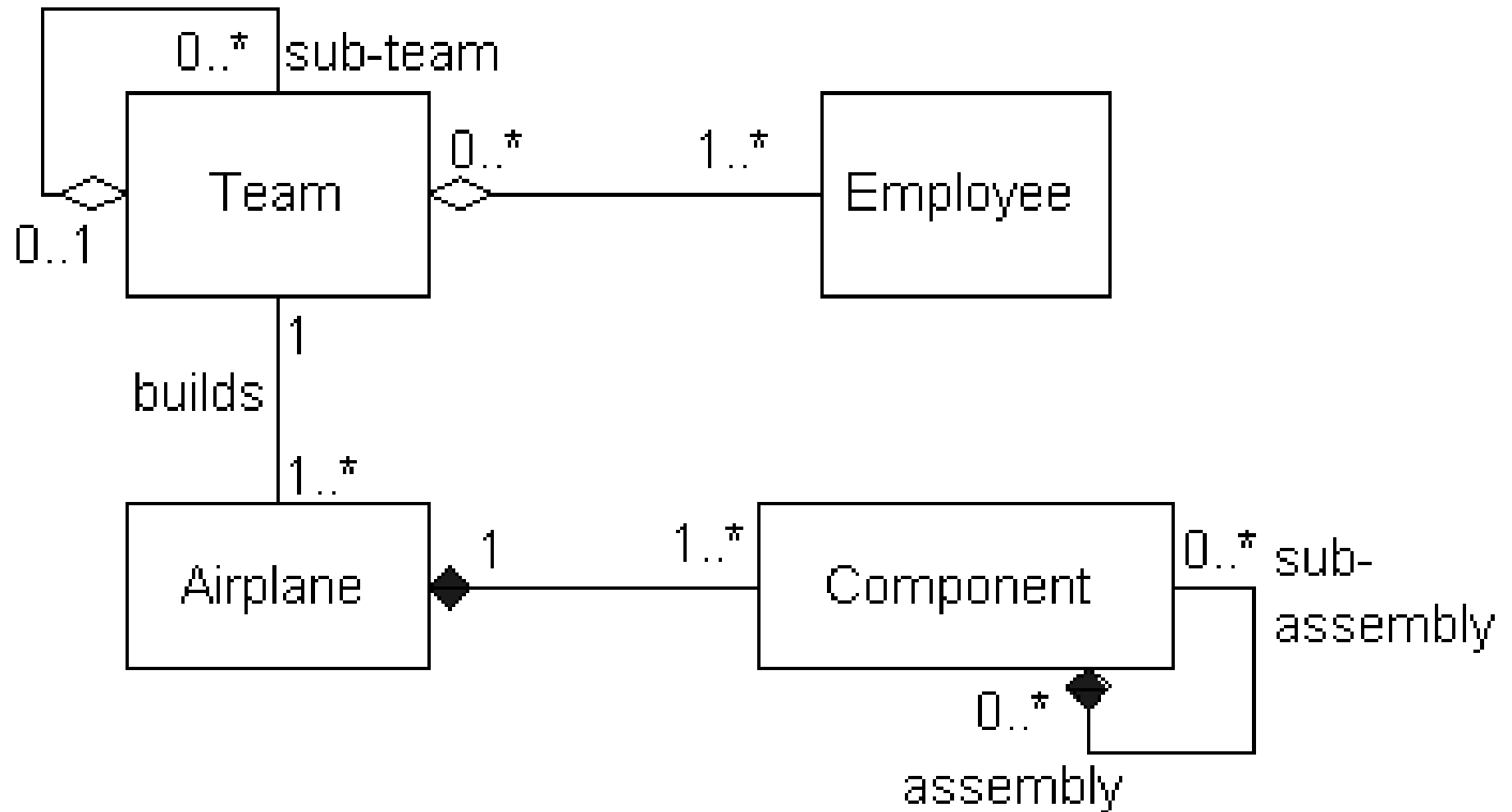
Generalization



Construct Class Diagram

Company XYZ has two types of customers, corporate customers and personal customers. All customers can place orders. Every order is placed by a customer.





UML Modeling – A Big Picture

Structure: class, component, deployment ...

...

Behavioral

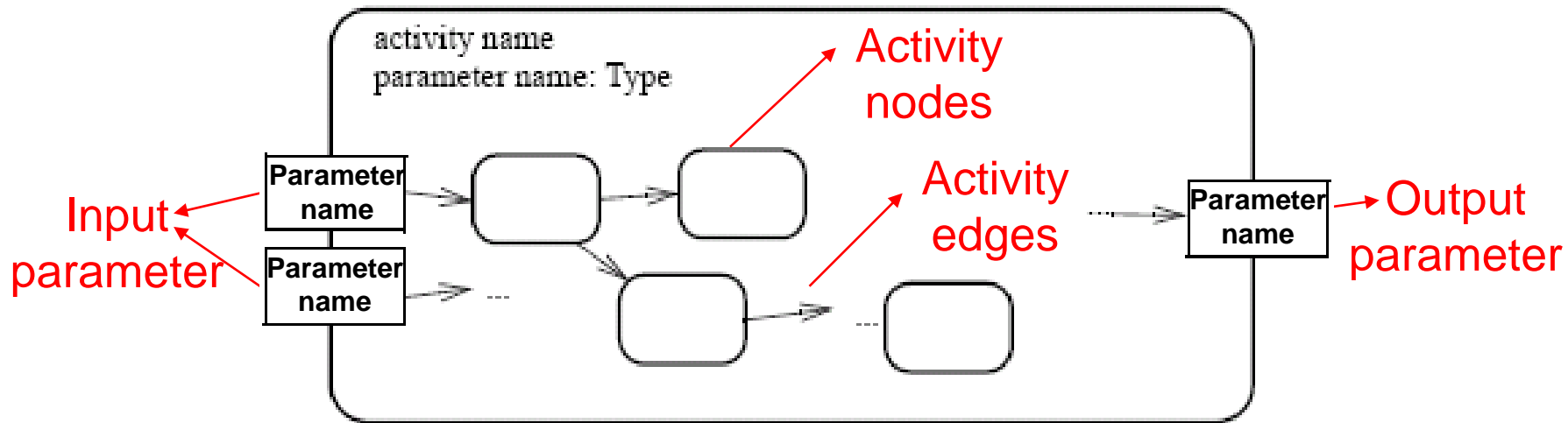
- Interaction: sequence
- Others: use case, activity, ...

UML Activity Diagram

- Capture activities/actions: unit of executable functionality
 - Mathematical functions
 - Calls to other behaviors
 - Processors of data
- When to use it
 - Business modeling
 - Dataflow
 - Concurrent program and parallel algorithm
- Model both control and data flow

Activities

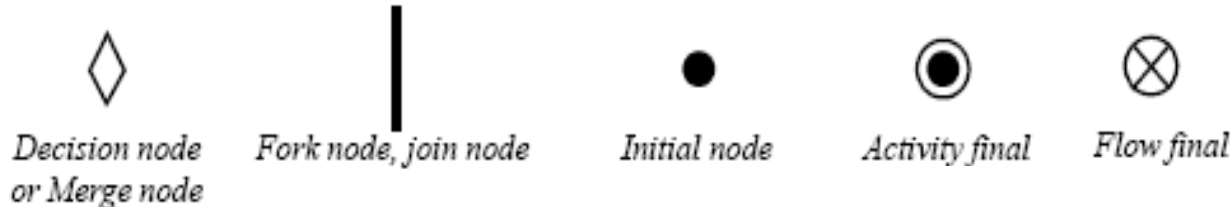
- Uses parameters to receive and provide data to the invoker



- An action can invoke an activity to describe its action more finely
- Token: information moving along an edge (data, object, control)

Activity nodes

- **Action nodes:** executable activity nodes
- **Parameter Nodes:** input/output parameters
- **Control nodes:** coordinate flows in an activity diagram between other nodes



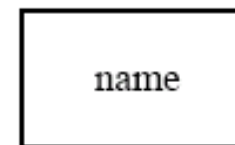
Activity final – entire activity; flow final – terminate a path

- **Object nodes:** data passing through the flow – *pin*



Buffer,
Datastore

modeling16b-yagel



{ordering = LIFO}

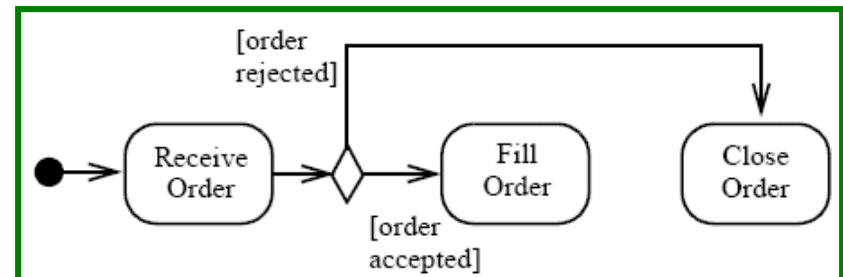
Control Nodes – initial node

- More than one initial node: a control token is placed in each initial node when the activity is started, initiating multiple flows
- More than one outgoing edge, only one of these edges will receive control



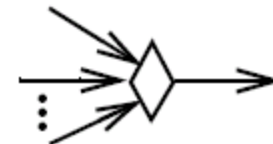
Control Nodes – decision node

- Choice of a flow
- Guards are specified on the outgoing edges or with the stereotype «decisionInput»
- There is also the predefined guard **[else]**, chosen only if the token is not accepted by all the other edges
- If all the guards fail, the token remains at the source object node until one of the guards accept it



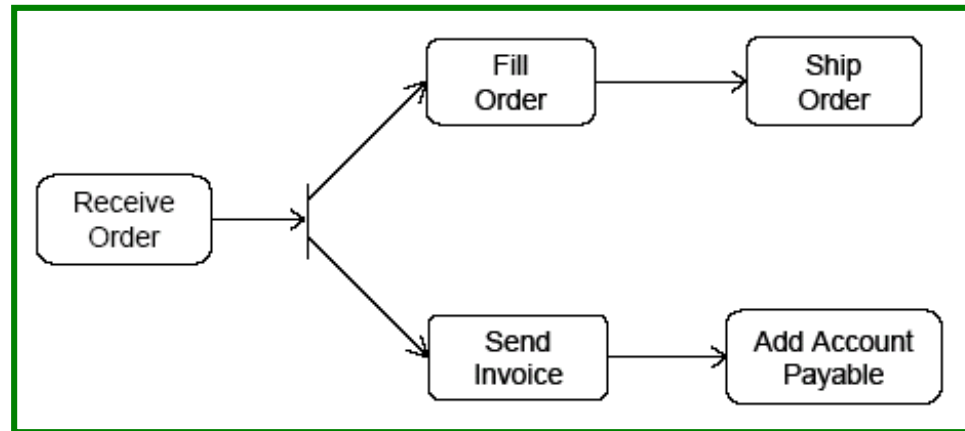
Control Nodes – merge node

- All controls and data arriving at a merge node are immediately passed to the outgoing edge
- There is no synchronization of flows or joining of tokens



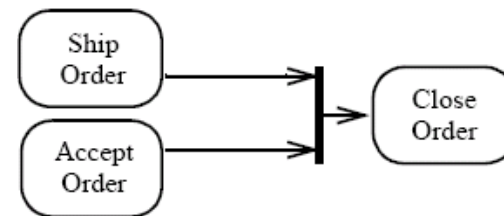
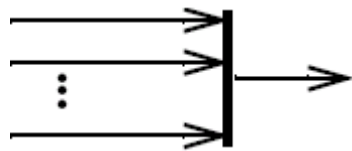
Control Nodes – fork node

- Split flows into multiple concurrent flows

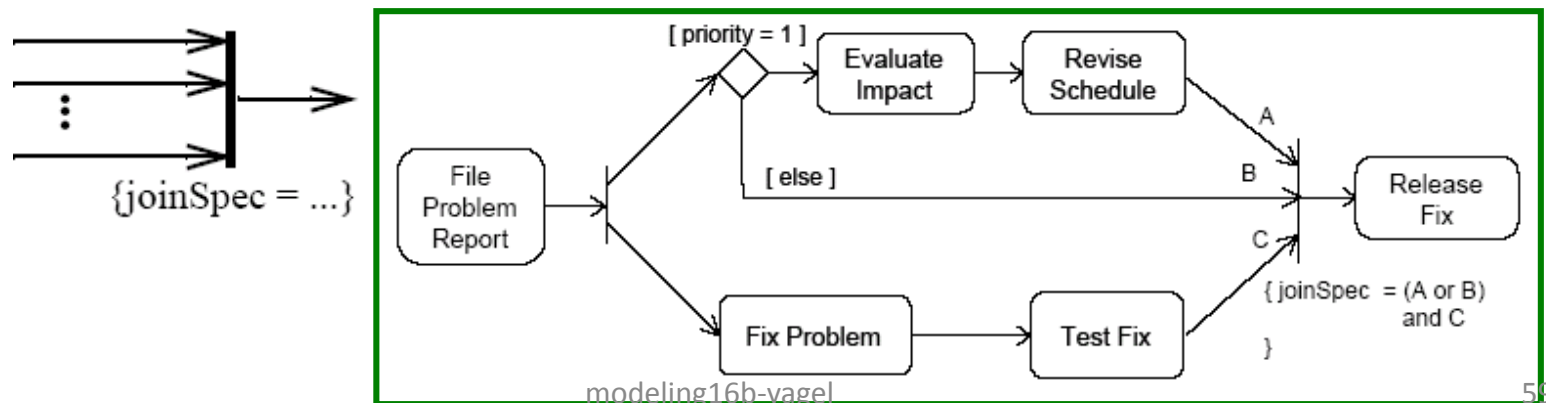


Control Nodes – join node

- Join nodes synchronize multiple flows



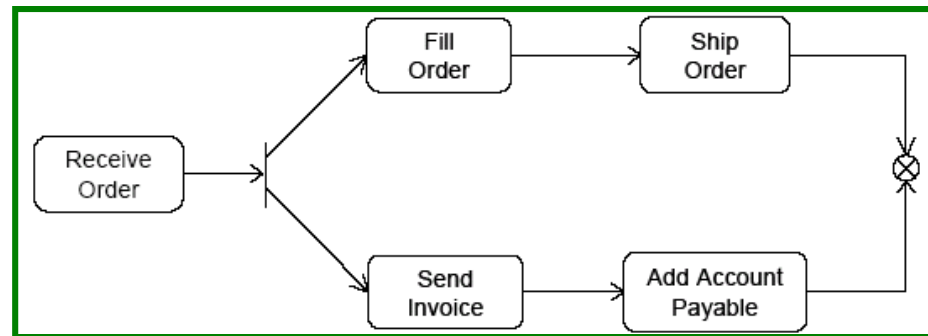
- Generally, controls or data must be available on every incoming edge in order to be passed to the outgoing edge, user can specify different conditions under which a join accepts incoming controls and data using a *join specification*



Control nodes – final nodes

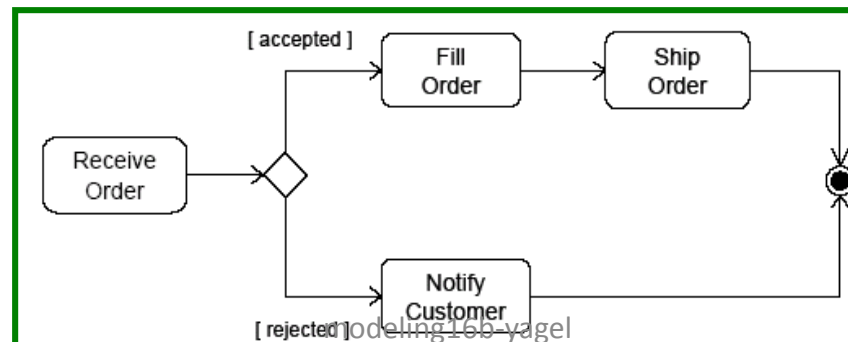
- **Flow final:**

- destroys the tokens that arrive into it
- the activity is terminated when all tokens in the graph are destroyed



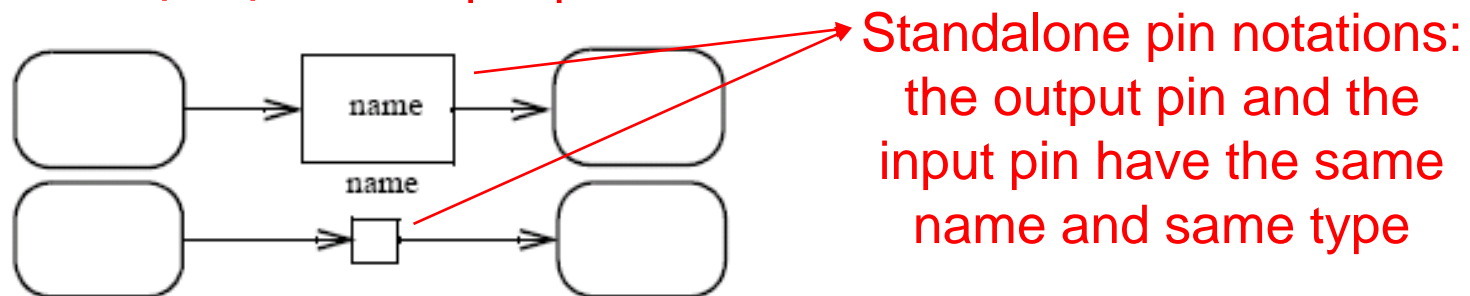
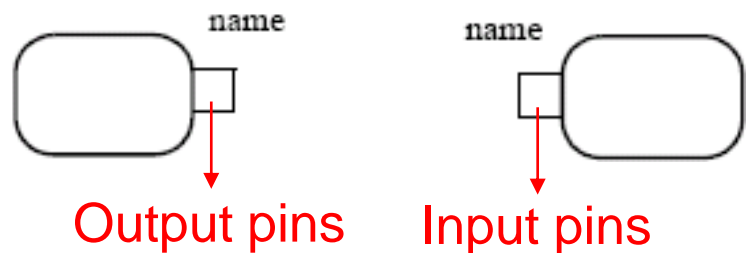
- **Final node:**

- the activity is terminated when the first token arrives



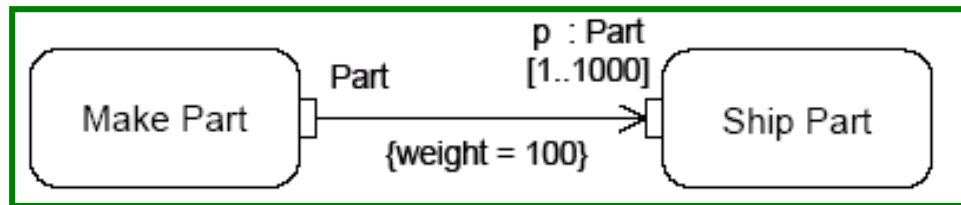
Components - Pin

- Input and output for an activity

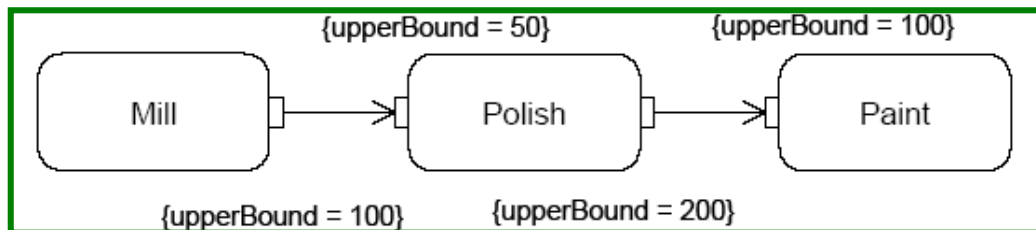
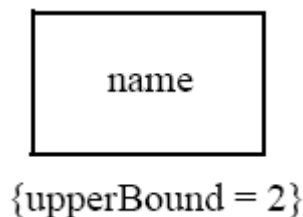


Multiplicities and upperBound

- ***Multiplicities***: the minimum (≥ 0) and maximum number of values each pin accepts or provides at each invocation of the action: when is available the minimum number of values, the action can start

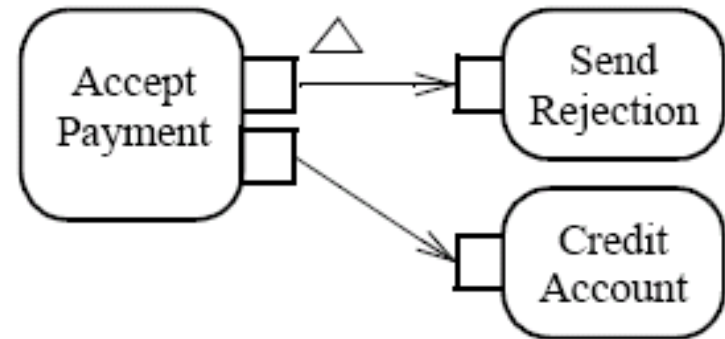
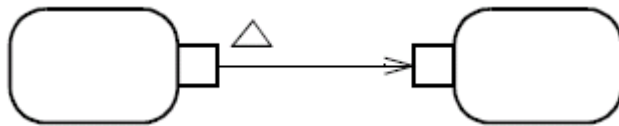


- ***UpperBound***: the maximum number of values that an object node can hold: at runtime, when the upper bound has been reached, the flow is stopped (buffering)

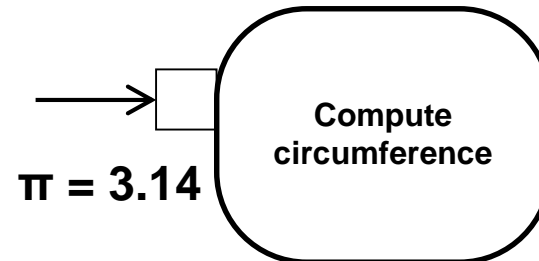
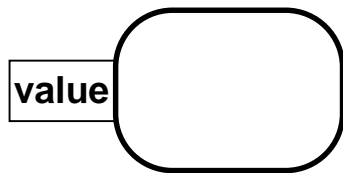


Special Pins

Exception Pin

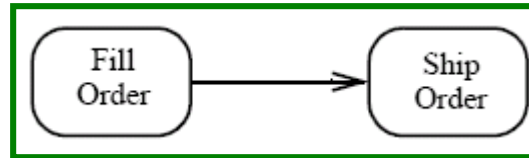


Value Pin

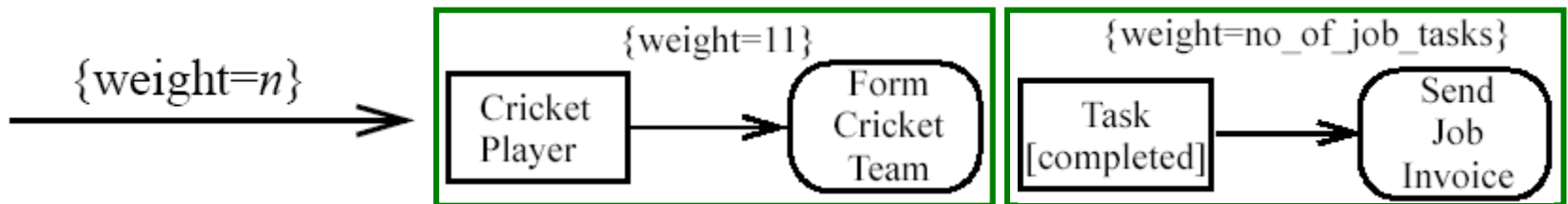


Activity edges

- Directed



- Weight:** determines the minimum number of tokens that must traverse the edge at the same time

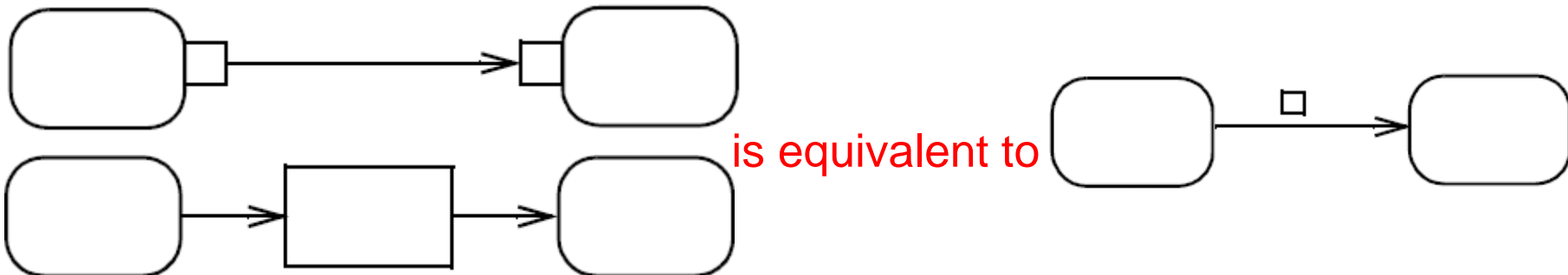


Two Types of Edges

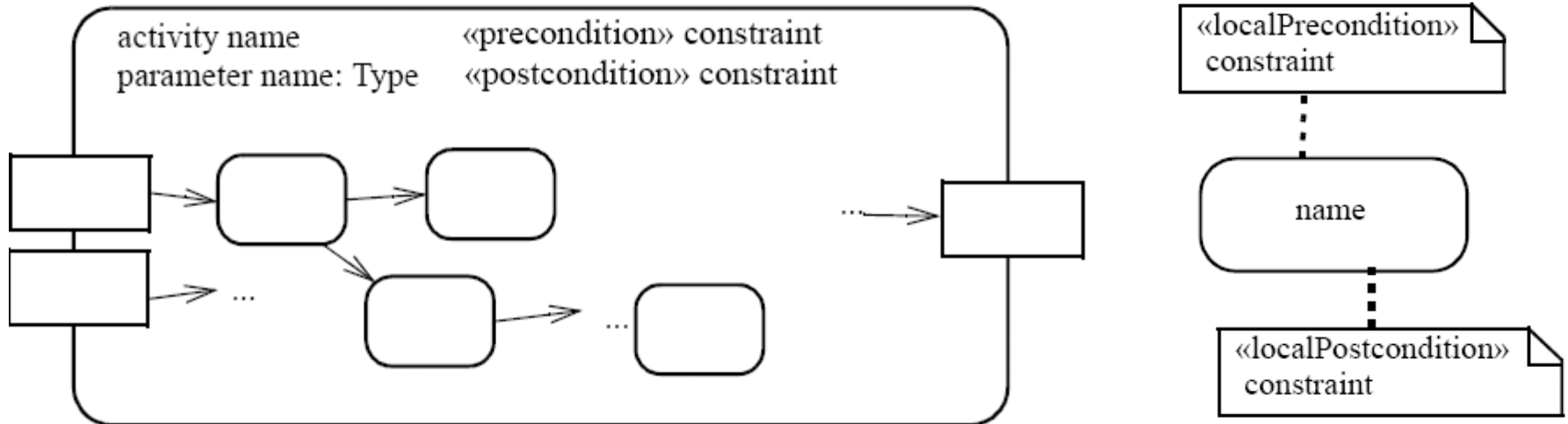
Control flow edge - is an edge which starts an activity node after the completion of the previous one by passing a control token



Object flow edge - models the flow of values to or from object nodes by passing object or data tokens



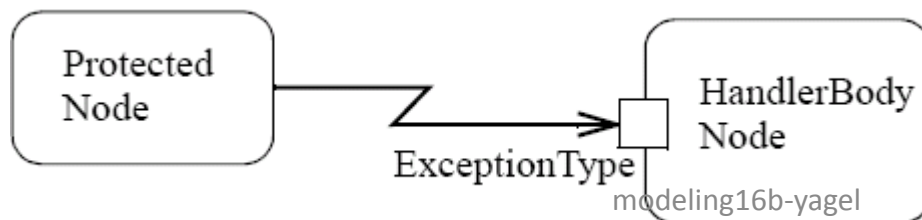
Pre & post condition



- Can be referred to an activity or to an action (local condition)
- UML also does not define what the runtime effect of a failed pre/post condition should be (error that stops execution, warning, no action)

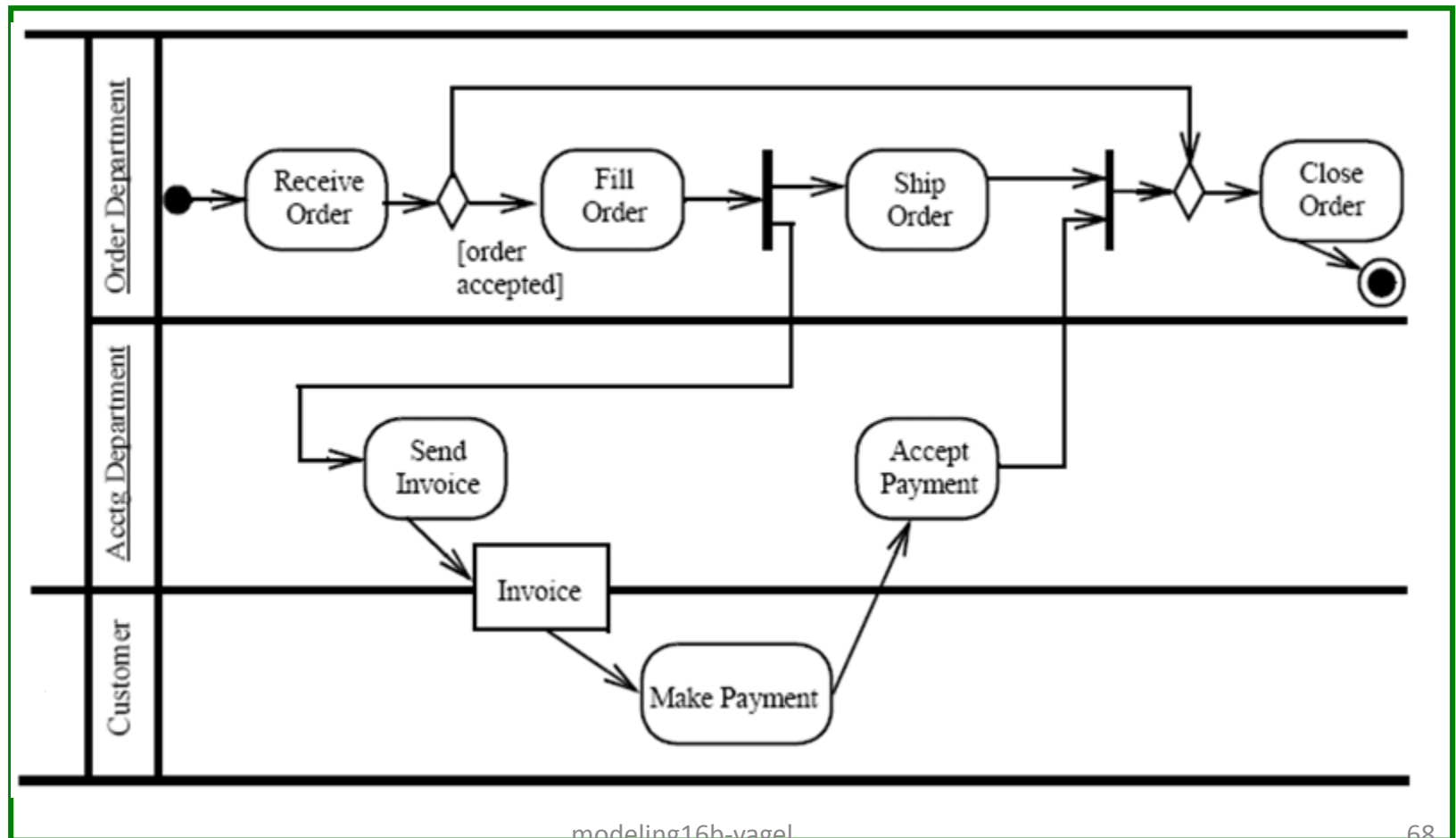
Exception

- Exception handler - specifies the code to be executed when the specified exception occurs during the execution of the protected node
- If the exception is not caught, it is propagated
- If the exception propagates to the topmost level of the system and is not caught, the behaviour of the system is unspecified; profiles may specify what happens in such cases



Activity Partition

- Partitions divide the nodes and edges for identifying actions that have some characteristics in common

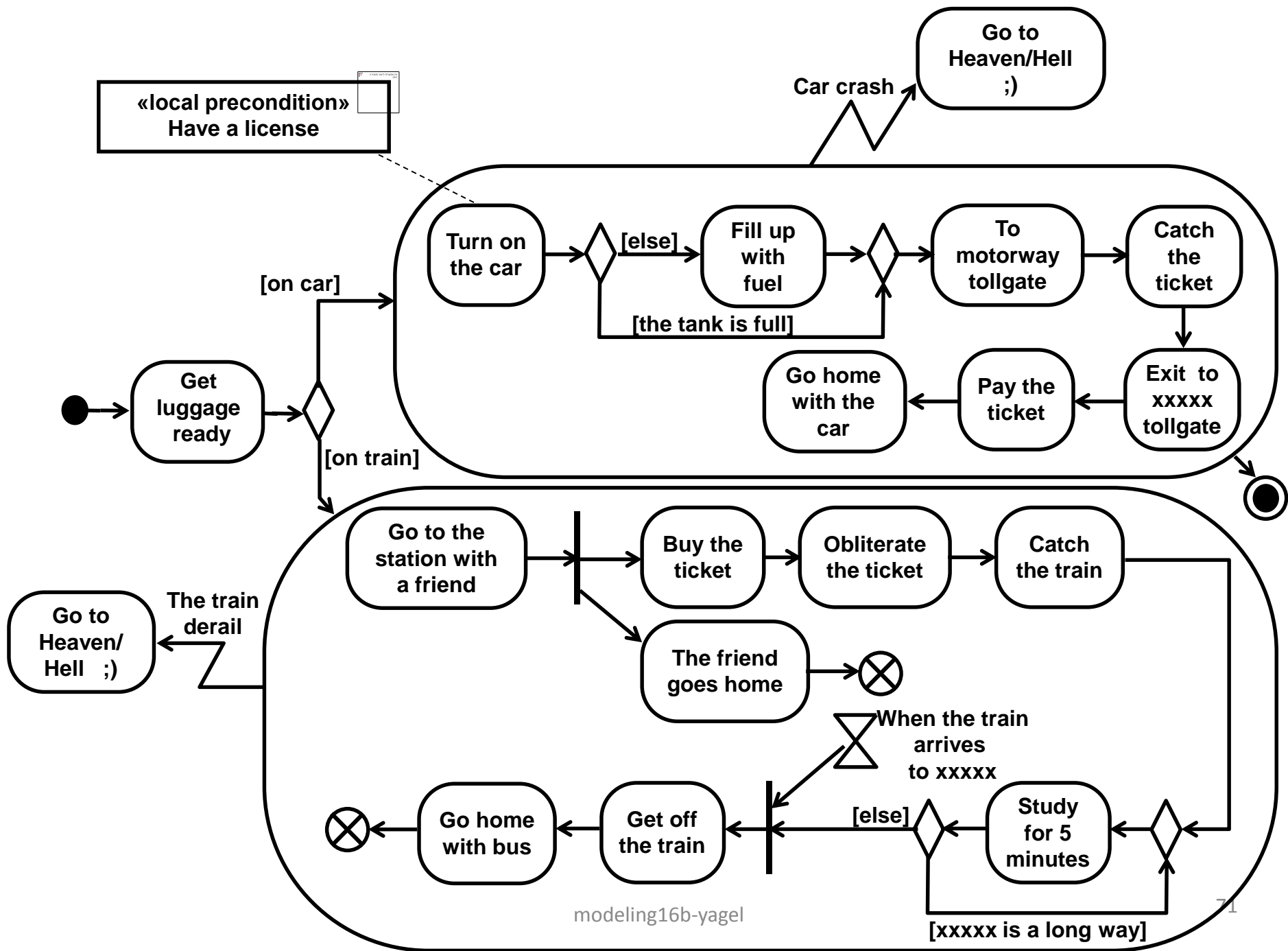


More

- History Point 
- Interaction with other diagrams, e.g. use-case

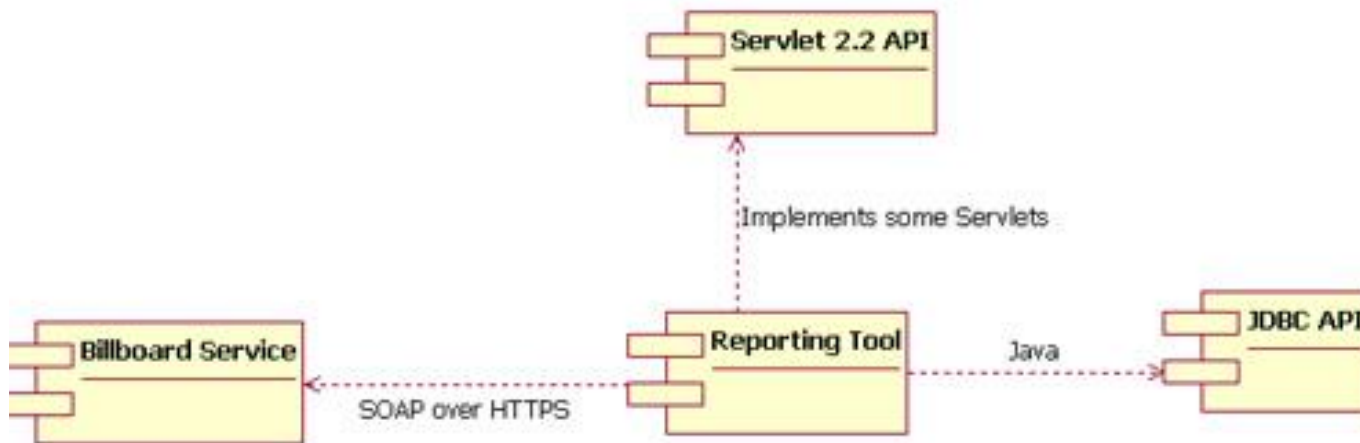
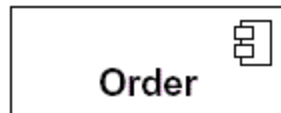
Example – Travel to xxxxx

- Go by car
- Go by train



Quick Review of Other UML Diagrams

- Component Diagram: interface and dependency
the structural relationships between the components of a system



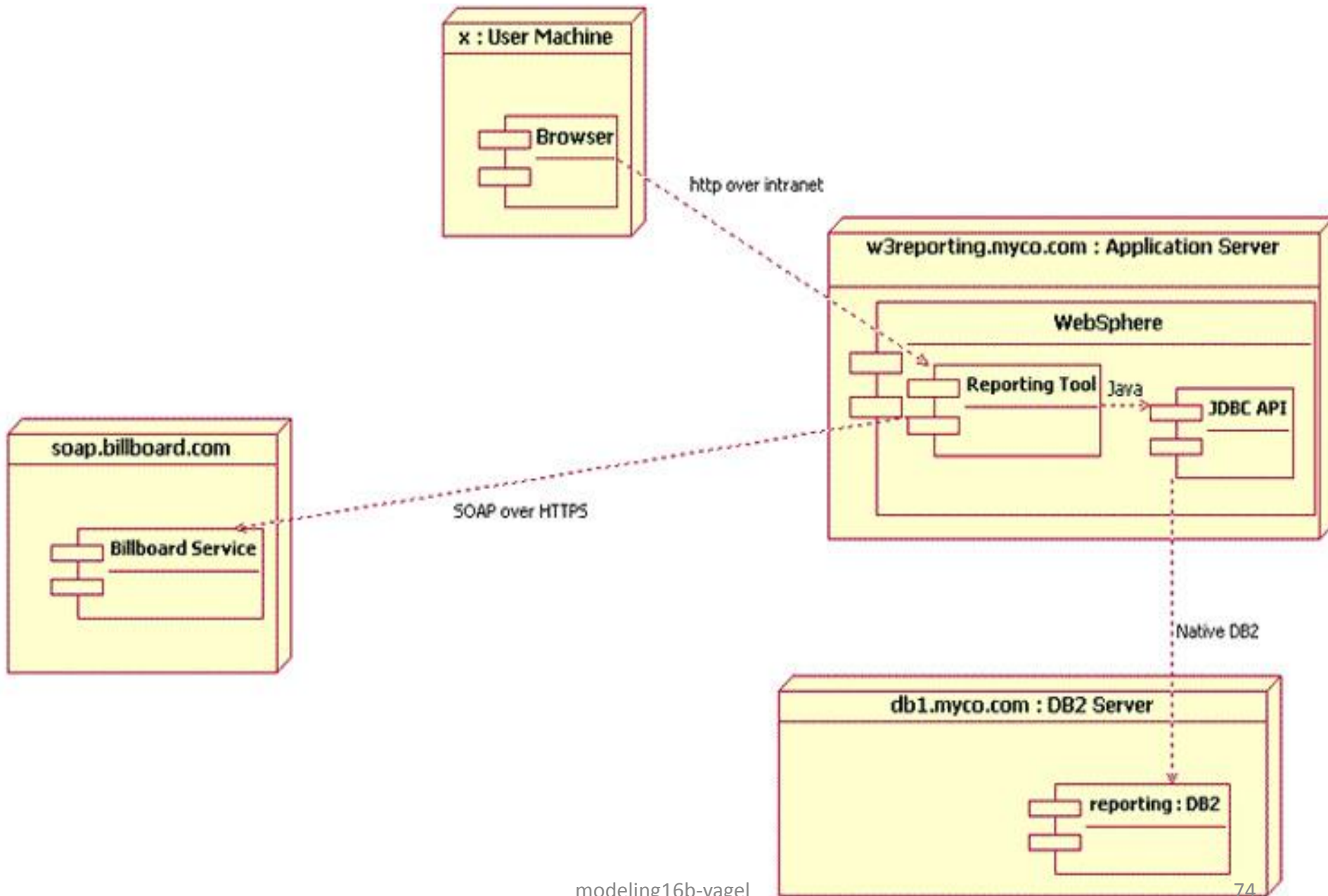
Deployment Diagram

Where the different components of the system will physically run and how they will communicate with each other.

- Deployment team/system staff

A node represents either a physical machine or a virtual machine node (e.g., a mainframe node).

Extension of Component Diagram

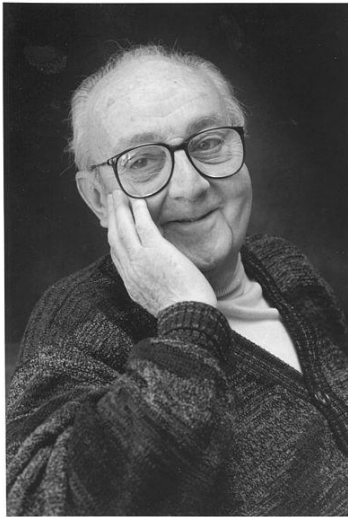


תרגיל 1- חלק ב'

- מידול עם כל אחד מסוגי הדיאגרמות שראינו
- הגשה עד ערב ההרצאה הבאה

סיכום

“all models are wrong but some are useful” G. E. P. Box



• UML

– תרחיש שימוש

– דיאגרמות נוספות

• בפעם הבאה

– סיכום UML

– OCL (בדרך למפרטים פורמליים)