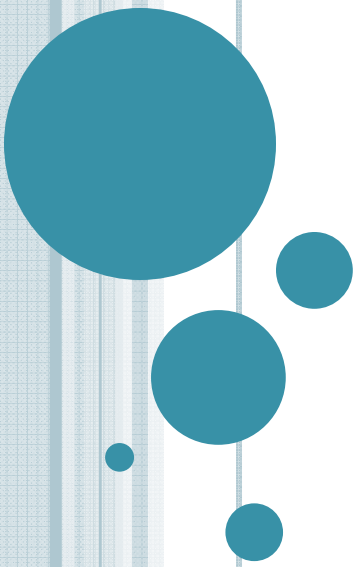
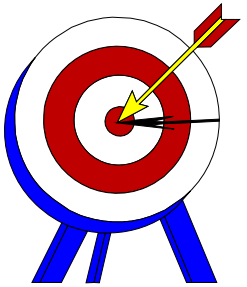
OBJECT-ORIENTED SYSTEMS DEVELOPMENT: USING THE UNIFIED MODELING LANGUAGE



Unified Modeling Language

GOALS



- Modeling
- Unified modeling language
  - Class diagram.
  - Use case diagram
  - Interaction diagrams
    - Sequence diagram
    - Collaboration diagram
    - State chart diagram
    - Activity diagram
    - Implementation diagrams
      - Component diagram
      - Deployment diagram
- N.B. In this chapter we look at UML notations and diagrams

INTRODUCTION

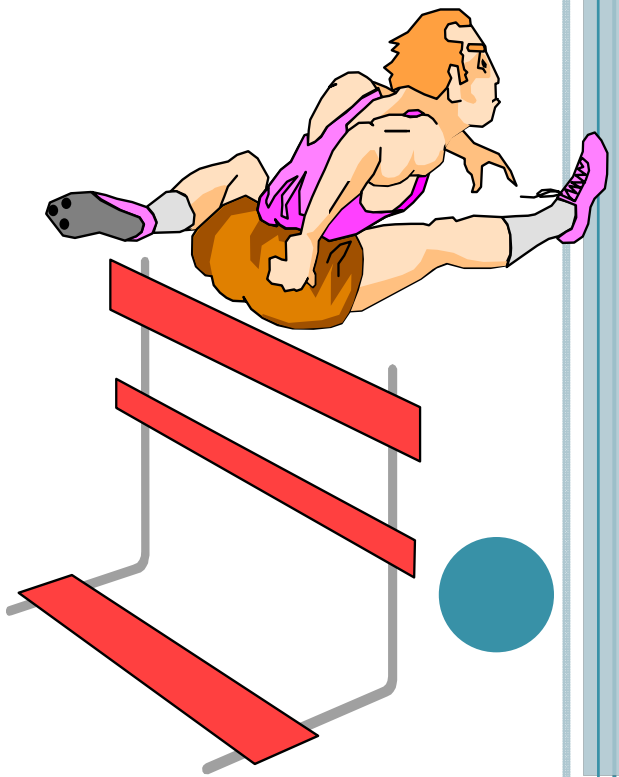
- A *model* is an abstract representation of a system, constructed to understand the system prior to building or modifying it
- Most of the modeling techniques involve graphical languages
- Modeling is also an iterative process- as a model progresses from analysis to implementation, more detail is added
- Modeling is simplified representation of reality
- Modeling provides means for conceptualization and communication of ideas in precise and unambiguous form

STATIC OR DYNAMIC MODELS

- Models can represent
  - static or
  - dynamic situations

STATIC MODEL

- A static model can be viewed as a "snapshot" of a system's parameters at rest or at a specific point in time
- For example customer can have more than accounts
- They are used to represent static or structural aspects of the system
- Class diagram is example of static model



DYNAMIC MODEL

- Is a collection of procedures or behaviors that, taken together, reflect the behavior of a system over time
- It shows how the business objects interact to perform tasks
- For example, an order interacts with inventory to determine product availability
- A system can be described by first developing static model(structure of objects and their relationships to each other)
- Then changing the objects and relationships overtime
- Dynamic modeling is useful during the design and implementation phases of system development
  - UML interaction diagrams and activity models are examples of UML dynamic models

WHY MODELING?

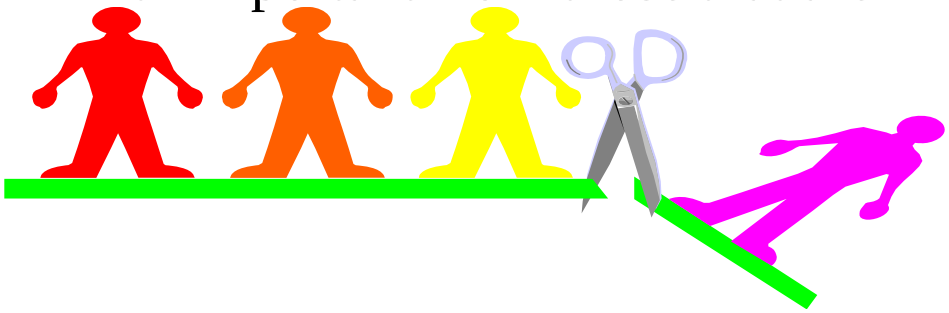
- Model is like blueprint of system
- Good models are necessary for complex system
- A modeling language must include:
  - Model elements
    - Fundamental modeling concepts and semantics
  - Notations
    - Visual rendering of modeling elements
  - Guidelines
    - Expression of usage within the trade

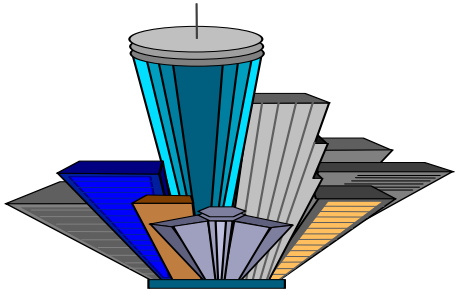
ADVANTAGES OF MODELING

- Models reduce complexity by separating those aspects that are unimportant from those that are important
- The use of visual notation to represent or model a problem can provide following benefits:
  - Clarity
    - To pick up errors from visual representations is far easy
  - Familiarity
    - Representation of model may be done similar as actual representation
  - Maintenance
    - We can make changes faster and fewer errors are likely to be introduced in the making changes
  - Simplification
    - Higher level representation provide more general construct, giving simplicity and conceptual understanding

ADVANTAGES OF MODELING

- Turban cites the following advantages:
  - Models make it easier to express complex ideas
  - For example, an architect builds a model to communicate ideas more easily to clients
  - Models enhance learning
  - The cost of the modeling analysis is much lower than the cost of similar experimentation conducted with a real system
  - Manipulation of the model (changing variables) is much easier than manipulating a real system
  - Models reduce complexity by separating those aspects that are unimportant from those that are important.



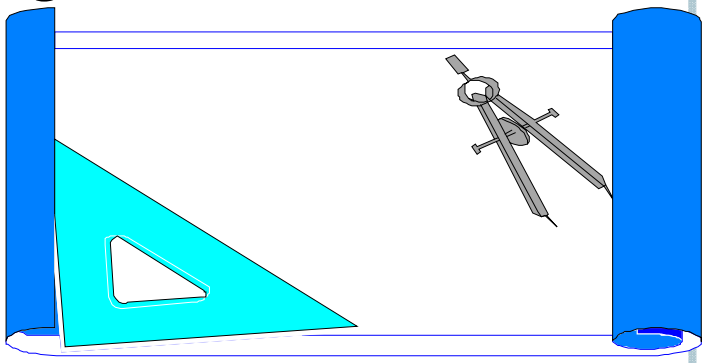


MODELING KEY IDEAS

- A model is rarely correct on the first try
- Always seek the advice and criticism of others
- Avoid excess model revisions, as they can distort the essence of your model. Let simplicity and elegance guide you through the process

THE UNIFIED MODELING LANGUAGE (UML)

- The unified modeling language (UML) is a language for specifying, constructing, visualizing, and documenting the software system and its components.
- UML is a graphical language with set of rules and semantics
- Rules and semantics are expressed in English called as Object Constraint Language (OCL)
- OCL is a specification language for specifying properties of the system
- UML has tight mapping with OO languages



CONT..

- Goal of unification was to keep everything simple:
  - Take away elements from Booch, OMT and OOSE methods that do not work and add elements from other methods that are more effective
  - To invent new methods only when existing solutions was unavailable
  - Other diagram like DFD is not included as they do not fit into OO paradigm

GOALS OF UML

- Provide users expressive modeling language so that they can develop and exchange meaningful models
- Provide extensibility and specialization mechanism to extend core concept
- Be independent of programming language or development processes
- Provide formal basis for understanding modeling language
- Encourage growth of OO tools
- Support higher level development concepts
- Integrate best practices and methodologies
- In this book UML ver 1.1 by Booch, Jacobson and Rumbaugh

UML DIAGRAMS

The UML defines nine graphical diagrams:

1. Class diagram (static)
2. Use-case diagram
3. Behavior diagrams (dynamic):
   - 3.1. Interaction diagram:
     - 3.1.1. Sequence diagram
     - 3.1.2. Collaboration diagram
   - 3.2. Statechart diagram
   - 3.3. Activity diagram
4. Implementation diagram:
   - 4.1. Component diagram
   - 4.2. Deployment diagram

- Choice of models and diagrams depends on problem and their solution

UML CLASS DIAGRAM

- Also referred to as object-modeling, the UML class diagram is the main static analysis diagram
- Class diagrams show the static structure of the model
- Class diagram is collection of static modeling elements, such as classes and their relationships
- Object modeling is a technique to represent(map) the logical objects in the real world as actual objects in the system.

CONT..

- To determine objects required in system ask following questions:
  - What are the goals of the system?
  - What must the system accomplish?
- Main task of object-modeling is to graphically show,
  - what each object will do in the problem domain, describe the structure
  - Describe the structure (e.g. class hierarchy or part-whole)
  - And relations among the objects by visual notations

CLASS NOTATIONS

- Class is drawn as rectangle with three components separated by horizontal lines
- Top on holds class name, middle has attributes and properties while last holds list of operations
- Either or both the attributes and operation components may be suppressed
- A convention of UML is to use italic font for abstract classes and normal (usually roman) font for concrete classes

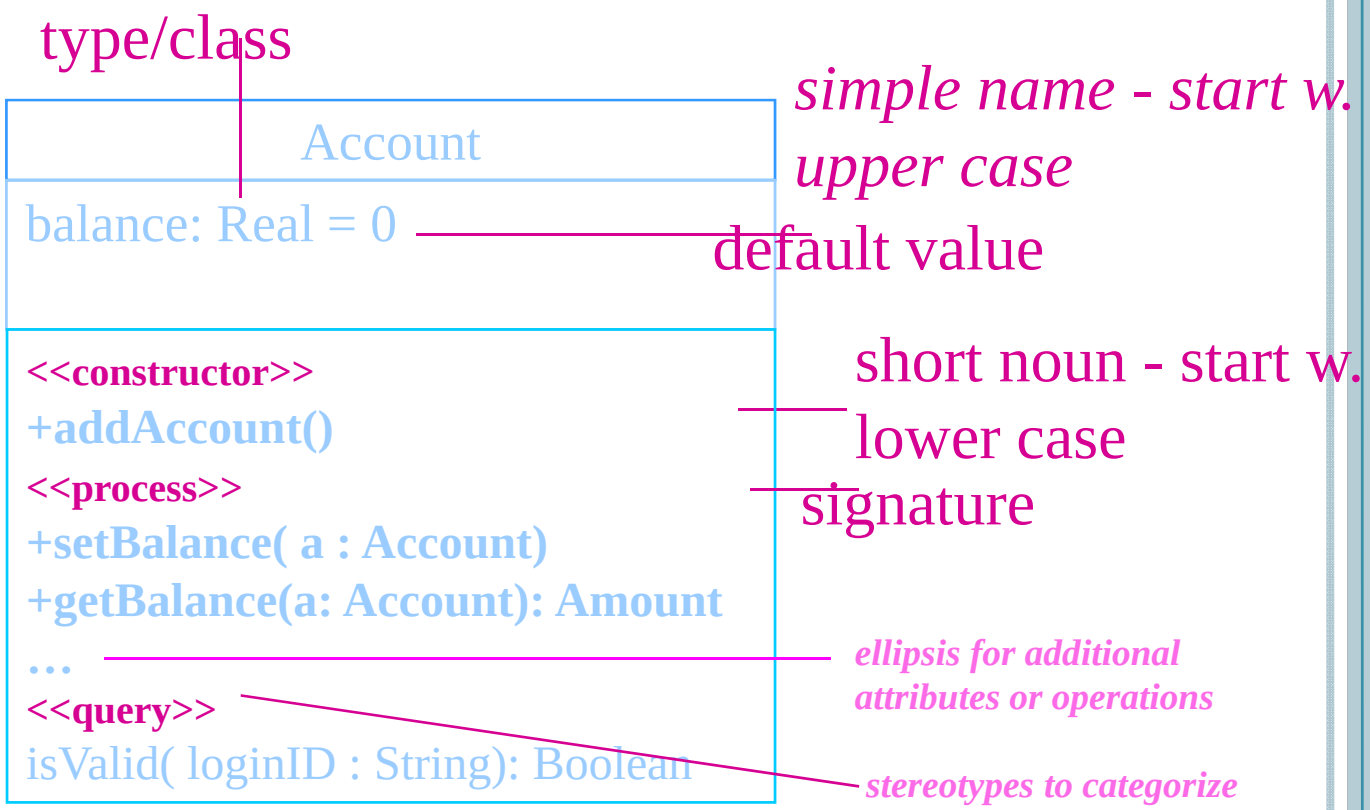
Classes

Names

Attributes

Operations

may cause object to change state



Bank

Customer

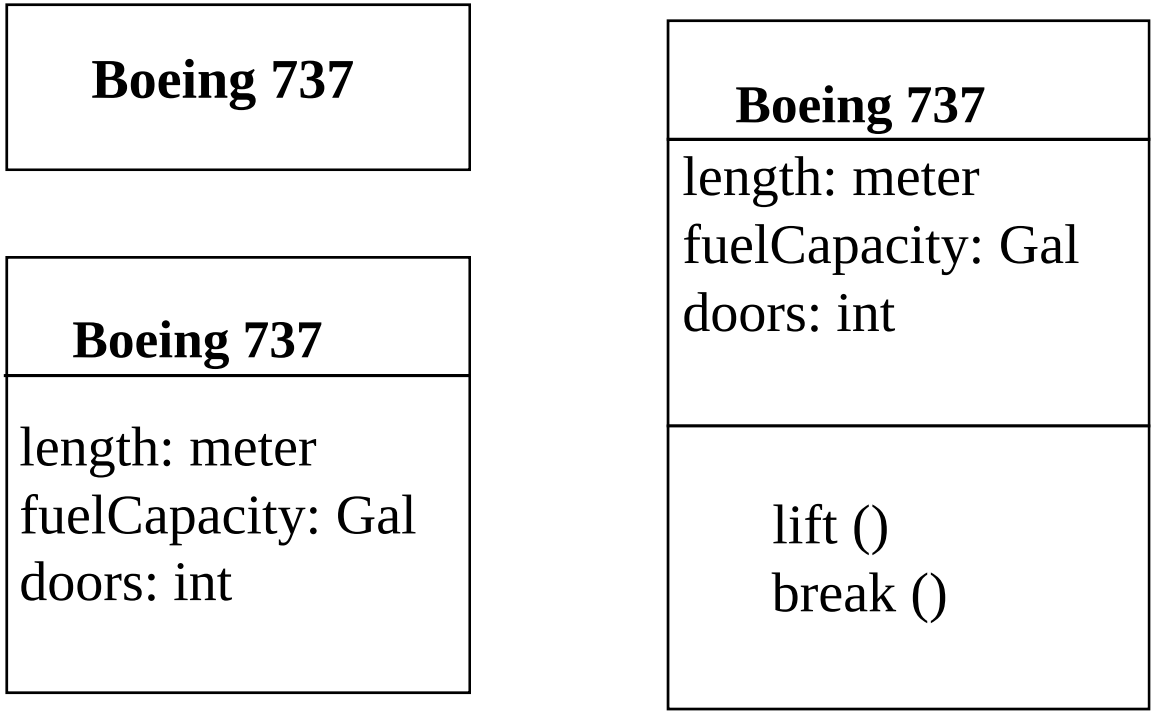
only the name compartment, ok

Java::awt::Polygon

path name = package name ::package name::name

CLASS NOTATION (CONT..)

- In class notation, either one or both the attributes and operation compartments may be suppressed

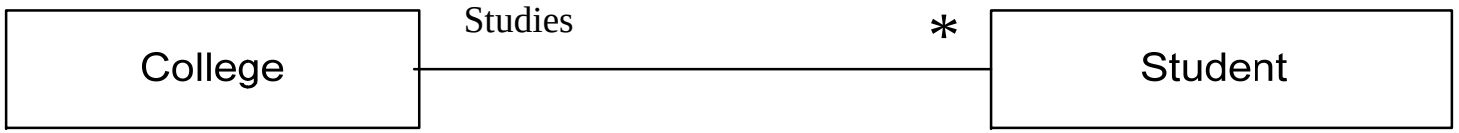


OBJECT DIAGRAM

- A static object diagram is an instance of class diagram
- It shows detailed state of system at a point in time
- Notation is same for object diagram and class diagram
- Class diagram can contain objects
- So class diagram with objects and no classes is an object diagram

OBJECT DIAGRAM

- There is a minor difference:
  - Class diagram shows a class with attributes and methods declared
  - While in object diagram, these attributes and method parameters are allocated values

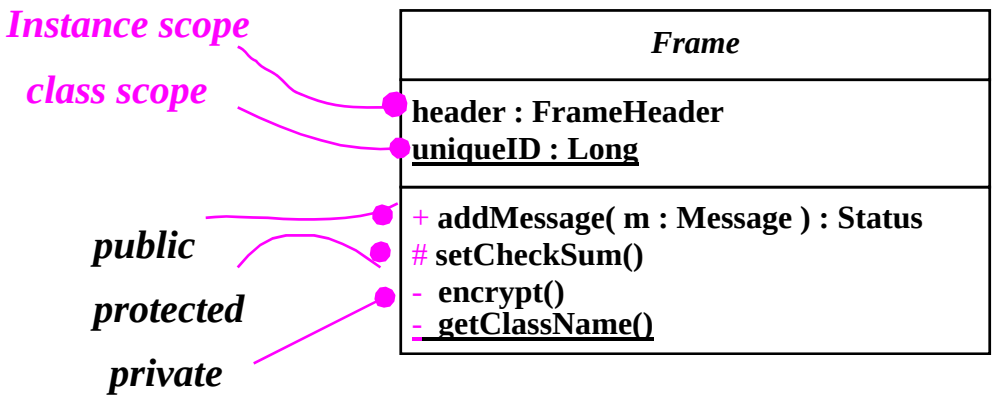


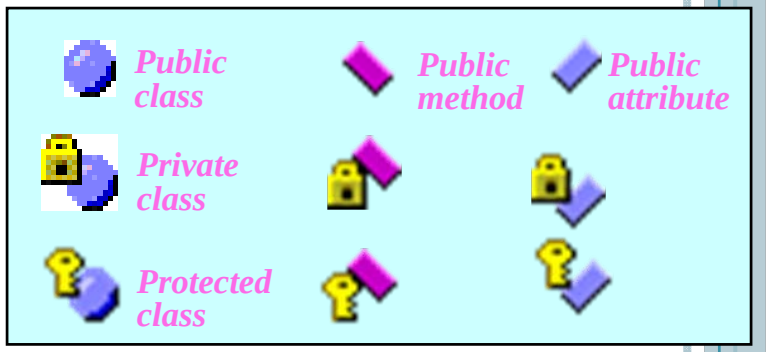
CLASS INTERFACE NOTATION

- Class interface notation is used to describe the externally visible behavior of a class
- For example, an operation with a public visibility
- UML notation is small circle with name of interface connected to class
- Class that requires operations in interface is attached to circle using dashed arrow
- Dependent class is not required to use all operations

Scope & Visibility

- *Instance* Scope — each instance of the classifier holds its own value.
- *Class* Scope — one value is held for all instances of the classifier (underlined).

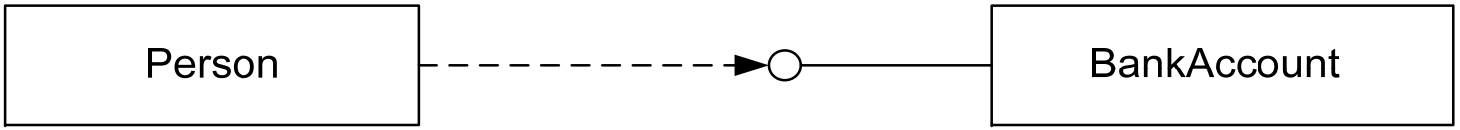




- Public - access allowed for any outside classifier (+).
- Protected - access allowed for any descendant of the classifier (#).
- Private - access restricted to the classifier itself (-).

CLASS INTERFACE NOTATION

- Class interface notation is used to describe the externally visible behavior of a class
- For example, an operation with a public visibility

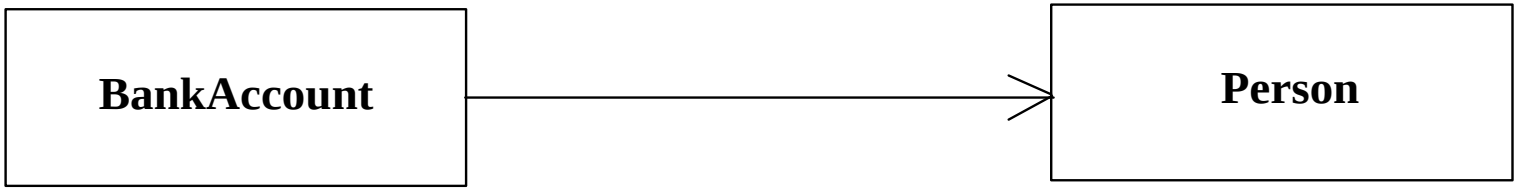


ASSOCIATION

- Association represents relationship between objects and classes
- Binary association is relationship between two classes
- A binary association is drawn as a solid path connecting two classes or both ends may be connected to the same class
- An association may have an association name
- Association name may have optional black triangle that indicates direction in which to read the name

UML ASSOCIATION NOTATION

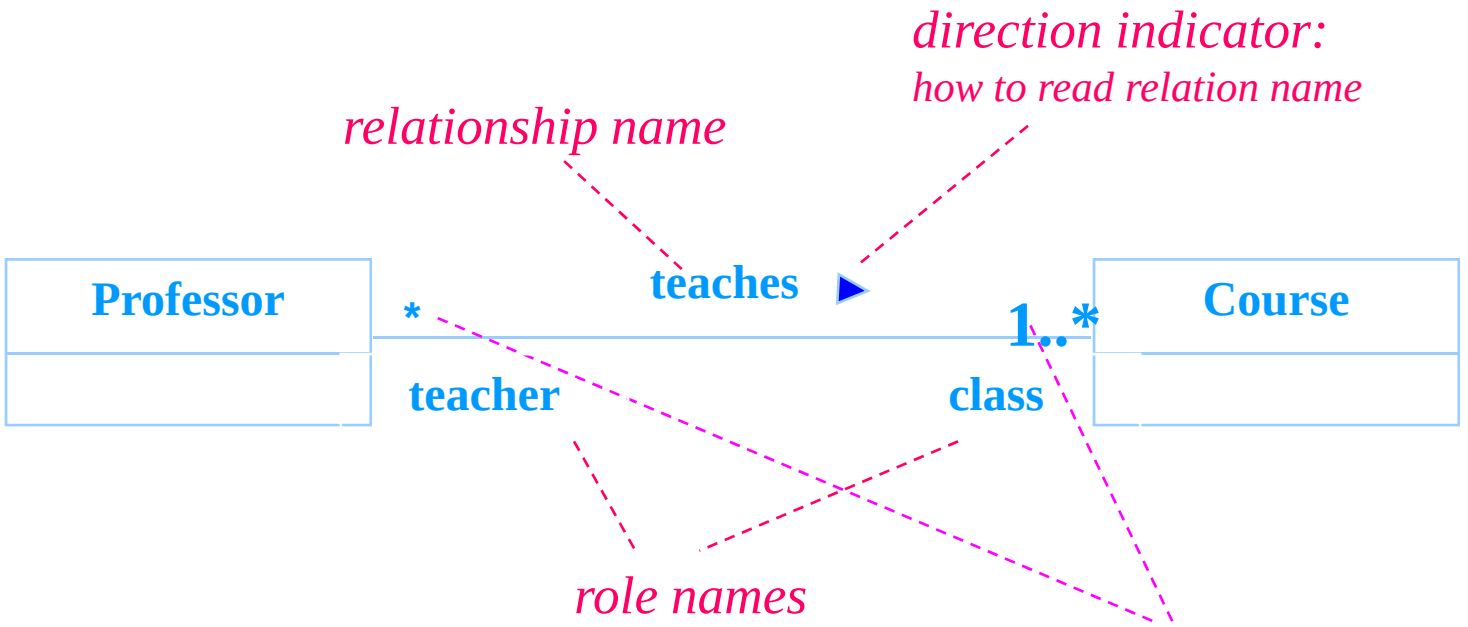
- UML uses term association navigation or navigability to specify role
- In the UML, association is represented by an open arrow
- Arrow may be attached at the end of path to indicate direction of navigation
- An arrow may be attached to neither, one or both ends of path
- In the UML, association is represented by an open arrow



- Person class can't know anything about BankAccount class, but BankAccount class can know about Person class

Associations (UML)

- Represent conceptual relationships between classes

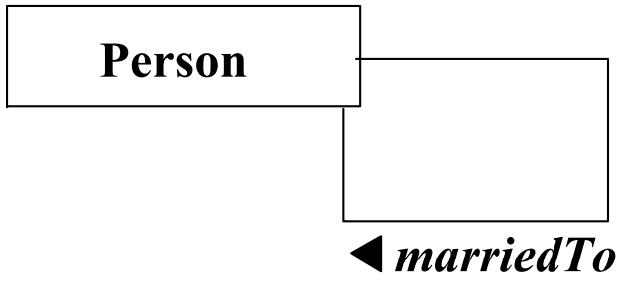


Multiplicity

defines the number of objects associated with an instance of the association. : Default of 1; Zero or more (*); n..m; range from n to m inclusive

BINARY ASSOCIATION NOTATION

- A binary association is drawn as a solid path connecting two classes or both ends may be connected to the same class

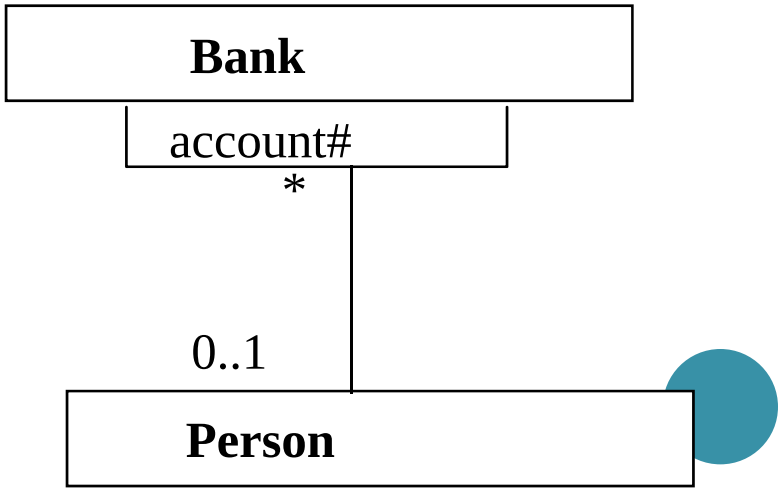


ASSOCIATION ROLE

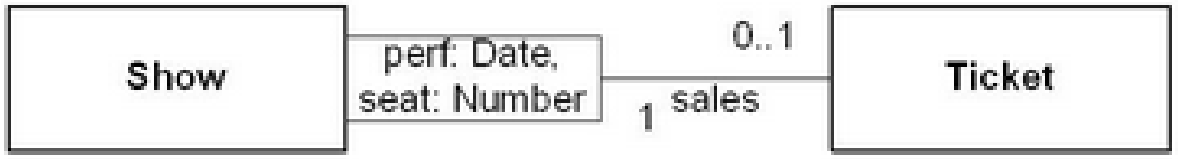
- A simple association—the technical term for it is *binary association*—is drawn as a solid line connecting two class symbols
- The end of an association, where it connects to a class, shows the association role
  - Role is a part of association, not class
  - Each association has two or more roles to which is connected

QUALIFIER

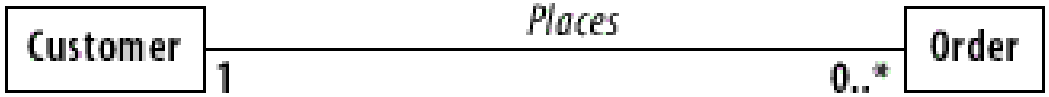
- A qualifier is an association attribute. For example, a person object may be associated to a Bank object
- An attribute of this association is the account#
- The account# is the qualifier of this association



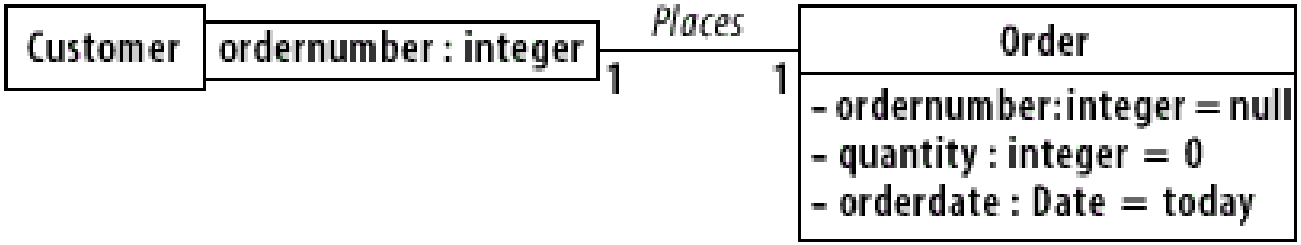
QUALIFIER



Without the qualifier

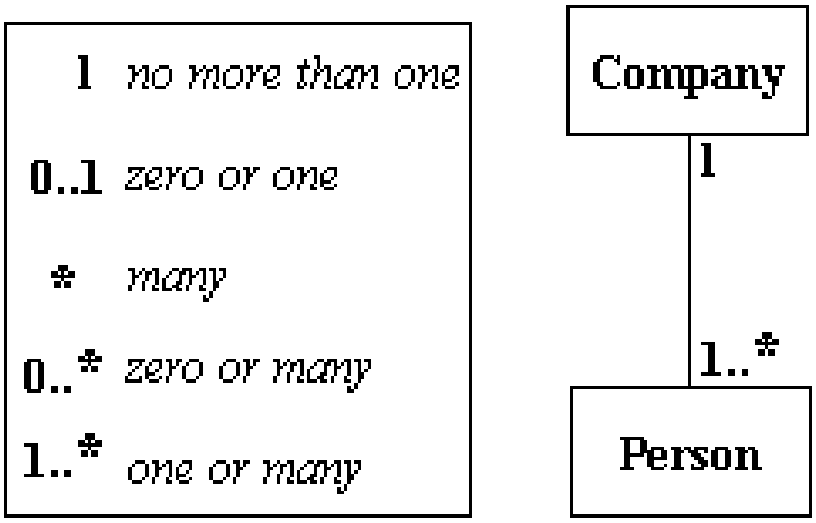


With the qualifier



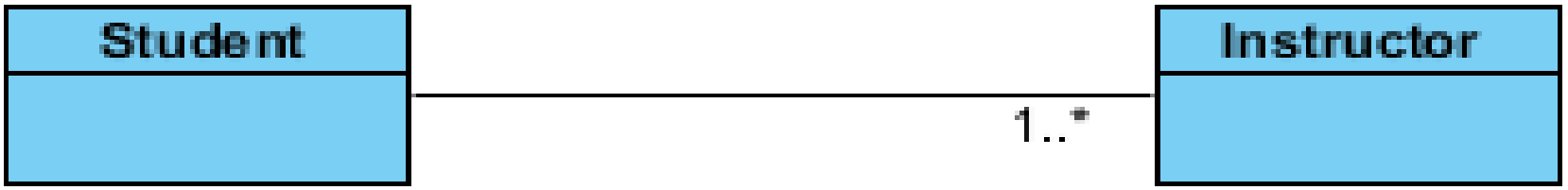
MULTIPLICITY

- Multiplicity specifies the range of allowable associated classes
- It is given for roles within associations, parts within compositions, repetitions, and other purposes
- lower bound .. upper bound

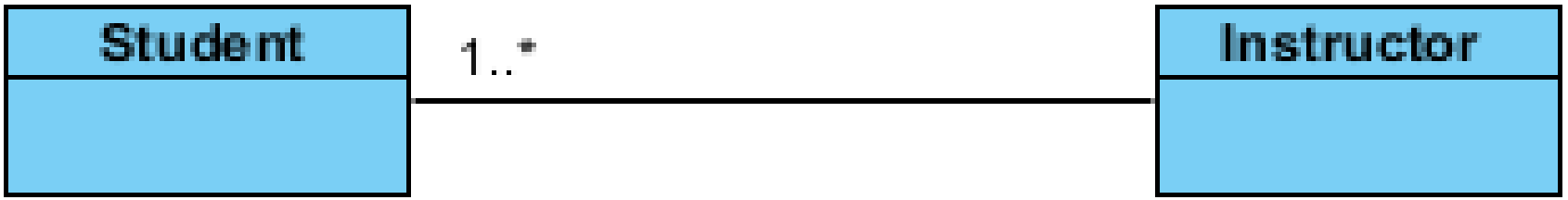


ASSOCIATION

- A single student can associate with multiple teachers:



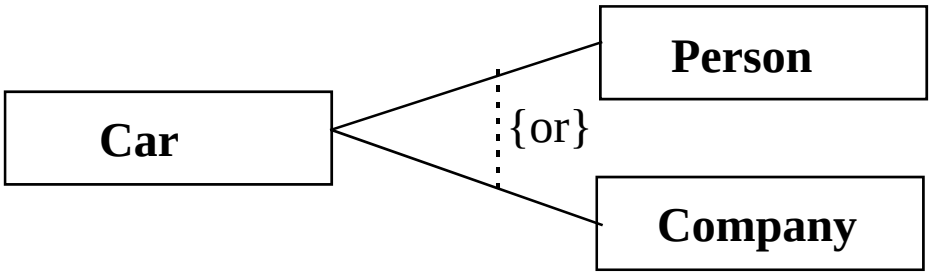
- The example indicates that every Instructor has one or more Students:



- We can also indicate the behavior of an object in an association (i.e., the role of an object) using role names.

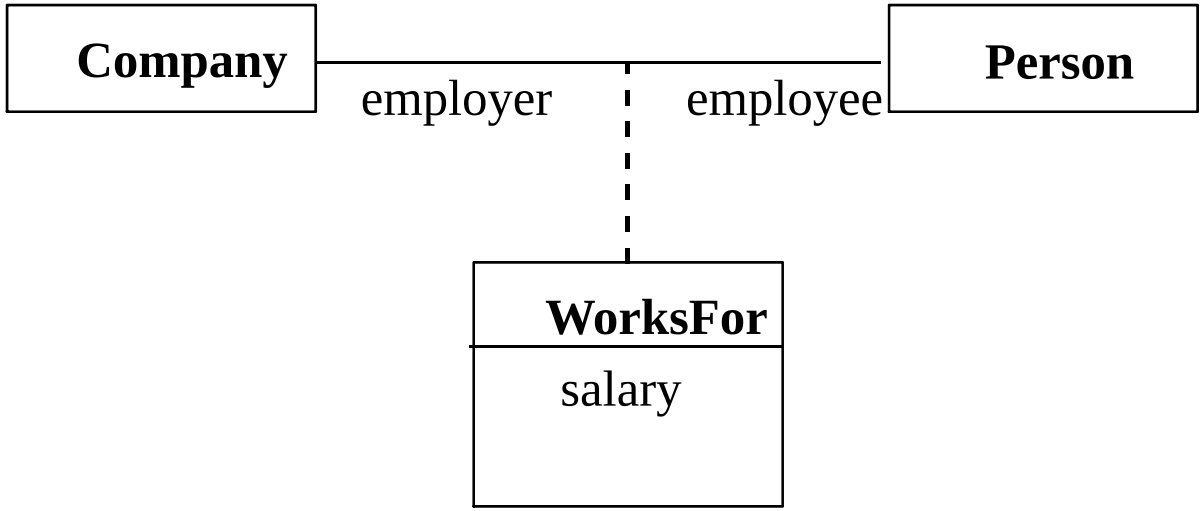
OR ASSOCIATION

- An OR association indicates a situation in which only one of several potential associations may be substantiated at one time for any single object

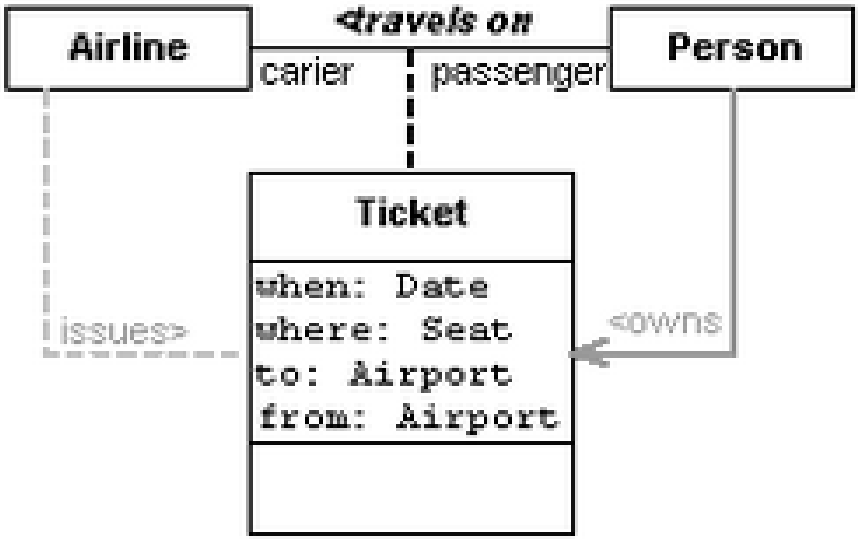


ASSOCIATION CLASS

- An association class is an association that also has class properties
- An association class is shown as a class symbol attached by a dashed line to an association path

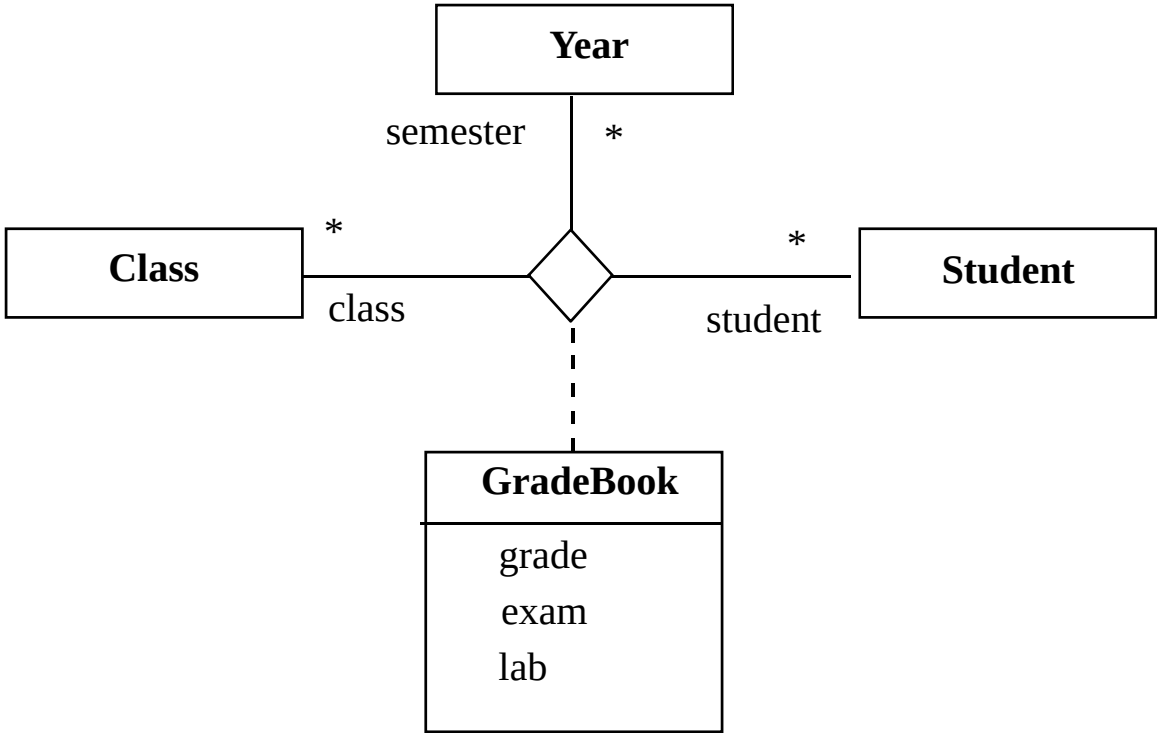


ASSOCIATION CLASS



N-ARY ASSOCIATION

- An n-ary association is an association among more than two classes
- Since n-ary association is more difficult to understand, it is better to convert an n-ary association to binary association



N-ARY ASSOCIATION

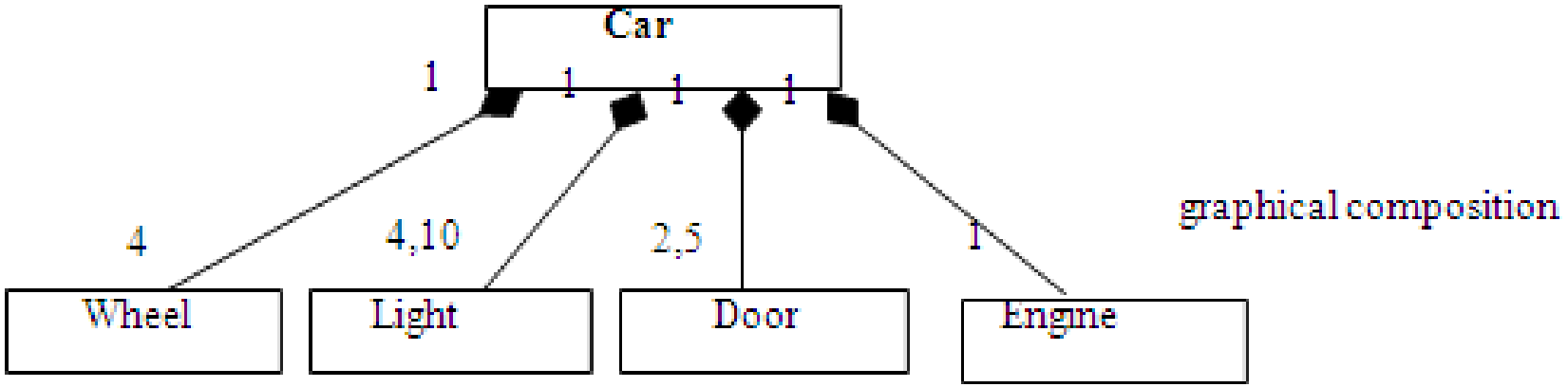
- Multiplicity, the name of the association like in binary one can be included
- But qualifiers and aggregations are not permitted
- An association class symbol may be attached to the diamond by a dashed line, indicating that an N-ary relation that has attributes, operations or associations

AGGREGATION

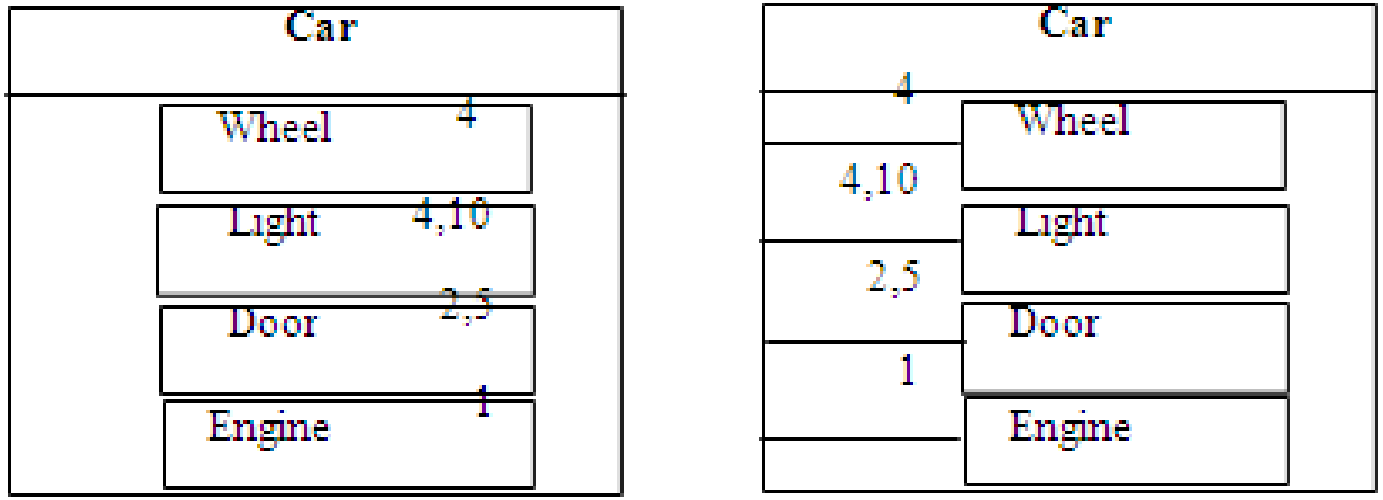
- Aggregation is a form of association
- A hollow diamond is attached to the end of the path to indicate aggregation
- Aggregation is used to represent ownership or a whole/part relationship

COMPOSITION

- Also known as the *a-part-of*, is a form of aggregation with strong ownership to represent the component of a complex object

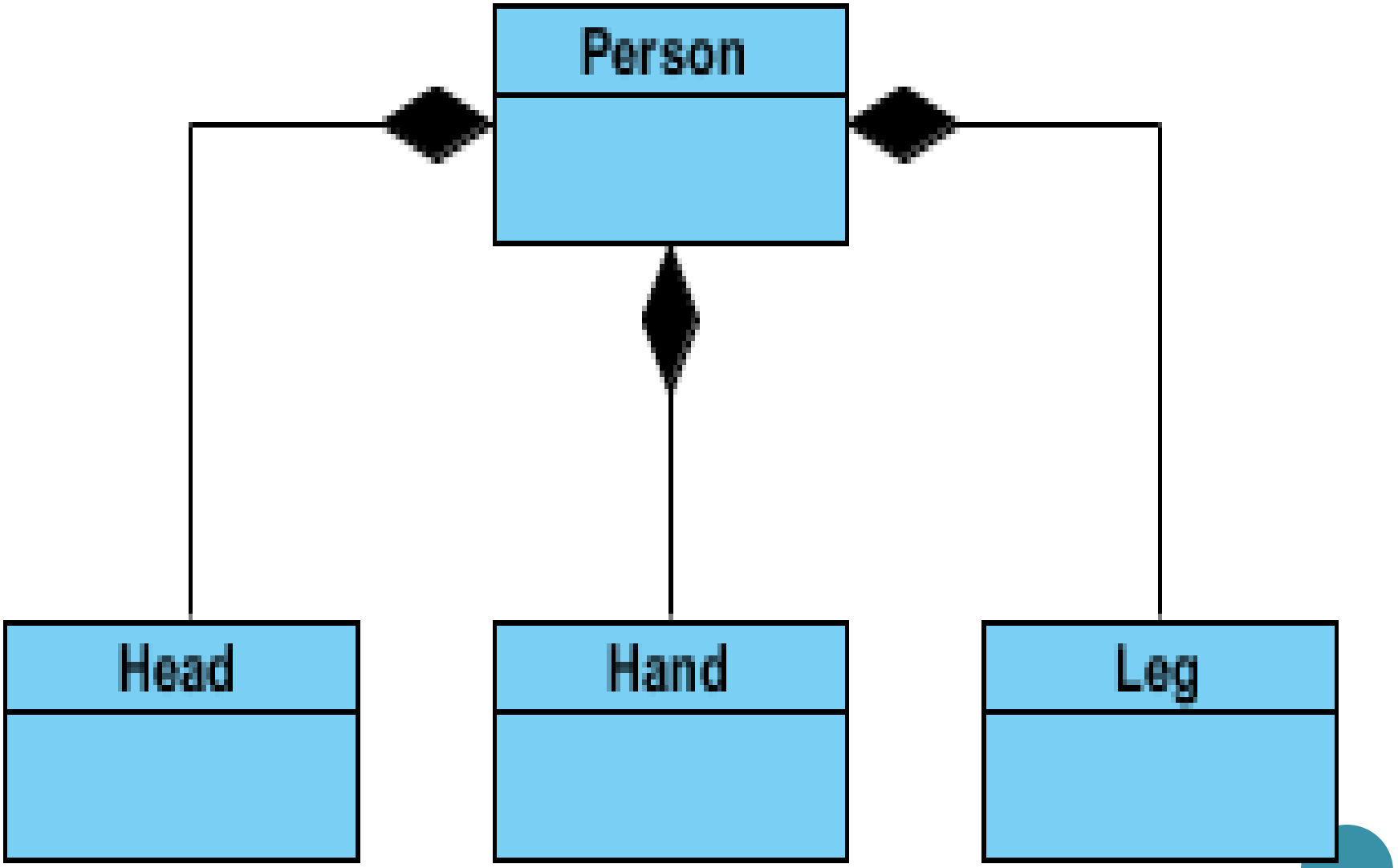


graphical composition



nested composition

COMPOSITION Example



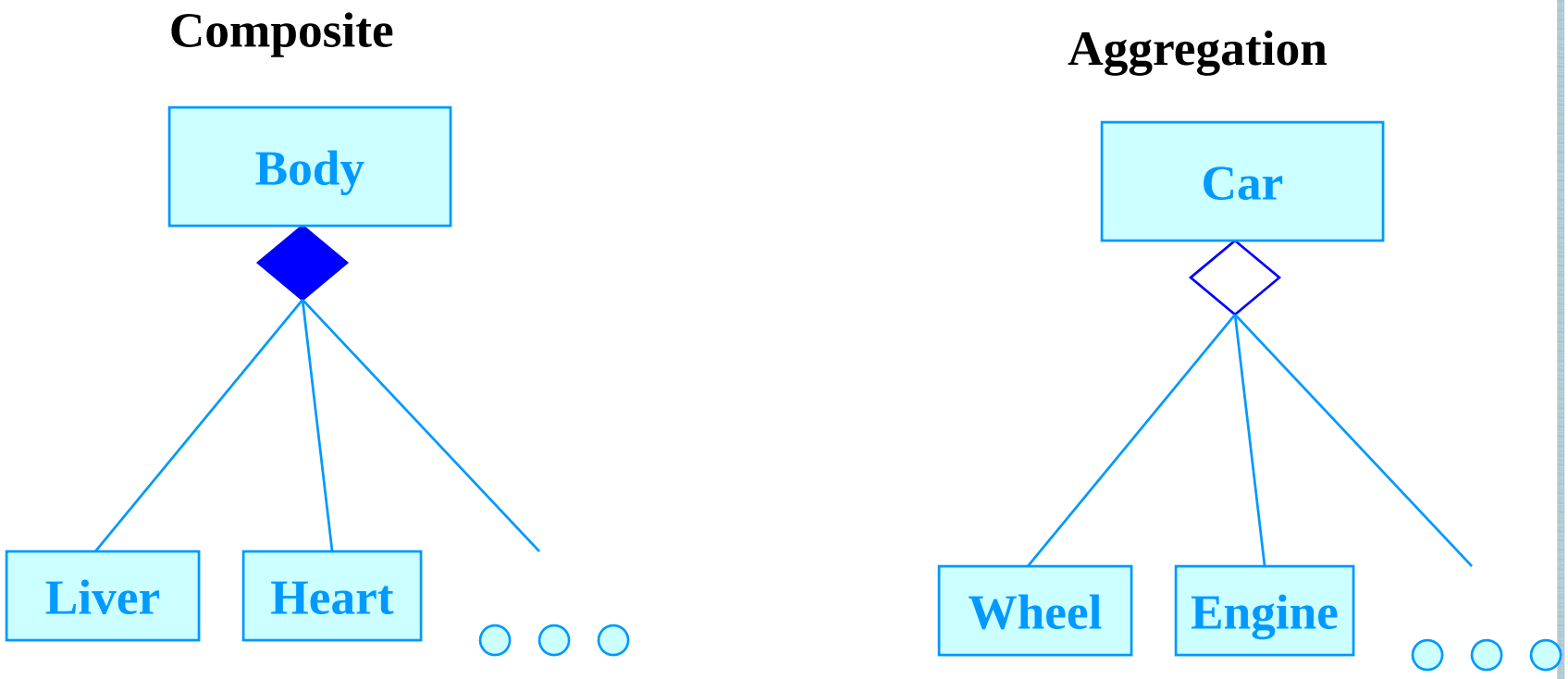
COMPOSITION

- The UML notation for composition is a solid diamond at the end of a path , and is used to represent an even stronger form of ownership.
- The composite object has sole responsibility for the disposition of its parts in terms of creation and destruction
- In implementation terms, the composite is responsible for memory allocation and deallocation
- The multiplicity of the aggregate end may not exceed one; i.e., it is unshared. An object may be part of only one composite at a time

COMPOSITION

- If the composite is destroyed, it must either destroy all its parts or else give responsibility for them to some other object
- A composite object can be designed with the knowledge that no other object will destroy its parts

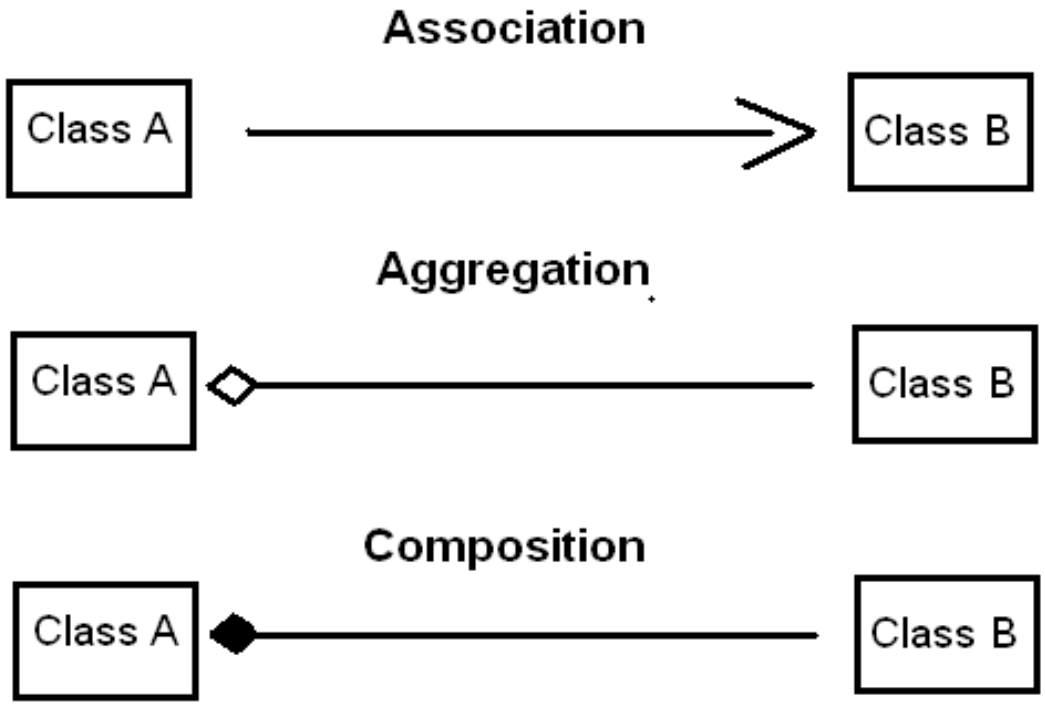
Modeling Structural Relationships



Composite is a stronger form of aggregation. Composite parts live and die with the

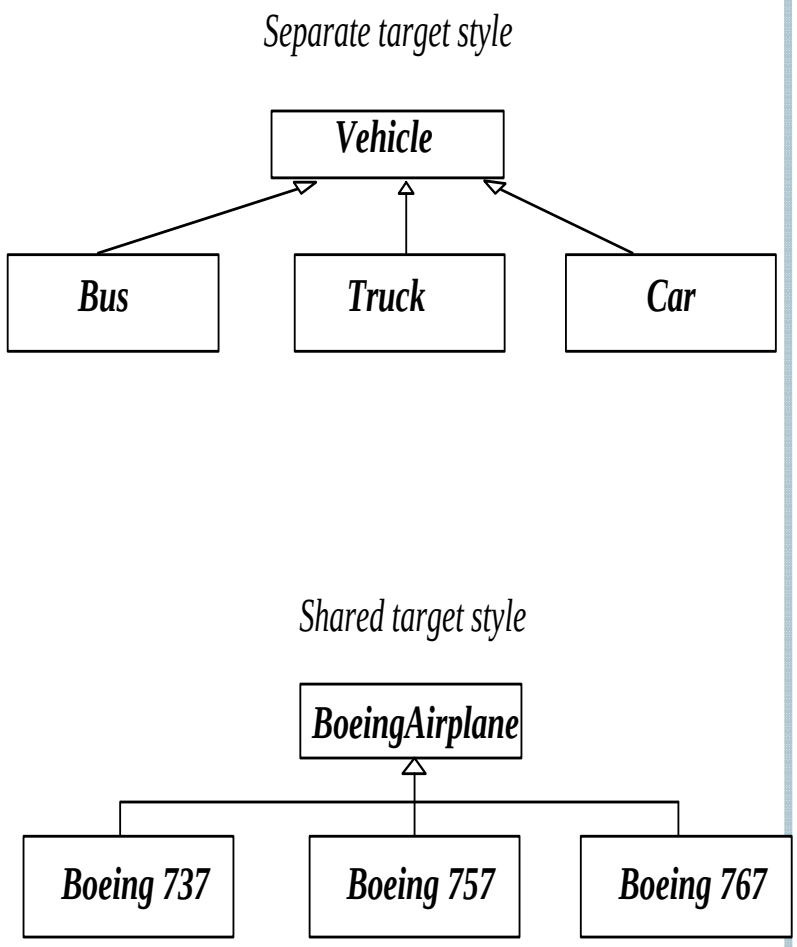
CONCLUSION

- The three links Association, Aggregation and Composition form a kind of scale on how closely two classes are related to each other.



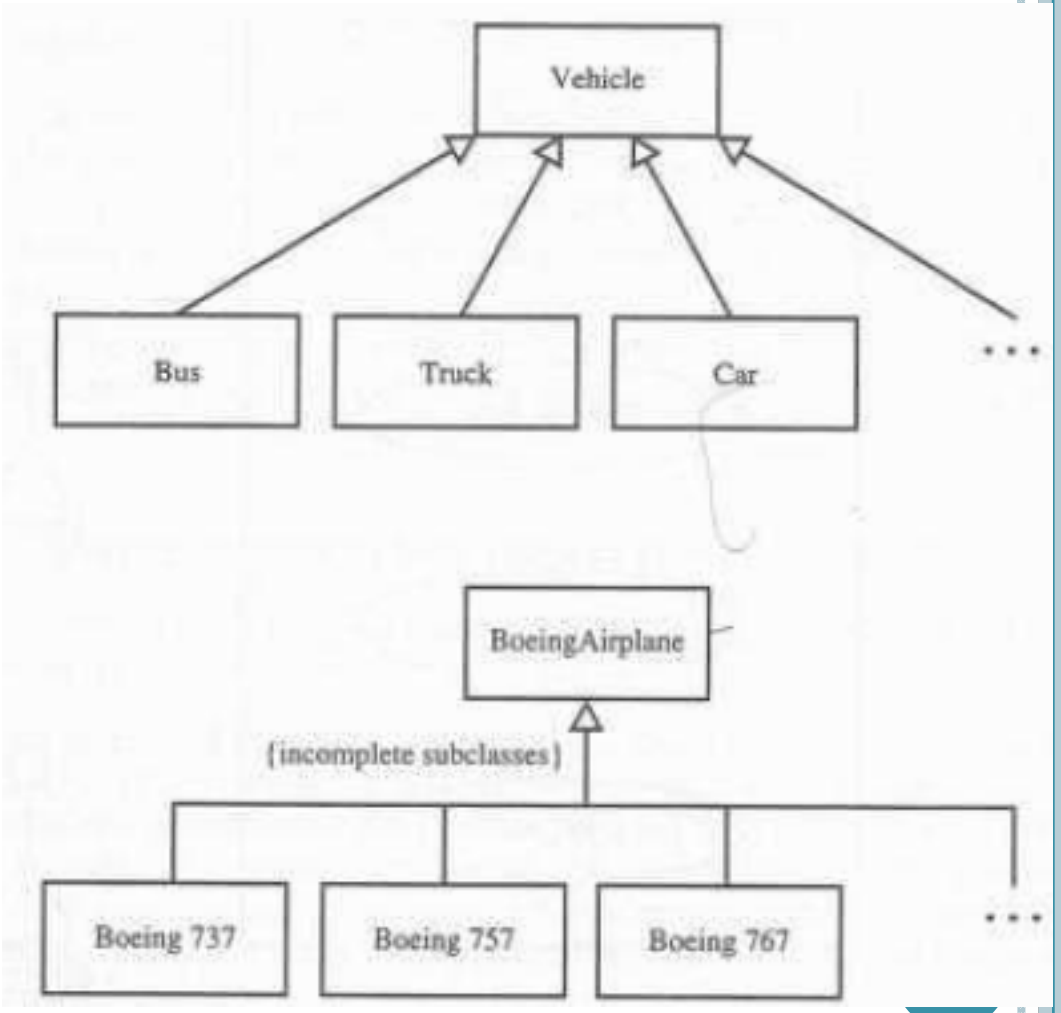
GENERALIZATION

- Generalization is the relationship between a more general class and a more specific class
- This relationship used when we want represents a class, which captures the common states of objects of different classes
- Generalization is displayed as directed line with a closed, hollow arrowhead at the superclass end



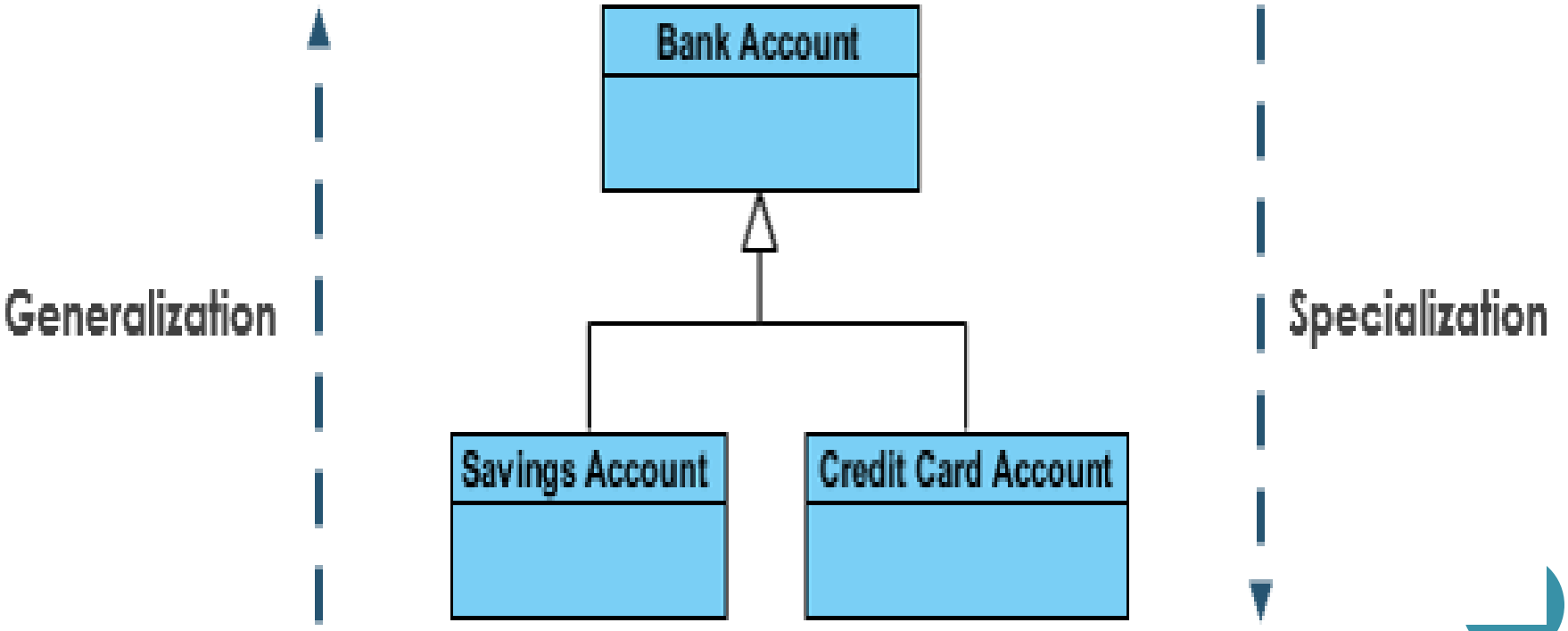
GENERALIZATION

- Ellipses indicate that the generalization is incomplete and more subclasses exist that are not shown
- If the text label is placed on hollow triangle, it applies to all the paths.

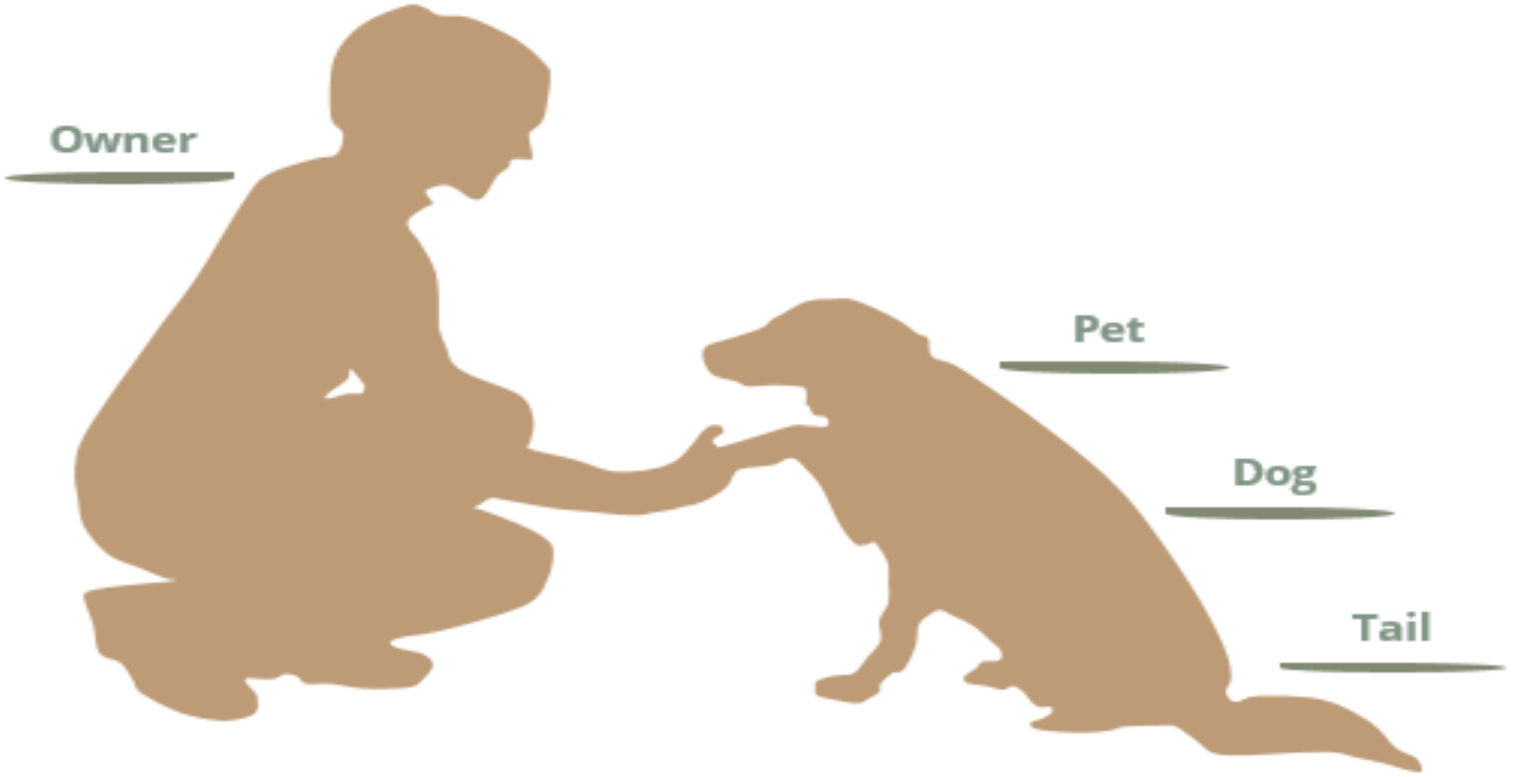


GENERALIZATION vs SPECIALIZATION

Specialization is the reverse process of Generalization means creating new sub-classes from an existing class.



Association • Aggregation • Composition



We see the following relationships:

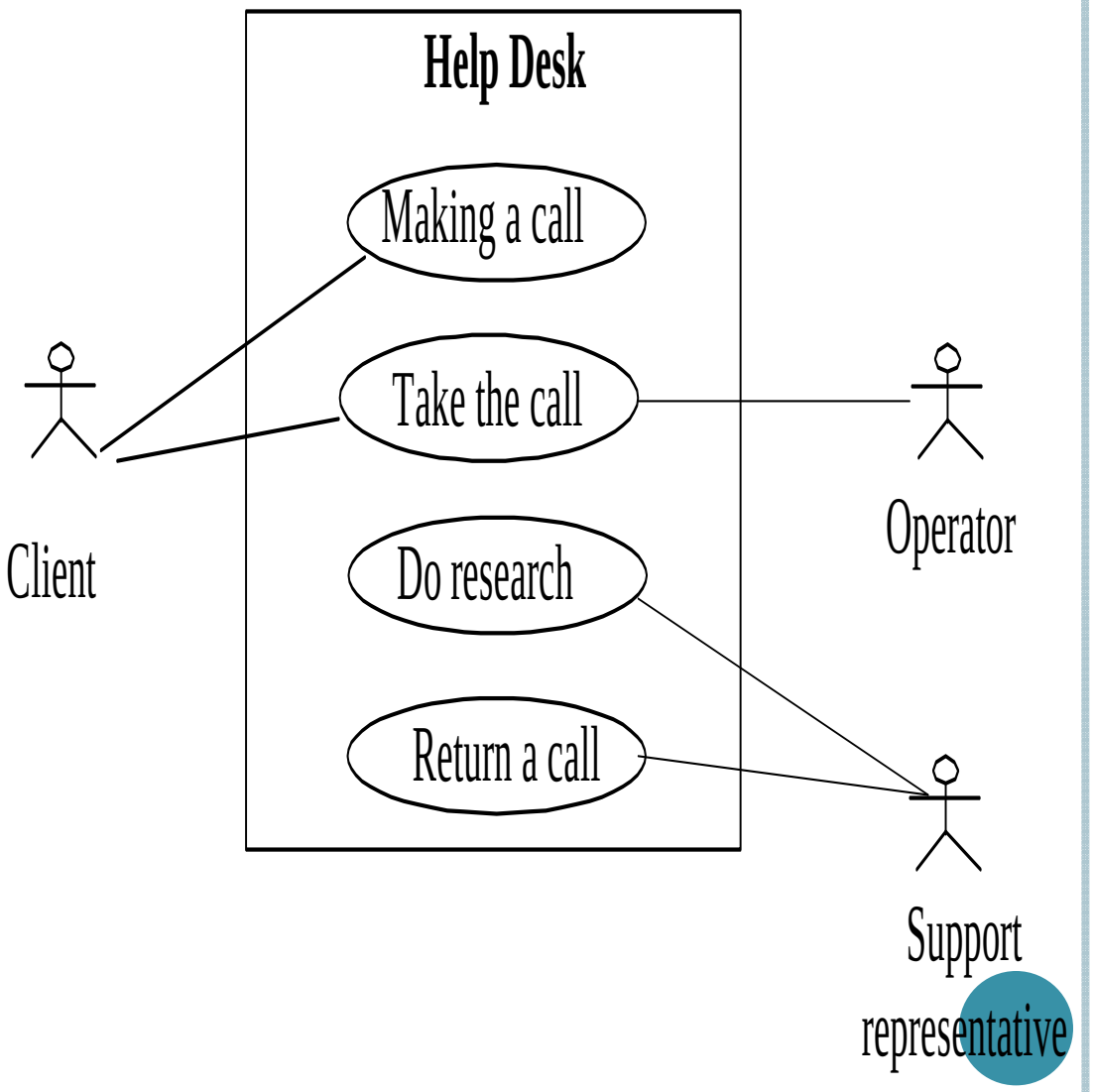
- owners feed pets, pets please owners (**association**)
- a tail is a part of both dogs and cats (**aggregation / composition**)
- a cat is a kind of pet (**inheritance / generalization**)

USE-CASE DIAGRAM

- A Use Case is a description of a set of sequence of actions that a system performs that yields an observable result of value to a particular action
- The description of a use case defines what happens in the system when the use case is performed
- In essence, the use-case model defines the outside (actors) and inside (use case) of the system's behavior
- An Actor is someone or something that must interact with the system. An Actor initiates the process(that is USECASE)

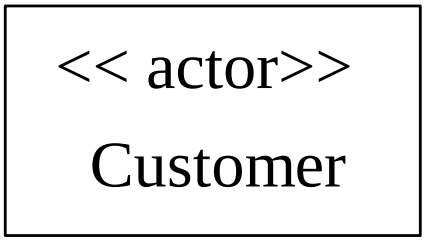
USE-CASE DIAGRAM (CON'T)

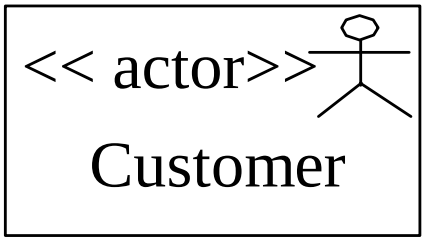
- A use-case diagram is a graph of actors, a set of use cases enclosed by a system boundary, communication (participation) associations between the actors and the use cases, and generalization among the use cases

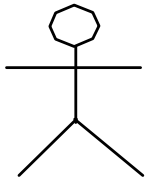


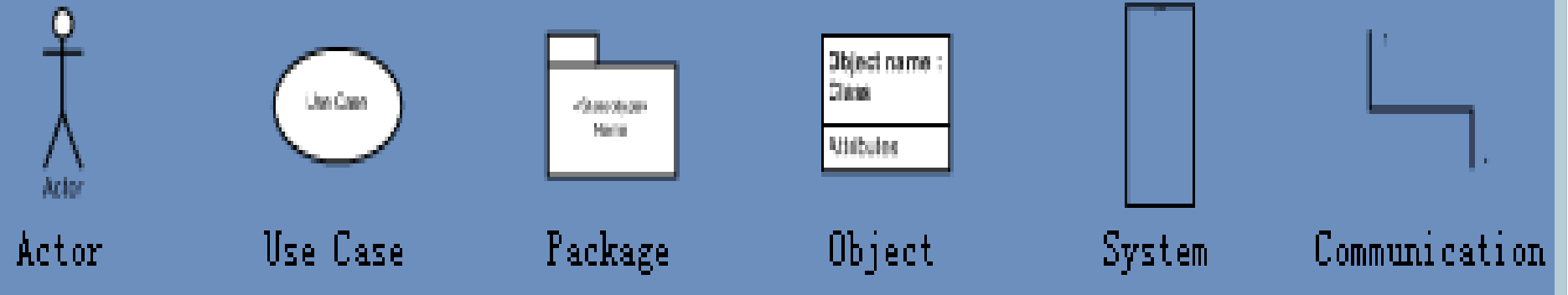
ACTOR NOTATIONS

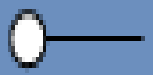
- The three representations of an actor are equivalent











Interface

Constraint

Note

USE-CASE DIAGRAM (CON'T)

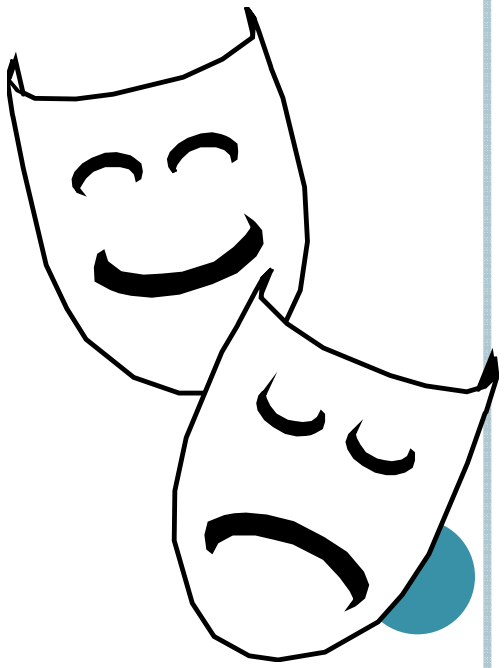
- These relationships are shown in a use-case diagram:
- Communication
  - The communication relation of an actor in a use case is shown by connecting the actor symbol to the use-case symbol with a solid path
- Uses
  - A uses relationship between use cases is shown by a generalization arrow from the use case
- Extends
  - The extends relationship is used when you have one use case that is similar to another use case but does a bit more. Like a subclass

BEHAVIOR OR DYNAMIC DIAGRAMS

- Its impossible to capture all the details of a system in just one model or view, the diagrams we have seen till are all static
- But events happen dynamic: objects are created and destroyed, objects send messages or some events may also trigger operations on certain objects
- E.g. a fixed line telephone
- Describing a systematic event in a static medium such as sheet of a paper is difficult – OOD, make it easy to express with following diagram

CONT..

- Interaction diagrams:
  - Sequence diagrams
  - Collaboration diagrams
- Statechart diagrams
- Activity diagrams

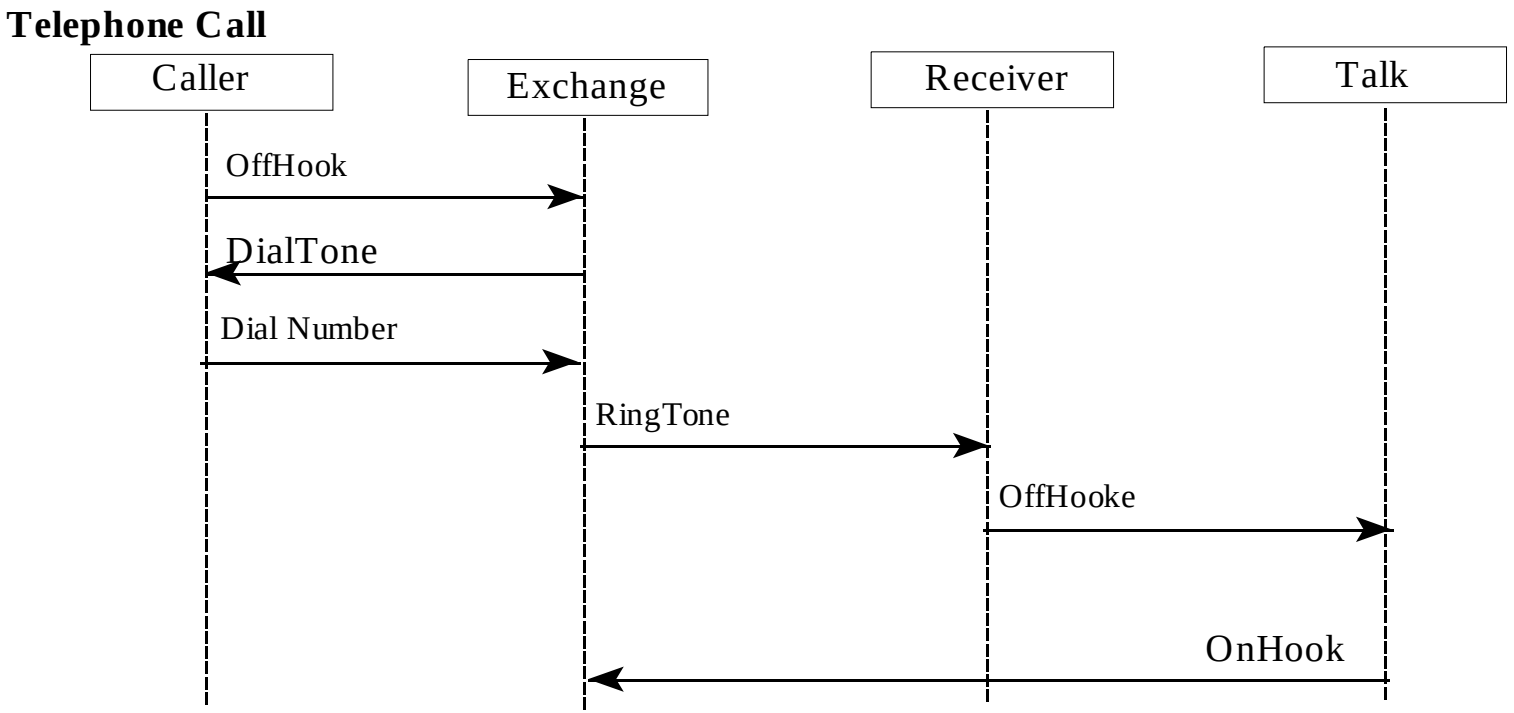


UML INTERACTION DIAGRAMS

- Interaction diagrams describe how groups of objects collaborate to get the job done
- Interaction diagrams capture the behavior of a single use case, showing the pattern of interaction among objects

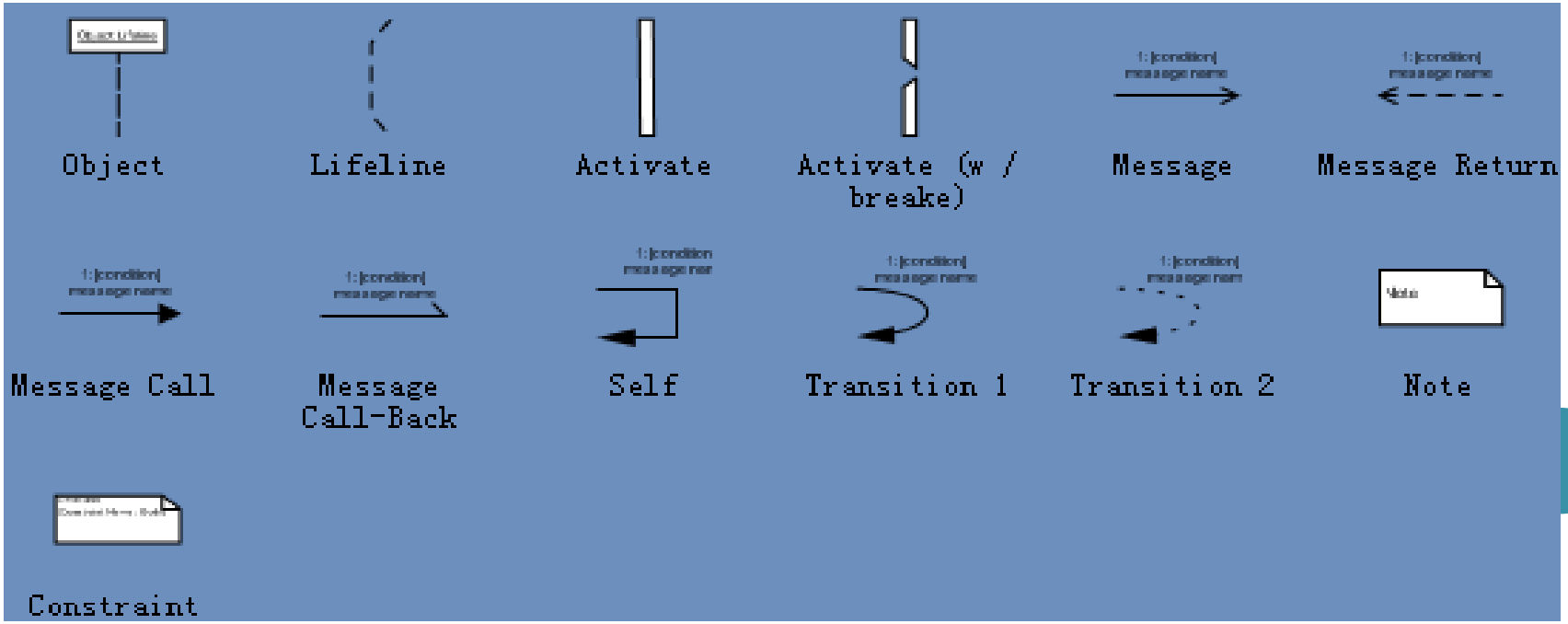
UML SEQUENCE DIAGRAM

- *Sequence diagrams* are an easy and intuitive way of describing the behavior of a system
- A sequence diagram shows an interaction arranged in a time sequence



CONT..

- It shows the objects participating in the interaction by their life-lines and the messages they exchange, arranged in a time sequence
- It has two dimensions: the vertical represents time and the horizontal represents different objects
- The life line represents the objects existence during the interaction



UML Sequence Diagram

- Label can also have argument or control information
- Self-delegation
  - Message that object sends to itself by sending arrow back to same lifeline
- Horizontal ordering of lifeline is arbitrary
- Sequence diagram does not show relationship among roles and association among objects
- Boxes on top of the diagram represent use cases, objects, classes, or actors.
- Objects have labels in the standard UML format name: ClassName, where “name” is optional
- Object labels are underlined while classes and actors are not.
- anonymous objects
- Classes have labels in the format ClassName, and actors have names in the format Actor Name.

UML Sequence Diagram

- Dashed lines hanging from the boxes are called object lifelines, representing life span of the object during the scenario
- The long, thin boxes on the lifelines are activation boxes, also called method-invocation boxes
- It indicates processing is being performed by the target object/class to fulfill a message.
- Draw activation boxes if needed
- The X at the bottom of an activation box indicates an object has been removed from memory

UML Sequence Diagram

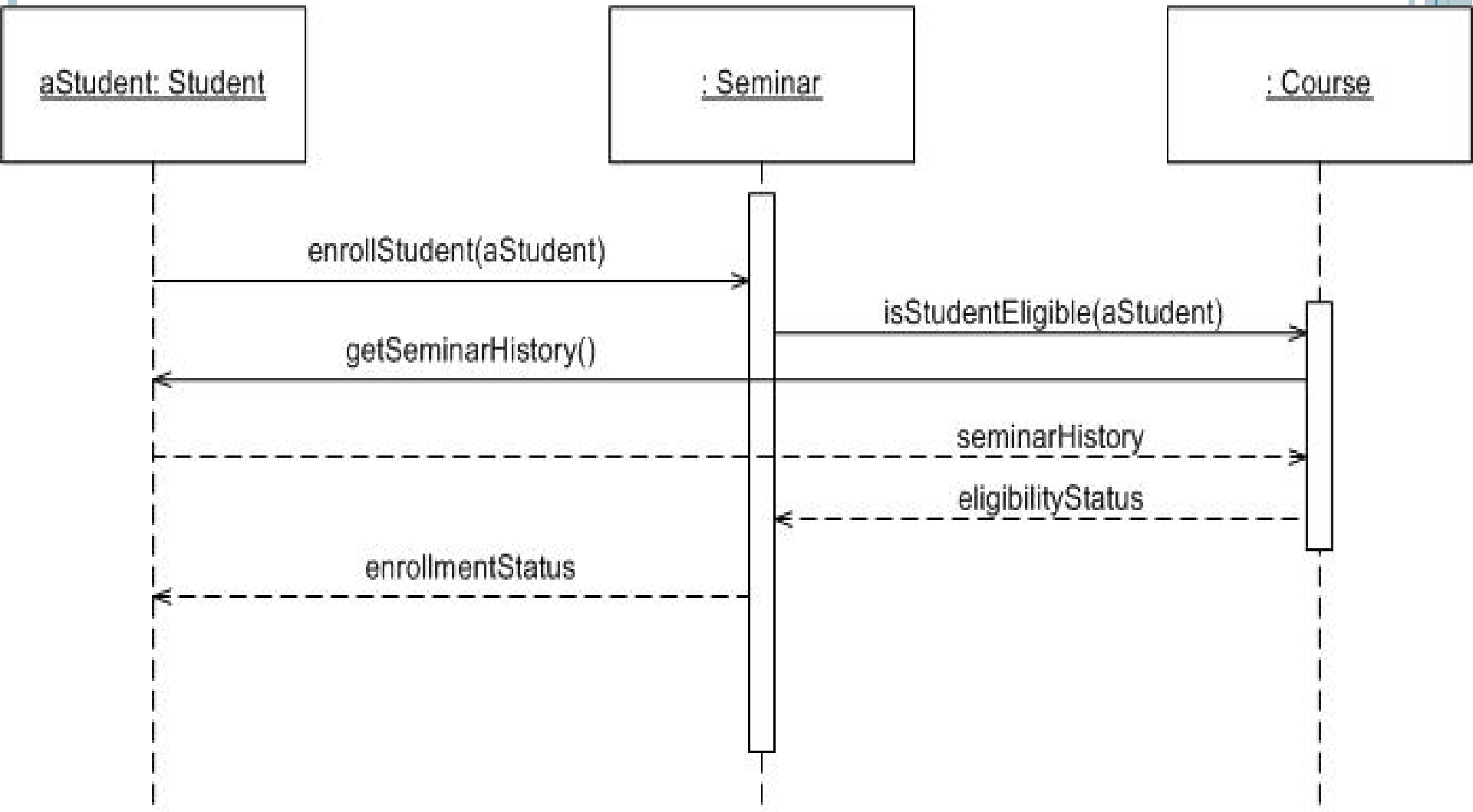


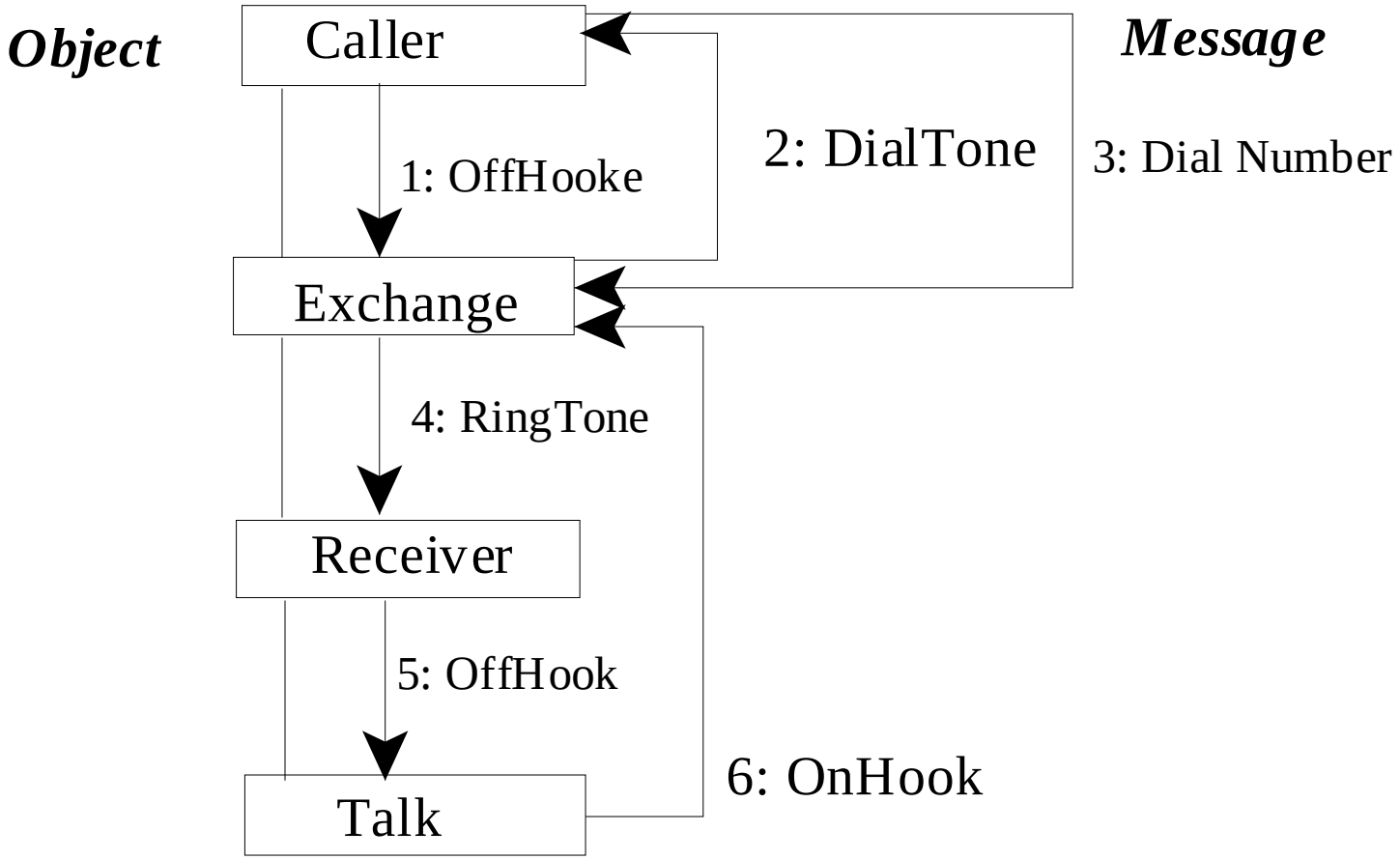
Fig - Enrolling in a seminar

UML Collaboration Diagram

- A collaboration diagram represents a set of objects related in a particular context, and the exchange of their messages to achieve a desired outcome.
- Objects are shown as figures
- Arrow indicate message sent within given use case
- But sequence is indicated by numbering messages
- Several numbering schemes are provided

UML COLLABORATION DIAGRAM

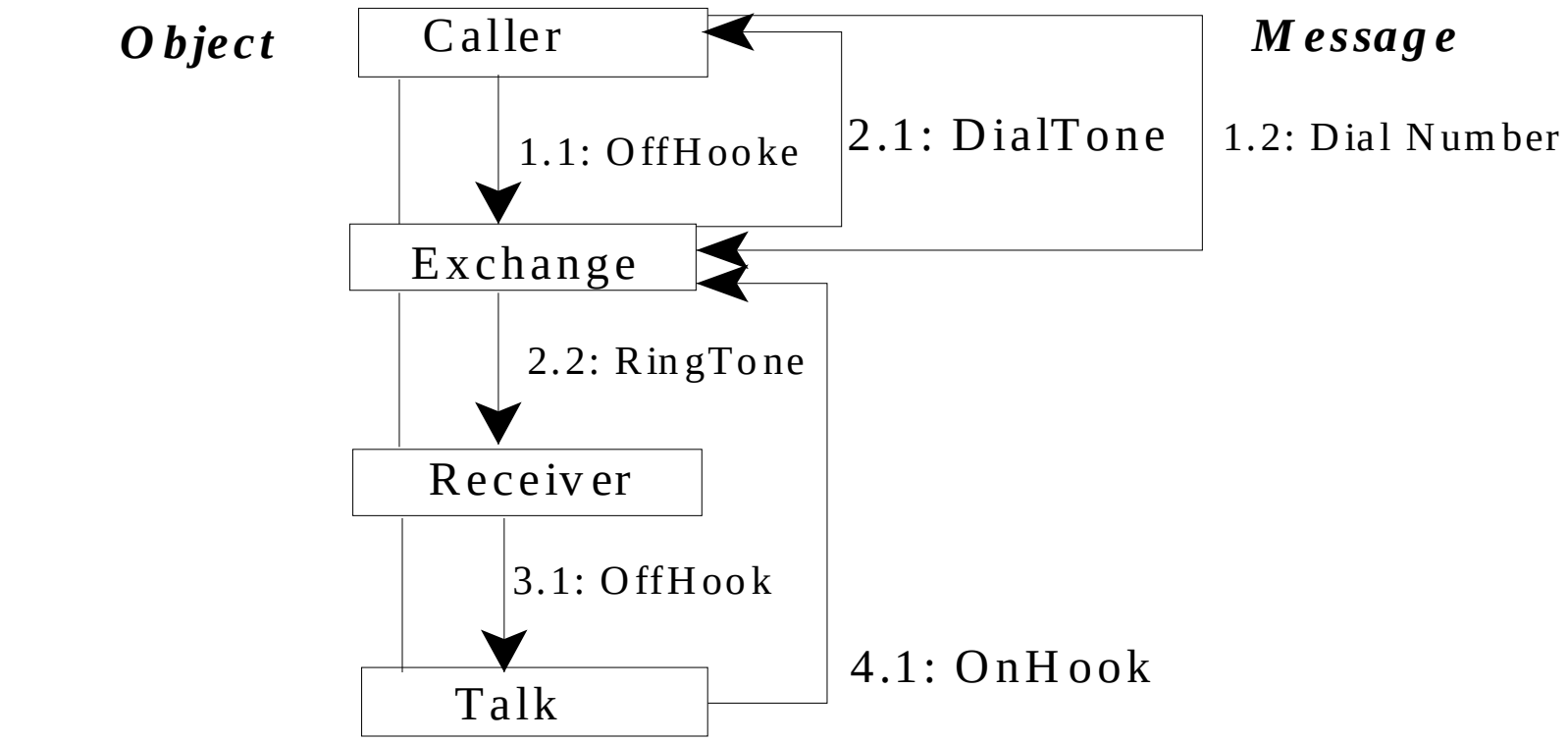
Telephone Call



CONT..

- In this diagram , objects are shown as figures
- The sequence is indicated by numbering the messages
- Specifically it shows how the objects are linked together – and the layout can be overlaid with packages or other information

UML COLLABORATION DIAGRAM (CON'T)



UML Collaboration Diagram

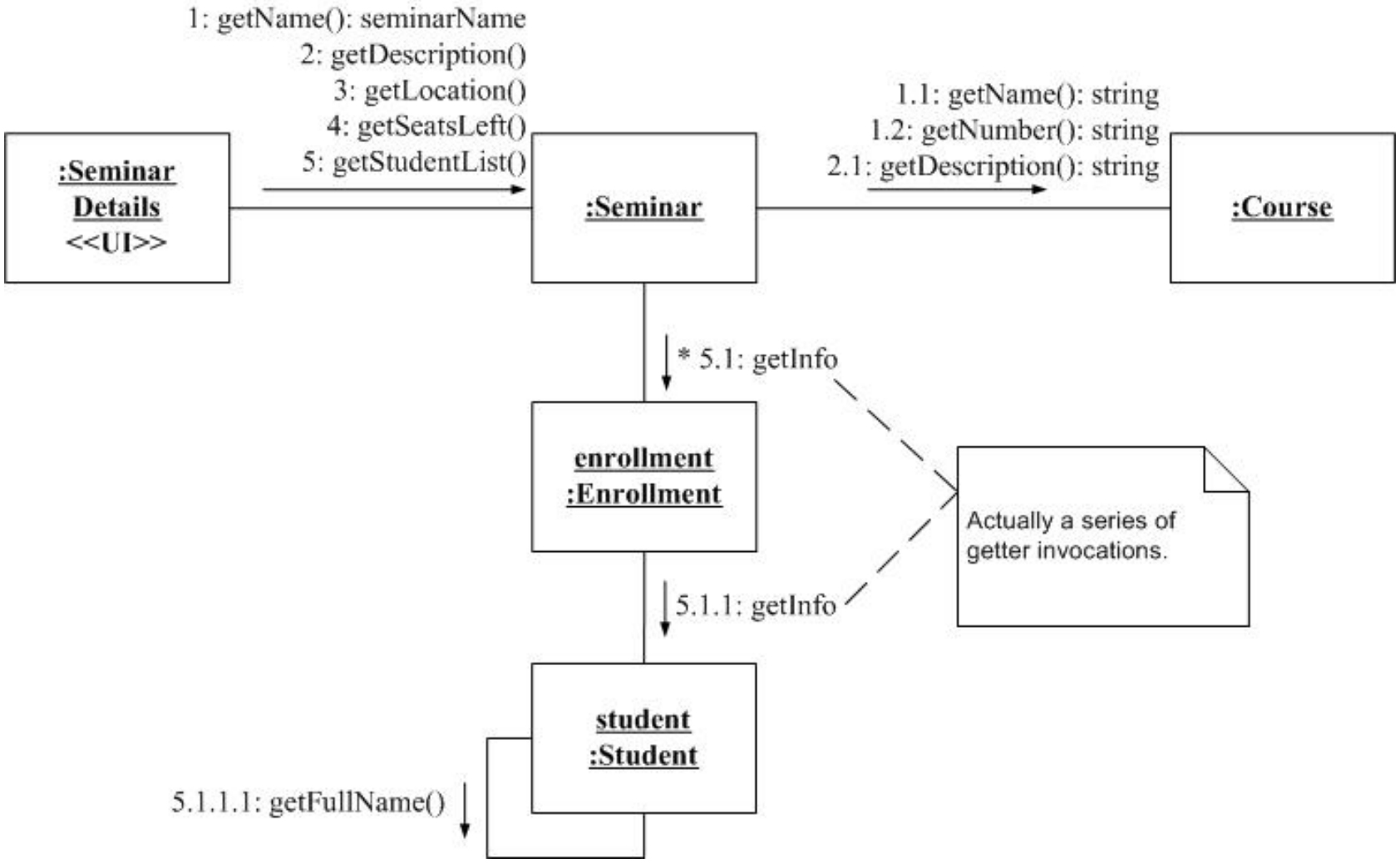


Fig – Collaboration diagram for displaying seminar details on screen

[sequenceNumber:] methodName(parameters) [: returnValue]

UML Collaboration Diagram

- Different people have different preferences for using sequence diagram or collaboration diagram
- Sequence diagram is easy to read
- While collaboration indicate how objects are statically connected
- Disadvantage of collaboration diagram is that it is good for single sequential process
- But break down to represent conditional looping behavior
- Conditional behavior can be represented in two ways:
- Use separate diagrams for each scenario
- Use conditions on messages to indicate behavior
- For complex behavior in single diagram use activity diagram
- Generally interaction diagrams are used to show behavior of objects within single use case

COLLABORATION AND SEQUENCE DIAGRAMS: DIFFERENCE

- Collaboration diagram illustrates object interactions in a graph or network format, in which objects can be placed anywhere on the diagram. It demonstrates how objects are statically connected.

- Sequence diagram illustrates interaction in a kind of fence format, in which each new object is added to the right. It generally shows the sequence of events that occur.

DISADVANTAGES OF INTERACTION DIA.

- They are great only for representing a single sequential process
- And begin to break down when you want to represent conditional looping behavior
- It can be solved by separate diagrams for each scenario Or use condition on the messages to indicate behavior
- At the end interaction diagrams can be used to examine the behavior of objects within a single use case.

UML STATE-CHART DIAGRAM

- A state chart diagram (also called a *state diagram*) shows the sequence of states that an object goes through during its life in response to outside stimuli and messages
- The state is set of values that describes an object at specific point in time and represented by state symbols and transitions are by arrows connecting states
- It may contain sub diagrams. It represents the state of a method execution. The activities represent activities of the objects that perform the method

UML STATE-CHART DIAGRAM

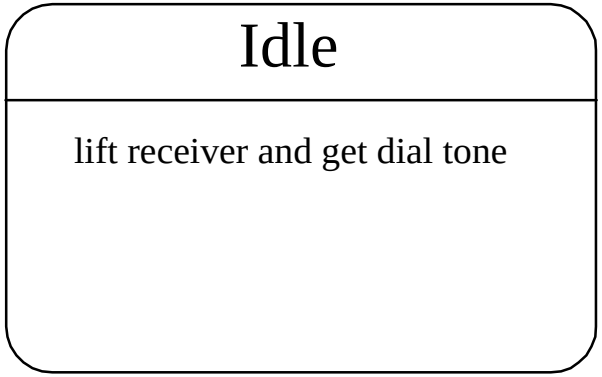
- State is represented as rounded box with optional compartments: *name and the internal transition*
- Compartments are optional
  - Name compartment:
    - Holds optional name of state
    - State without name are anonymous and all are distinct
    - Do not show same named state twice in state diagram because it can be confusing
  - Internal transition compartment:
    - Holds list of internal actions or activities performed in response to events received while object is in the state without changing state
    - ie activities within state symbol indicates execution of operation
    - Syntax is event-name argument-list / action-expression
    - Same operation name may appear more than once indicating

UML STATE-CHART DIAGRAM

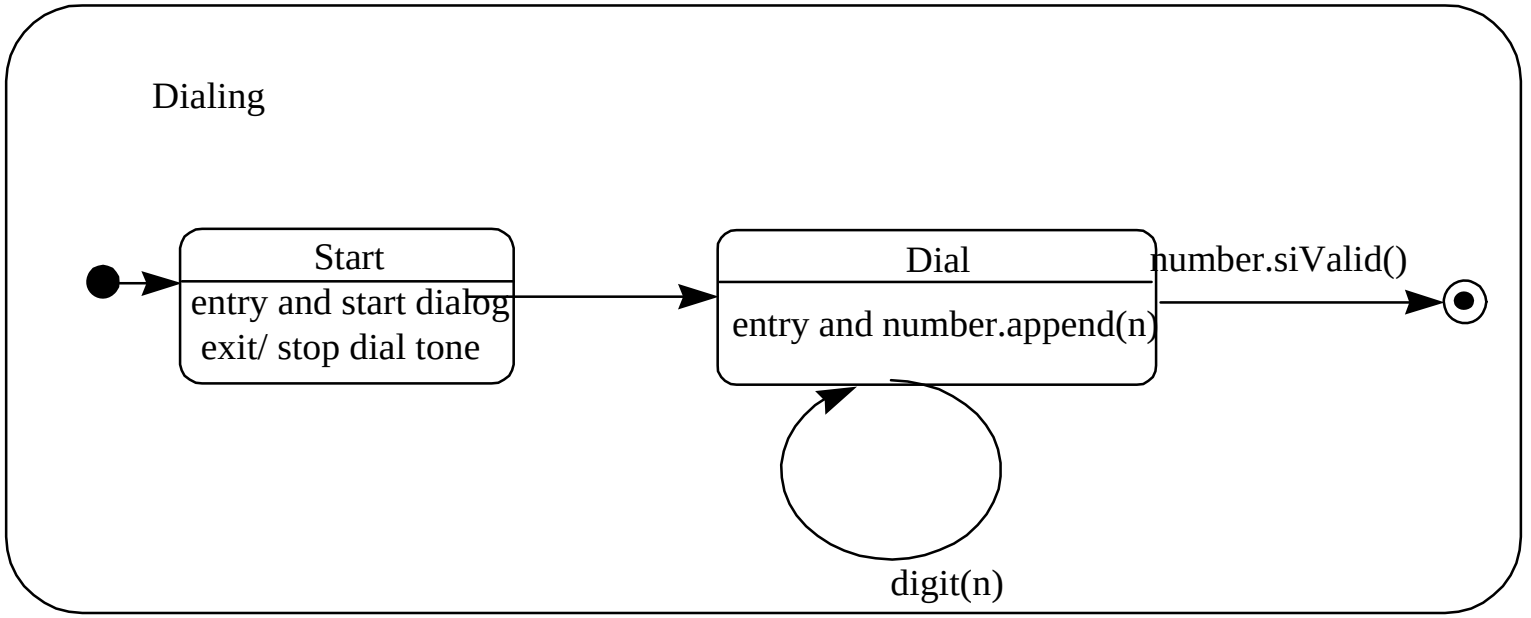
- An outgoing solid arrow indicates a transition triggered by the completion of the activity.
- The syntax is: event-name argument list/ action-expression; e.g help/display help
- Two special events are entry and exit(reserved words : can not be used for event names)
  - Used as: entry/action-expression(the action is to be performed on entry to the state)
  - exit/action-expression(the action is to be performed on exit from the state)

UML Statechart Diagram

- Transition is represented by solid arrow connecting state symbols
- Transition is triggered by completion of activity
- Message is data passed from one object to another
- Message is name that will trigger operation associated with target object



State



Substates

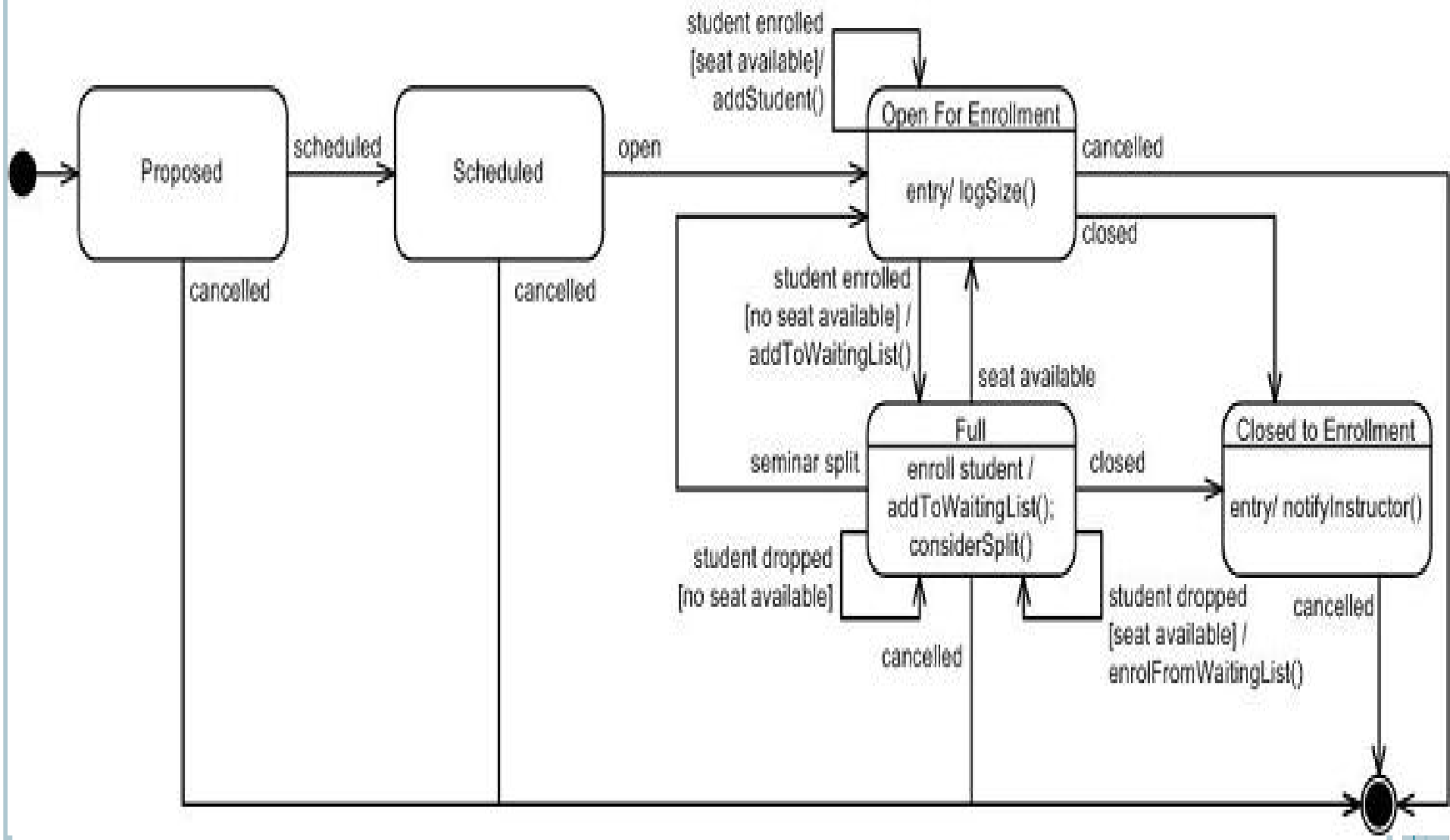
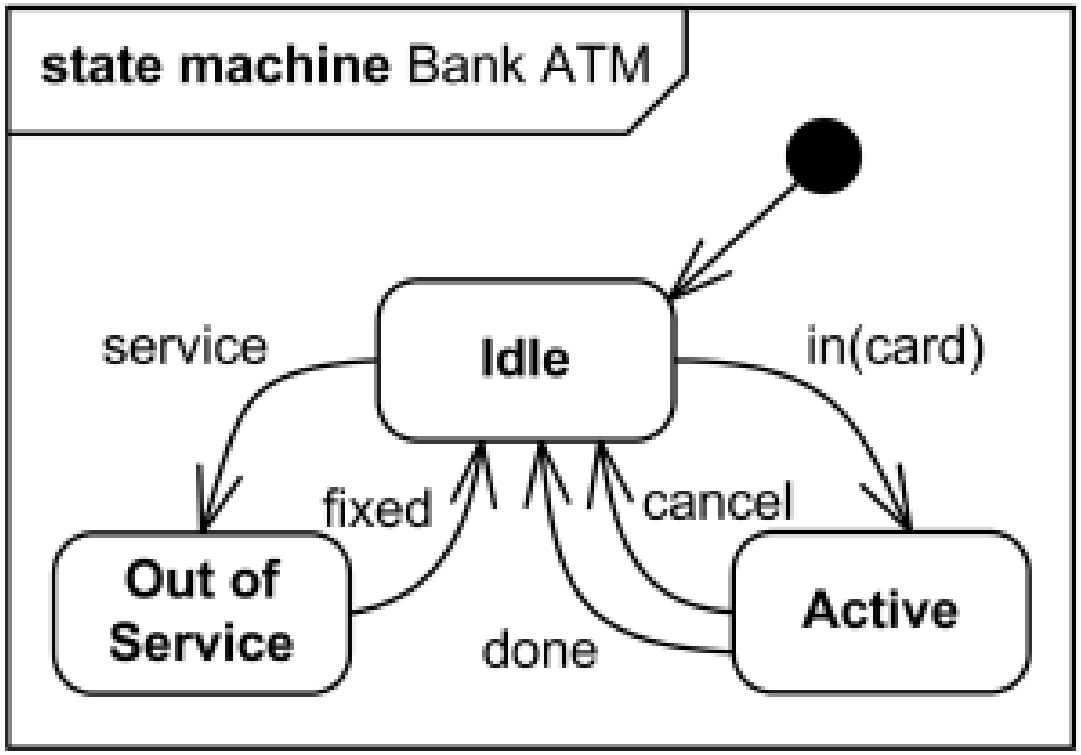


Fig – Statechart diagram of registration during seminar

event [guard][/method list]

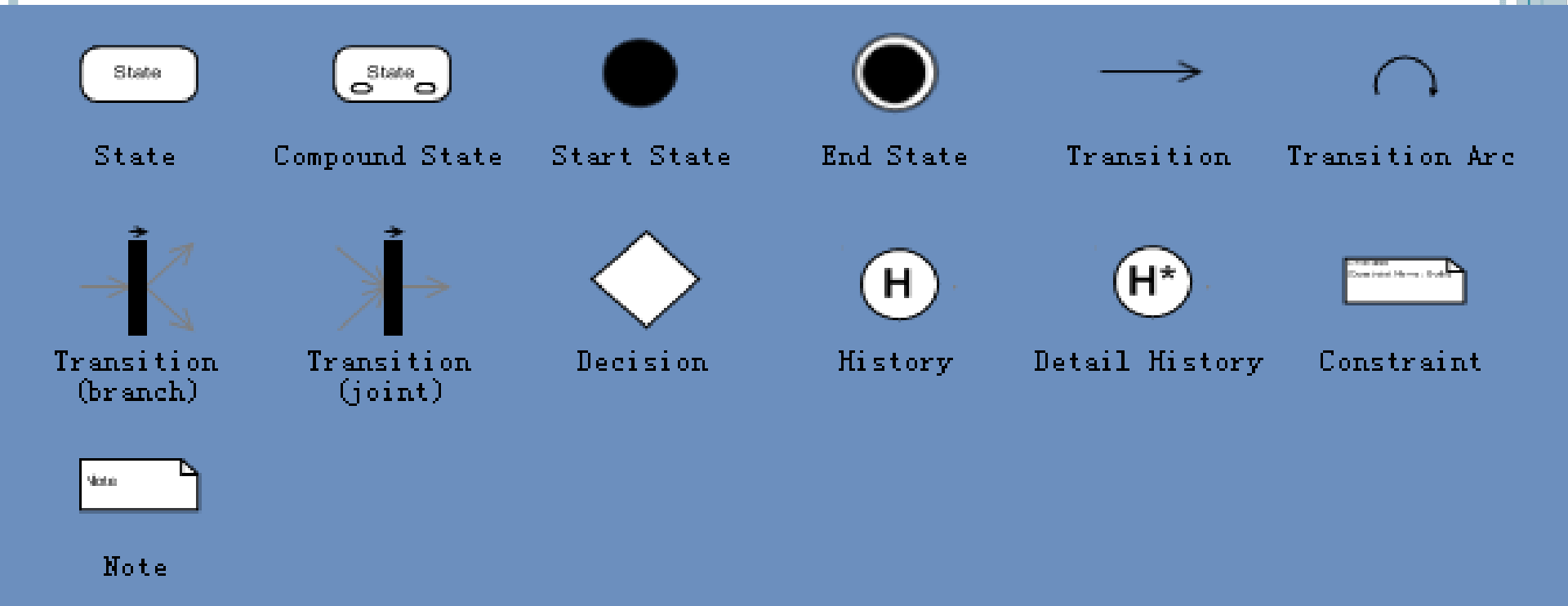
UML STATECHART DIAGRAM

- Purpose of state chart is to understand the algorithm involved in performing the method.



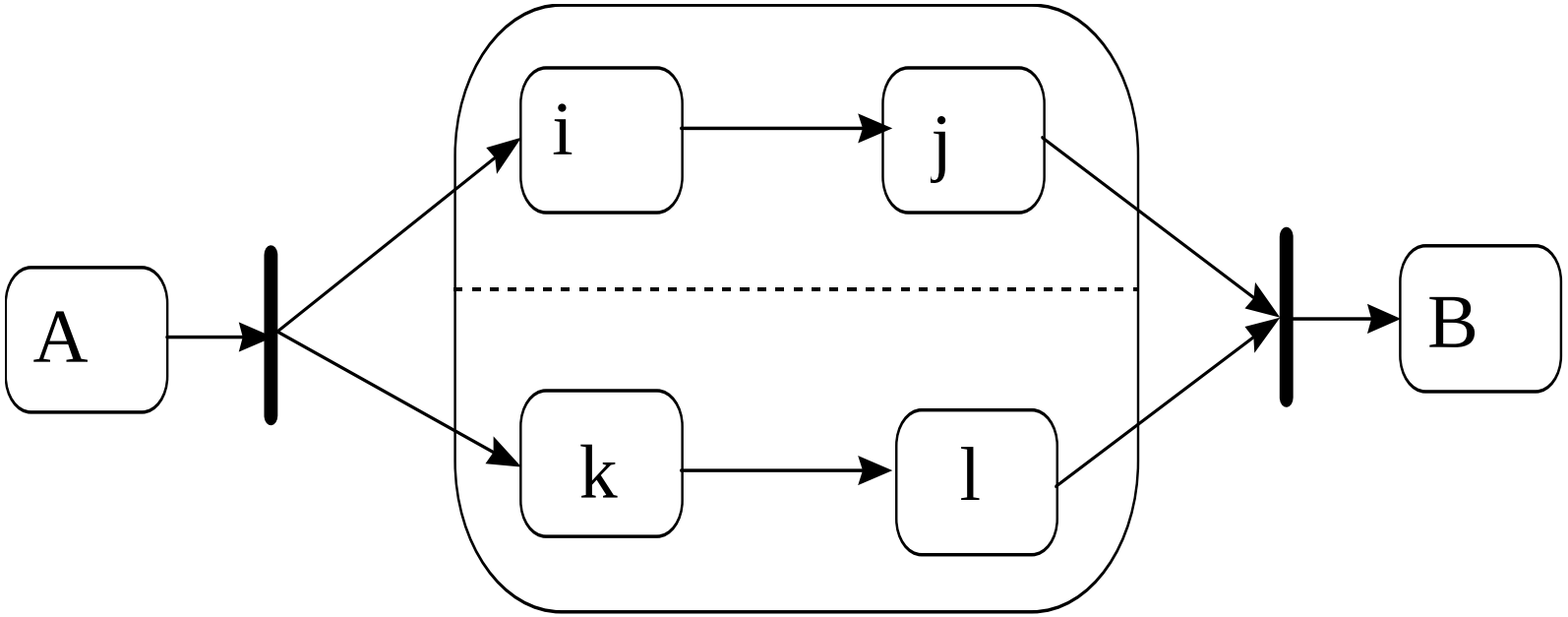
COMPLEX TRANSITION

- Complex transition may have multiple source and target states, with synchronization or splitting of control into concurrent threads.
- A complex transition is shown as a solid heavy bar and may have one or more solid arrows from states(source states) to bars and bars to states(destination states)



COMPLEX TRANSITION

- Concurrent state machines sometimes need to be synchronized with each other
  - start together
  - run through independently until their terminal state
  - re-sychronize at the end

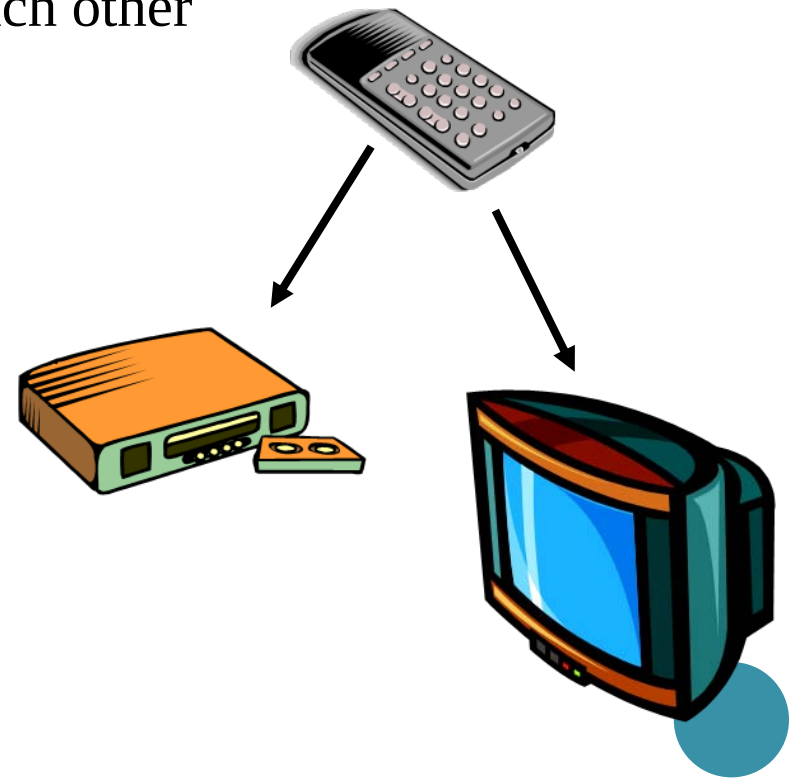


UML Statechart diagram

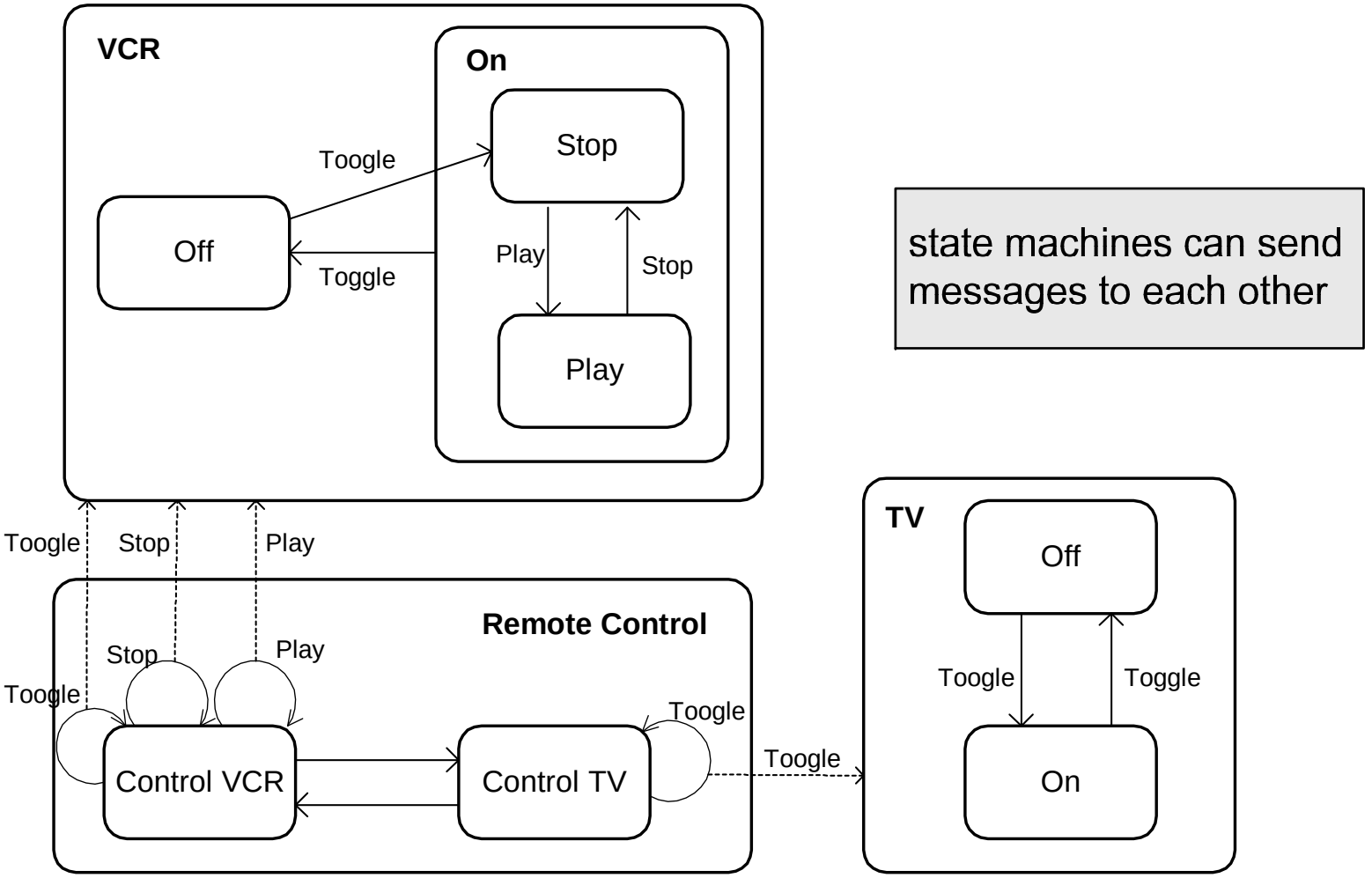
- No need for state diagram of each class
- Useful when class is very dynamic

COOPERATING INDEPENDENT STATES

- Example: remote control of television and VCR
  - three independent systems
  - three independent state machines
  - but clearly they cooperate with each other
- How to model this in UML?



COOPERATING INDEPENDENT STATES



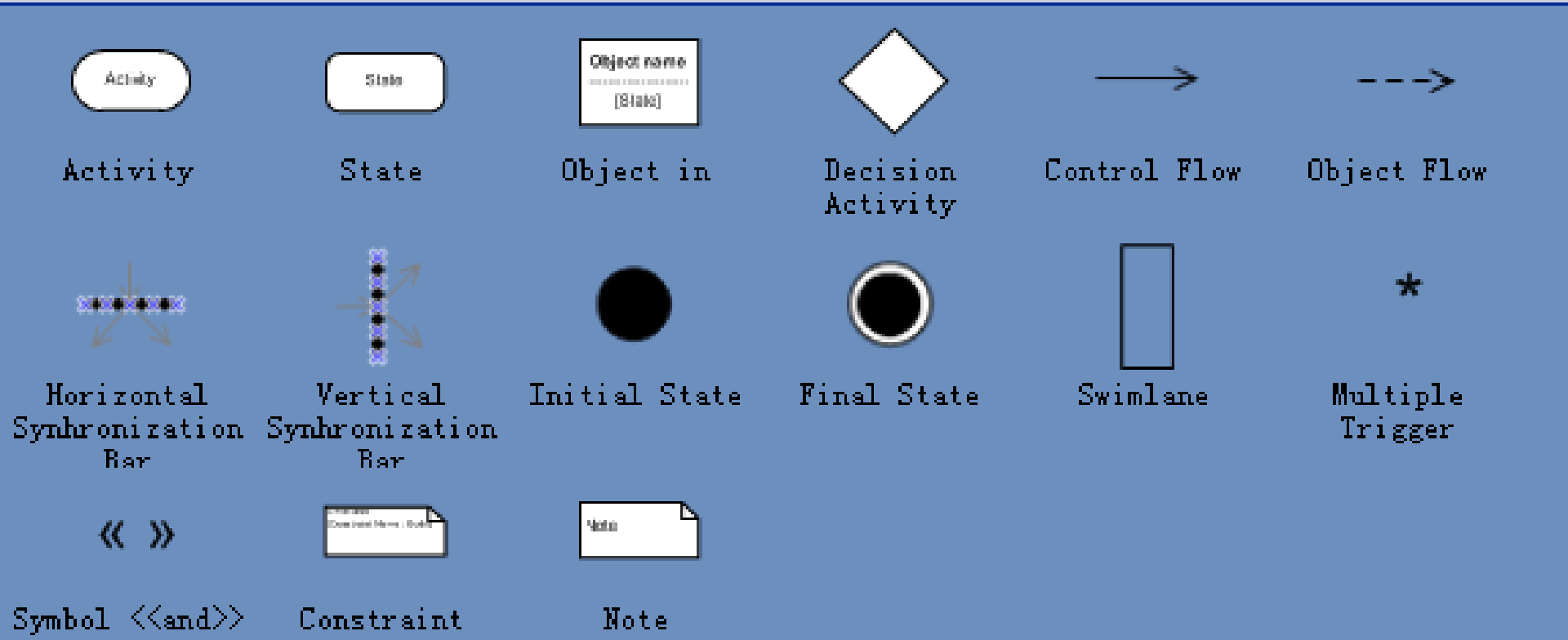
state machines can send messages to each other

UML Activity Diagram

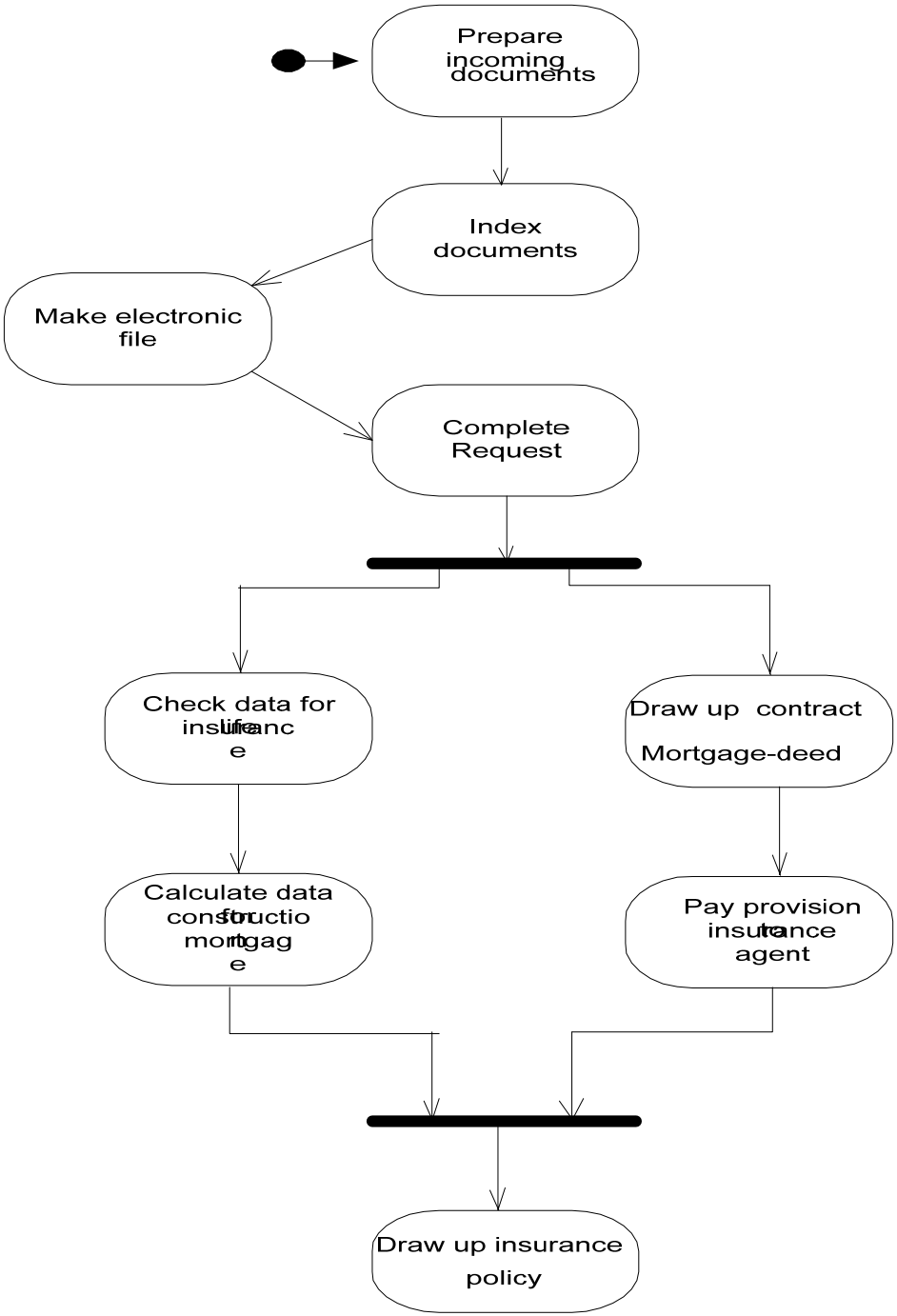
- An activity diagram is a variation or special case of a state machine, in which the states are activities
- Activities represents operation is performed and the transitions are triggered by the completion of the operations.
- State diagrams focus on events occurring in single object
- Activity diagram can be used to model entire business process
- Purpose of activity diagram is to provide view of flows in use case or several classes
- It can also be used to represent class's method implementation

UML ACTIVITY DIAGRAM

- An activity diagram is a variation or special case of a state machine, in which the states are activities representing the performance of operations and the transitions are triggered by the completion of the operations



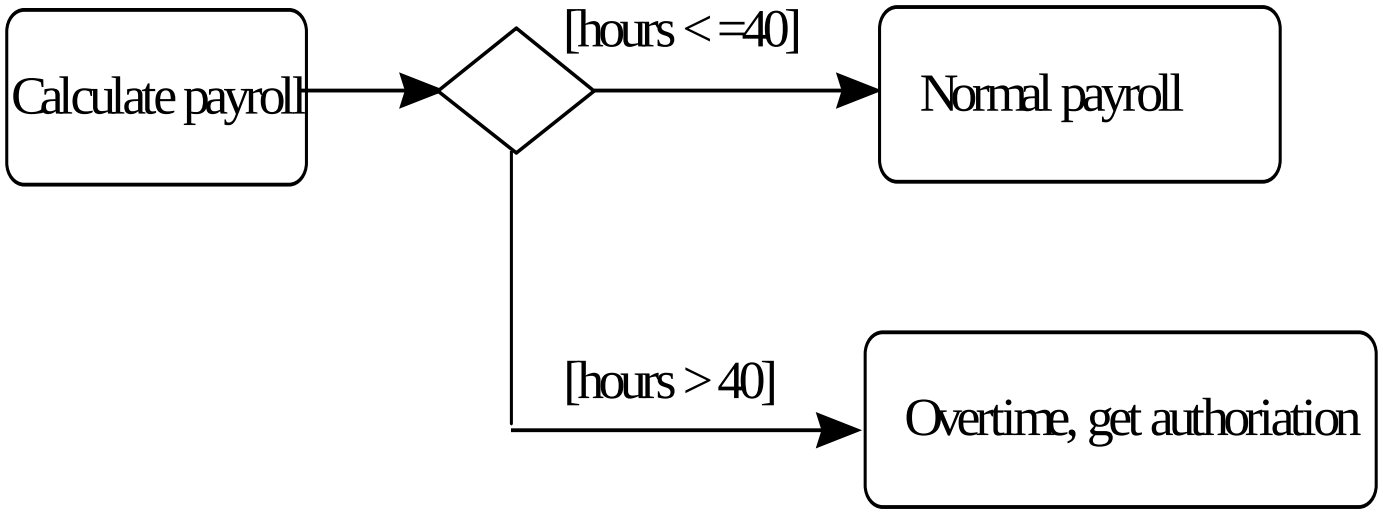
UML ACTIVITY DIAGRAM (CON'T)



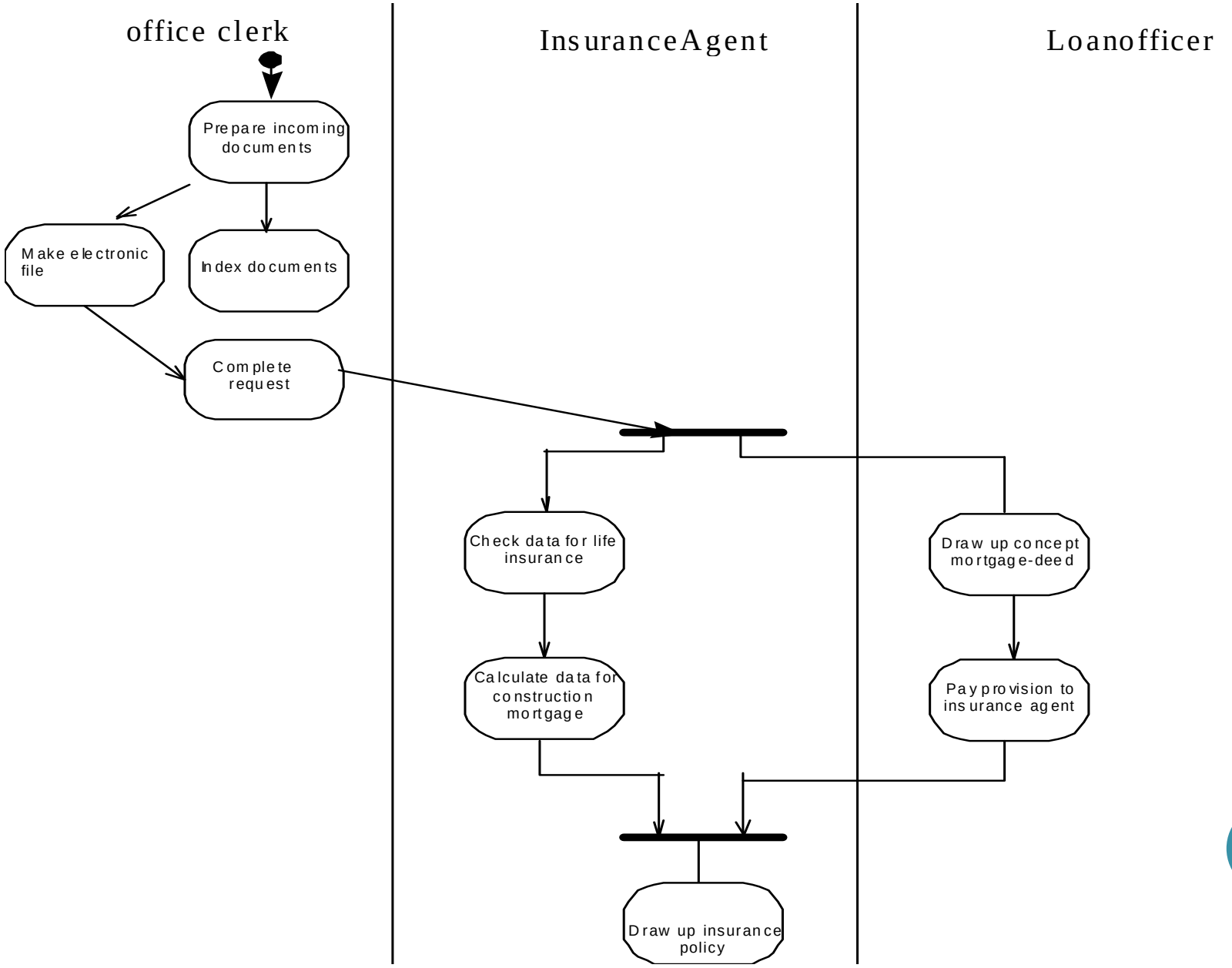
UML ACTIVITY DIAGRAM

- The purpose of activity diagram is to provide a view of flows and what is going on inside a use case or among several classes
- Another use is to represent a class's method implementation

UML ACTIVITY DECISION



SWIMLANES.

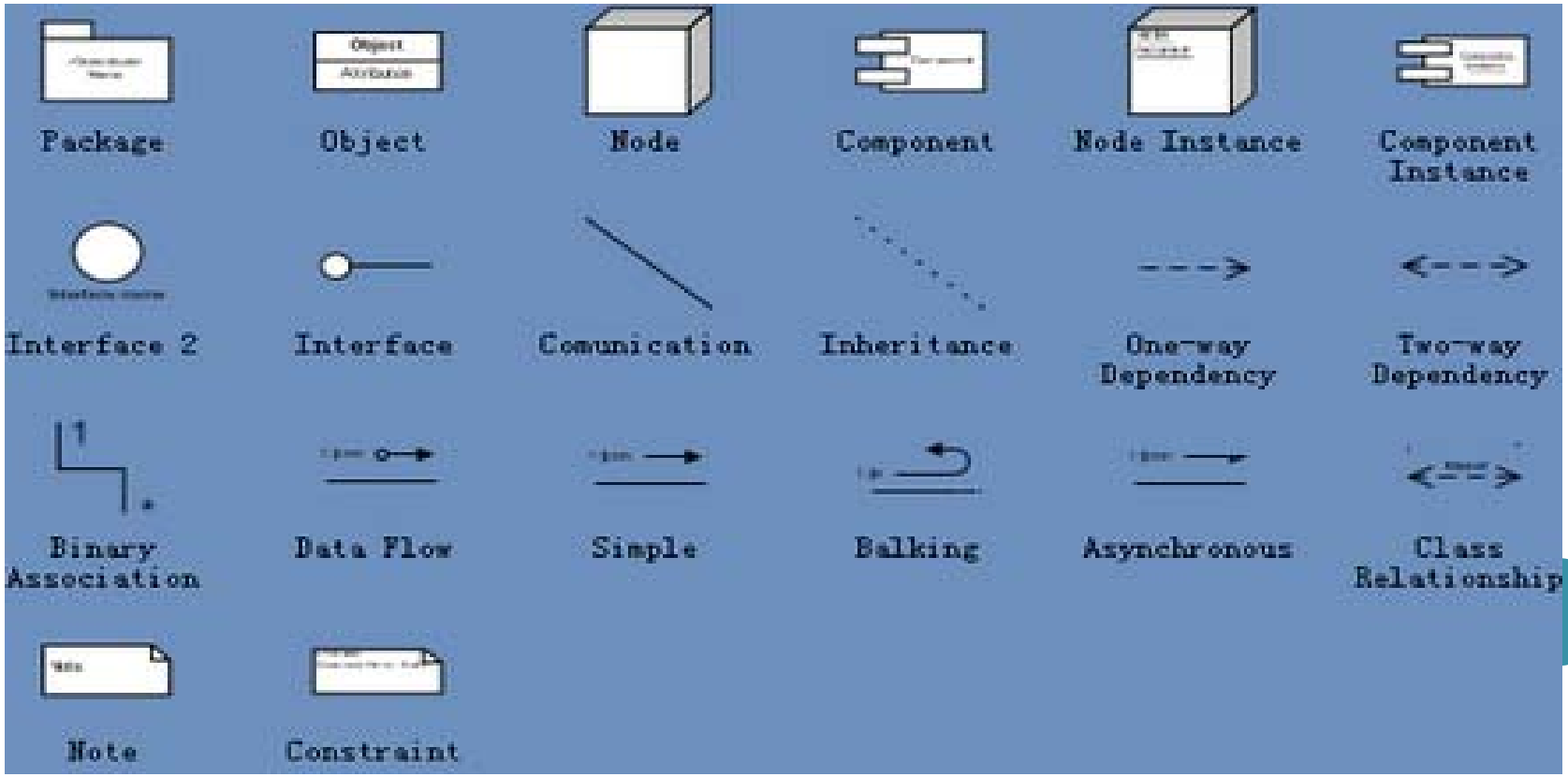

office clerk
InsuranceAgent
Loanofficer
Prepare incoming documents
Make electronic file
Index documents
Complete request
Check data for life insurance
Calculate data for construction mortgage
Draw up concept mortgage-deed
Pay provision to insurance agent
Draw up insurance policy

IMPLEMENTATION DIAGRAMS

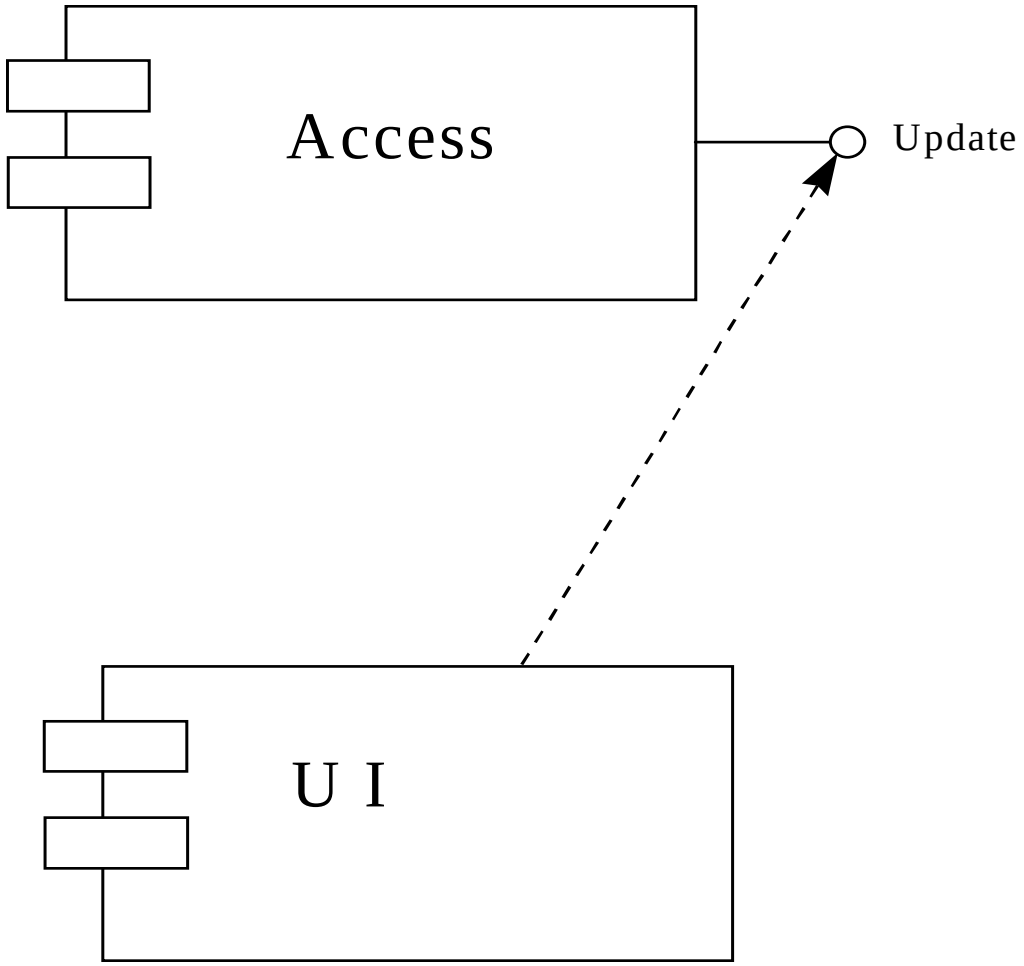
- These diagrams show the implementation phase of systems development
- Such as the source code structure and the run-time implementation structure

IMPLEMENTATION DIAGRAMS (CON'T)

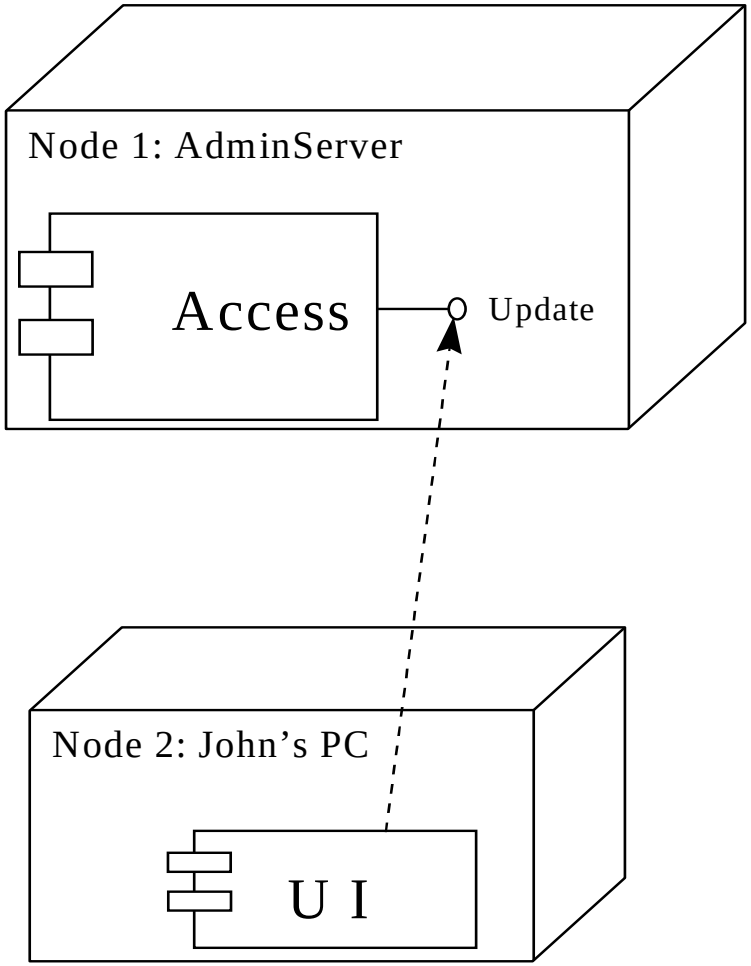
- There are two types of implementation diagrams:
  - Component diagrams show the structure of the code itself
  - Deployment diagrams show the structure of the run-time system



COMPONENT DIAGRAMS



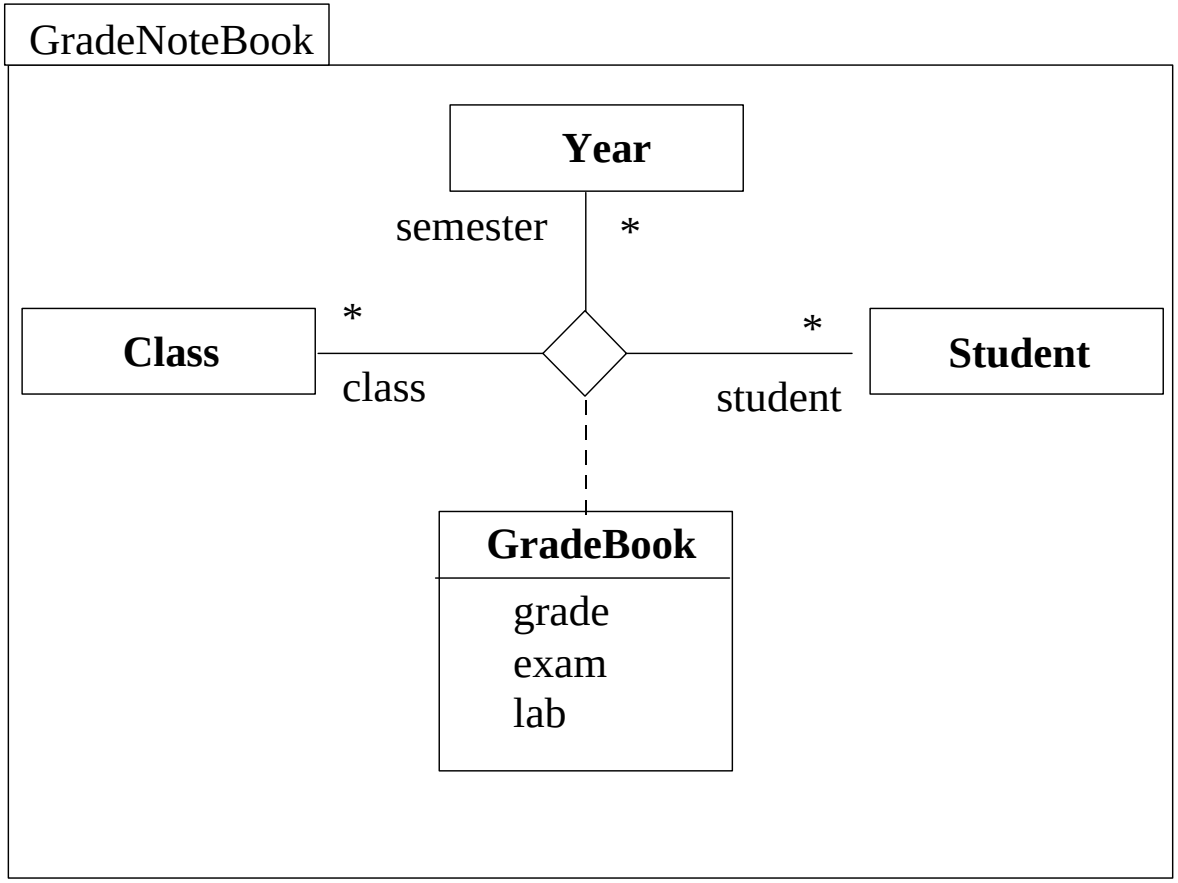
DEPLOYMENT DIAGRAM



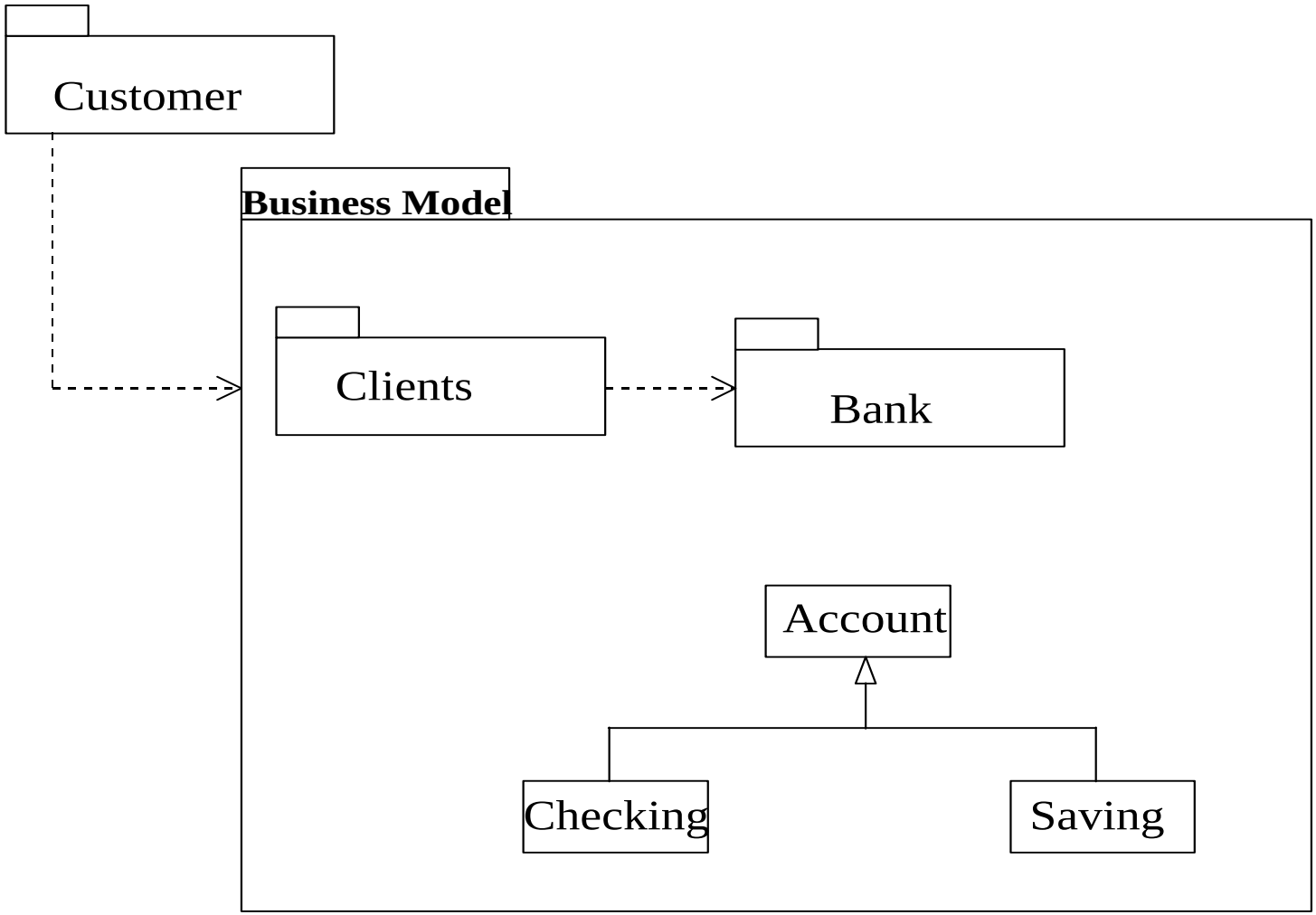
MODEL MANAGEMENT: PACKAGE

- A package is a grouping of model elements
- Packages themselves may contain other packages
- A package may contain both subordinate packages and ordinary model elements

A PACKAGE AND ITS CONTENTS

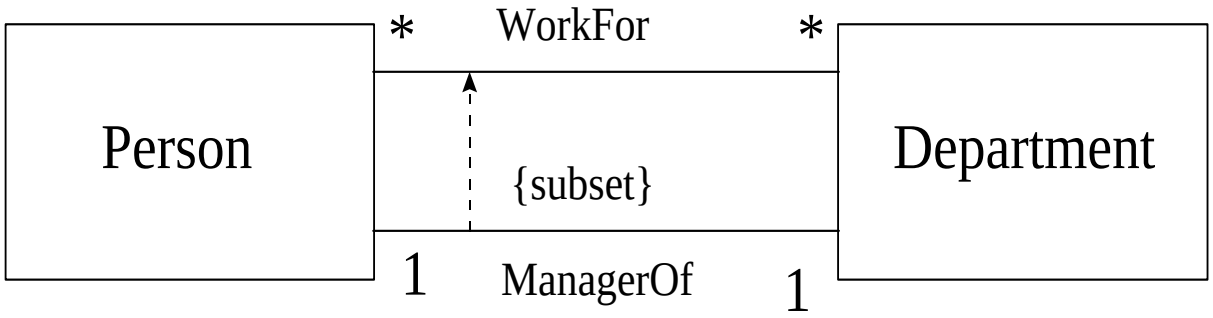


A PACKAGE AND ITS DEPENDENCIES



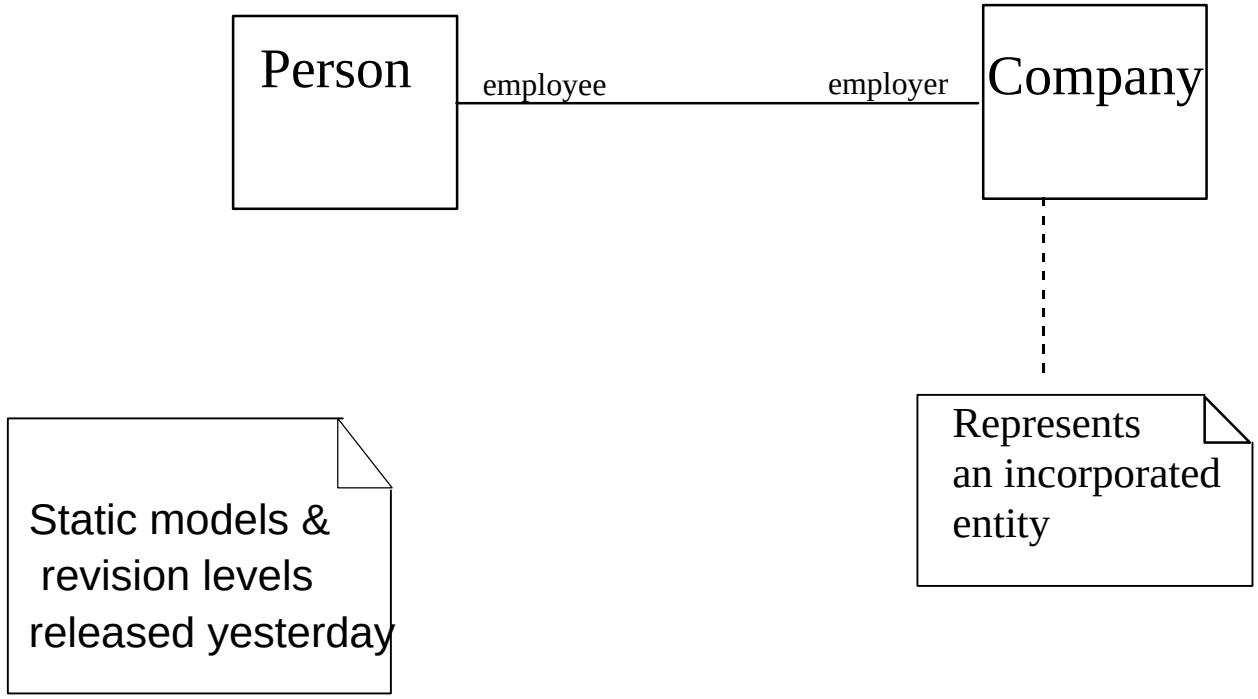
MODEL CONSTRAINTS AND COMMENTS

- Constraints are assumptions or relationships among model elements specifying conditions and propositions that must be maintained as true otherwise the system described by the model would be invalid



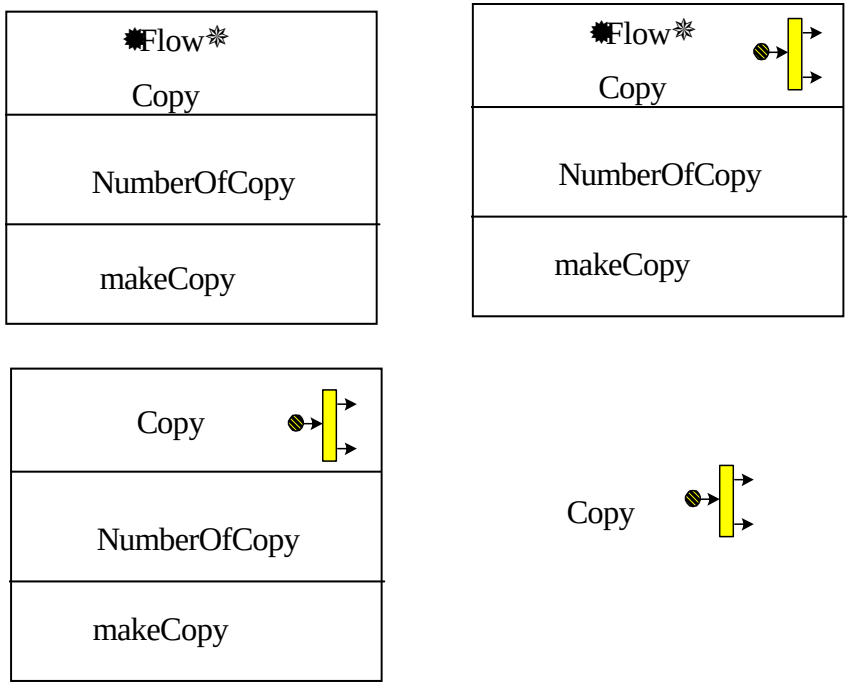
NOTE

- A *note* is a graphic symbol containing textual information; it also could contain embedded images



STEREOTYPE

- *Stereotypes* represent a built-in extendibility mechanism of the UML
- User-defined extensions of the UML are enabled through the use of stereotypes and constraints



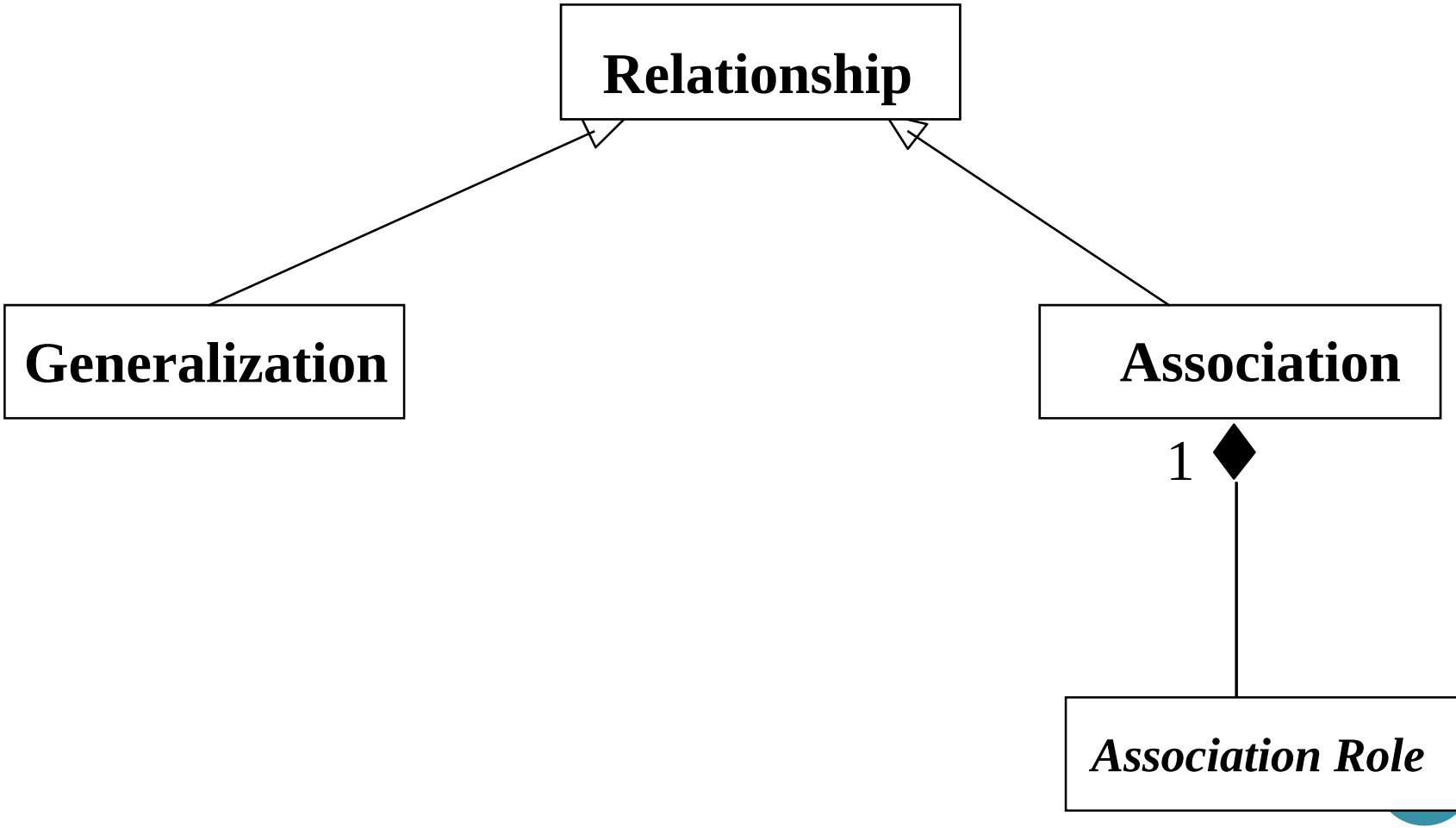
Stereotype

- Stereotype can also be graphic figure, texture and color
- Figure can be used in two ways:
  - Instead of or in addition to stereotype keyword string as part of symbol for base model element
  - Entire base model element
- Limitation – makes model less universal

UML META-MODEL

- Along with systems components, UML notations can be used to describe a model itself, called meta model.
- A meta-model is a model of modeling elements
- The purpose of the UML meta-model is to provide a single, common, and definitive statement of the syntax and semantics of the elements of the UML

UML META-MODEL (CON'T)



SUMMARY

- A model is a simplified representation of reality
- The unified modeling language (UML) was developed by Booch, Jacobson, and Rumbaugh and encompasses the unification of their modeling notations

SUMMARY (CON'T)

- UML consists of the following diagrams:
  - Class diagram
  - Use case diagram
  - Sequence diagram
  - Collaboration diagram

SUMMARY (CON'T)

- Statechart diagram.
- Activity diagram.
- Component diagram.
- Deployment diagram.