Unit II

Static Modeling

Unified Modeling Language

- UML is a standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems.
- UML was created by Object Management Group and UML 1.0 specification draft was proposed to the OMG in January 1997.
- This tutorial gives a complete understanding on UML.

- A model is simplified because reality is too complex or large and much of the complexity actually is irrelevant to the problem we are trying to describe or solve
- A Model provides a means for conceptualization and communication of ideas in a precise and unambiguous form.
- The characteristics of simplification and representation are difficult to achieve in the real world
- Most modeling technique used during many of the phases of the SLC such as analysis, design and implementation

- For example, objectory is build around several different models:
  - Use-case model

    defines the outside(actor) and inside (use-case) of the system's behavior
  - Domain object model

    Objects of real world are mapped into the domain object model
  - Analysis object model

    The analysis object model presents how the source code should be carried out and written
  - Implementation Model

    The implementation model represents the implementation of the system
  - Test model

    The test model constitutes the test plans, specifications, and reports

Static and Dynamic Models

- Models can represent static and dynamic
- Each representation has different implications for how the knowledge about the model might be organized and represented
- Static Model
  - A static model can be viewed as a snapshot of a system's parameters at rest or a specific point in time.
  - Static models are needed to represent the structural static aspect of a system
  - For example a customer could have more than one account or an order could be aggregated from one or more line items.
  - Static models assume stability and an absence of change in data over time .
  - The unified modeling language class diagram is an example of a static model

- Dynamic model
  - In contrast to a static model, can be viewed as a collection of procedures or behaviors that, taken together, reflect the behavior of a system over time.
  - Dynamic relationships show how the business objects interacts to perform tasks.
  - For example, an order interacts with inventory to determine product availability.
  - The UML interaction diagram and activity models are examples of UML dynamic models

Why Modeling?

- Building a model for a software system prior to its construction is as essential as having a blueprint for building a large building.
- Good models are essential for communication among project teams.
- A modeling language must include
  - Model elements – fundamental modeling concepts and semantics
  - Notation – visual rendering of model elements
  - Guidelines – expression of usage within the trade

- The use of visual notation to represent or model a problem can provide us several benefits relating to
  - Clarity : we are much better at picking out errors and omissions from a graphical or visual representation than from listing of code or tables of numbers.
  - Familiarity : The representation form for the model may turn out to be similar to the way in which the information actually is represented and used by the employee currently working in the problem domain.
  - Maintenance : visual notation can improve the maintainability of a system.
  - Simplification : Use of a higher level representation generally results in the use of fewer but more general constructs, contributing to simplicity and conceptual understanding

Advantages of modeling

- Models make it easier to express complex ideas
- Reduction of complexity
- Enhance and reinforce learning and training
- Cost is low
- Manipulation is much easier

Key ideas of modeling

- A model is rarely correct on the first try
- Always seek the advice and criticism of others. You can improve a model by reconciling different perspective
- Avoid excess model revision, as they can distort the essence of your model.

Introduction to the UML

- The UML is a languague for specifying, constructing, visualizing and documenting the software system and its components.
- The UML is a graphical language with sets of rules and semantic
- The rules and semantics of a model are expressed in english, in a form known as object constraint language(OCL).
- OCL is a specification language that uses simple logic for specifying the properties of a system.
- The UML is not intended to be a visual programming language in the sense of having all the necessary visual and semantic support to replace programming languages.

- The primary goal in the design of the UML were
  - Provide users a ready –to-use, expressive visual modeling language develop and exchange meaningful models
  - Provide extensibility and specialization mechanisms to extend the core concepts
  - Be independent of particular programming languages and development processes
  - Provide a formal basis for understanding the modeling language
  - Encourage the growth of the OO tools market
  - Support higher-level development concepts
  - Integrate best practices and methodologies

UML Diagrams

- Every complex system is best approached through a small set of nearly independent views of a model; no single view is sufficient.
- The UML defines nine graphical diagrams
  - Class diagram(static)
  - Use-case diagram
  - Behavior diagram(Dynamic)
    - Interaction diagram
      - Sequence diagram
      - Collaboration diagram
    - Statechart diagram
    - Activity diagram
  - Implementation diagram
    - Component diagram
    - Deployment diagram

Use Case Diagram

- Used for describing a set of user **scenarios**
- Mainly used for capturing user requirements
- Work like a **contract** between end user and software developers

Use Case Diagram (core components)

Actors: A role that a user plays with respect to the system,including human users and other systems. e.g.,inanimate physical objects (e.g. robot); an external system that needs some information from the current system.

Use case: A set of scenarios that describing an interaction between a user and a system, including alternatives.

Actor

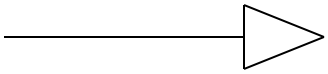
Use Case

System boundary: rectangle diagram representing the boundary between the actors and the system.

Use Case Diagram(core relationship)

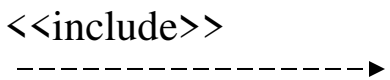
Association: communication between an actor and a use case; Represented by a solid line.

Generalization: relationship between one general use case and a special use case (used for defining special alternatives)
Represented by a line with a triangular arrow head toward the parent use case.



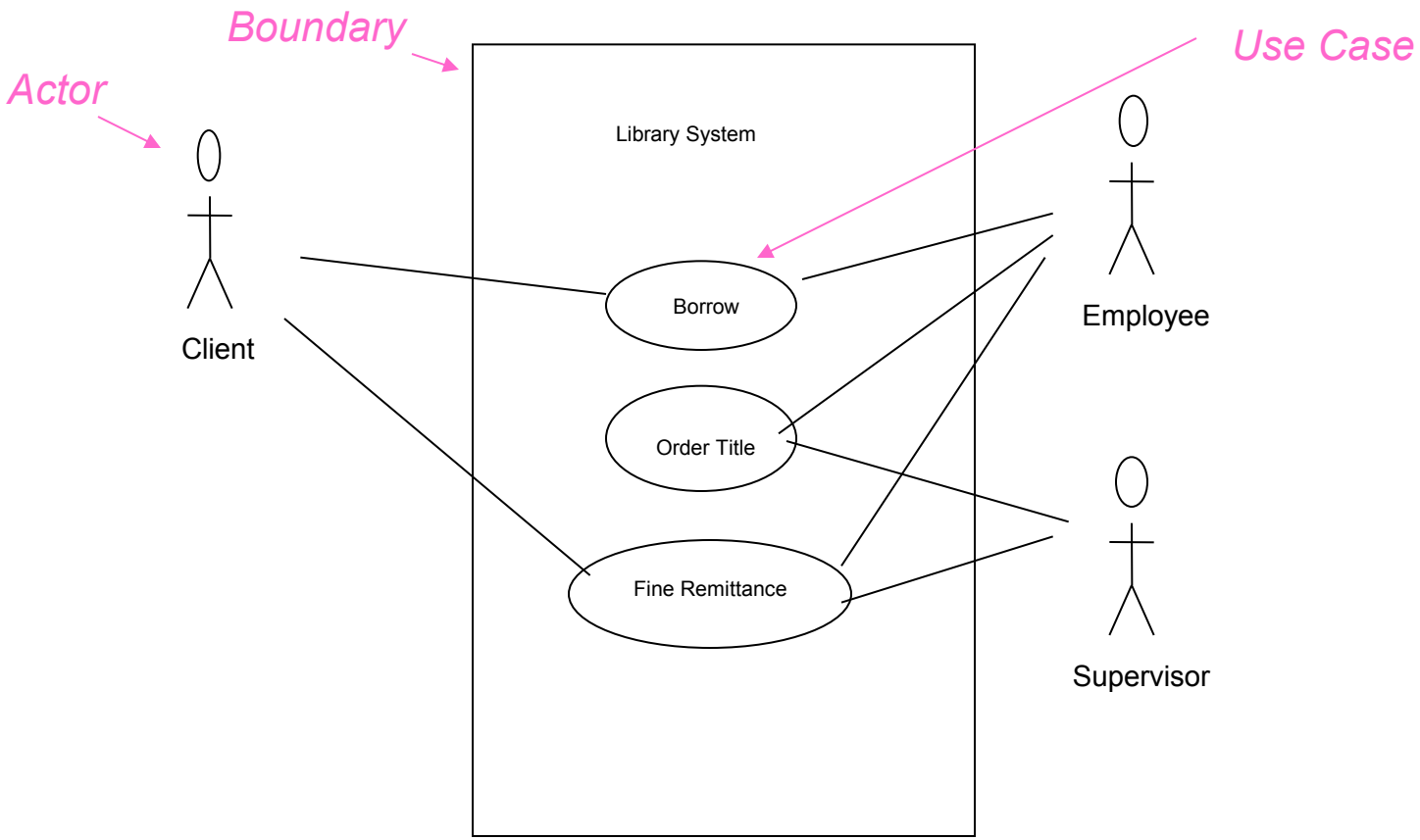
Use Case Diagram(core relationship)

Include: a dotted line labeled <<include>> beginning at base use case and ending with an arrows pointing to the include use case. The include relationship occurs when a chunk of behavior is similar across more than one use case. Use "include" in stead of copying the description of that behavior.



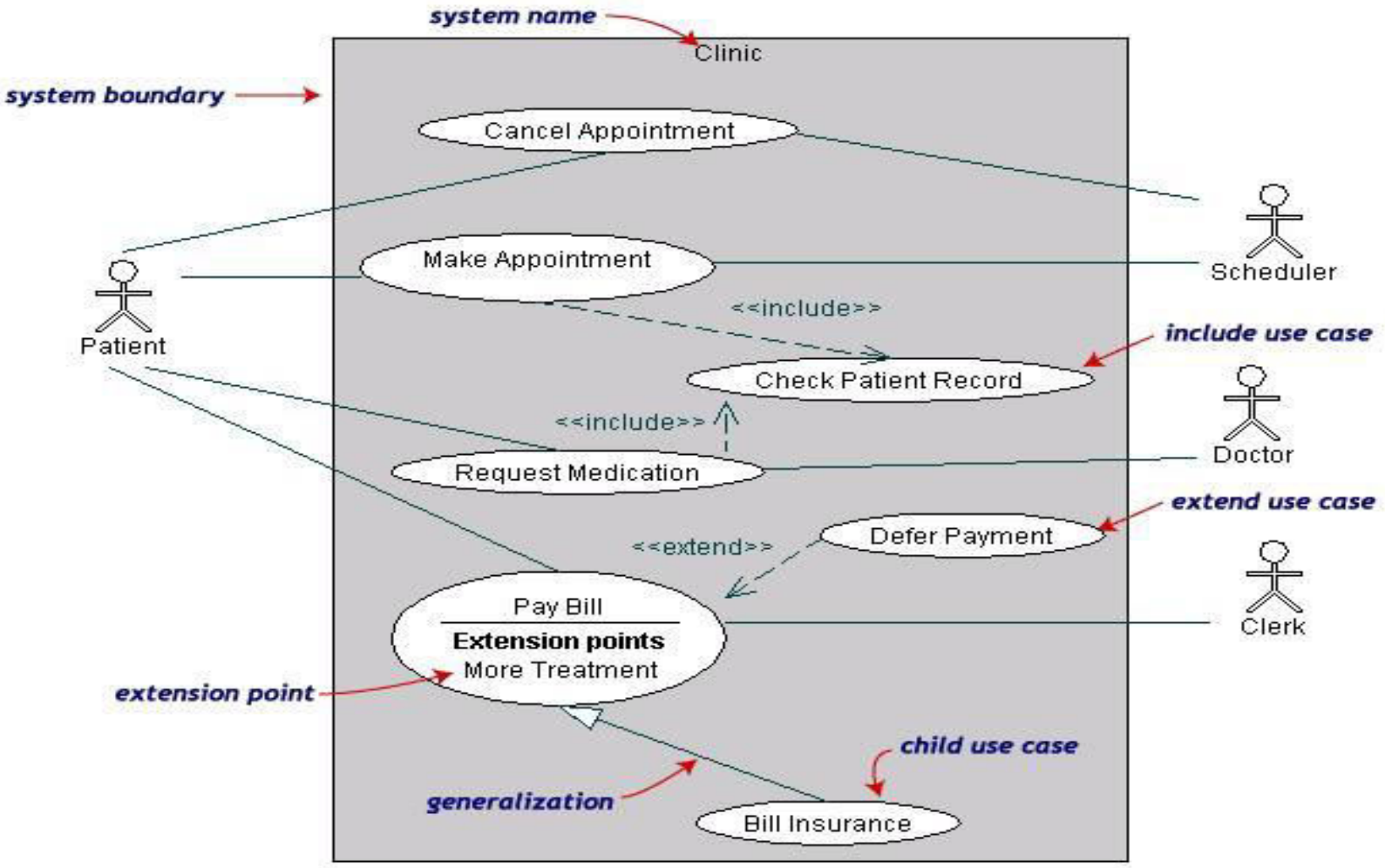
Extend: a dotted line labeled <<extend>> with an arrow toward the base case. The extending use case may add behavior to the base use case. The base class declares "extension points".

Use Case Diagrams



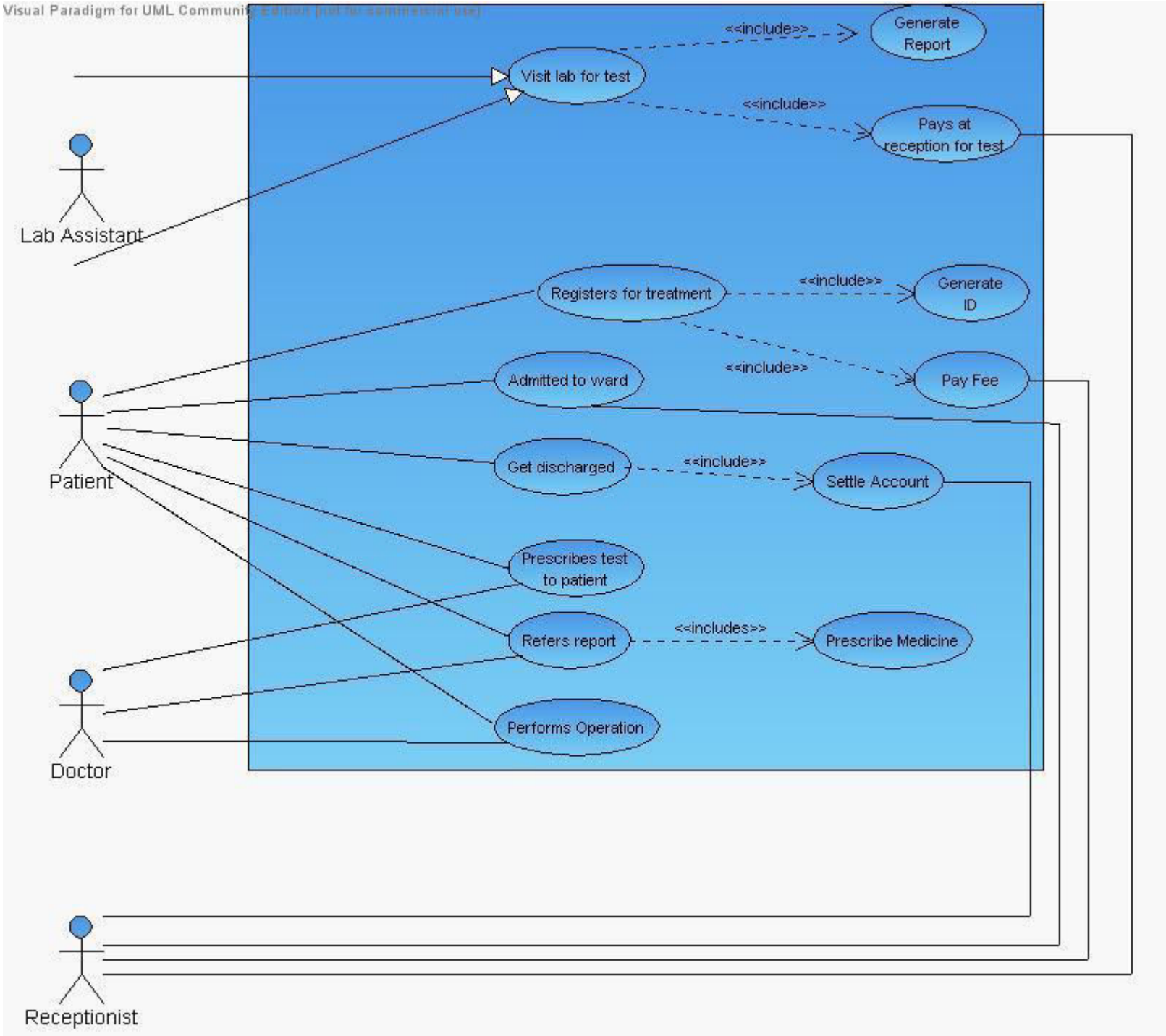
- A generalized description of how a system will be used.
- Provides an overview of the intended functionality of the system

Use Case Diagrams(cont.)



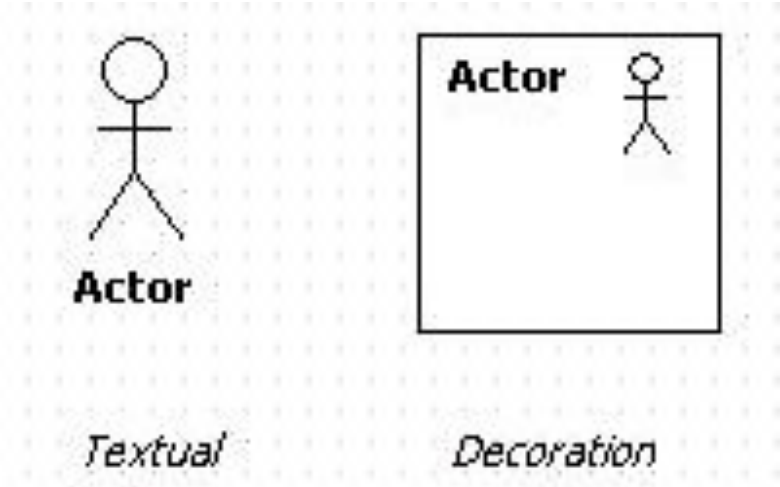
(TogetherSoft, Inc)

Visual Paradigm for UML Community Edition [not for commercial use]



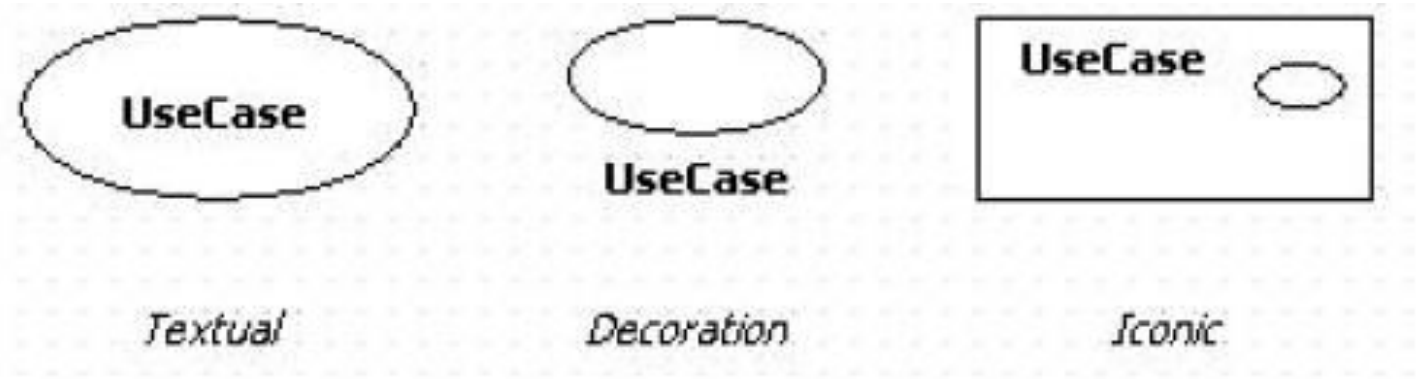
Procedure for creating Actor

- In order to create Actor, click [Toolbox] -> [UseCase] -> [Actor] button and click the position where to place Actor. Actor is shown in the form of stick man or rectangle with icon, that is decoration view. To display actor in decoration view, select [Format] -> [Stereotype Display] -> [Decoration] menu item or select [Decoration] item in combo button on toolbar.



Procedure for creating UseCase

- In order to create UseCase, click [Toolbox] -> [UseCase] button and click the position where to place UseCase on the [main window].
- UseCase is expressed in the forms of textual, decoration, iconic. To change UseCase's view style, select menu item under [Format] -> [Stereotype Display] or select button's combo item.

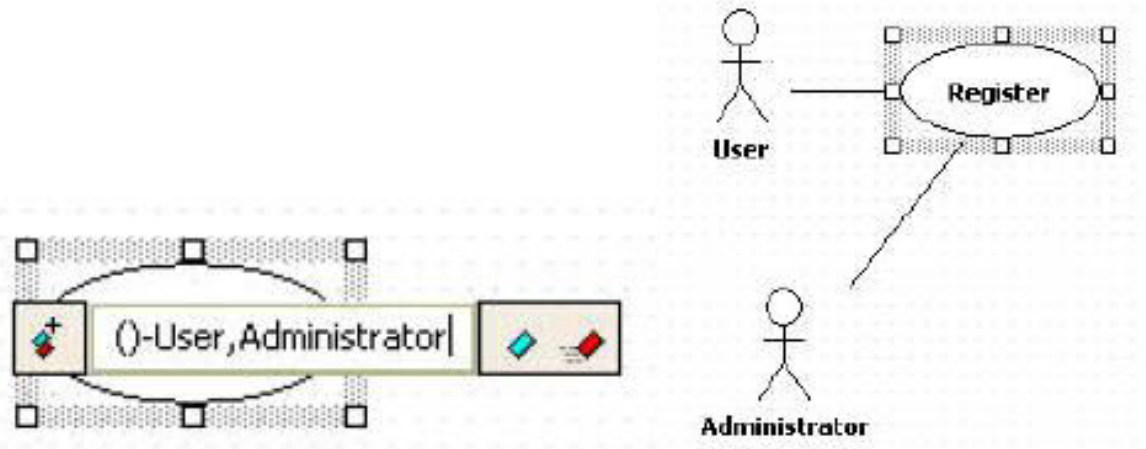


Procedure for creating Actor from UseCase

- In order to create multiple Actors related to UseCase at once, use shortcut creation syntax.

  1. Double-click UseCase, or select UseCase and press [Enter] key. At quick dialog, enter Actor's name after "()-" string and separate Actor names by "," character.

  2. And press [Enter] key. Several Actors associated with the UseCase are created and arranged vertically.

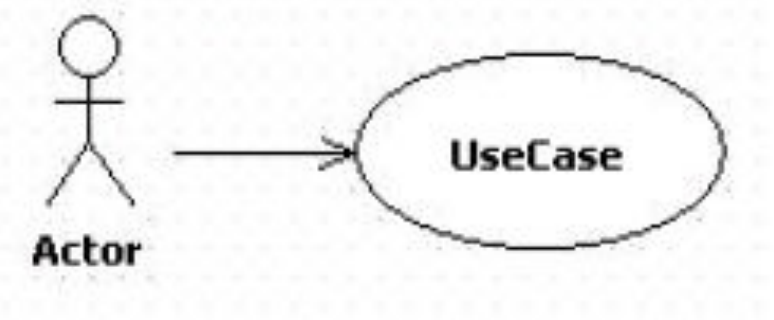


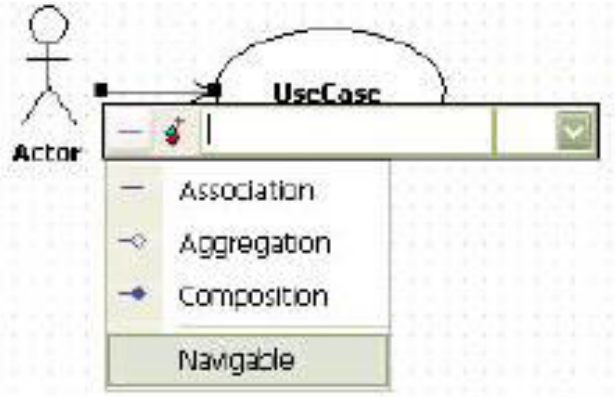
Procedure for creating association

- In order to create association, click [Toolbox] -> [UseCase] -> [Association] button, drag from first element, and drop to second element in the [main window].

Procedure for creating directed association

- The procedure is equal to the association's, drag and drop in the arrow direction.
- Or create association, click the actor-side association end. At the quick dialog, uncheck navigable and association becomes directed.



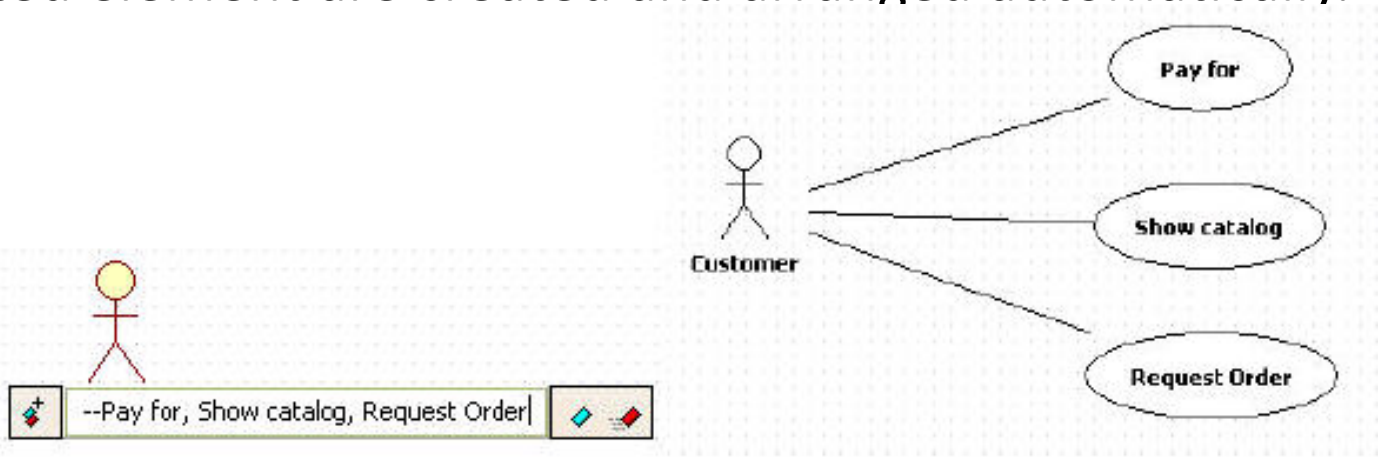


Procedure for creating element related to association/directed associtition

- In order to create element associated with current element, use shortcut creation syntax.

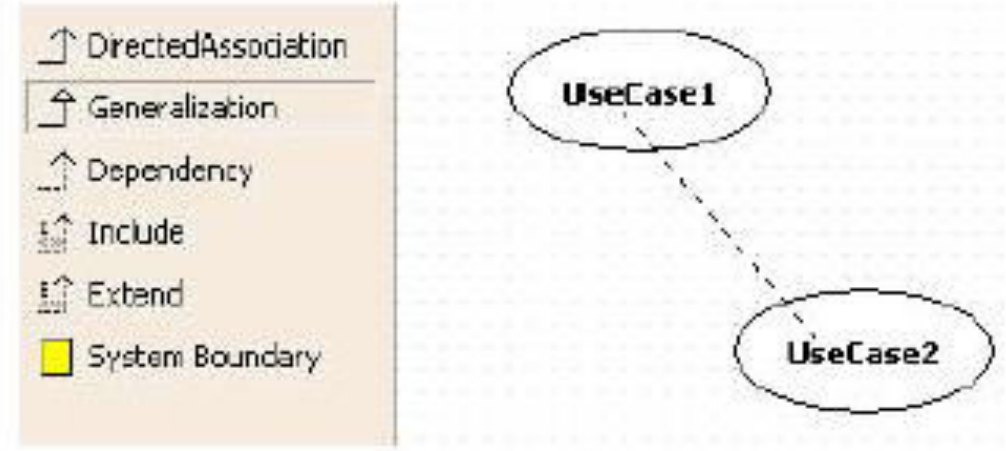
  1. Double-click element and enter element's names associated after "--" or "->" string at the quick dialog. Separate element names with "," character to relate multiple elements.

  2. Press [Enter] key and several elements associated with selected element are created and arranged automatically.



Procedure for creating generalization

- In order to make generalization, click [Toolbox] -> [UseCase] ->[Generalization] button, drag from child element and drop to parent element in the [main window].

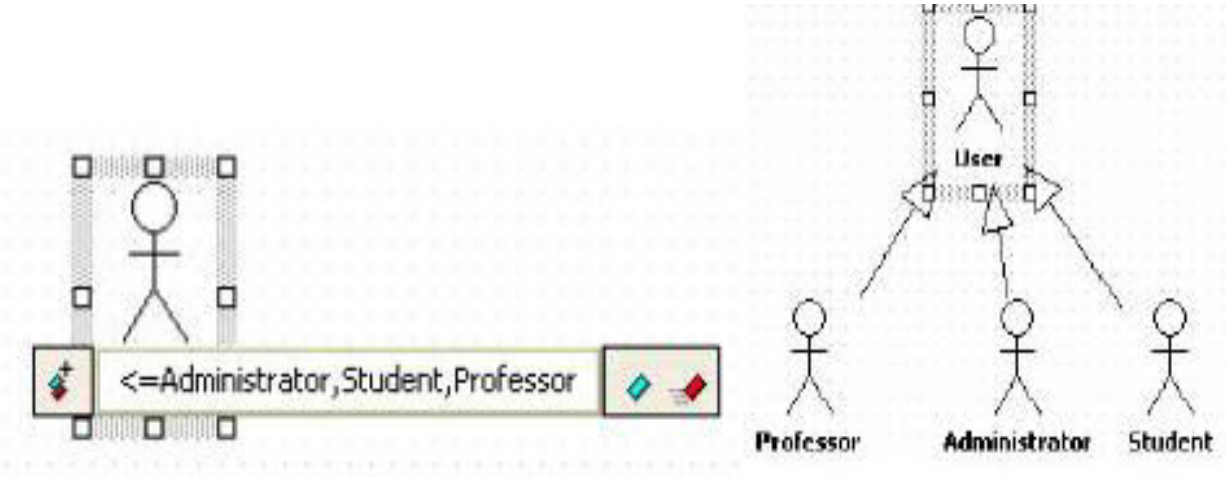


Procedure for creating multiple child actors inherited from actor

- To create multiple elements inherited from some element,

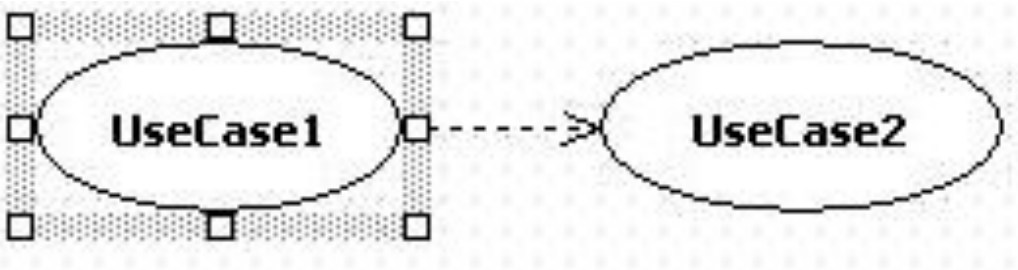
  1. Enter with "<=" string as following at the quick dialog, and several elements inherited from selected element are created at once.

  2. Child elements are generated below selected element and arranged automatically.



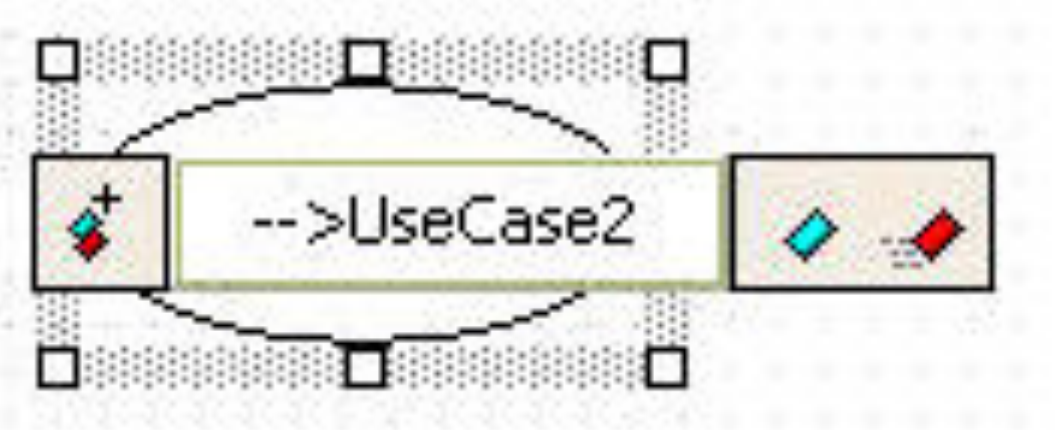
Procedure for creating dependency

- In order to create dependency, click [Toolbox] -> [UseCase] -> [Dependency] button, drag element and drop to other element depended



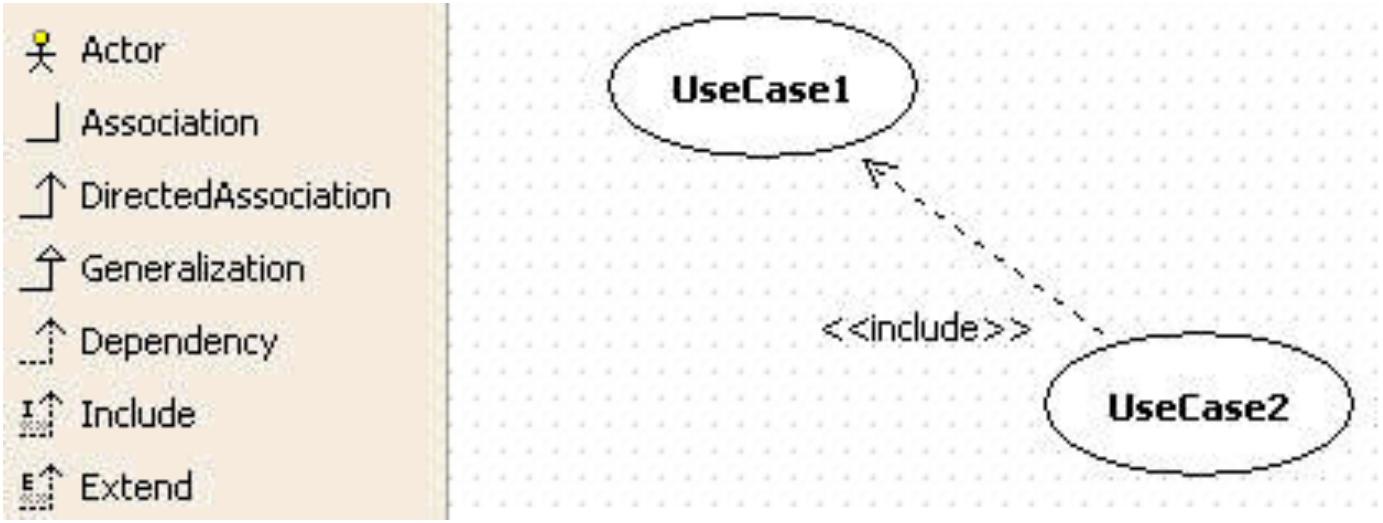
Procedure for creating other usecase depended by current usecase

- Enter with "-->" string at the quick dialog as following.
- So dependency relationship is created between two elements.



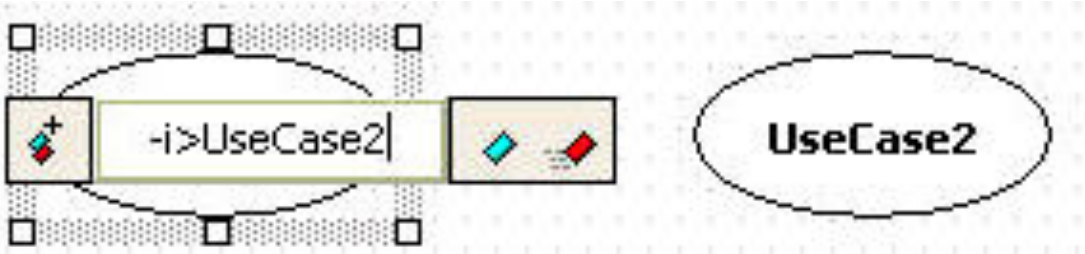
Procedure for creating include

- In order to create include relationship, click [Toolbox] -> [UseCase] -> [Include] button, drag from element including and drop to element included in the [main window].

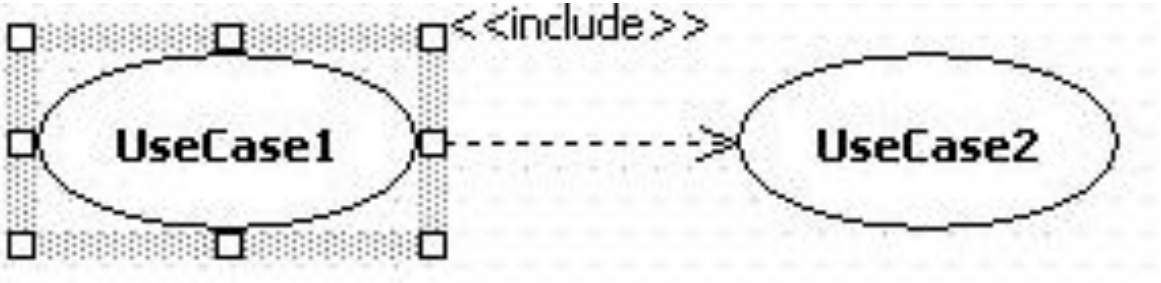


Procedure for creating other usecase included by current usecase

- Enter with "-i>" string at the quick dialog as following

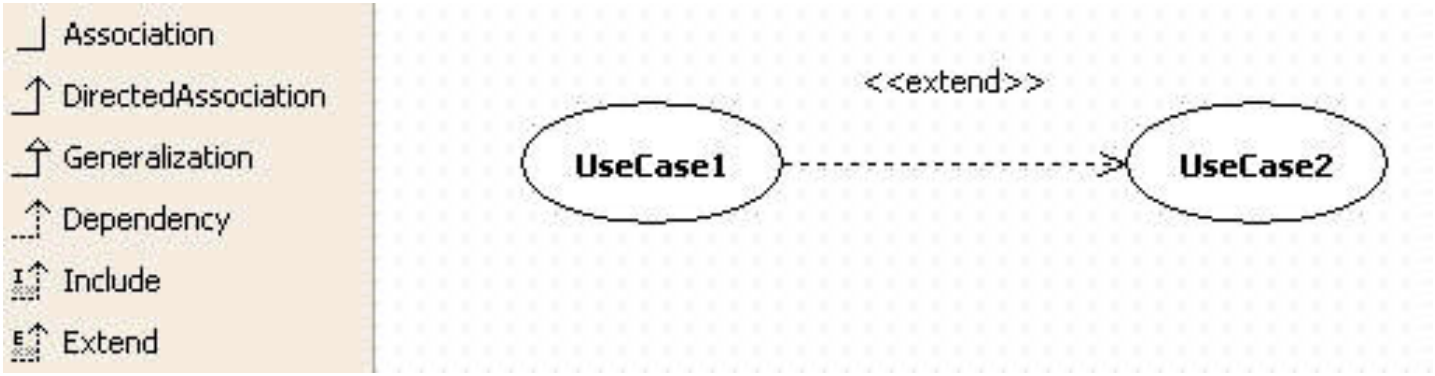


- So include relationship is created between two elements.



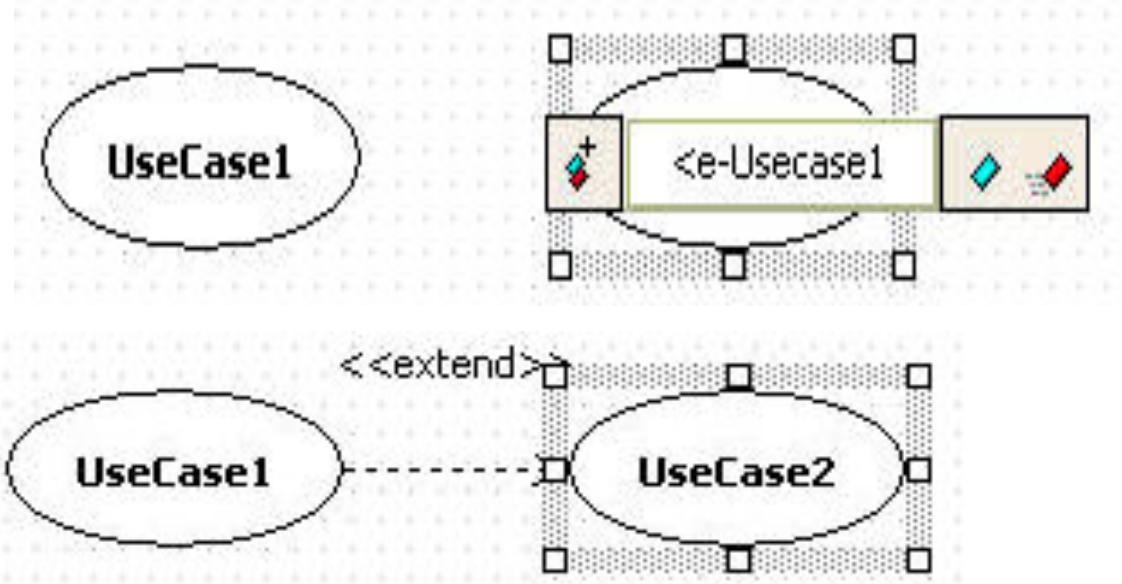
Procedure for creating extend

- In order to create extend, click [Toolbox] -> [UseCase] -> [Extend] button, drag from element extending and drop to element extended in the [main window].



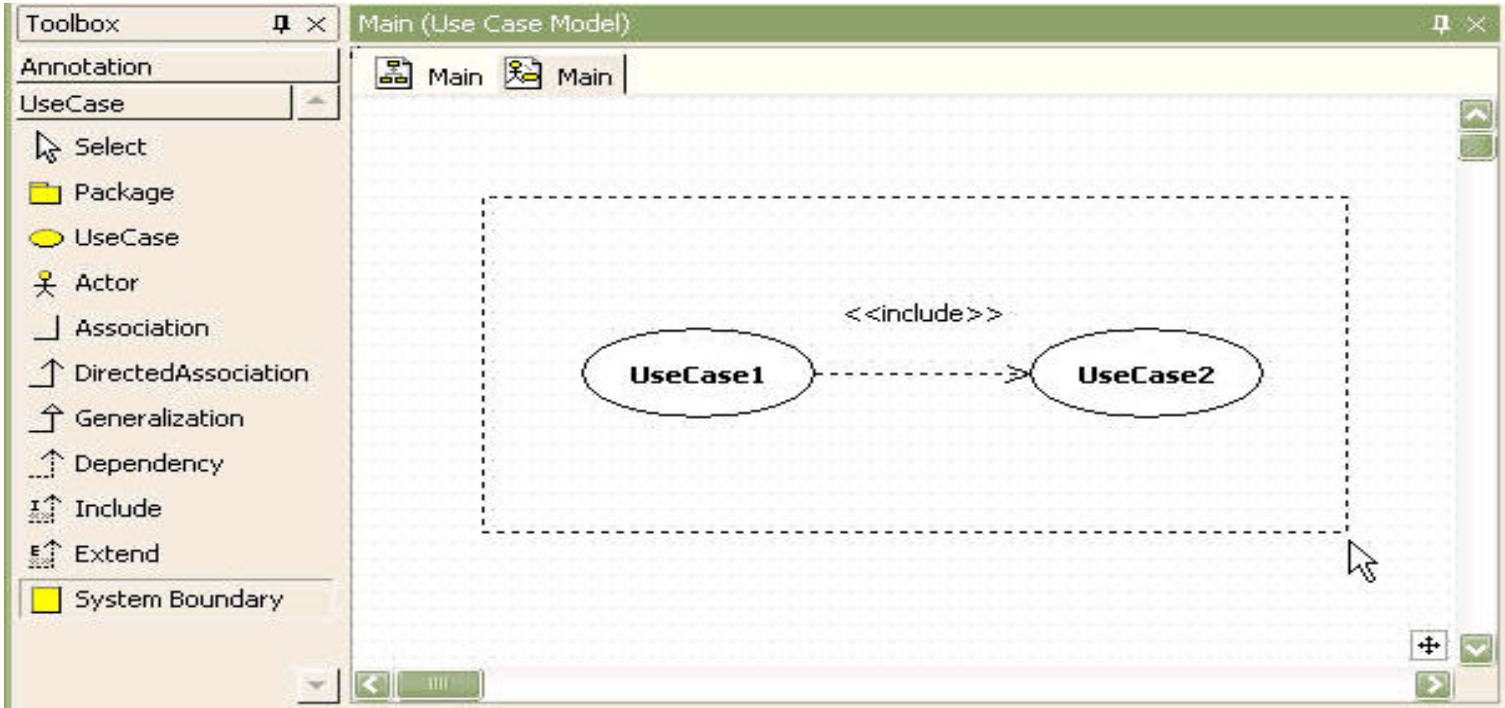
Procedure for creating other usecase extending current usecase

- Enter with "<e-" string at the quick dialog as following.
- So extend relationship is created between two elements



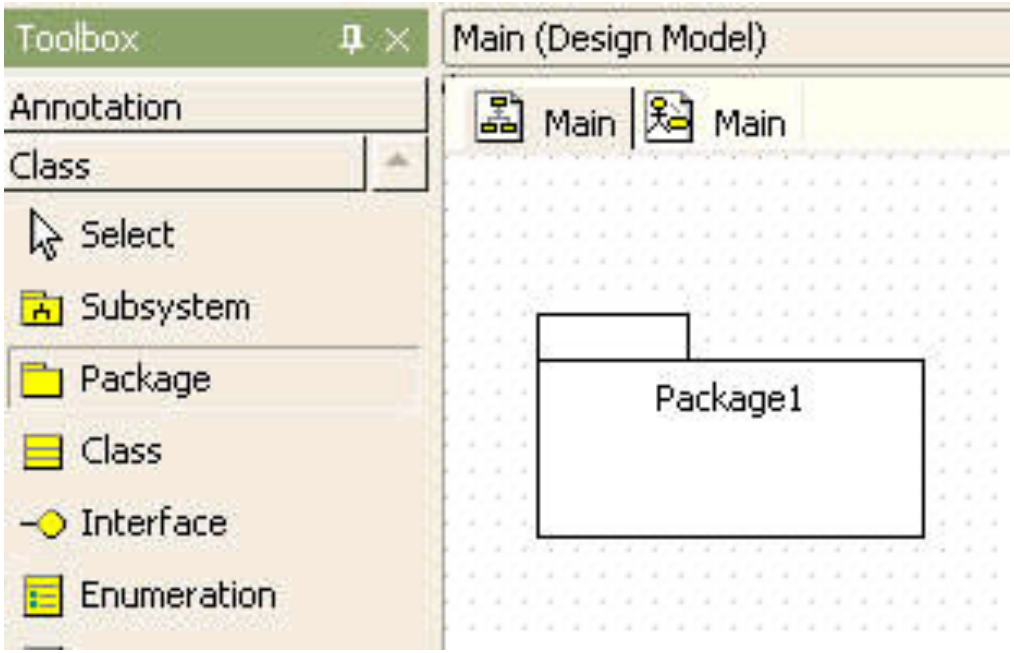
Procedure for creating system boundary

- In order to create system boundary, click [Toolbox] -> [UseCase] -> [System Boundary] button, drag from the starting point of system boundary and drag to right-bottom point of system boundary.



Procedure for creating package

- In order to create package, click [Toolbox] -> [UseCase] -> [Package] button and click at the location where package will be placed in the [main window].



Class Diagram

- It is also referred to as object modeling, is the main static analysis diagram.
- A class diagram is a collection of static modeling elements, such as classes and their relationships, connected as a graph to each other and to their contents.
- For example the things that exist, their internal structures, and their relationships to other classes.
- Class diagrams do not show temporal information, which is required in dynamic modeling.

Class Diagram

- Object modeling is the process by which the logical objects in the real world are represented by the actual objects in the program.
- This visual representation of the objects, their relationships, and their structures is for ease of understanding.
- To effectively develop a model of the real world and to determine the objects required in the system, you first must ask what objects are needed to model the system.

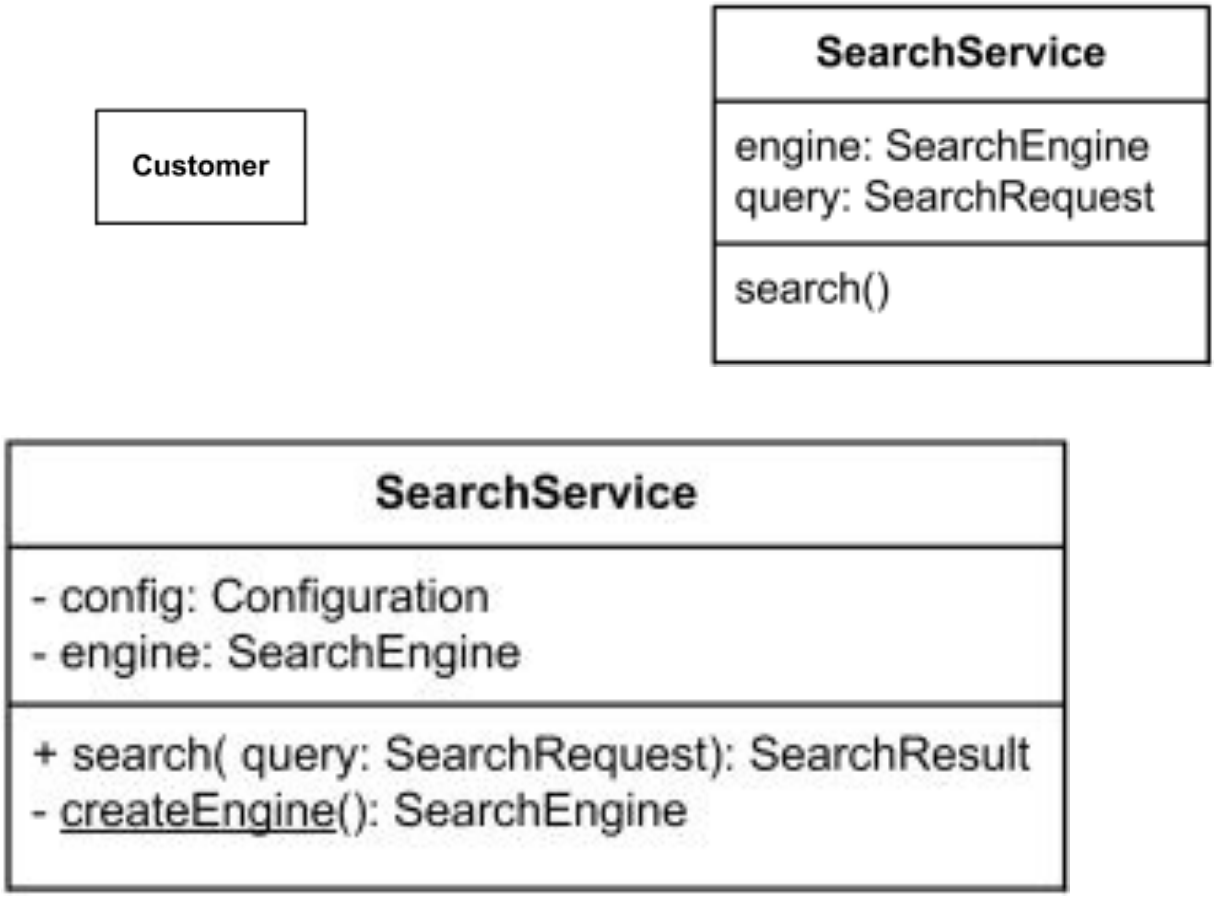
Class Diagram

- Answering the following questions will help you to stay focused on the problem at hand and determine what is inside the problem domain and what is outside it:
  - What are the goals of the system?
  - What must the system accomplish?
- You need to know what objects will form the system because , in the object oriented view point, objects are the primary abstraction,.
- The main task of object modeling is to graphically show what each will do in the problem domain, describes the structure and the relationships among objects by visual notations, and determine what behaviors fall within and outside the problem domain.

Class Notation: Static Structure

- A class is drawn as a rectangle with three components separated by horizontal lines.
- The top name compartment holds the class name, other general properties of the class, such as attributes, are in the middle compartment, and the bottom compartment holds a list of operations.
- Either or both the attribute and operation compartments may be suppressed.
- A separator line is not drawn for a missing compartment if a compartment is suppressed; no inference can be drawn about the presence or absence of elements in it.
- The class name and other properties should be displayed in up to three sections.
- A stylistic convention of UML is to use an italic font for abstract classes and a normal (roman) font for concrete classes.

Class Diagram

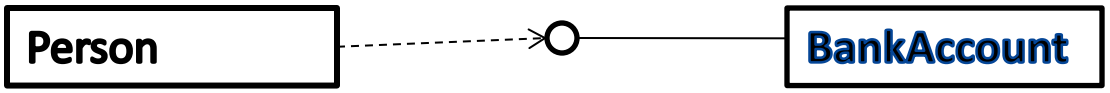


Object Diagram

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

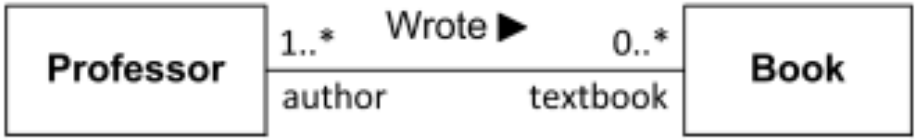
Class Interface notation

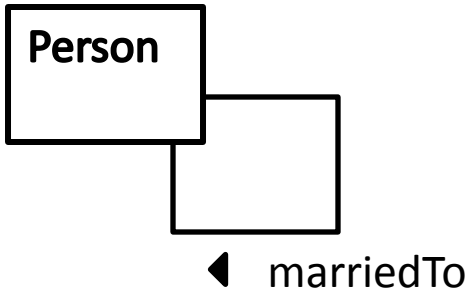
- Class interface notation is used to describe the externally visible behavior of a class; for example, an operation with public visibility.
- Identifying class interfaces is a design activity of OOSD.
- The UML notation for an interface is a small circle with the name of the connected to the class.
- A class that requires the operations in the interface may be attached to the circle by a dashed arrow .
- The dependent class is not required to actually use all of the operations.
- For example, a person object may need to interact with the BankAccount object to get the balance; this relationship is depicted with UML class interface notation.



Binary association notation

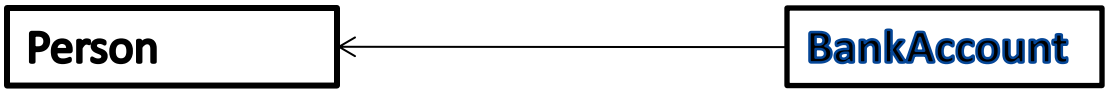
- A binary association is drawn as a solid path connecting two classes, or both ends may be connected to the same class.
- An association may have an association name.
- The association name may have an optional black triangle in it, the point of the triangle indicating the direction in which to read the name.
- The end of an association, where it connects to a class, is called the association role.





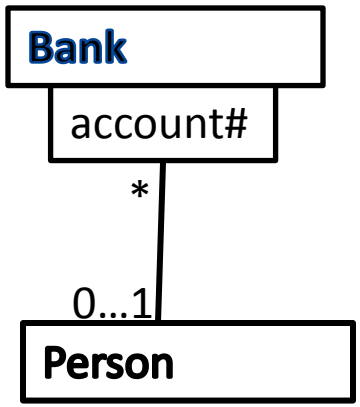
Association Role

- 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.
- The role is part of the association, not part of the class.
- Each association has two or more roles to which it is connected.
- The UML uses the term association navigation or navigability to specify a role affiliated with each end of an association relationship.
- An arrow may be attached to the end of the path to indicate that navigation is supported in the direction of class pointed to.
- An arrow may be attached to neither, one, or both ends of the path.



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.
- A qualifier is shown as a small rectangle attached to the end of an association path, between the final path segment and the symbol of the class to which it connects.
- The qualifier rectangle is part of the association path, not part of the class.
- The qualifier rectangle usually is smaller than the attached class rectangle.



Multiplicity

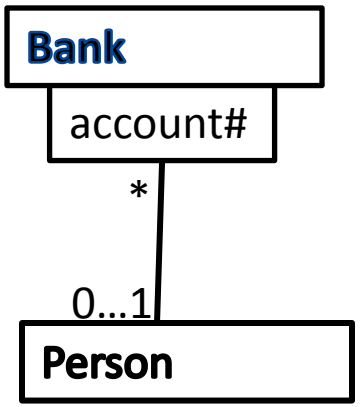
- Multiplicity specifies the range of allowable associated classes.
- It is given for roles within association, parts within compositions, repetitions, and other purposes.
- A multiplicity specification is shown as a text string comprising a period separated sequence of integer intervals, where an interval represents a range of integers in this format

  lower bound .. Upper bound

  0..1

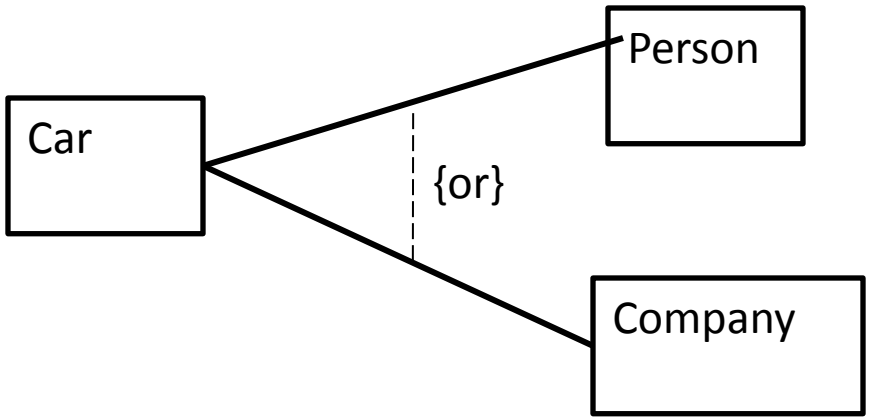
  0 .. *

  1..3, 7..10,15,19 .. *



OR Association

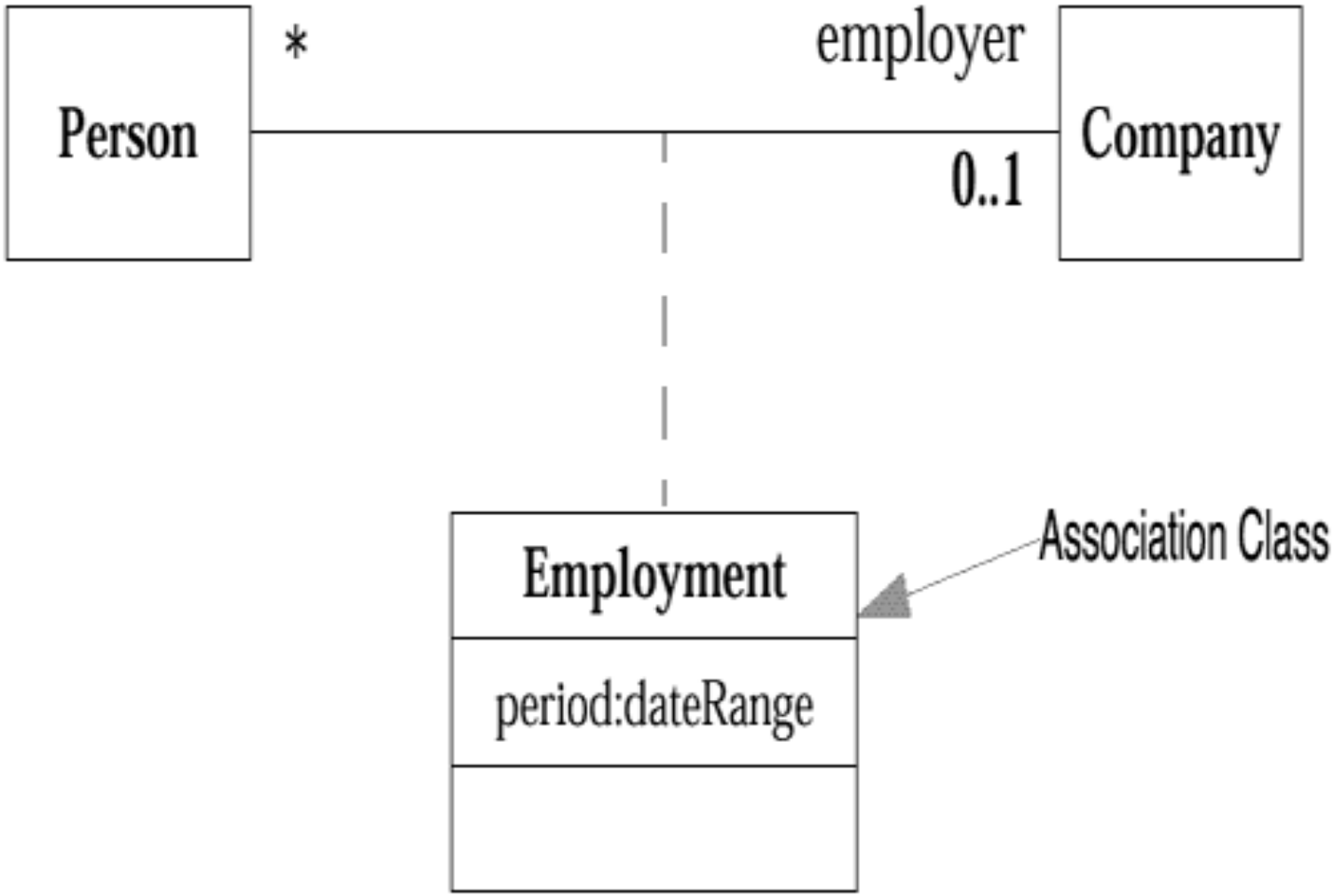
- An Or association indicates a situation in which only one of several potential association may instantiated at one time for any single object.
- This is shown as a dashed line connecting two or more association, all of which must have a class in common, with the constraint string{0r} labeling the dashed line.
- In other words, any instance of the class may participate in, at most, one of the association at one time.



Association Class

- An association class is an association that also has class properties.
- An association class in shown as a class symbol attached by a dashed line to an association path.
- The name in the class symbol and the name string attached to the association path are the same.
- The name can be shown on the path or class symbol or both.
- If an association class has attributes but no operations or other association, then the name be displayed on the association path and omitted from the association class to emphasize its “ association nature”.
- If it has operations and attributes, then the name may be omitted from the path and placed in the class rectangle to emphasize its “class nature”.

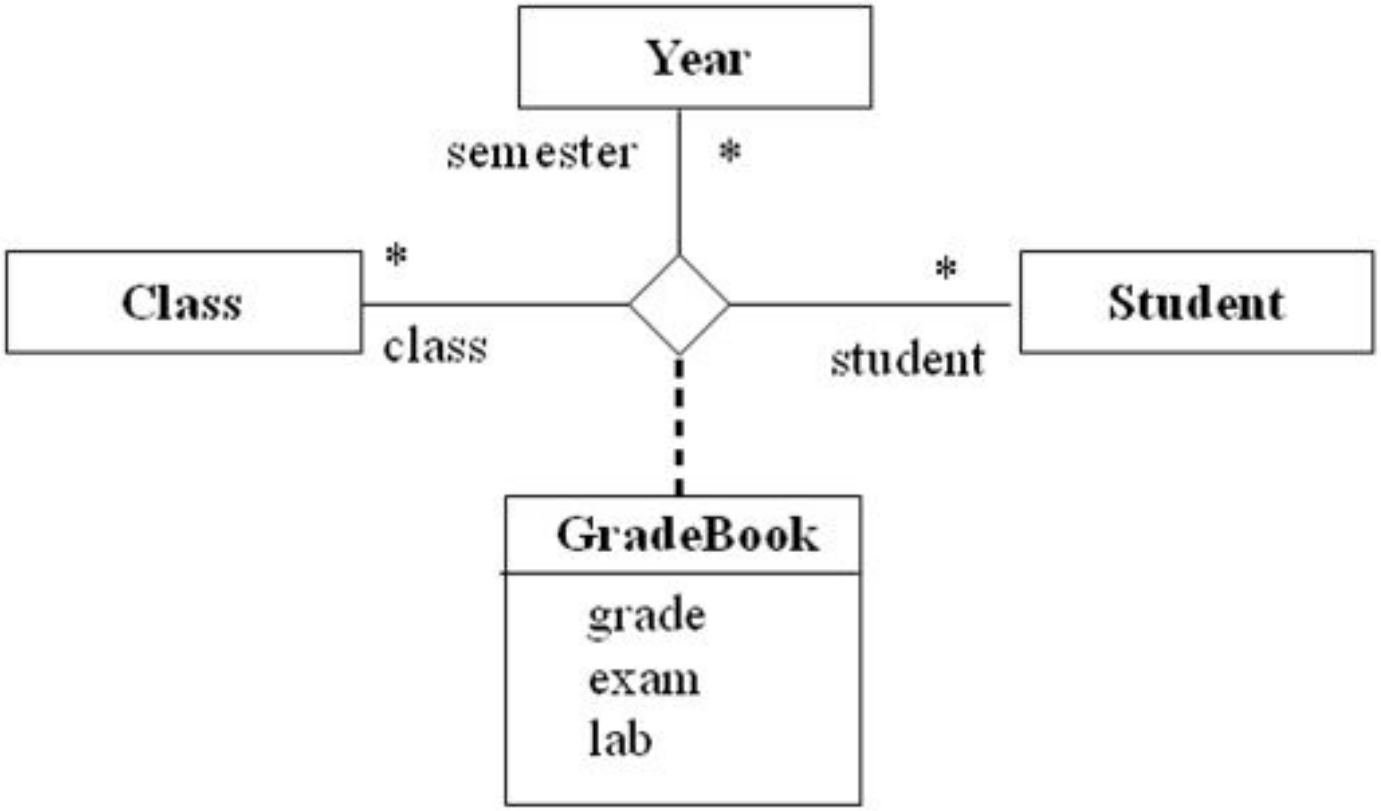
Association Class



N-ary Association

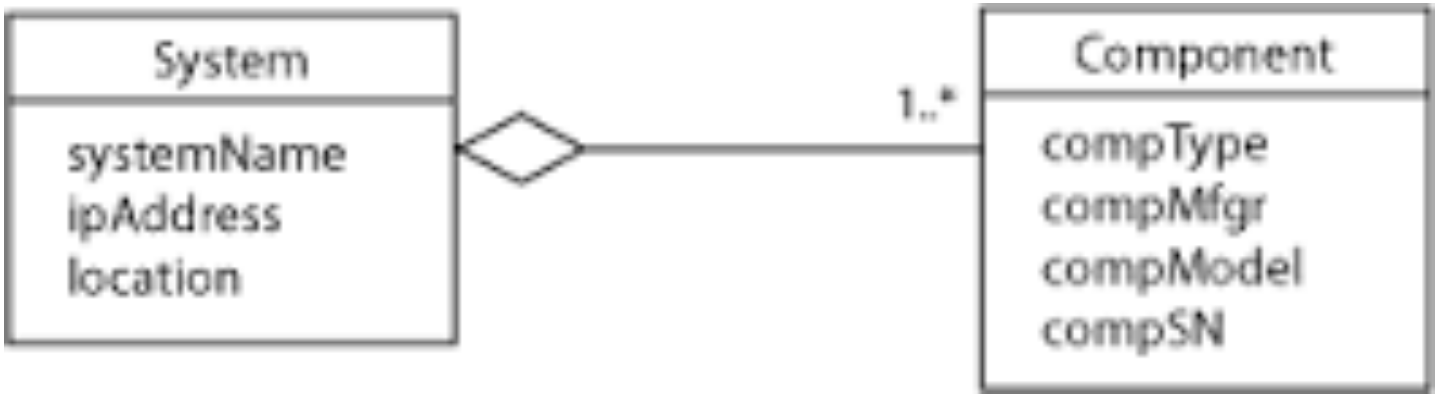
- An n-ary association is an association among more than two classes.
- An n-ary association is shown as a large diamond with a path from the diamond to each participant class.
- The name of the association is shown near the diamond.
- The role attachment may appear on each path as with a binary association.
- Multiplicity may be indicated; however, qualifiers and aggregation are not permitted.
- An association class symbol may be attached to the diamond by a dashed line, Indicating an n-ary association that has attributes, operation, or associations.

N-ary Association



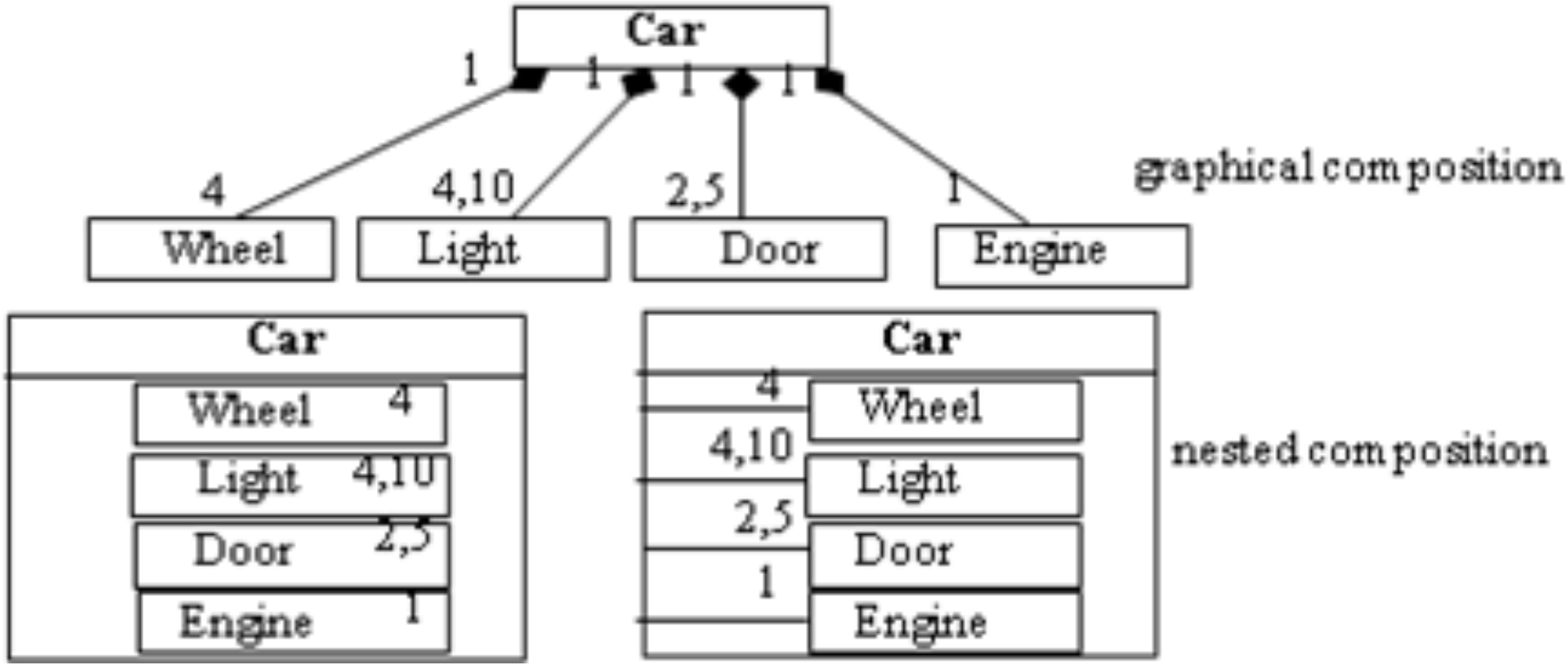
Aggregation

- Aggregation is a form of association.
- A hollow diamond is attached to the end of the path to indicate aggregation.
- The diamond may not be attached to both ends of a line, and it need not be presented at all.



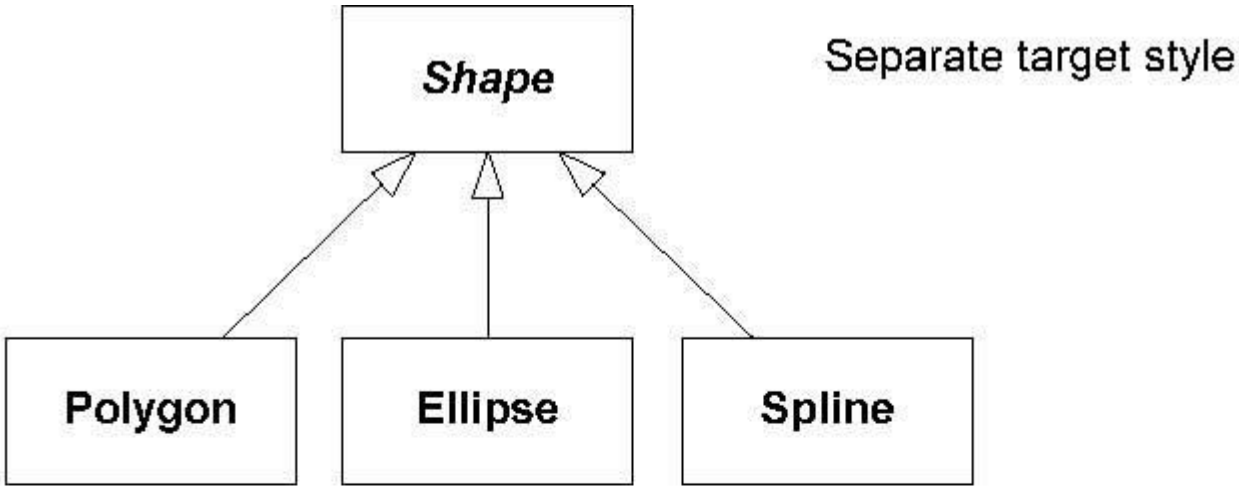
Composition

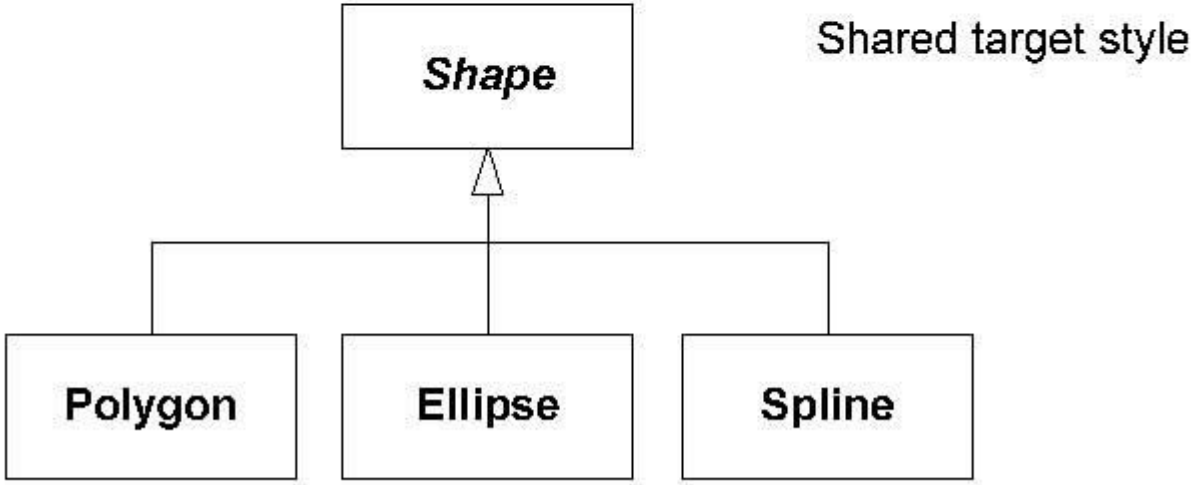
- It also know as the a-part-of, is a form of aggregation with strong ownership to represent the component of a complex object.
- Composition also is referred to as a part-whole relationship.
- The UML notation for composition is a solid diamond at the end of a path.
- Alternatively, the UML provides a graphically nested form that, in many cases, is more convenient for showing composition.
- Parts with multiplicity greater than one may be created after the aggregate itself but, once created, they live and die with it.
- Such parts can also explicitly removed before the death of the aggregate.

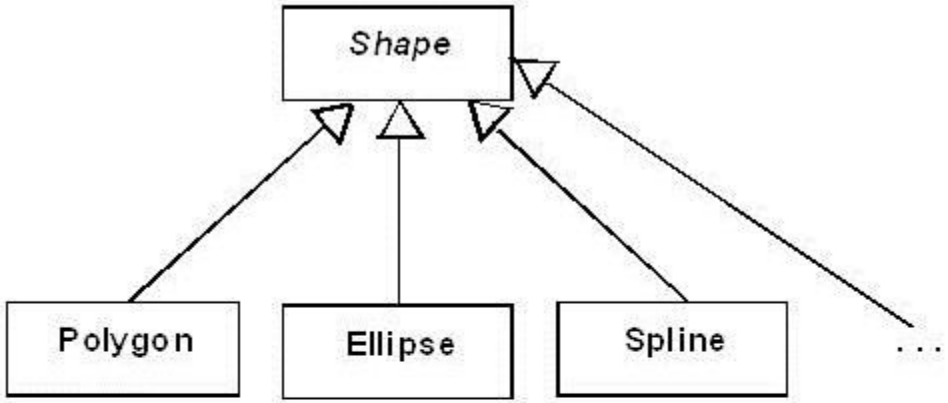

Car
1
1
1
1
4
4,10
2,5
1
Wheel
Light
Door
Engine
graphical composition
Car
Wheel 4
Light 4,10
Door 2,5
Engine 1
Car
4 Wheel
4,10 Light
2,5 Door
1 Engine
nested composition

Generalization

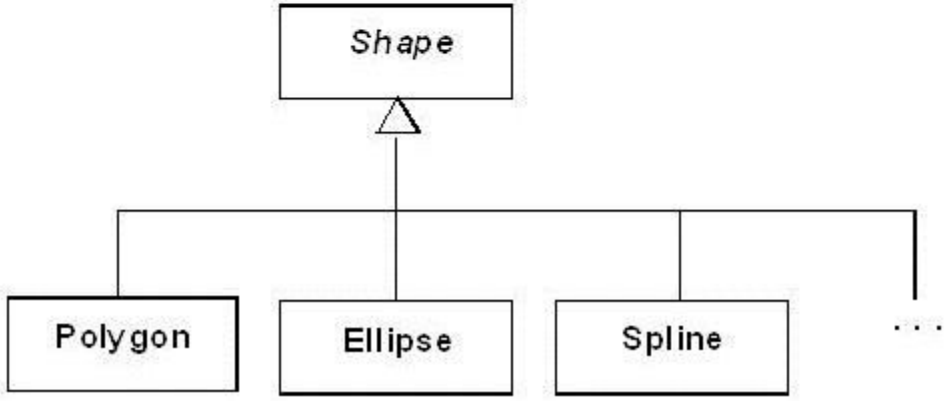
- Generalization is the relationship between a more general class and a more specific class.
- Generalization is displayed as a directed line with a closed, hollow arrowhead at the superclass end.
- The UML allows a discriminator label to be attached to a generalization of the superclass.
- Ellipse(….) indicate that the generalization is incomplete and more subclasses exit that are not shown.
- The constructor complete indicates that the generalization is complete and no more subclasses are needed.
- If a text label is placed on the hollow triangle shared by several generalization paths to subclasses, the label applies to all of the paths.
- In other words, all subclasses share the given properties.


Shape
Separate target style
Polygon
Ellipse
Spline


Shape
Shared target style
Polygon
Ellipse
Spline



Separate Target Style



Shared Target Style

Visual Paradigm for UML Community Edition [not for commercial use]