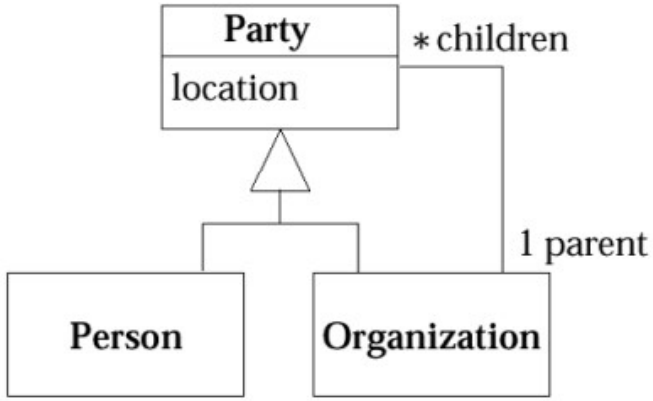
Class Diagrams: Advanced Concepts

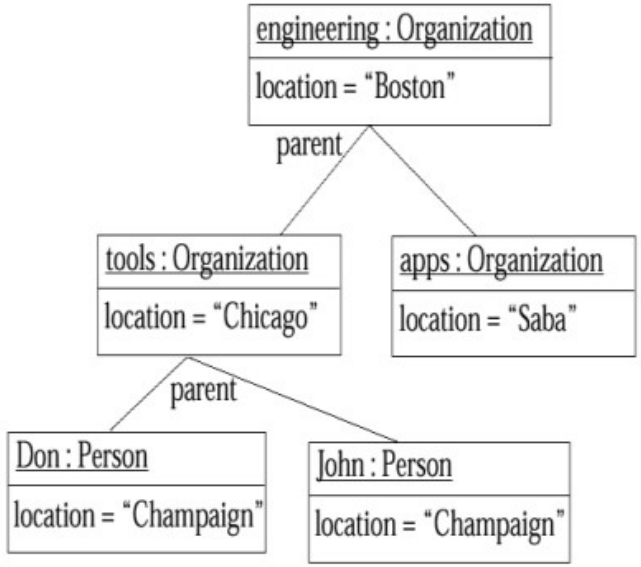
Stereotypes

- core extension mechanism of the UML
  - For not in UML
  - treat your construct as a stereotype of the UML construct
- UML interface is a class that has only public operations with no method bodies or attributes.
- Stereotypes are usually shown in text between guillemets (for example, «interface»)
  - an icon can be used for the stereotype
- stereo-types as subtypes of the meta-model types Class, Association, and Generalization
  - For example abstract class
  - Denoting contraint

Object Diagram

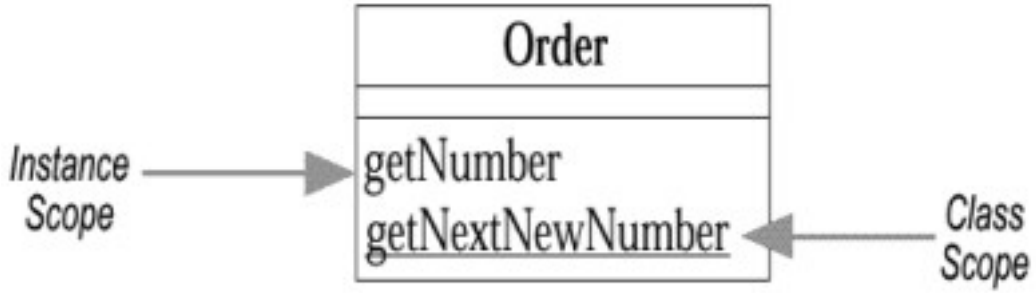
- a snapshot of the objects in a system at a point in time
  - often called an instance diagram
- Class Diagram of Party Composition Structure
- Instance Diagram showing instances of the classes





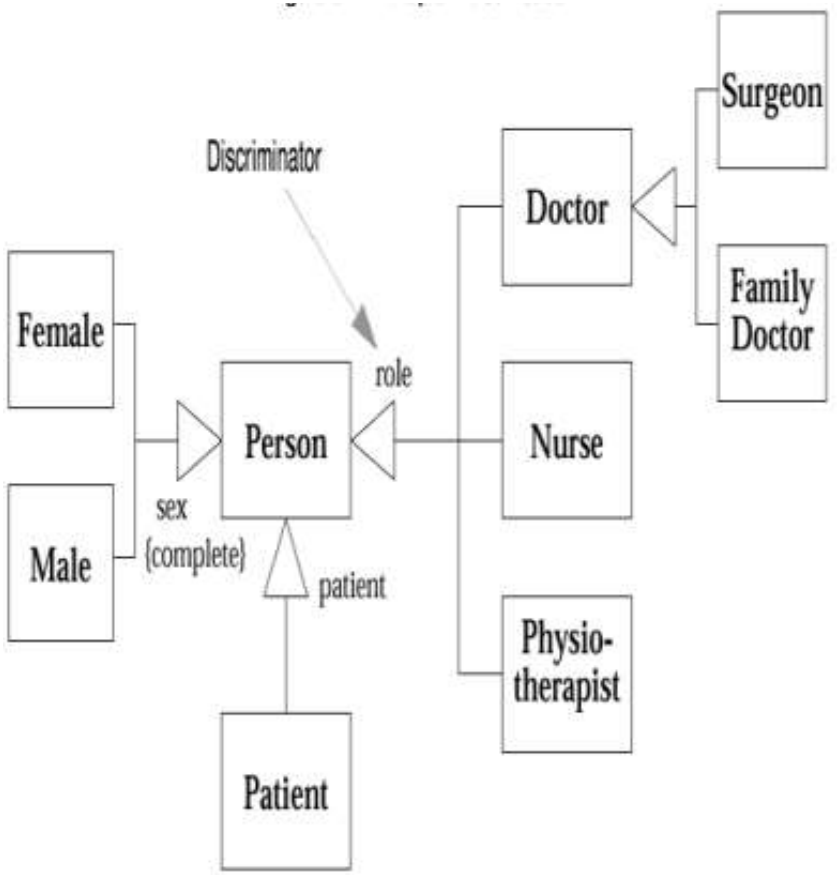
Class Scope Operations and Attributes

- Operations and attributes indicates the scope
- Some of them are instance scope
  - Making the attributes available
- Others for class scope
  - For manipulating the attributes



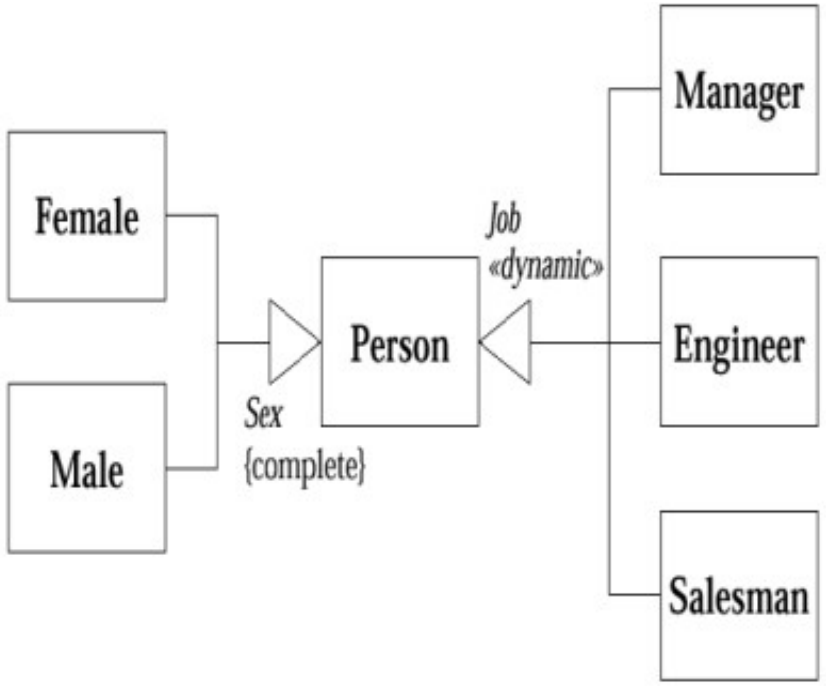
Multiple Classification

- single classification,
  - an object belongs to a single type,
  - inherit from supertypes.
- multiple classification,
  - an object may be described by several types
  - not necessarily connected by inheritance.
  - Different from multiple inheritence



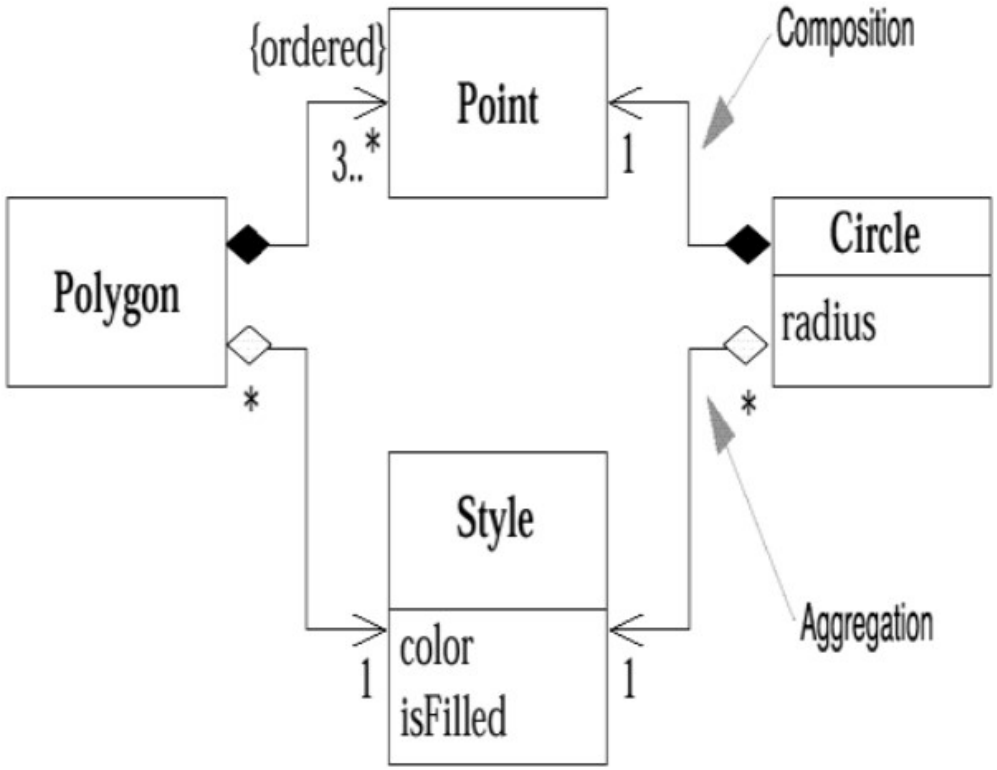
Dynamic classification

- allows objects to change type within the subtyping structure
  - static classification does not.
- With static classification, a separation is made between types and states;
- dynamic classification combines these notions.



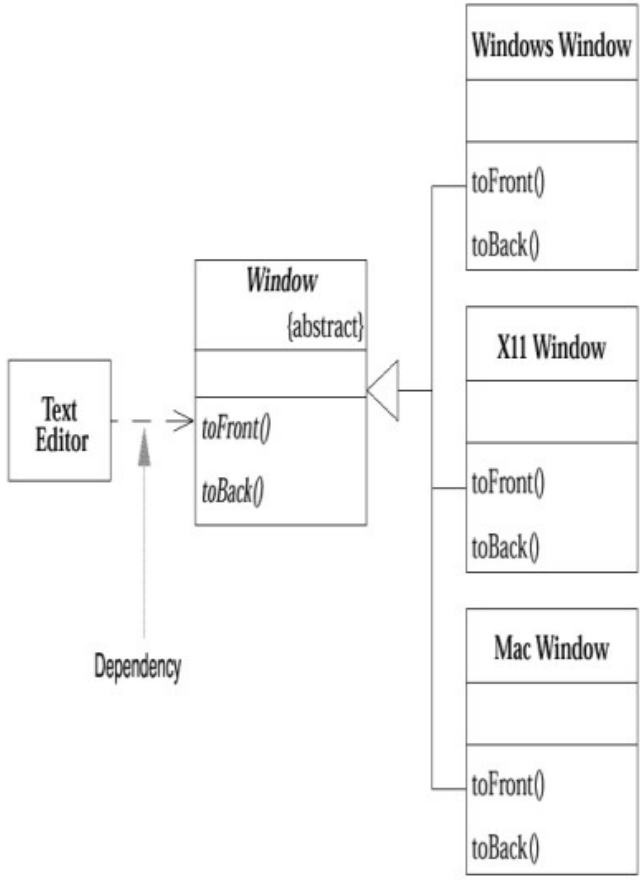
Aggregation and Composition

- Composition
  - a class is structured by multiples of other class
  - If the composite is deleted the elements of the composite is destroyed
- Aggregation
  - Structured by multiple classes
  - Deletion does not affect the elements



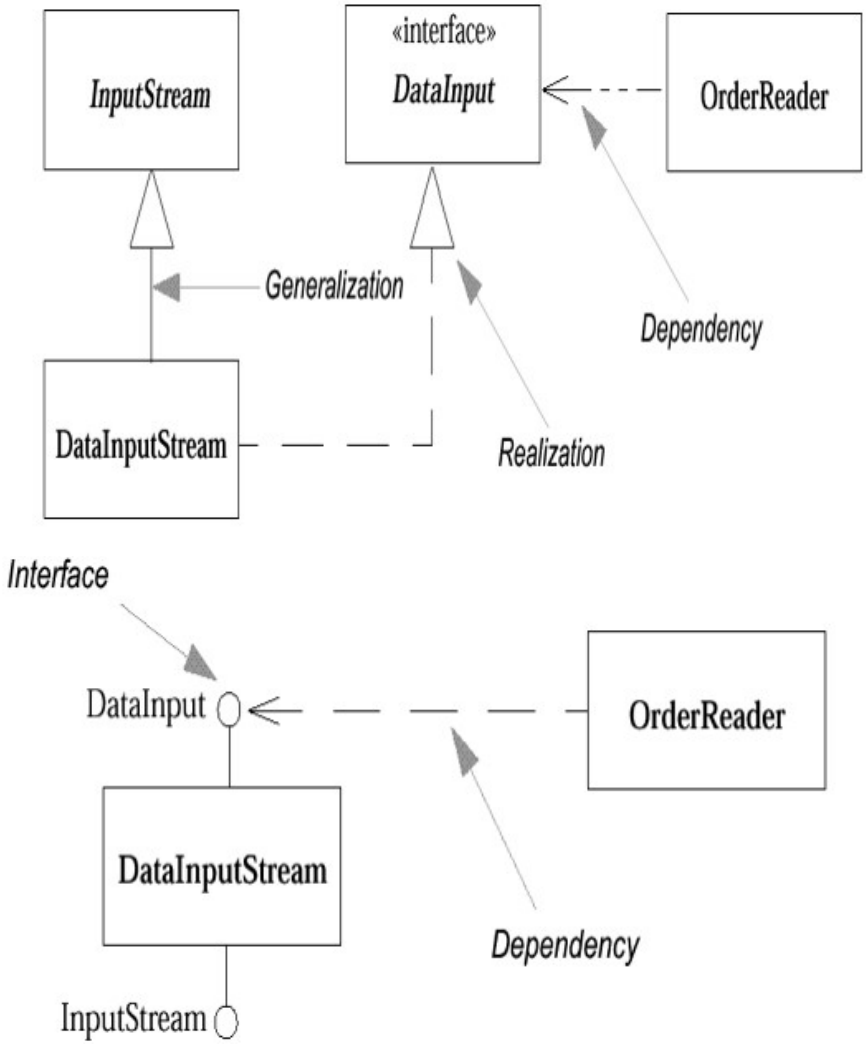
Interfaces and Abstract Classes

- A pure interface, as in Java, is a class with no implementation
  - has operation declarations but no method bodies and no fields
- abstract classes may provide some implementation,
  - but often they are used primarily to declare an interface
  - Written in italic font



Interfaces and Abstract Classes

- Realization is the implementation
  - Not necessarily the inheritance
- Dependency indicated by broken association to the interface
  - Dependent classes might need to change if the interface is changed
- Lollipop Notation for Interfaces
  - more compact notation. Here, the interfaces are represented by small circles (often called lollipops)



Collections for Multivalued Association Ends

- A multivalued end is one whose multiplicity's upper bound is greater than 1 (for instance, *)
- The {ordered} constraint
  - there is an ordering to the target objects-that is,
  - the target objects form a list.
- {bag} constraint to indicate that target objects may appear more than once, but there is no ordering
- {hierarchy} constraint to indicate that the target objects form a hierarchy
- {dag} constraint to indicate a directed acyclic graph.

Frozen

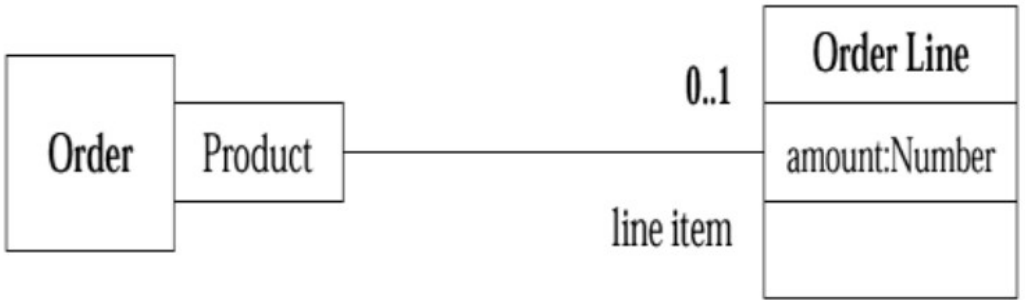
- constraint that the UML defines as applicable
  - to an attribute
  - an association end
  - useful for classes as well.
- indicates that the value of that attribute or association end may not change during the lifetime of the source object
- Frozen is not the same as read-only
  - value cannot be changed directly but may change due to a change in some other value
  - a person has a date of birth and an age, the age may be read-only, but it cannot be frozen.
- Marked by {*frozen*} and {*read only*}

Classification and Generalization

- subtyping as the "is a" relationship
  - 1. Shep is a Border Collie. (classification, Instantiation)
  - 2. A Border Collie is a Dog. (classification, Instantiation)
  - 3. Dogs are Animals. (subtyping or generalization)
  - 4. A Border Collie is a Breed. (subtyping or generalization)
  - 5. Dog is a Species (subtyping or generalization)
- If all are considered as generalization
  - 1 and 4: "Shep is a Breed.“
  - Wrong conclusion
- Generalization is transitive; classification is not.

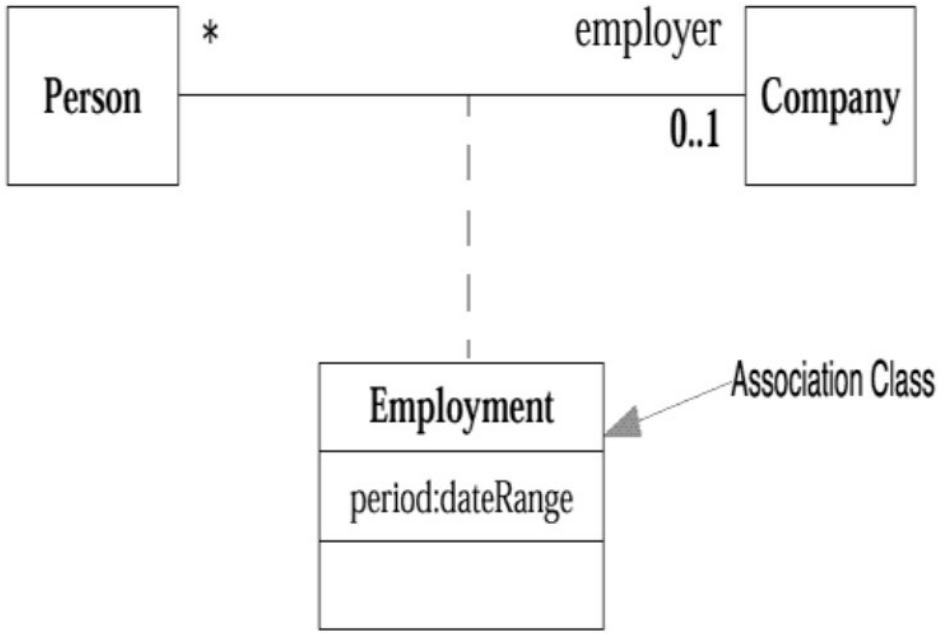
Qualified Association

- you cannot have two Order Lines within an Order for the same Product

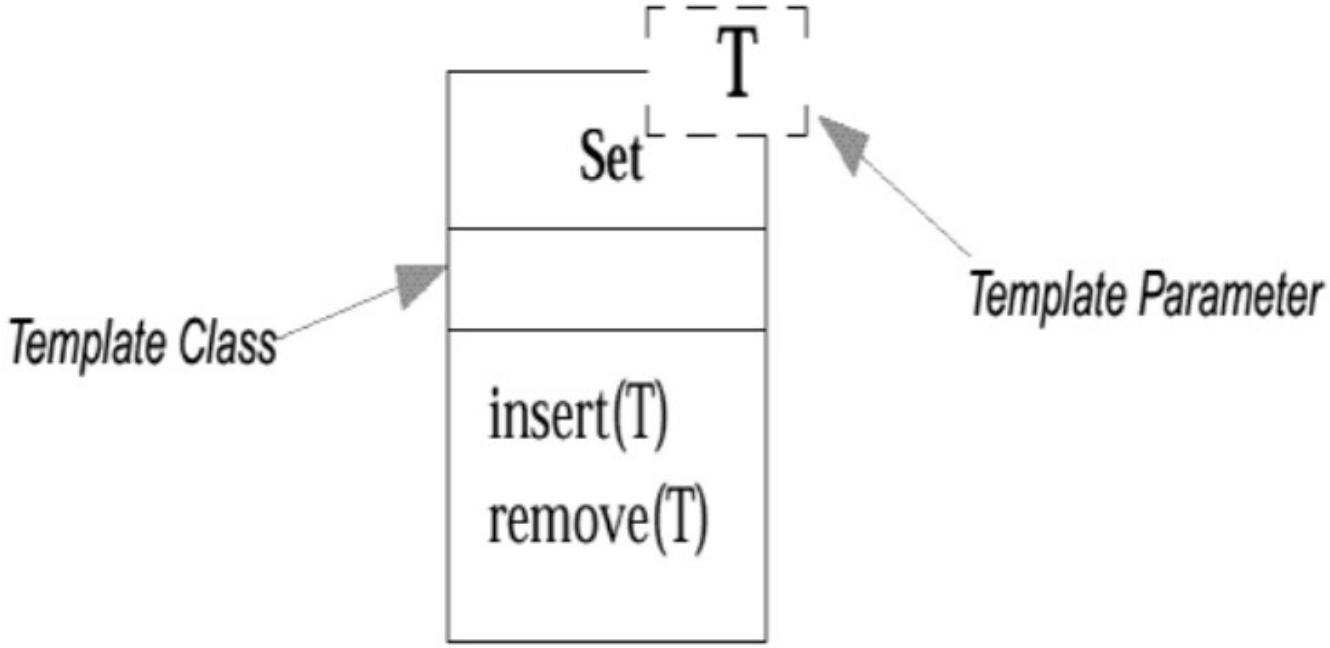


Association Class

- allow you to add attributes, operations, and other features to associations



Parameterized Class



Visibility

- A (+) public member is visible anywhere in the program and may be called by any object within the system.
- A (-) private member may be used only by the class that defines it.
- A (#) protected member may be used only by (a) the class that defines it or (b) a subclass of that class.

Class Diagram: State

- State of an instance is defined by
  - The values of the attributes
  - The number of links
- An object may show different behavior in different state
- State transition is initiated by an event
- State can be changed only by executing an operation

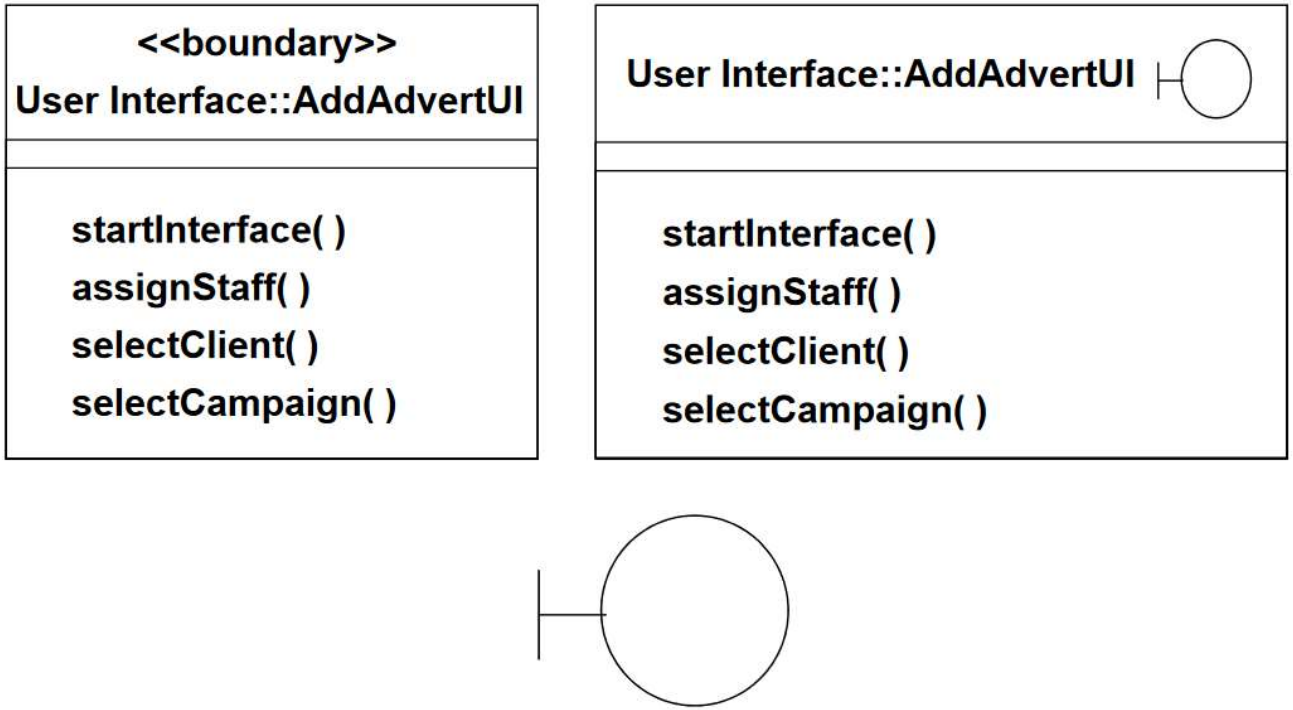
Different types of object

- Boundary objects
- Entity objects
- Control objects

Boundary Classes

- Models interaction between the system and actors
- May include interfaces to other software or devices
- Main task is to manage the transfer of information across system boundaries

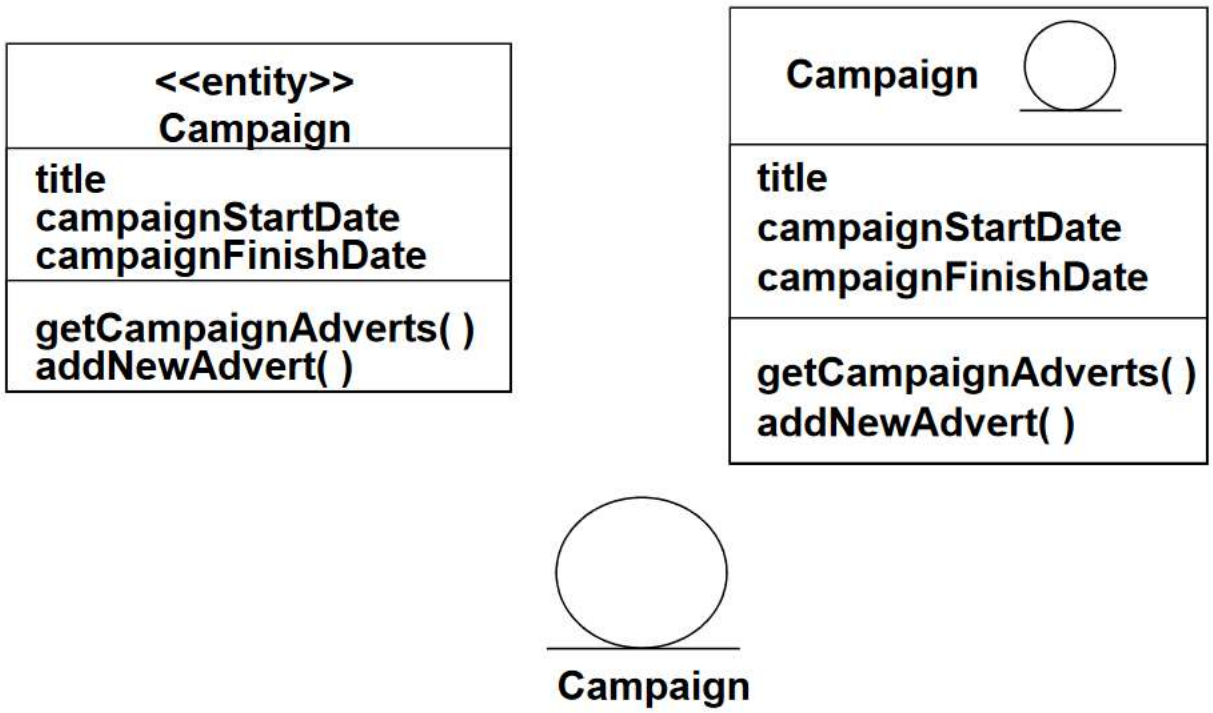
Notations for boundary class



Entity Classes

- Models information and their related behavior
- Maybe about a person, a real-life object or an event
- Often require persistent storage

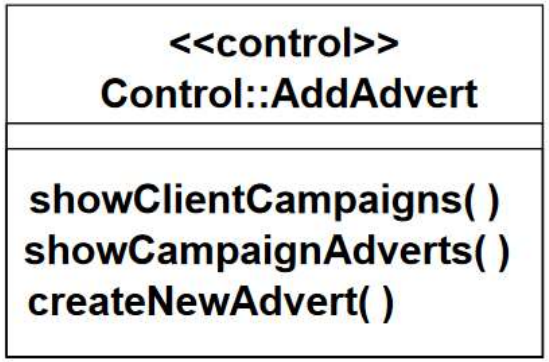
Notations for entity class

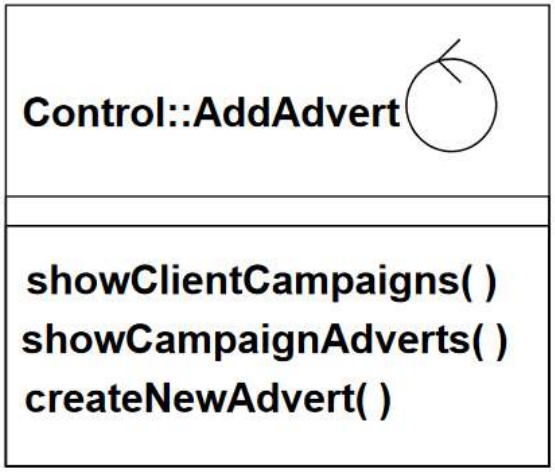


Control Classes

- Model the coordination, sequencing, transactions and control of other objects
- One use-case should result in one control class

Notations for control class





From Use-Case to Classes

- Start with one use case
- Identify the likely classes involved (the use case collaboration)
- Draw a collaboration diagram that fulfills needs of the use case
- Translate this collaboration into a class diagram
- Repeat for other use cases
- Combine the diagrams

From Use-Case to Classes: Step 1

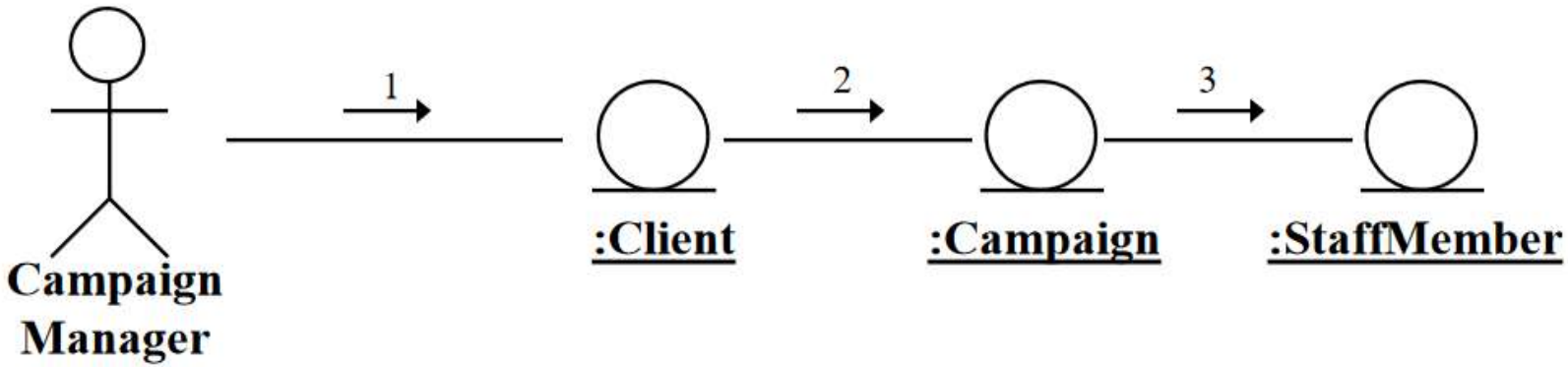
Use Case : Assign staff to a campaign	
Actor Action	System Response
1. None	2. Display List of Client Name
3. Select the client name	4. List the titles of the campaigns related to that client
5. Select the relevant campaign	6. Display list of staff not assigned to that campaign
7. Select a staff member to assign to the campaign	8. Present a message confirming the allocation of staff

From Use-Case to Classes: Step 2 (Guideline to eliminate candidate classes)

- A number of tests help to check whether a candidate class is reasonable
  - Is it beyond the scope of the system?
  - Does it refer to the system as a whole?
  - Does it duplicate another class?
  - Is it too fuzzy?
  - Is it too tied up with physical inputs and outputs?
  - Is it really an attribute?
  - Is it really an operation?
  - Is it really an association?
- If any answer is ‘Yes’, consider modeling the potential class in some other way (or do not model it at all)
- The identified classes (Client, Campaign, Staff Member)

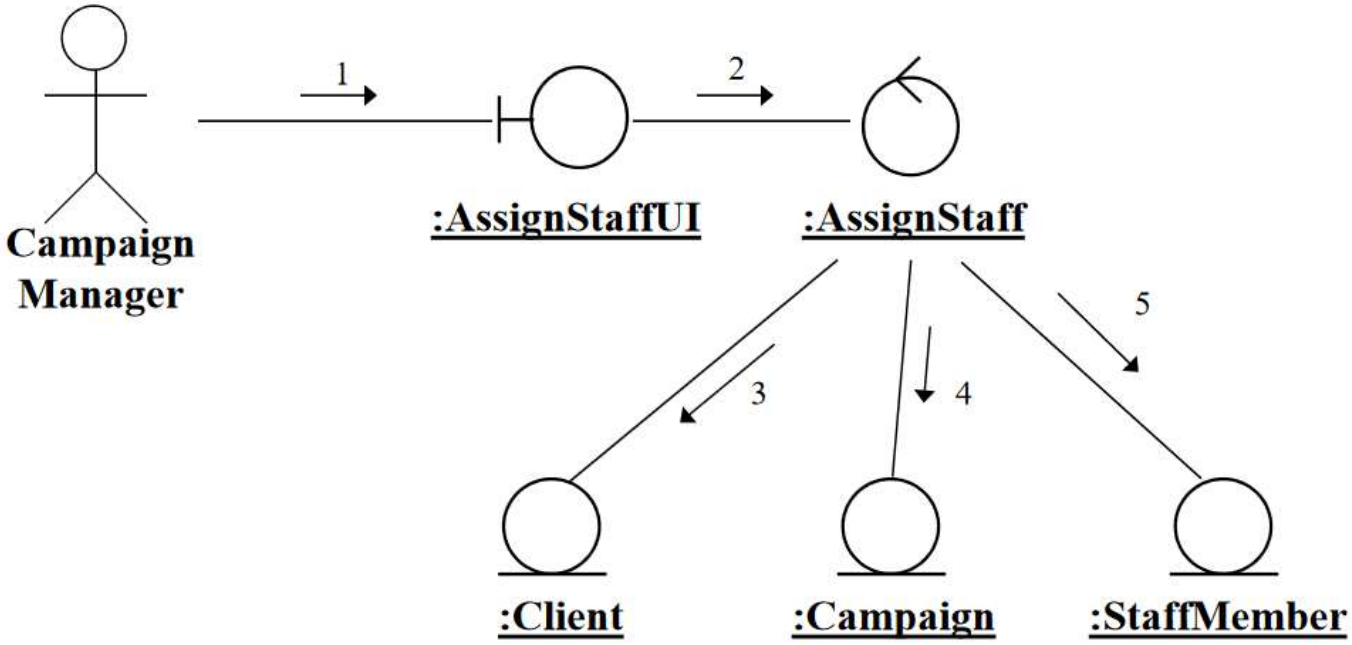
From Use-Case to Classes: Step 3

- Initial collaboration diagram



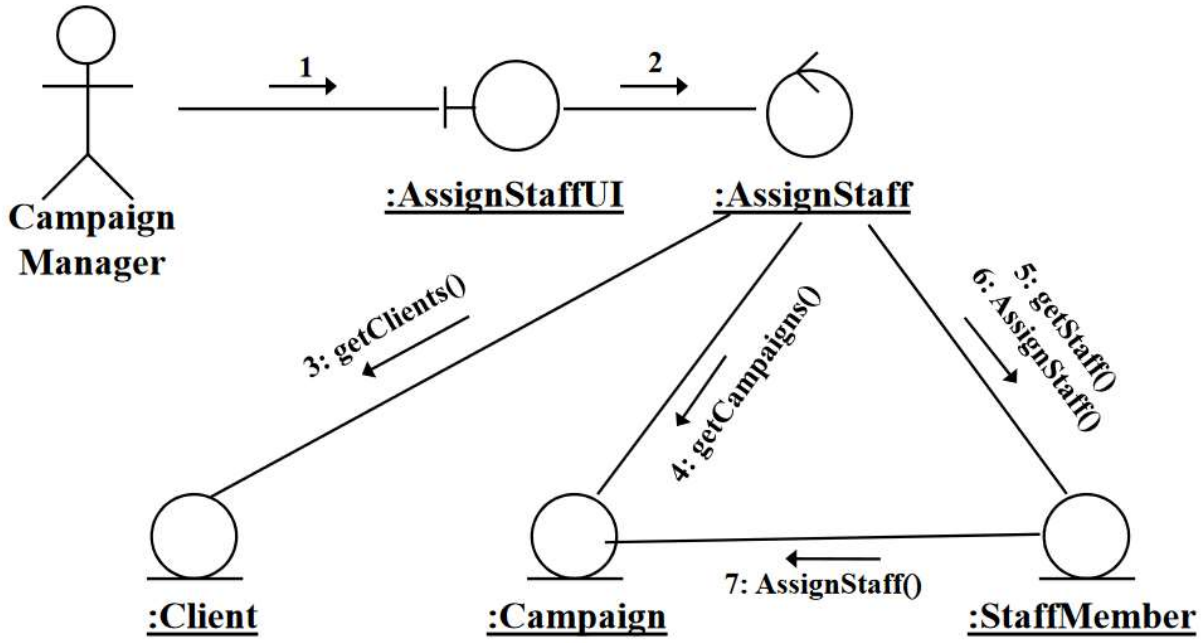
From Use-Case to Classes: Step 3

- Adding boundary and control classes



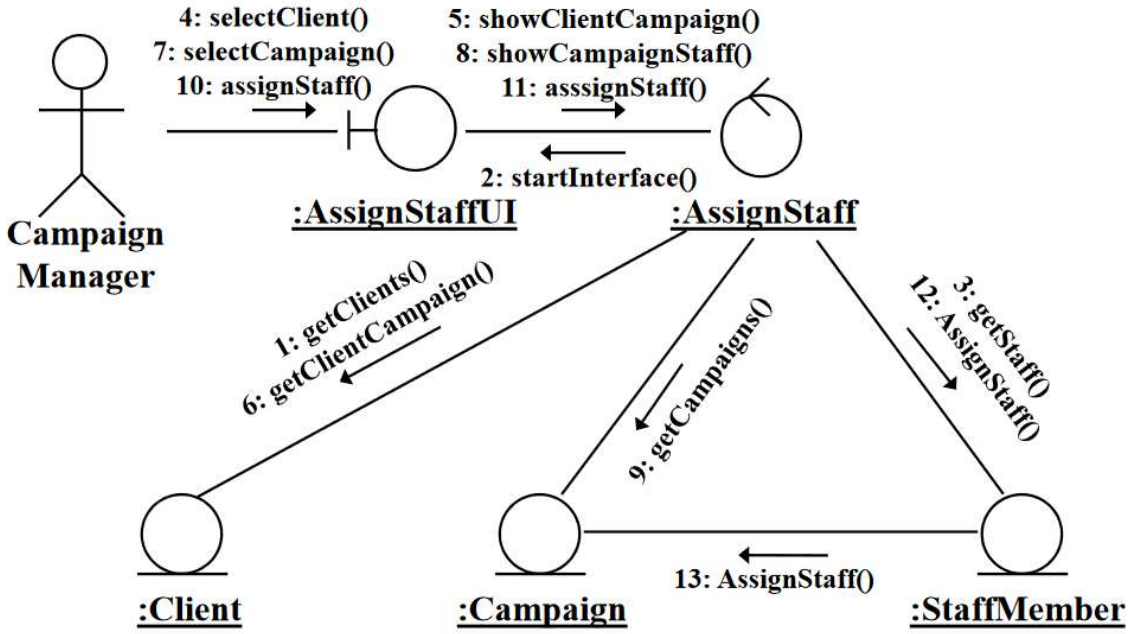
From Use-Case to Classes: Step 3

- Adding messages



From Use-Case to Classes: Step 3

- Finalizing



From Use-Case to Classes: Step 4