Object Oriented Analysis,
UML,
Class Diagrams

Overview

- Define object modeling and explain its benefits.
- Recognize and understand the basic concepts and constructs of object modeling.
- Define the UML and its various types of diagrams.
- Evolve a business requirements use-case model into a system analysis use-case model.
- Discover objects and classes, and their relationships.
- Construct a class diagram.

Introduction to Object Modeling

- Object-oriented analysis (OOA) – an approach used to
  - study existing objects to see if they can be reused or adapted for new uses
  - define new or modified objects that will be combined with existing objects into a useful business computing application

- Object modeling – a technique for identifying objects within the systems environment and the relationships between those objects.

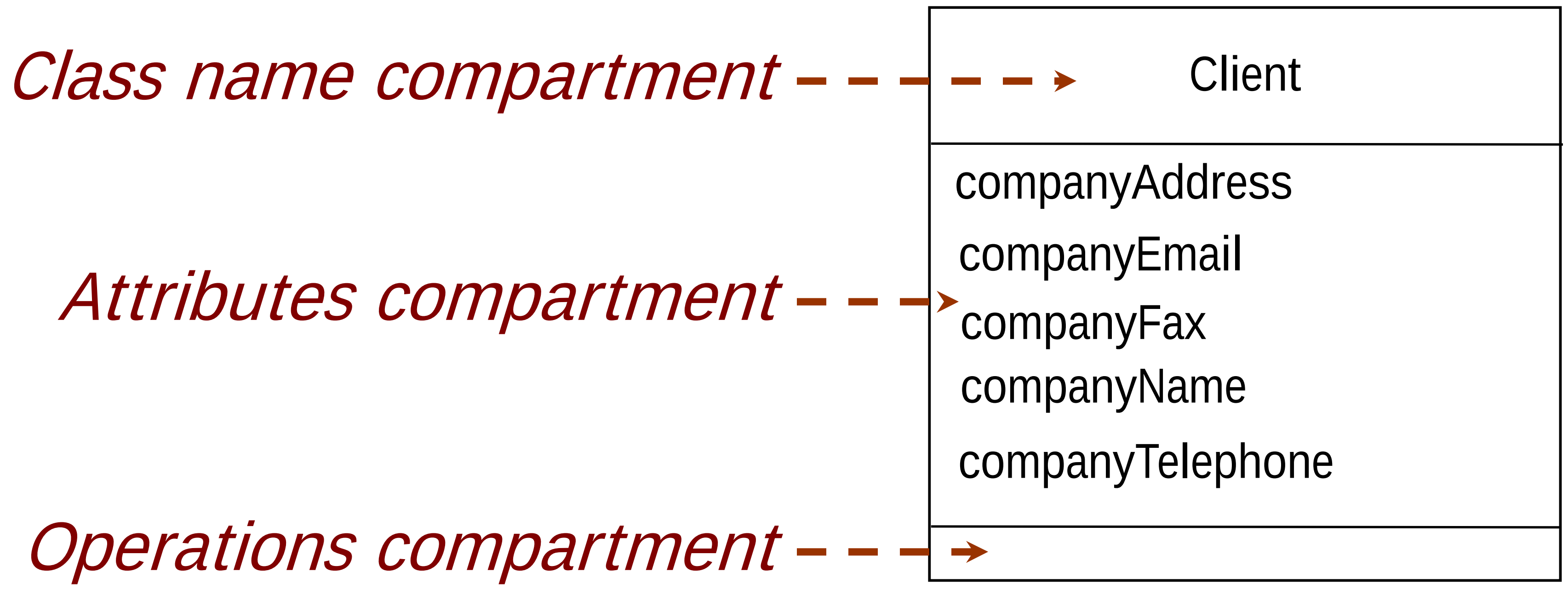
Introduction to the UML

- Unified Modeling Language (UML) – a set of modeling conventions that is used to specify or describe a software system in terms of objects.
  - The UML does not prescribe a method for developing systems—only a notation that is now widely accepted as a standard for object modeling.

UML Diagrams

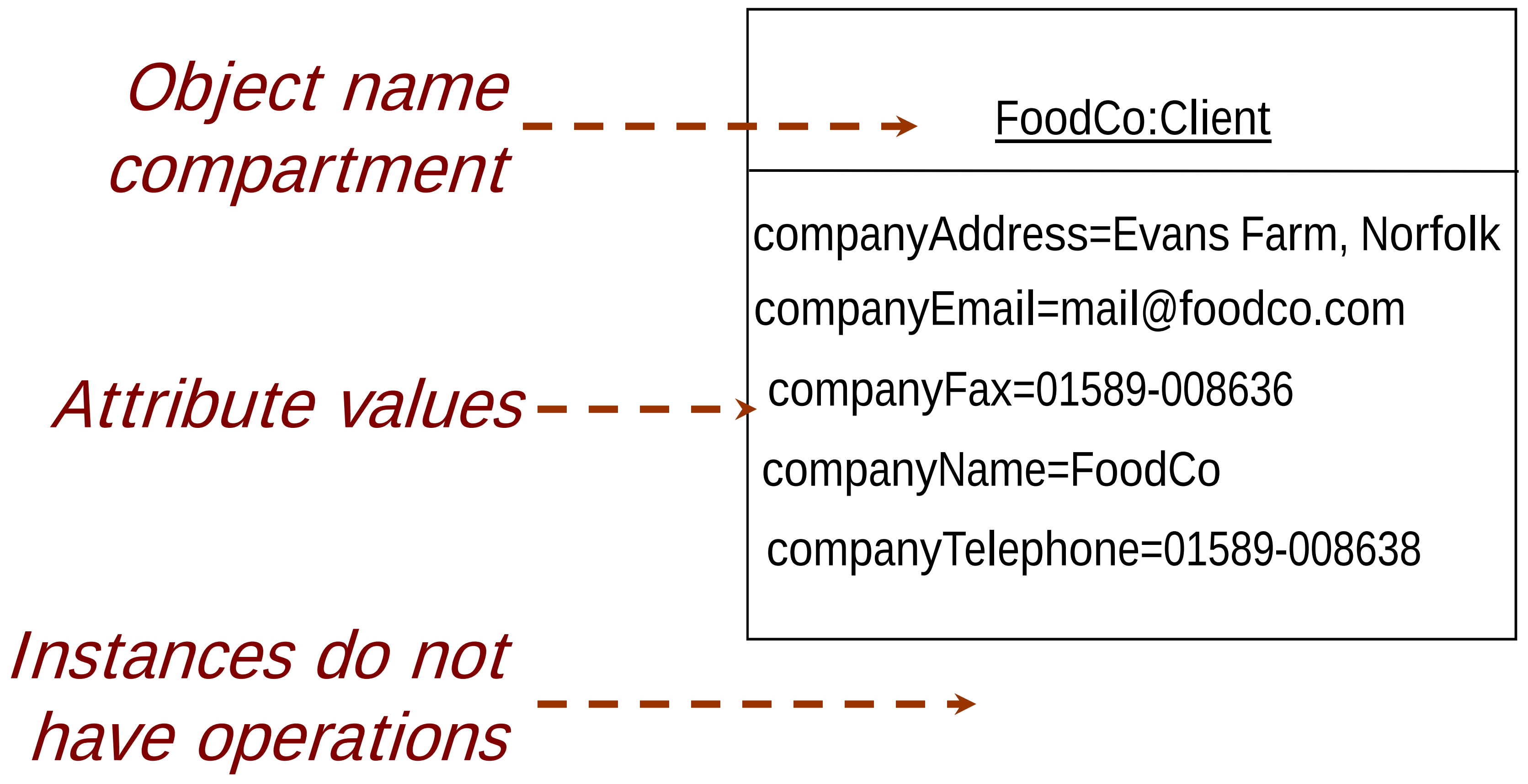
- Use-Case Model Diagrams
- Static Structure Diagrams
  - Class diagrams
  - Object diagrams
- Interaction Diagrams
  - Sequence diagrams
  - Collaboration diagrams
- State Diagrams
  - Statechart diagrams
  - Activity diagrams
- Implementation Diagrams
  - Component diagrams
  - Deployment diagrams

Class Diagram: Class vs Instance



Class

Class Diagram: Class vs Instance


Object name compartment
FoodCo:Client
companyAddress=Evans Farm, Norfolk
companyEmail=mail@foodco.com
companyFax=01589-008636
companyName=FoodCo
companyTelephone=01589-008638
Attribute values
Instances do not have operations

Instance

Class Diagram: Class vs Instance

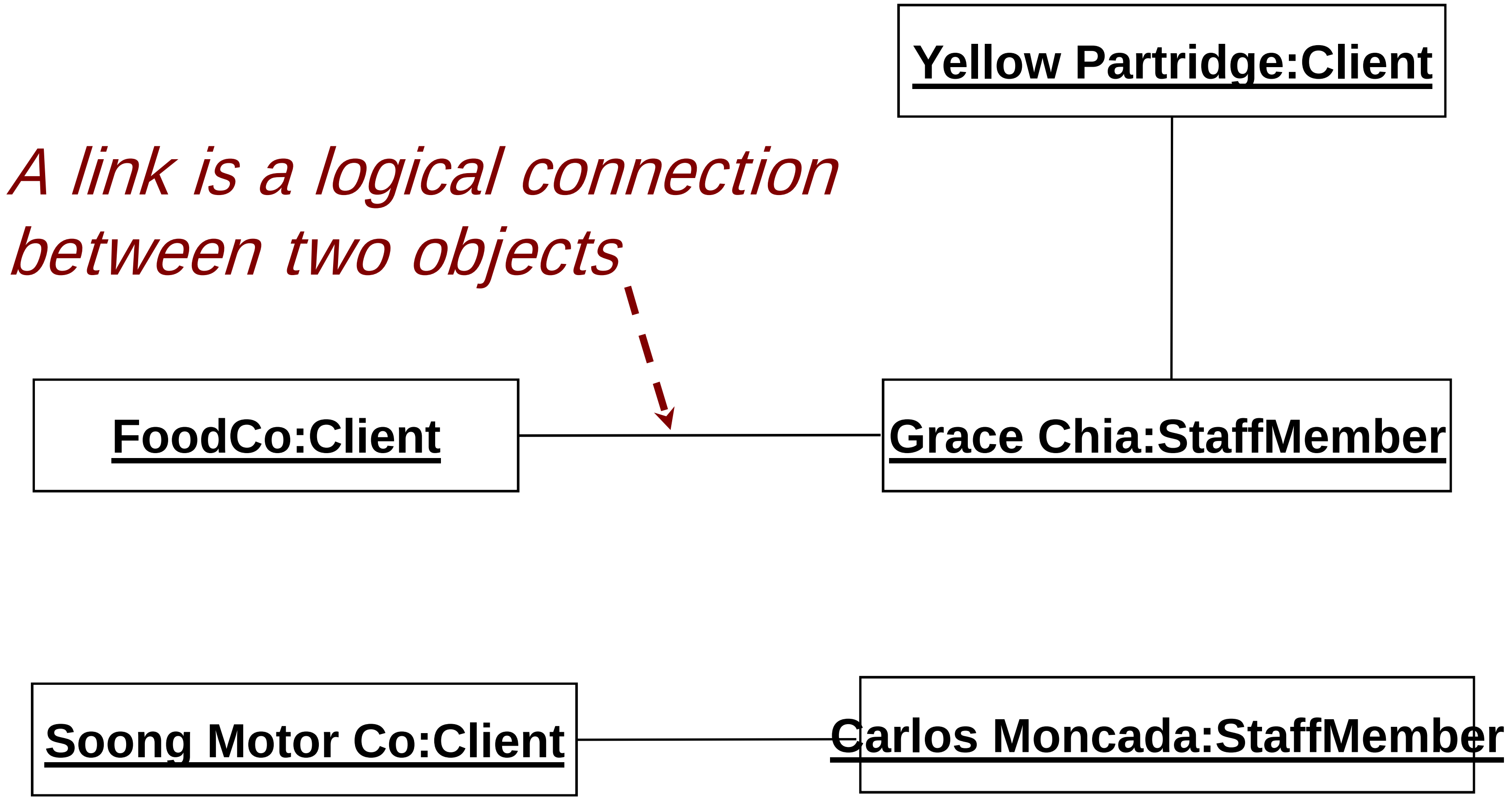
- Object Instances often changes frequently while classes are generally permanent.
- Instances can be destroyed.
- Object instances can be updated.

Class Diagram: Attributes

Attributes are:

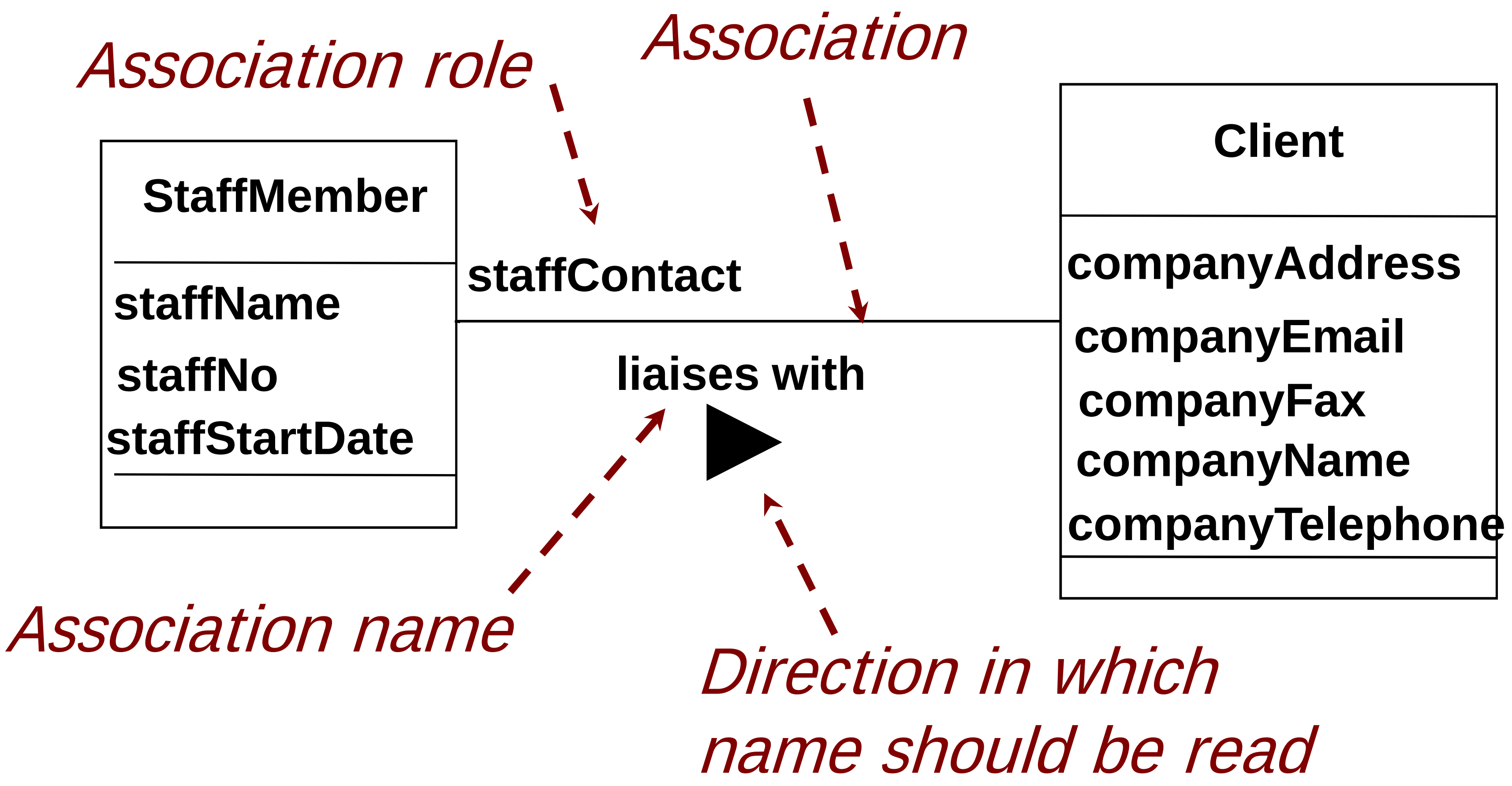
- Part of the essential description of a class
- The common structure of what the class can ‘know’
- Each object has its own *value* for each attribute in its class

Class Diagram: Link vs Association



Links

Class Diagram: Link vs Association



Association

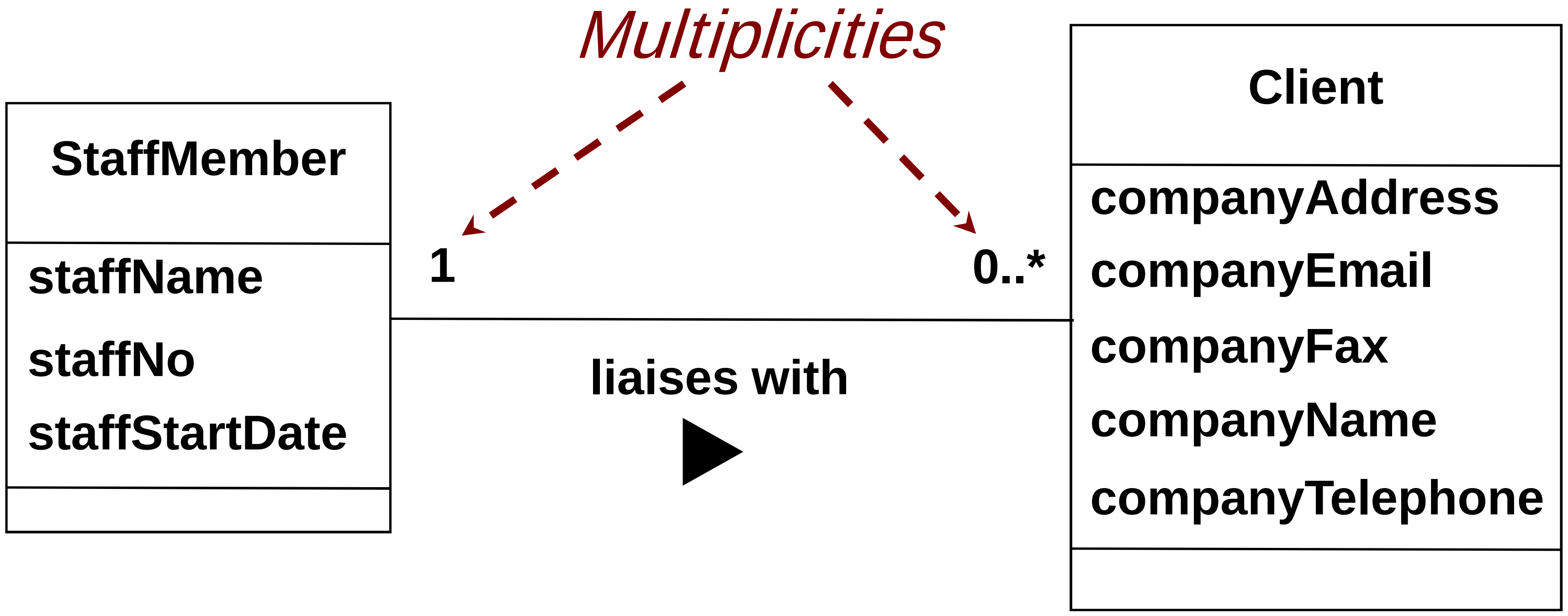
Class Diagram: Link vs Assosiation

- An association describes set of similar instances.
- Linked instances may be of different class or from the same class.
- Link can connect instances o itself.

Class Diagram: Multiplicity

- Associations have multiplicity
- Multiplicity is the range of permitted cardinalities of an association
- Represent *enterprise* (or *business*) *rules*
- For example:
  - Any bank customer may have one or more accounts
  - Every account is for one, and only one, customer

Class Diagram: Multiplicity



- Exactly one staff member liaises with each client
- A staff member may liaise with zero, one or more clients

Class Diagram: Multiplicity Notations

Multiplicity	Notations	Example
Exactly one	1 blank	A student studies in exactly one department
Zero or one	0..1	A bed may have zero or one patient
Zero or more	0..* *	Student may issue for zero or more IC
One or more	1..*	A course may have one or more teachers
Specific Range	0..3	A student may borrow at most 3 books from the library

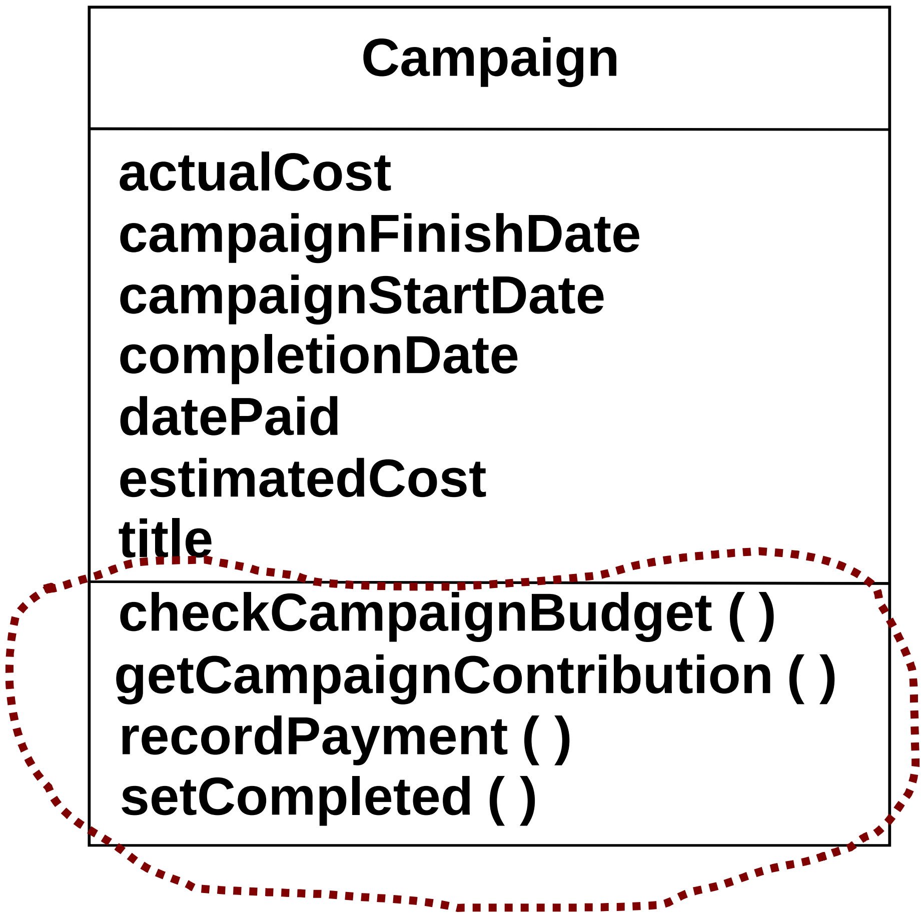
Class Diagram: Operations

Operations are:

- An essential part of the description of a class
- The common behaviour shared by all objects of the class
- Services that objects of a class can provide to other objects

Class Diagram: Operations

- Operations describe what instances of a class can do:
  - Set or reveal attribute values
  - Perform calculations
  - Send messages to other objects
  - Create or destroy links



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

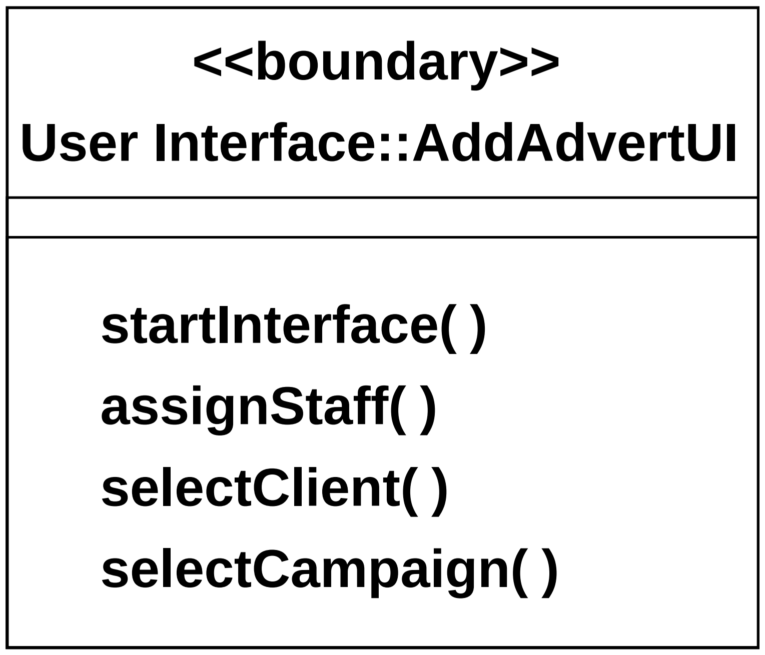
Class Diagram: Stereotypes

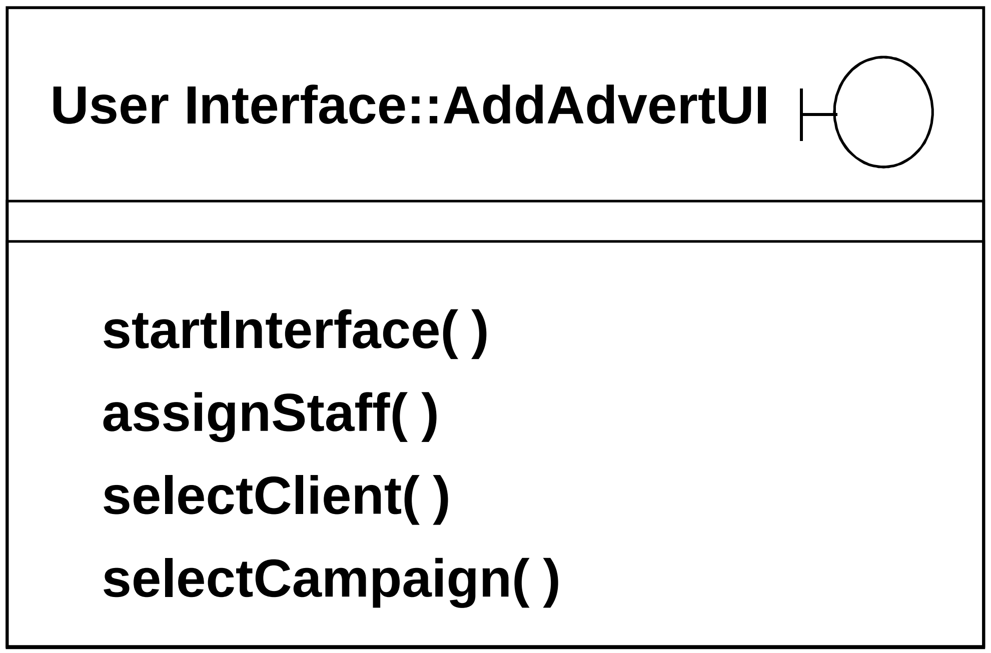
- Instances of a class stereotype have a shared focus on certain kind of things.
- Analysis class stereotypes differentiate the roles objects can play:
  - Boundary objects
  - Entity objects
  - Control objects

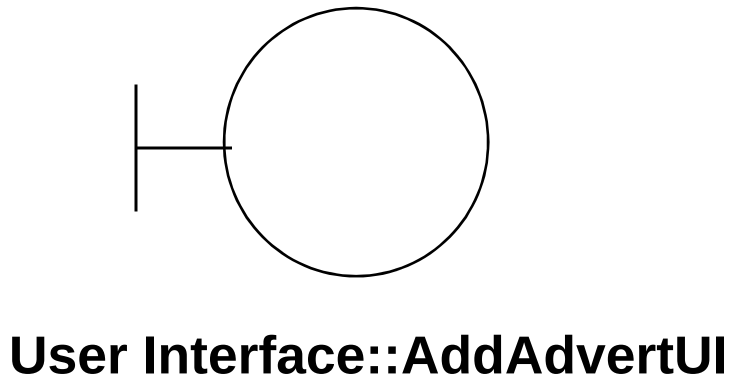
Class Diagram: Stereotypes

- 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



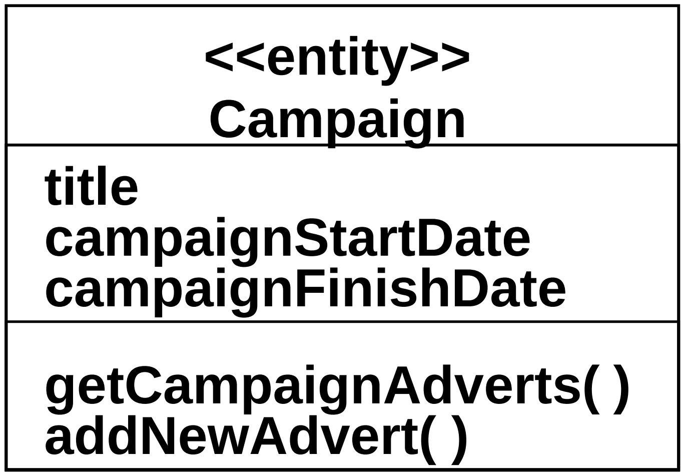


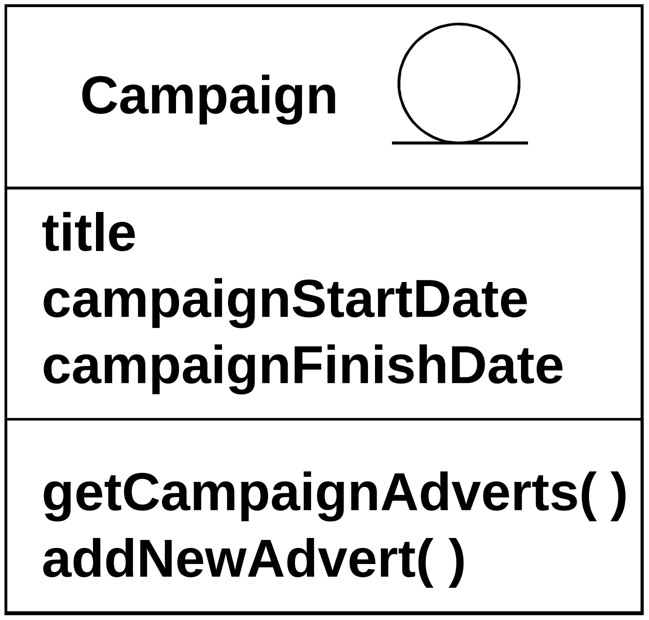


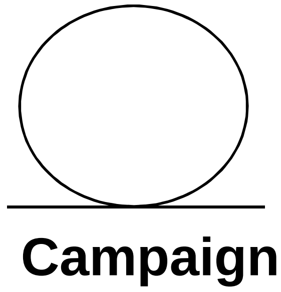
Class Diagram: Stereotypes

- 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



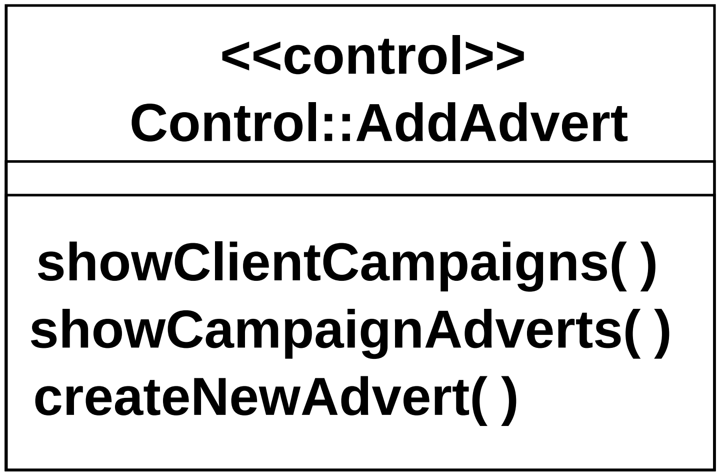


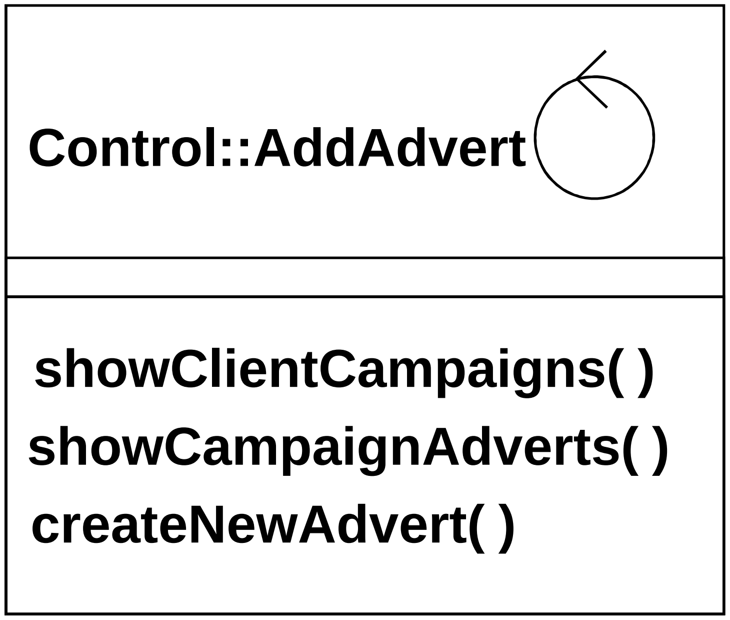


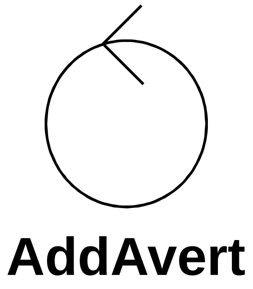
Class Diagram: Stereotypes

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

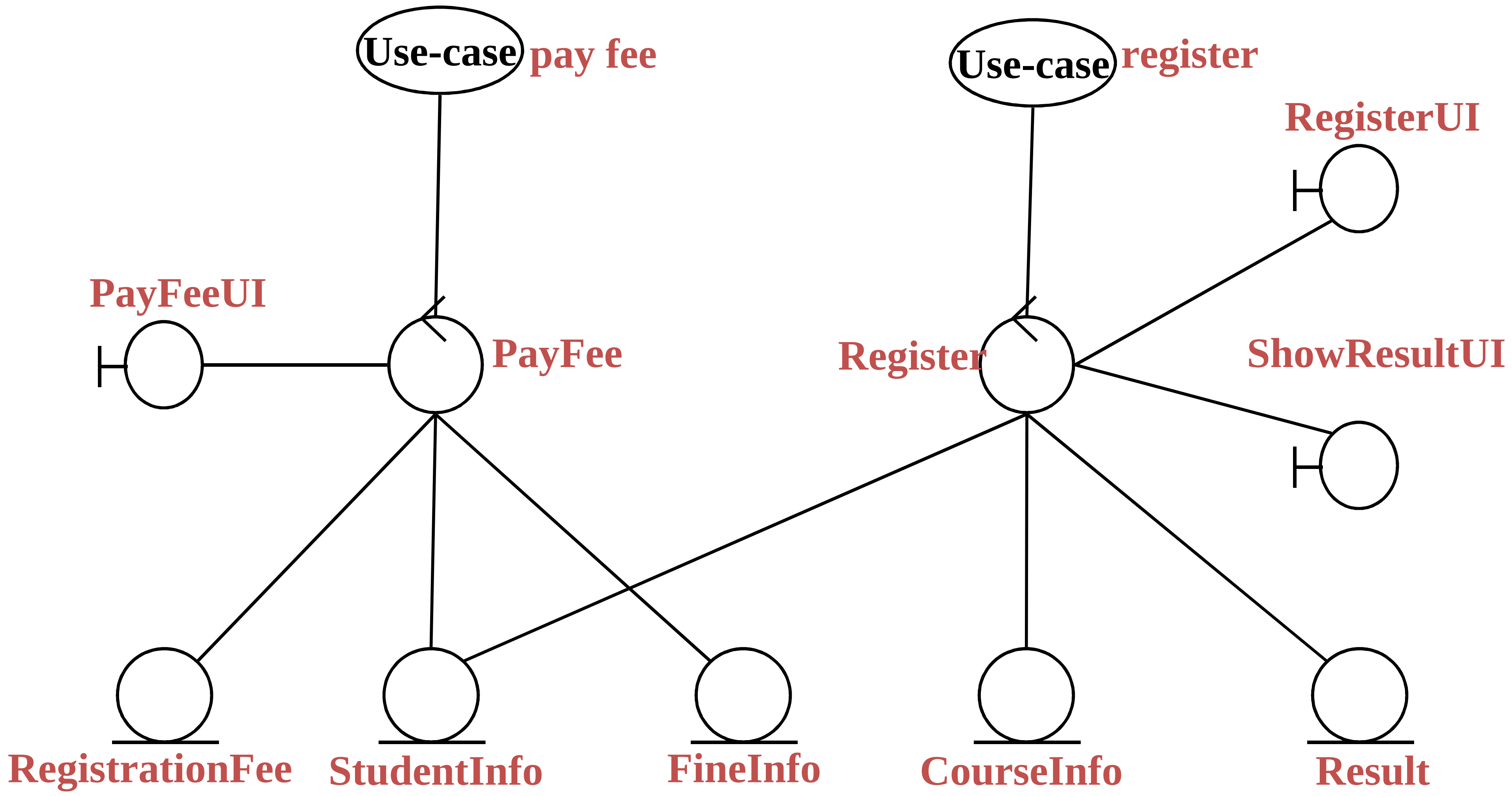
Notations for control class







Class Diagram: Stereotypes



From Use-Case to Classes

- Start with one use case
- Identify the likely classes involved (the use case collaboration)
- Draw a collaboration diagram that fulfils the 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?
  - (More on next slide)

From Use-Case to Classes: Step 2

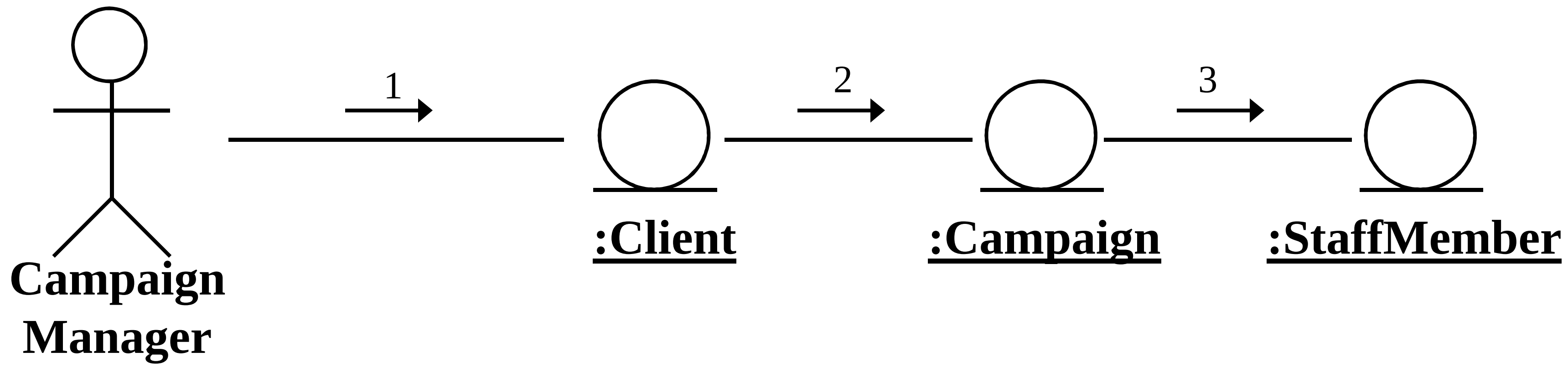
  - Is it too tied up with physical inputs and outputs?
  - Is it really an attribute?
  - Is it really an operation?
  - Is it really an association?
- If any answer is ‘Yes’, consider modelling the potential class in some other way (or do not model it at all)

From Use-Case to Classes: Step 2

- The identified classes
  - Client
  - Campaign
  - StaffMember

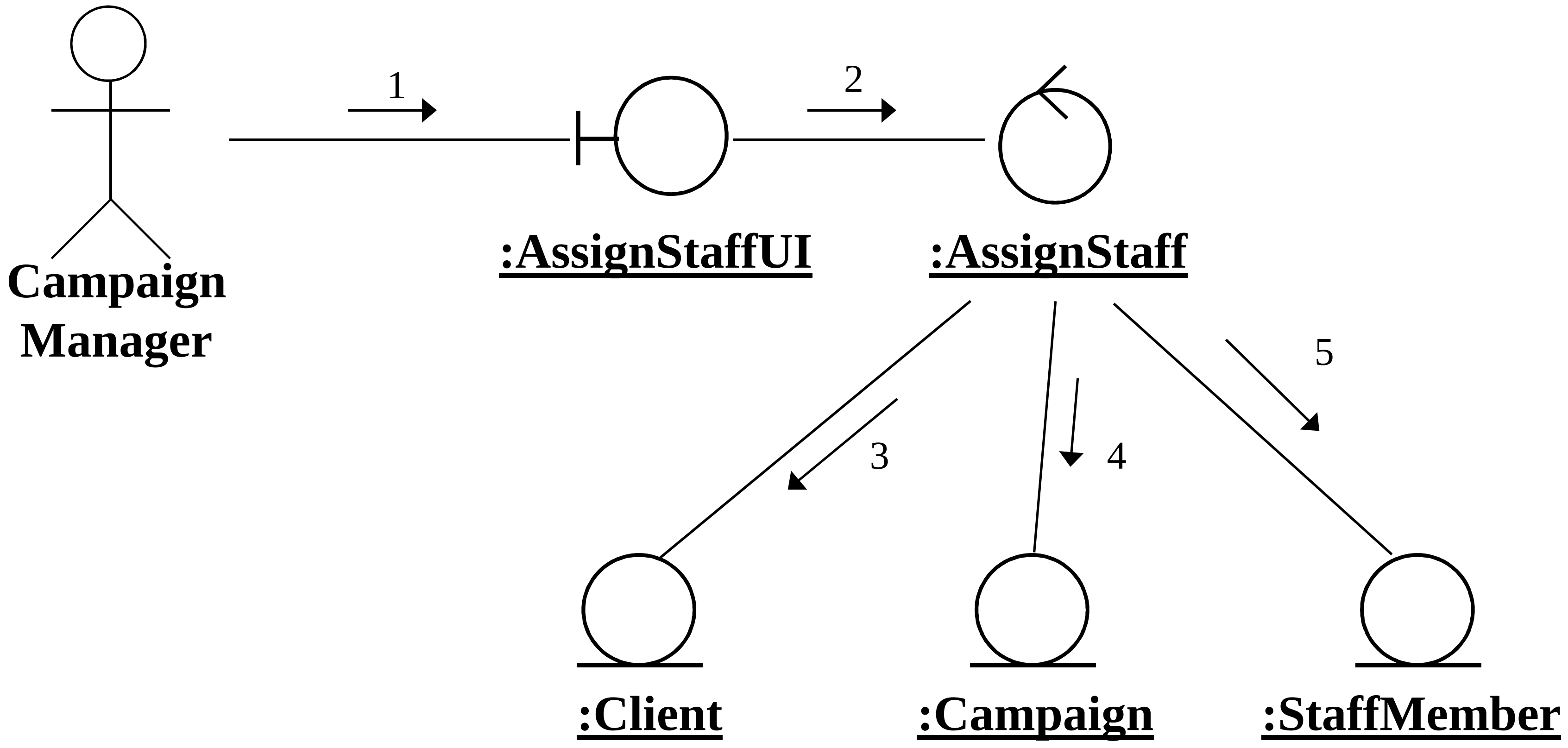
From Use-Case to Classes: Step 3

- Initial collaboration diagram



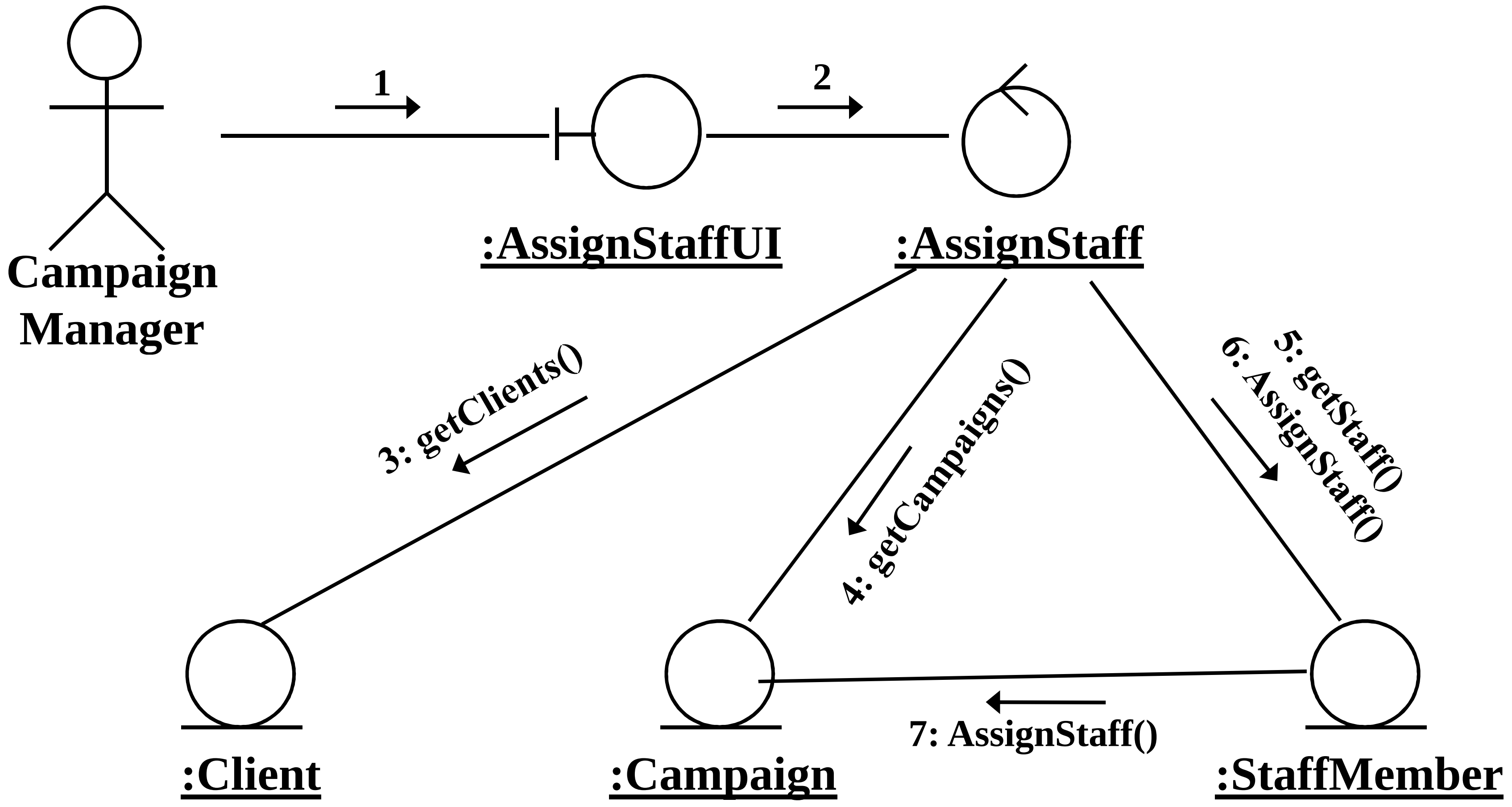
From Use-Case to Classes: Step 3

- Adding boundary and control classes



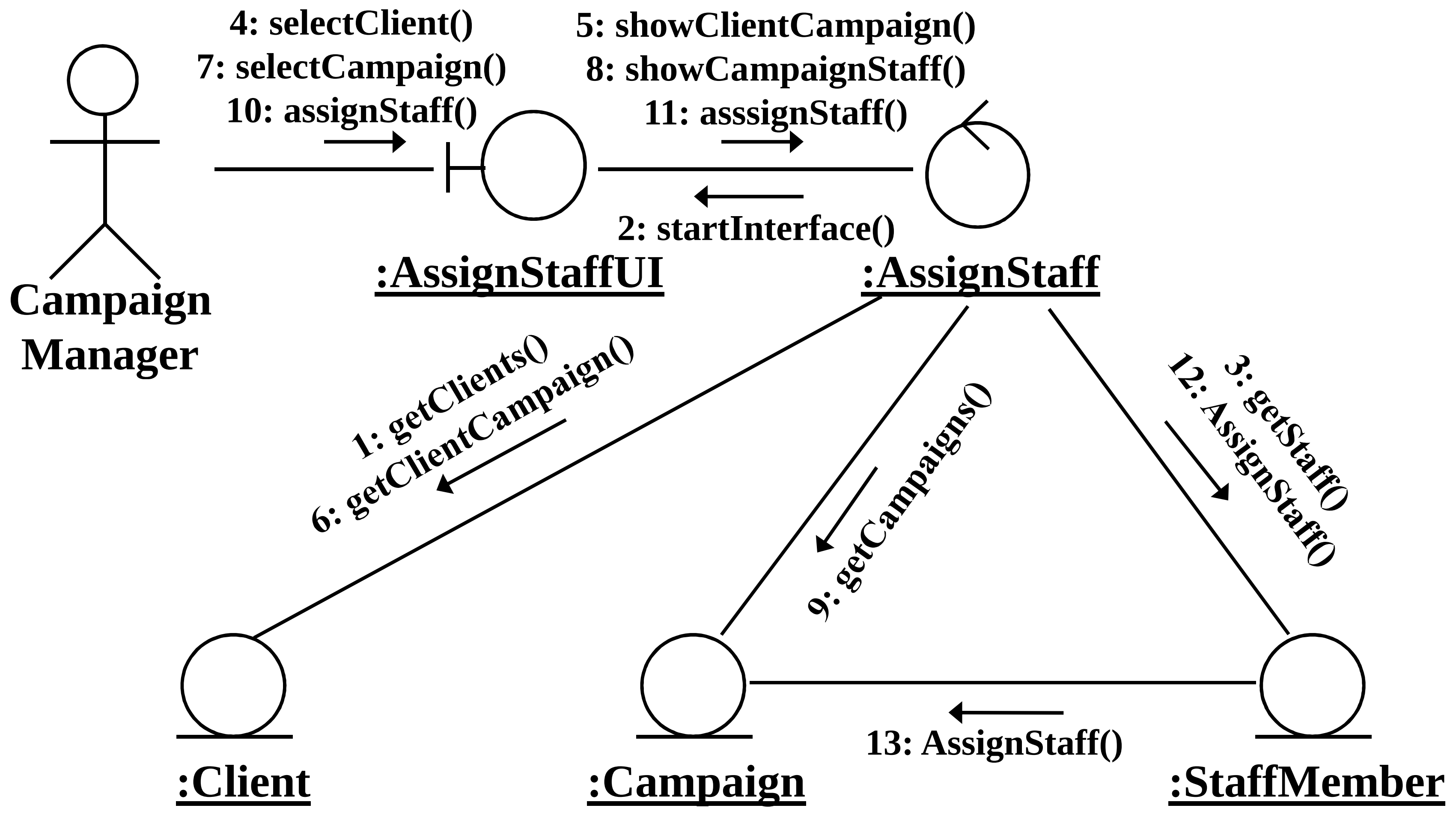
From Use-Case to Classes: Step 3

- Adding messages

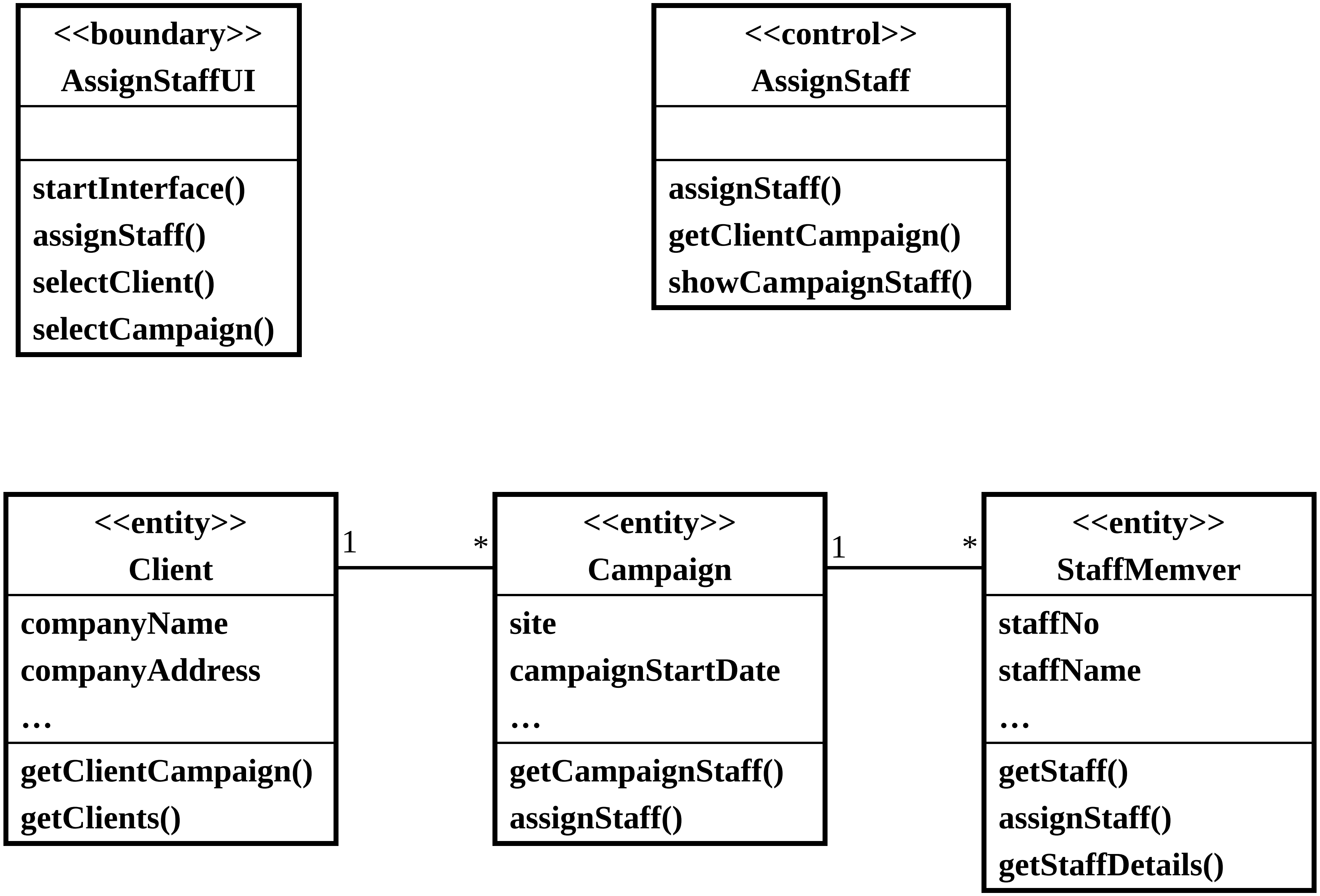


From Use-Case to Classes: Step 3

- Finalizing



From Use-Case to Classes: Step 4



Assigning Operations: CRC Cards

- Class–Responsibility–Collaboration cards help to model interaction between objects
- For a given scenario (or use case):
  - Brainstorm the objects
  - Allocate to team members
  - Role play the interaction

CRC Cards

Class Name:	
Responsibilities	Collaborations
Responsibilities of a class are listed in this section.	*Collaborations with other classes are listed here, together with a brief description of the purpose of the collaboration.*

Class Name	Client
Responsibilities	**Collaborations**
Provide client information.	
Provide list of campaigns.	Campaign provides campaign details.

Class Name	Campaign
Responsibilities	**Collaborations**
Provide campaign information.	
Provide list of adverts.	Advert provides advert details.
Add a new advert.	Advert constructs new object.

Class Name	Advert
Responsibilities	**Collaborations**
Provide advert details.	
Construct adverts.	

CRC Cards

- Effective role play depends on an explicit strategy for distributing responsibility among classes
- For example:
  - Each role player tries to be lazy
  - Persuades other players *their* class should accept responsibility for a given task