Software analysis and design: realisation

- Why analyse requirements
- Class diagrams

CE202 Software Engineering, Autumn term

Dr Michael Gardner, School of Computer Science and Electronic Engineering, University of Essex

Why Analyse Requirements?

- Requirements (Use Case) model alone is not enough
  - There may be repetition
  - Some parts may already exist as standard components
  - Use cases give little information about structure of software system

The Purpose of Analysis

- Analysis aims to identify:
  - A software structure that can meet the requirements
  - Common elements among the requirements that need only be defined once
  - Pre-existing elements that can be reused
  - The interaction between different requirements

What an Analysis Model Does

An analysis model must confirm what users want a new system to do:

- Understandable for users
- Correct scope
- Correct detail
- Complete
- Consistent between different diagrams and models

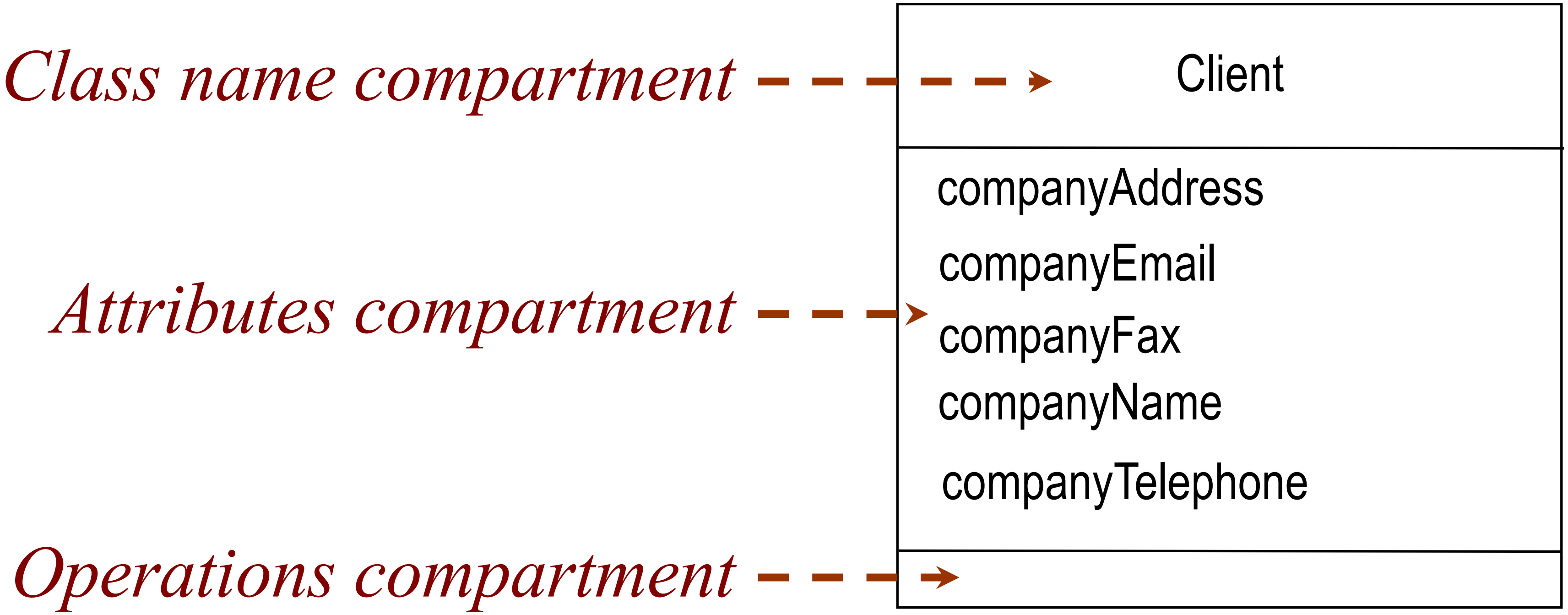
How to Model the Analysis

- The main technique for analysing requirements is the class diagram
- Two main ways to produce this:
  - Directly based on knowledge of the application domain (from a Domain Model)
  - By producing a separate class diagram for each use case, then assembling them into a single model (an Analysis Class Model)
- We will look at both approaches

Introduction to Class Diagrams

Class Diagram: Class Symbol

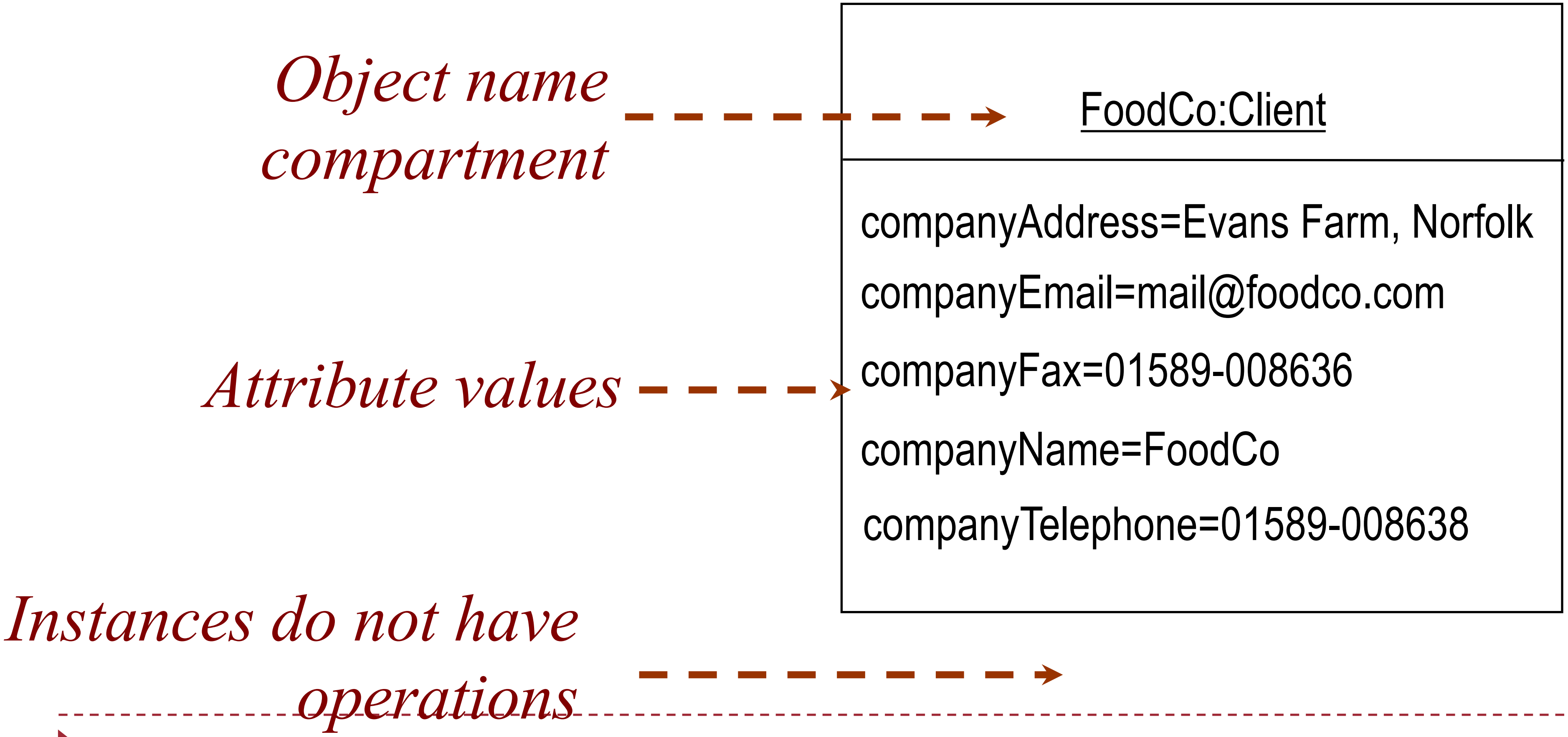
- A Class is "a description of a set of objects with similar features, semantics and constraints" (OMG, 2009)



Some class diagrams may only contain the class name information

Class Diagram: Instances

An object (instance) is: "an abstraction of something in a problem domain…"



Object name compartment

FoodCo:Client

Attribute values

companyAddress=Evans Farm, Norfolk
companyEmail=mail@foodco.com
companyFax=01589-008636
companyName=FoodCo
companyTelephone=01589-008638

Instances do not have operations

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:
  - *Attribute= “value ”*
  - companyName=FoodCo

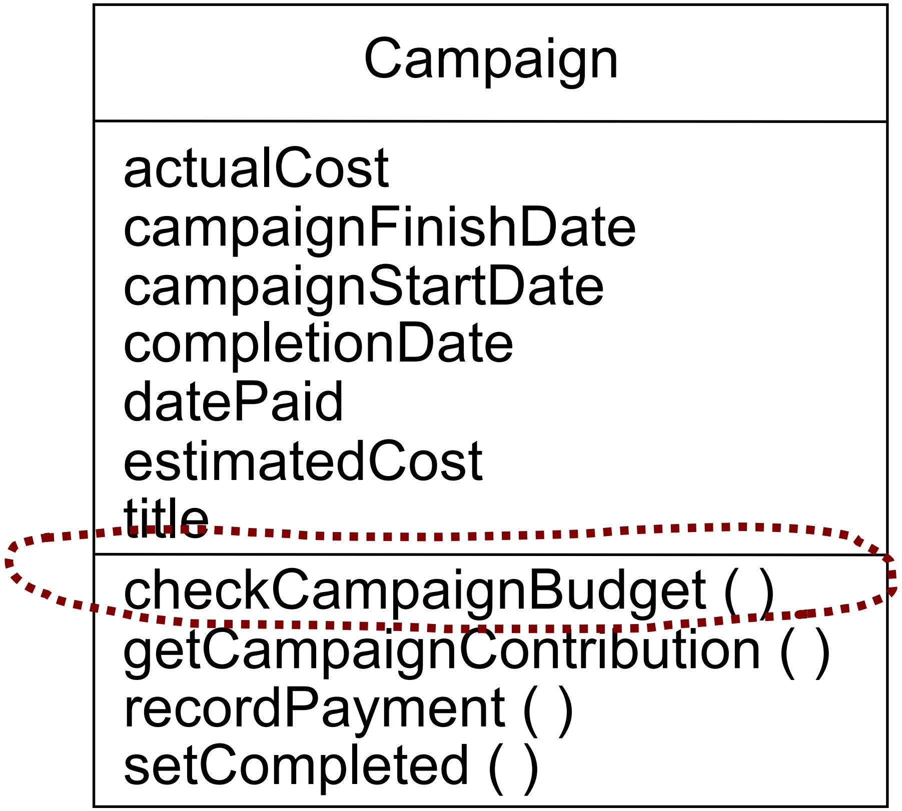
Class Diagram: Operations

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



Campaign

actualCost
campaignFinishDate
campaignStartDate
completionDate
datePaid
estimatedCost
title

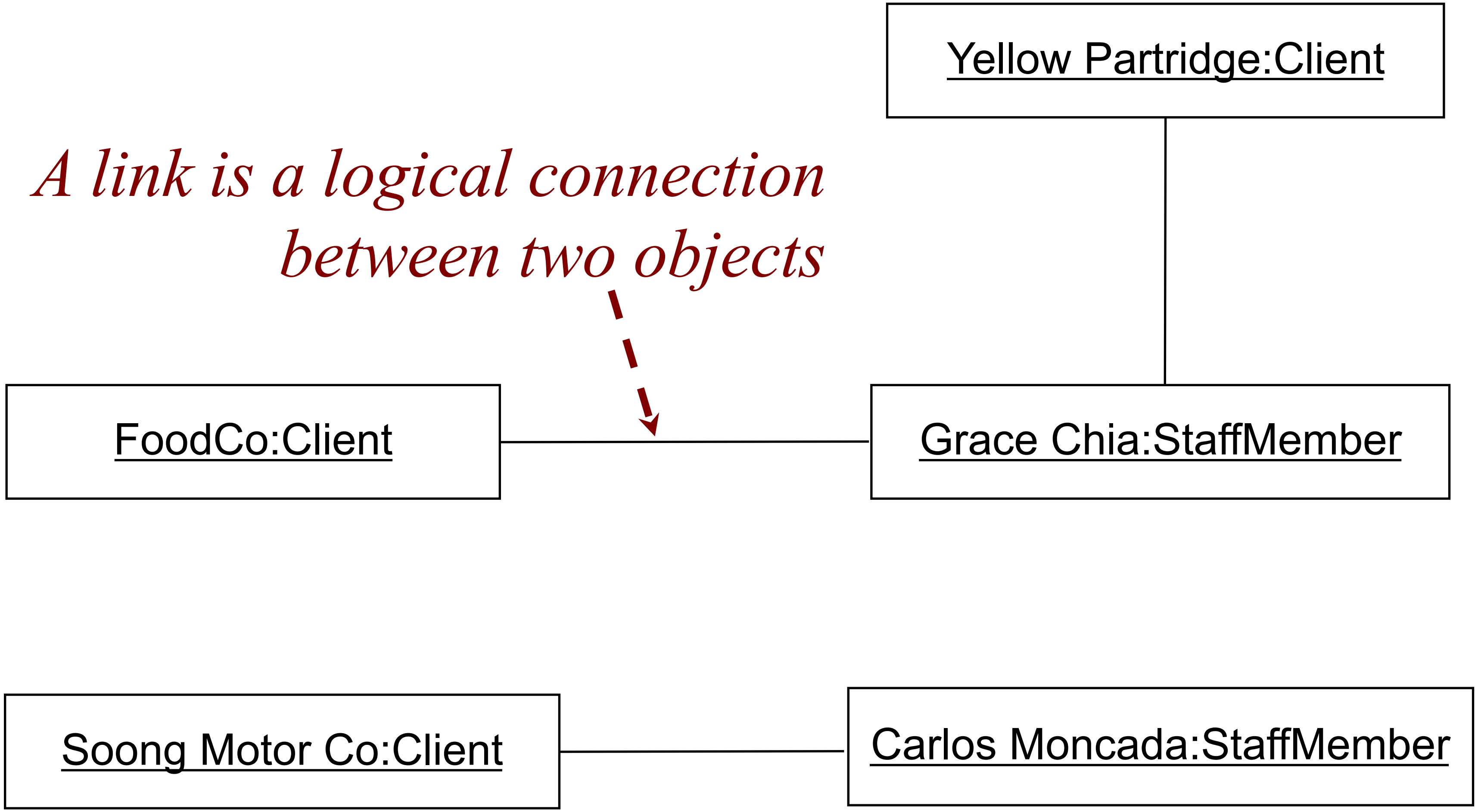
checkCampaignBudget ( )
getCampaignContribution ( )
recordPayment ( )
setCompleted ( )

Class Diagram: Associations

Associations represent:

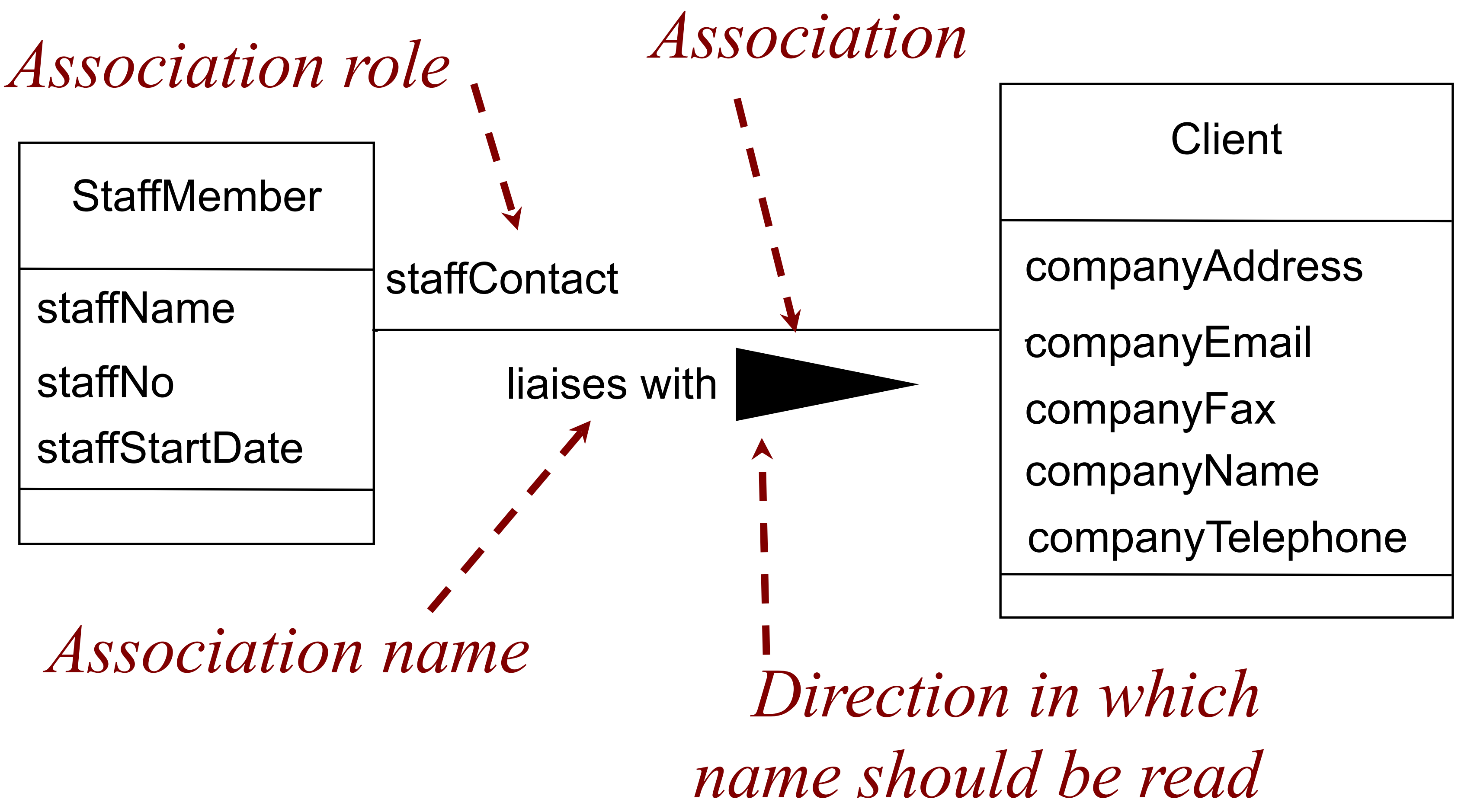
- The possibility of a logical relationship or connection between objects of one class and objects of another
  - "Grace Chia is the staff contact for FoodCo"
  - *An **employee** object is linked to a **client** object*
- If two objects are linked, their classes are said to have an association

Class Diagram: Links



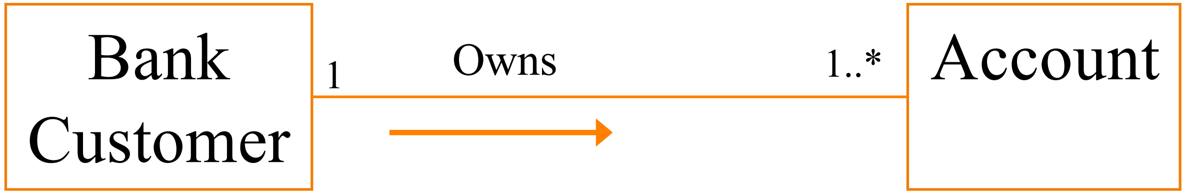
A link is a logical connection between two objects

Class Diagram: Associations

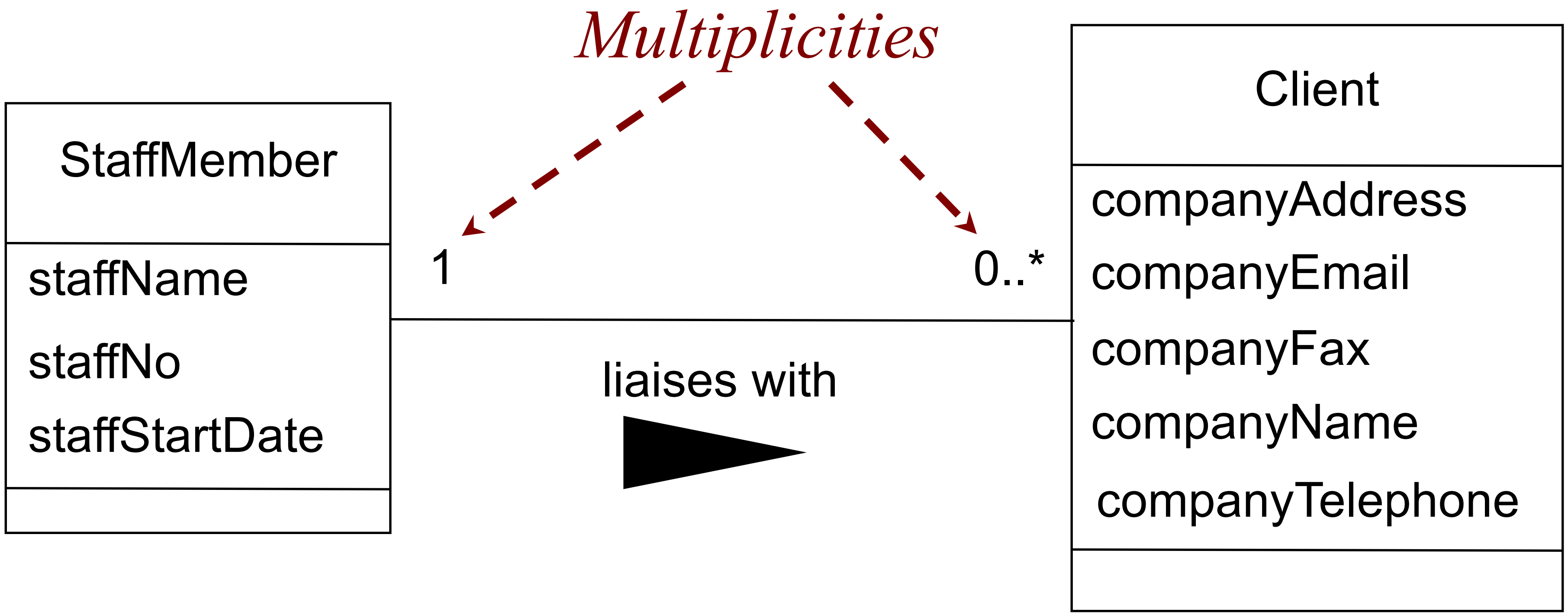


Class Diagram: Multiplicity

- Associations have multiplicity: the range of permitted cardinalities of an association
- Represent *enterprise* (or *business*) *rules*
- These always come in pairs:
  - Associations must be read separately from both ends
  - Each **bank customer** may have 1 or more **accounts**
  - Every **account** is for 1, and only 1, **customer**



Class Diagram: Multiplicity



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

Relationships in class diagrams

Between CLASSES

- Named association with cardinality and direction (can be recursive, can have association classes). See previous slides.
- Inheritance or △
- Whole/Part relationships (discussed next week):
  - Aggregation or ◇
  - Composition or ◆

  (can be included in a named association)

How to create a Class Diagram

Robustness Analysis

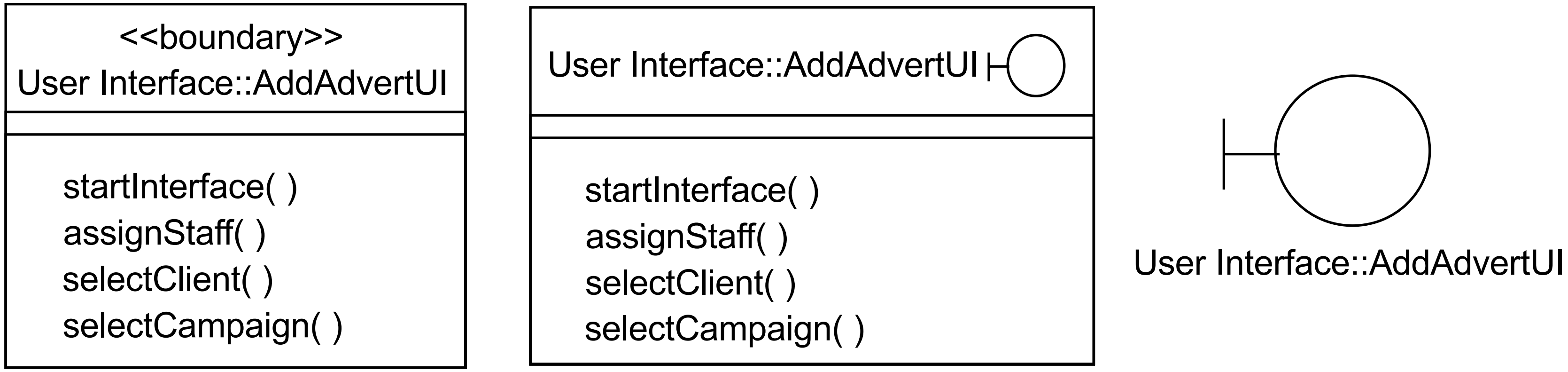
- Aims to produce a set of classes robust enough to meet the requirements of a use case
- Makes some assumptions about the interaction:
  - Assumes some class or classes are needed to handle the user interface
  - Abstracts logic of the use case away from *entity* classes (that store persistent data)

Robustness Analysis: Class Stereotypes

- Class stereotypes differentiate the roles objects can play:
  - **Boundary** objects model interaction between the system and actors (and other systems)
  - **Control** objects co-ordinate and control other objects
  - **Entity** objects represent information and behaviour in the application domain
  - Entity classes may be imported from domain model
  - Boundary and control classes are more likely to be unique to one application

Boundary Class Stereotype

- Boundary classes represent interaction with the user - likely to be unique to the use case but inherited from a library
- Alternative notations:



<<boundary>>
User Interface::AddAdvertUI

startInterface( )
assignStaff( )
selectClient( )
selectCampaign( )

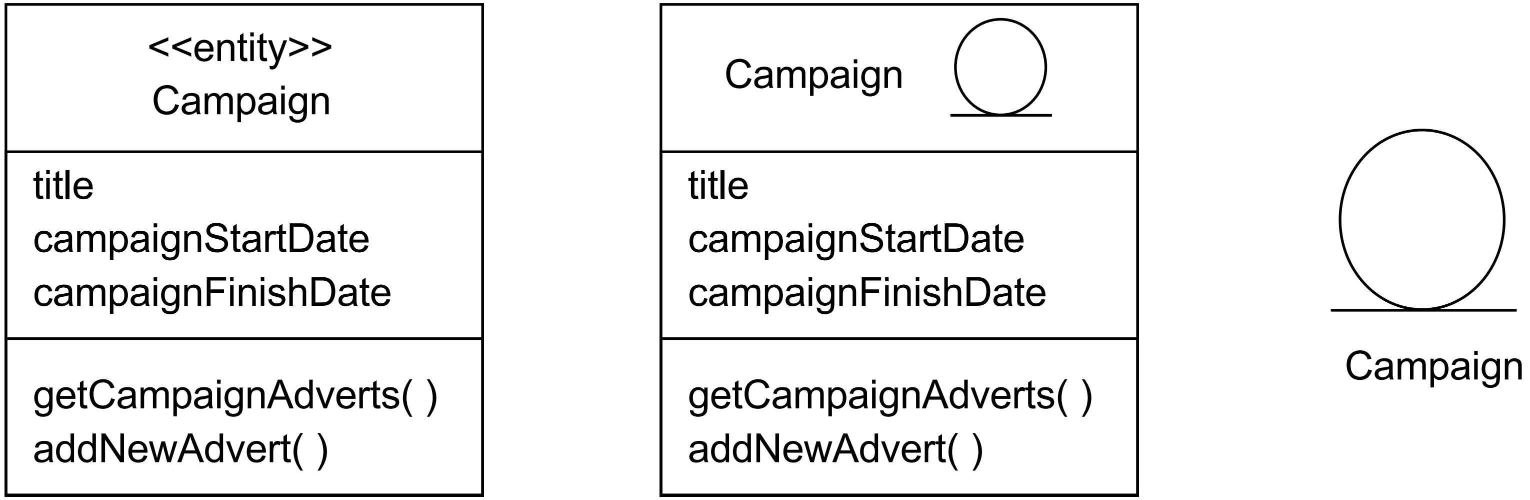
User Interface::AddAdvertUI

startInterface( )
assignStaff( )
selectClient( )
selectCampaign( )

User Interface::AddAdvertUI

Entity Class Stereotype

- Entity classes represent persistent data and common behaviour likely to be used in more than one application system
- Alternative notations :



<<entity>>
Campaign

title
campaignStartDate
campaignFinishDate

getCampaignAdverts( )
addNewAdvert( )

Campaign

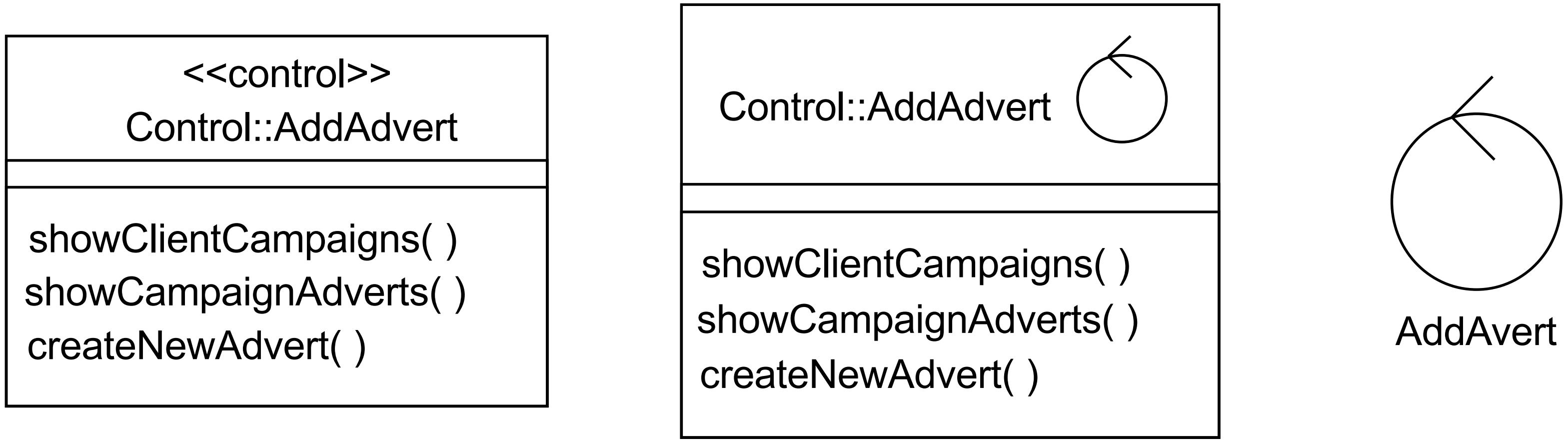
title
campaignStartDate
campaignFinishDate

getCampaignAdverts( )
addNewAdvert( )

Campaign

Control Class Stereotype

- Control classes encapsulate unique behaviour of a use case
- Specific logic kept separate from the common behaviour of entity classes
- Alternative notations:



<<control>>
Control::AddAdvert

showClientCampaigns( )
showCampaignAdverts( )
createNewAdvert( )

Control::AddAdvert

showClientCampaigns( )
showCampaignAdverts( )
createNewAdvert( )

AddAvert

From Requirements to Classes

- Requirements (use cases) are usually expressed in user language
- Use cases are units of development, but they are not structured like software
- The software we will implement consists of classes
- We need a way to translate requirements into classes

Goal of Realization

- An analysis class diagram is only an interim product
- This in turn will be realized as a design class diagram
- The ultimate product of realization is the software implementation of that use case

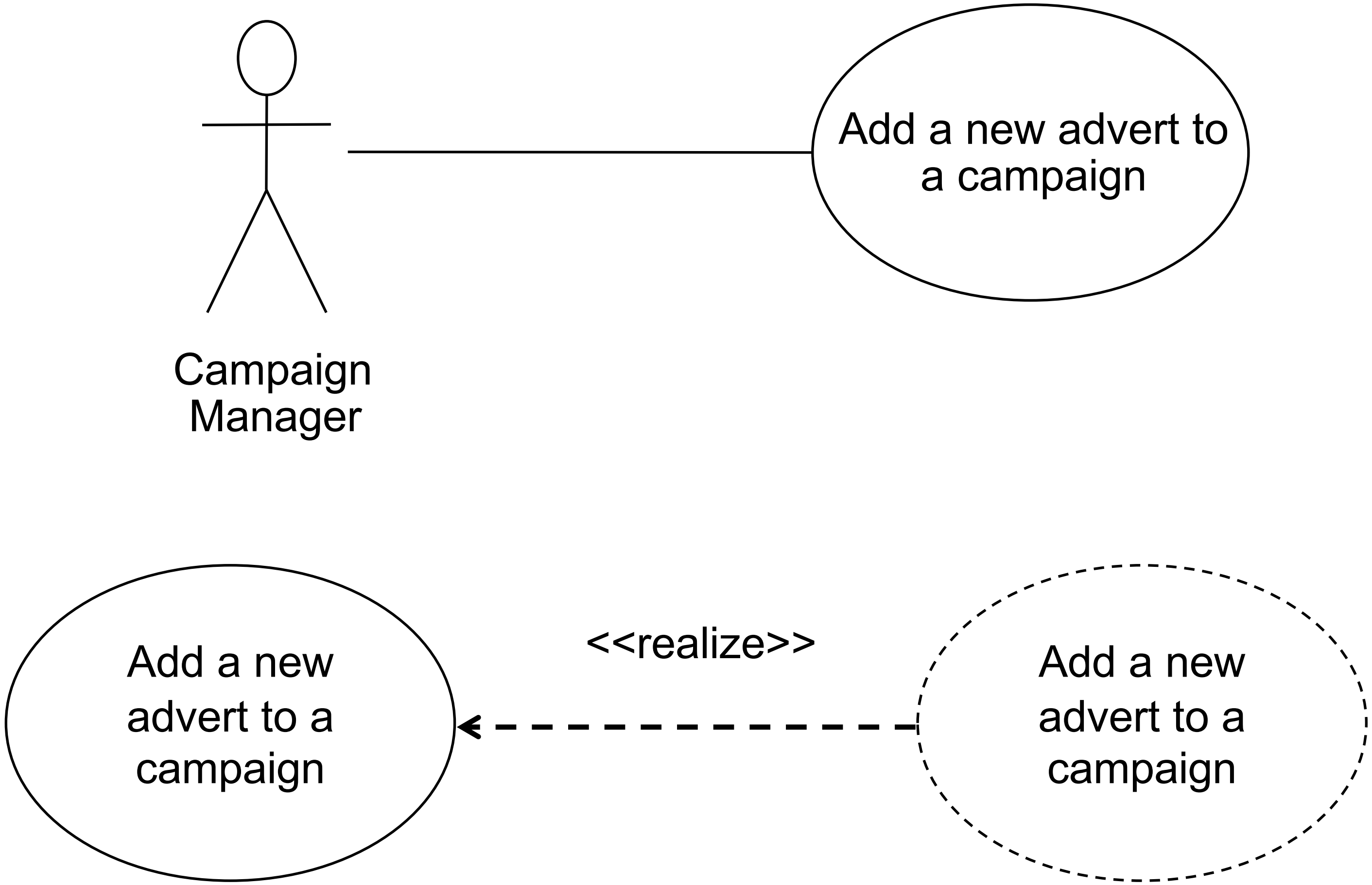
Assembling the Class Diagram from use cases

- However individual use cases are analysed, the aim is to produce a single analysis class diagram
- This models the application as a whole
- The concept is simple:
  - A class in the analysis model needs *all* the details required for that class in each separate use case
  - You then assemble a final single class diagram from the separate class diagrams (from each use case)

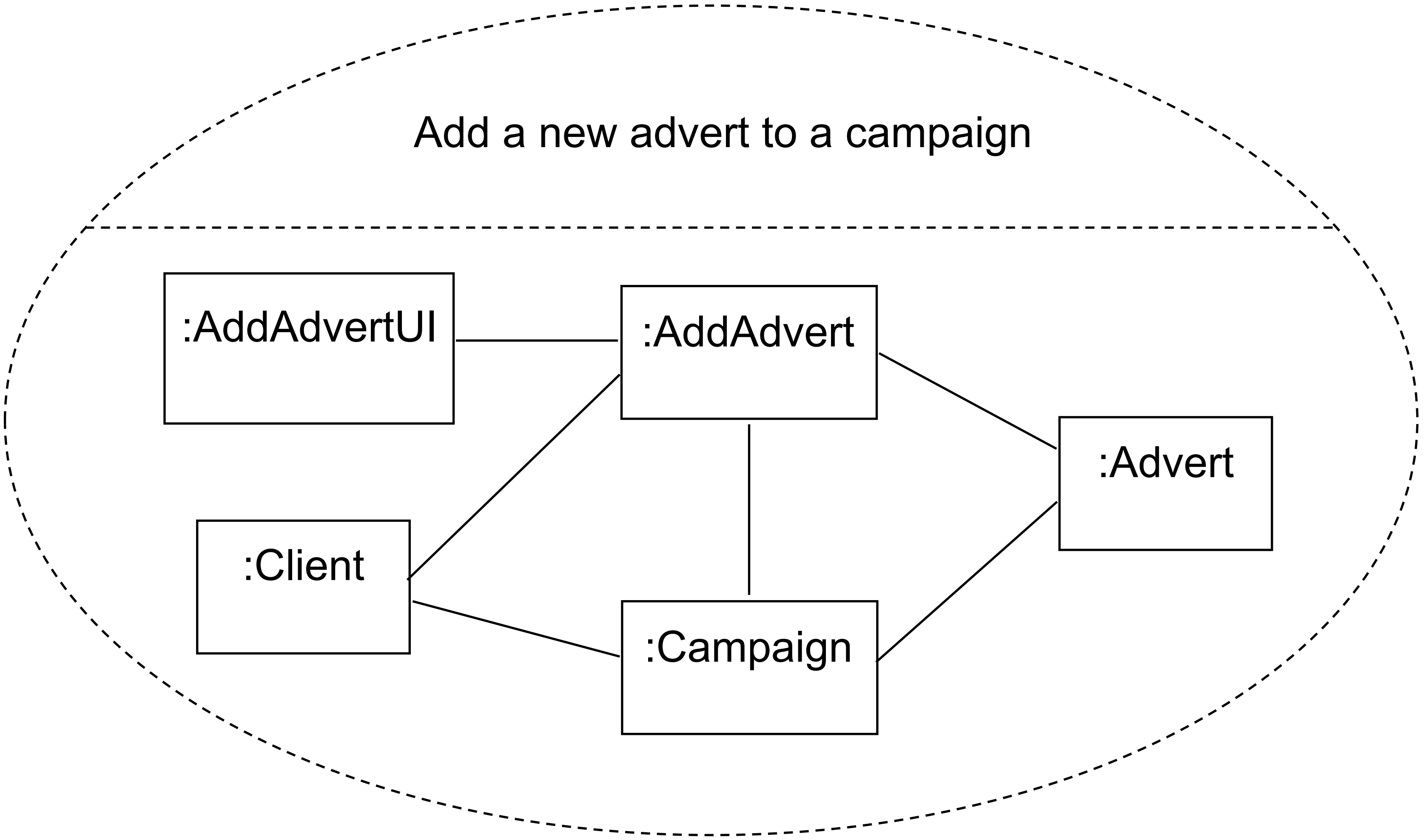
Realization of class diagram (based on knowledge of the application domain) - process

- 1. Define Entity objects
- 2. Then add in Control objects
- 3. Then add in Boundary objects

Example: assembling a class diagram from a Use Case

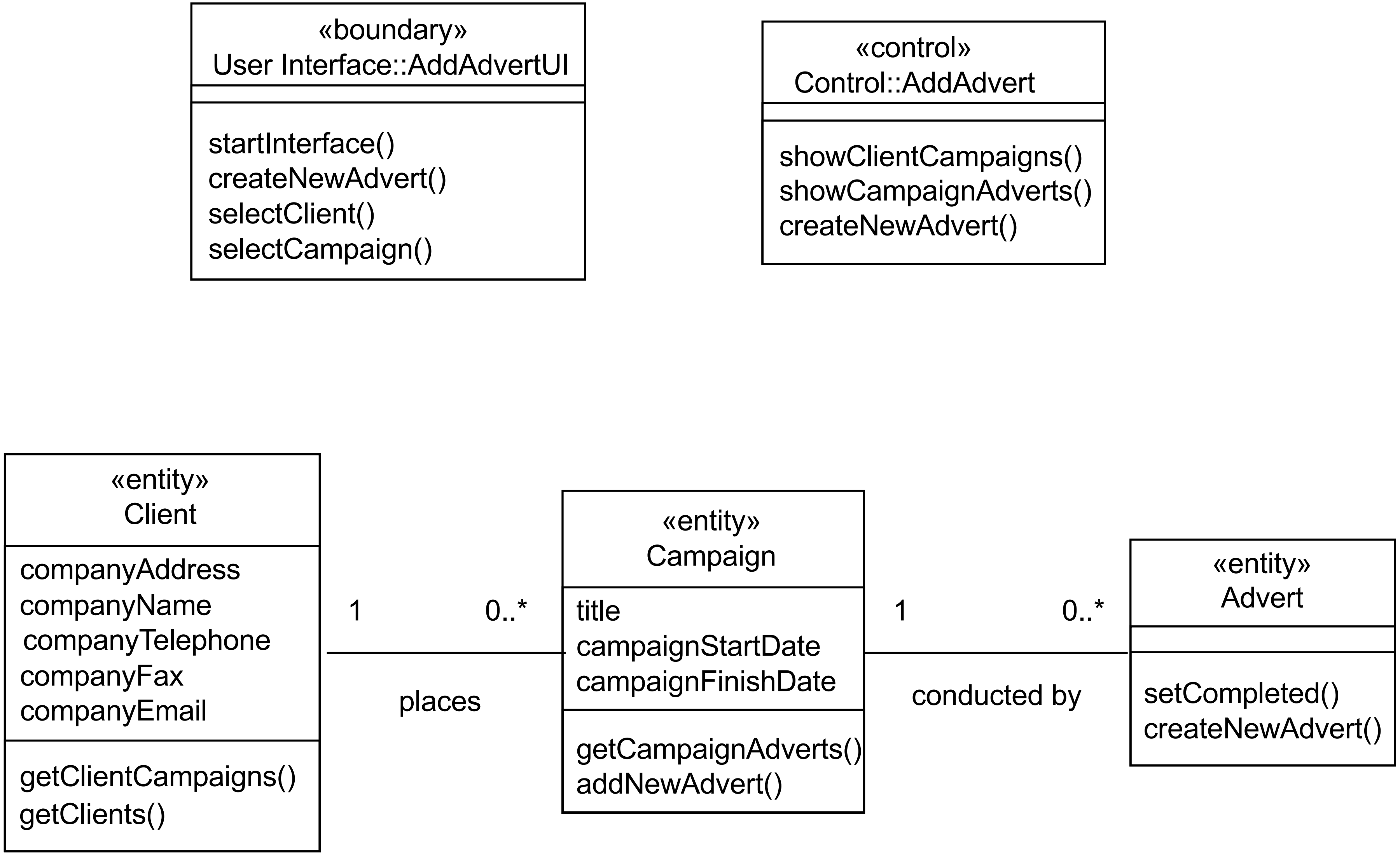

Campaign Manager
Add a new advert to a campaign
Add a new advert to a campaign
<<realize>>
Add a new advert to a campaign

A Possible Collaboration (diagram)



Add a new advert to a campaign

Resulting (incomplete) Class Diagram



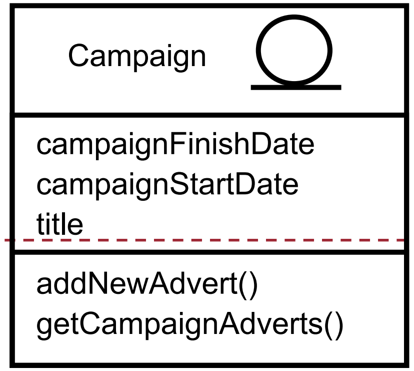
Reasonability Checks for Candidate Classes

- 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 vague?

(More on next slide)

Reasonability Checks for Candidate Classes (cont'd)

  - 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)
- Only show permanent/static relationships on the class diagram. Leave out temporary relationships associated with particular instances

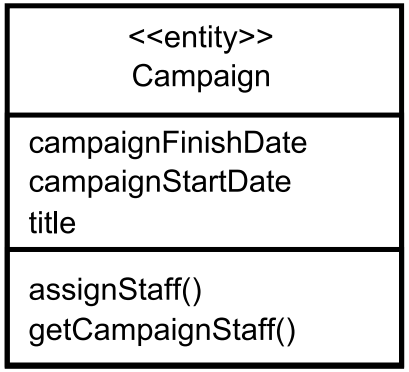


Campaign

campaignFinishDate
campaignStartDate
title

addNewAdvert()
getCampaignAdverts()

(a) Campaign class that meets the needs of `Add new advert to a campaign`



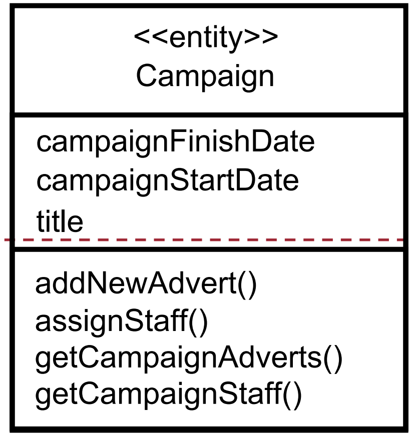
<<entity>>
Campaign

campaignFinishDate
campaignStartDate
title

assignStaff()
getCampaignStaff()

(b) Campaign class that meets the needs of `Assign staff to work on a campaign`

(c) Campaign class that meets the needs of both use cases

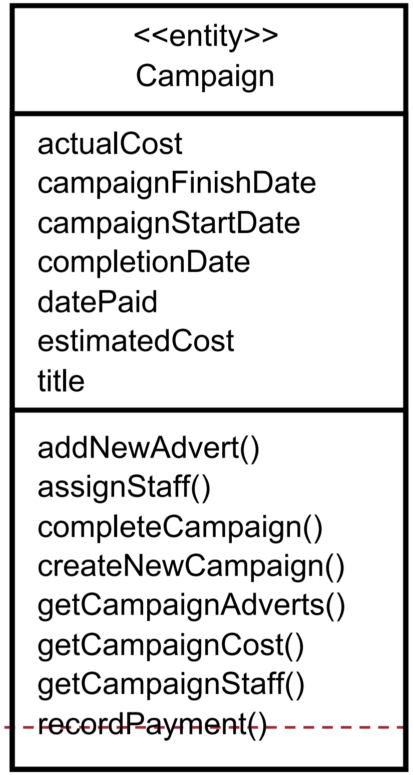


<<entity>>
Campaign

campaignFinishDate
campaignStartDate
title

addNewAdvert()
assignStaff()
getCampaignAdverts()
getCampaignStaff()

(d) A more fully developed Campaign class meets the requirements of these and several other use cases too

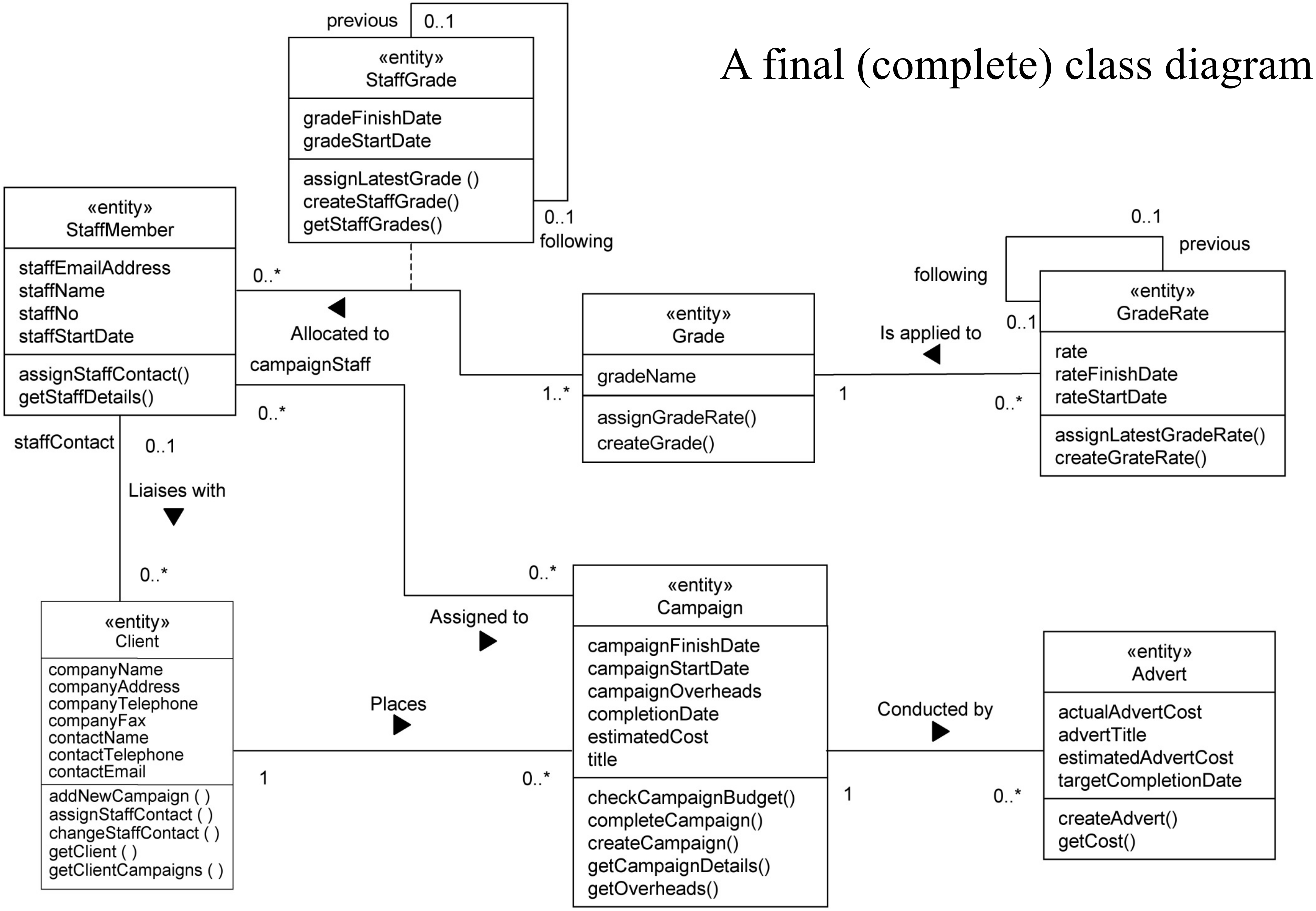


<<entity>>
Campaign

actualCost
campaignFinishDate
campaignStartDate
completionDate
datePaid
estimatedCost
title

addNewAdvert()
assignStaff()
completeCampaign()
createNewCampaign()
getCampaignAdverts()
getCampaignCost()
getCampaignStaff()
recordPayment()

A final (complete) class diagram



Communication Diagram Approach

- **Analyse one use case at a time**
- **Identify likely classes involved (the use case collaboration)**
  - **These may come from a domain model**
- **Draw a communication diagram that fulfils the needs of the use case (see next lecture)**
- Translate this into a use case class diagram
- Repeat for other use cases
- Assemble the use case class diagrams into a single analysis class diagram
- (see next lecture)

Summary

In this lecture you have learned:

- What is meant by *use case realization*
- How to realize use cases with robustness analysis and communication diagrams
- How to assemble the analysis class diagram
- Started looking at the syntax of class diagrams

Exercises

1. Draw a class diagram representing a book defined by the following statement: "A book is composed of a number of parts, which in turn are composed of a number of chapters. Chapters are composed of sections." Focus only on classes and relationships.
2. Add multiplicity to the class diagram you produced.
3. Extend the class diagram to include the following attributes:
   - a book includes a publisher, publication date, and an ISBN
   - a part includes a title and a number
   - a chapter includes a title, a number, and an abstract
   - a section includes a title and a number
4. Note that the Part, Chapter, and Section classes all include a title and a number attribute. Add an abstract class and a generalization relationship to factor out these two attributes into the abstract class.

Further reading

- Parnas (1985) - SOFTWARE ASPECTS OF STRATEGIC DEFENSE SYSTEMS, 1985 ACM OflOl-0782/85/1200-1326 750
- Designing Object-Oriented Software, Rebecca J Wirfs-Brock, Brian Wilkerson, and Lauren Wiener, Prentice Hall, 1990
- See Chapter 7 Bennett