WIA2002: Software Modelling

Semester 1, Session 2016/17

Lecture 6: Structural Modelling - UML Class Diagrams

(Part 2)

Learning Objectives

- Understand the concepts and notation of the object diagram.
- Understand what is meant by use case realization.
- Explore two approaches for realizing use cases:
  - Class-Responsibility-Collaboration (CRC).
  - Robustness analysis combined with communication diagrams.
- Understand how to combine use case class diagrams into a single analysis class model.

UML OBJECT DIAGRAMS

Object Diagrams

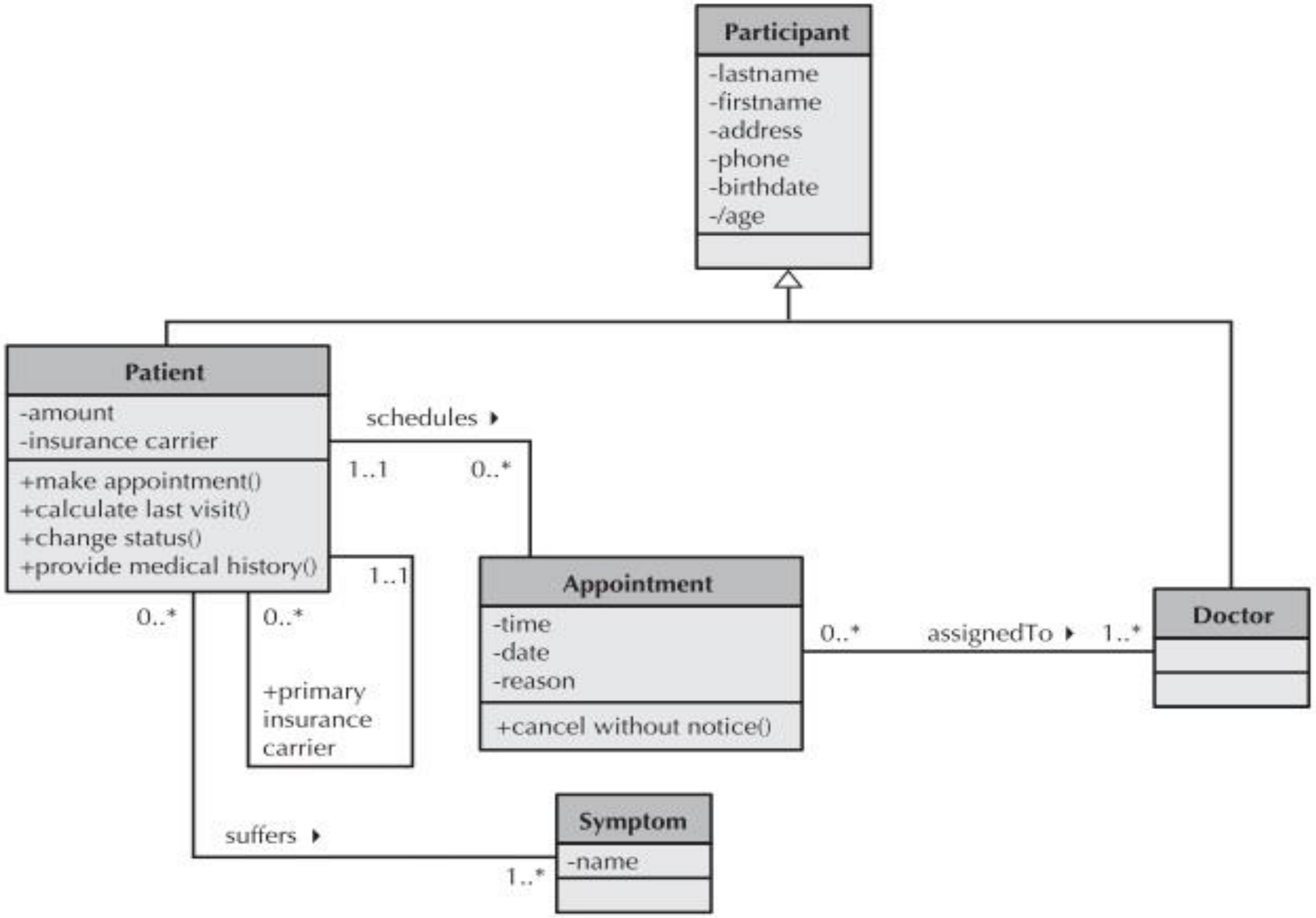
- A second type of static structure diagram.
- Can be useful in revealing additional information.
- An object diagram is essentially **an instantiation of all or part of a class diagram**.
- Instantiation means to create an instance of the class with *a set of appropriate attribute values*.
- It is easier to think in terms of concrete objects (instances) rather than abstractions of objects (classes).

Object Diagrams

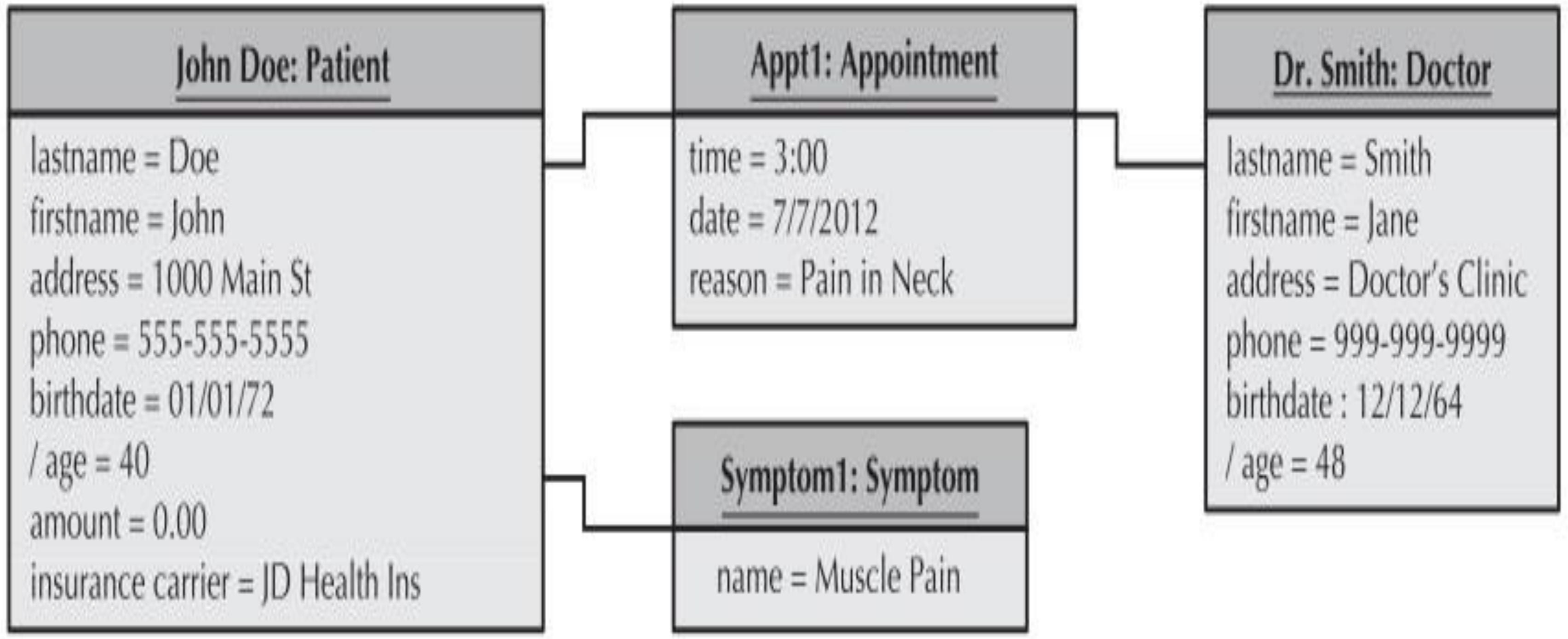
- Model the instances of things contained in class diagrams
  - Used to model the static design view or static process view of a system.
  - Involves modelling a snapshot of part of the structure of the system being modelled.

- Same basic appearance as a class diagram, except that it shows objects, and actual values for attributes, instead of classes.

Class Diagrams (Example)



Object Diagrams (Example)



Object Diagrams

Notation for object diagrams:

1. Top compartment of the class box: the name of the class to which the object belongs to appears after a colon.
    - The object may have a name that appears before the colon, or it may be anonymous (no name).
2. The contents of the top compartment are underlined for an object.
3. Lower compartment of the class box: each attribute defined for the given class has a specific value for each object that belong to that class.

REALIZING USE CASES

Analysis vs Design

- In analysis, analysts draw a conceptual model
  - To show the logical organization of the objects without indicating how the objects are stored, created, or manipulated.
  - Free from any implementation or technical details, can focus more easily on matching the model to the real business requirements of the system.
- In design, analysts evolve the conceptual structural model into a design model
  - To reflect how the objects will be organized in databases and software.
  - The model is checked for redundancy, and the analysts investigate ways to make the objects easy to retrieve.

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.

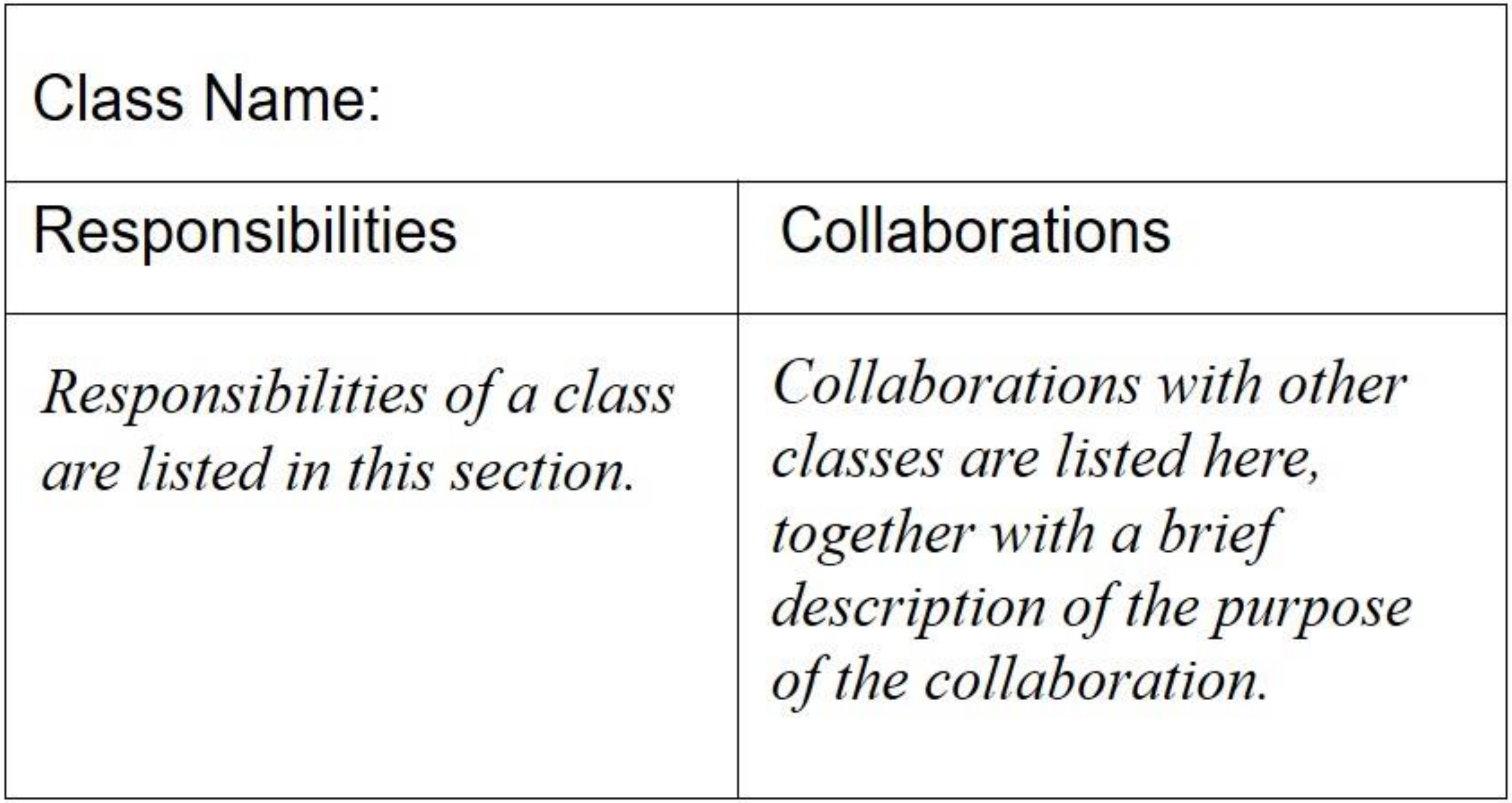
Approach 1: CRC Cards

- CRC (Class–Responsibility–Collaboration) cards are used to:
    - document the responsibilities and collaborations of a class.
    - model interaction between objects.
    - identify the classes, along with the attributes, operations, and relationships, involved with a use case.

Approach 1: CRC Cards

- Used as a way of:
  - Identifying classes that participate in a scenario.
  - Allocating responsibilities - both operations and attributes (*what can I do?* and *what do I know?*).
- 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.*

CRC Cards

- Responsibilities of a class can be broken into two separate types: **knowing** and **doing**.
- Knowing responsibilities are those things that an instance of a class must be capable of knowing.
  - An instance of a class typically knows the values of its attributes and its relationships.
- Doing responsibilities are those things that an instance of a class must be capable of doing.
  - An instance of a class can execute its operations or it can request a second instance, to execute one of its operations on behalf of the first instance.

Sample CRC Cards

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. Add a new advert.	Advert provides advert details. Advert constructs new object.

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

Role-Playing CRC Cards with Use Cases

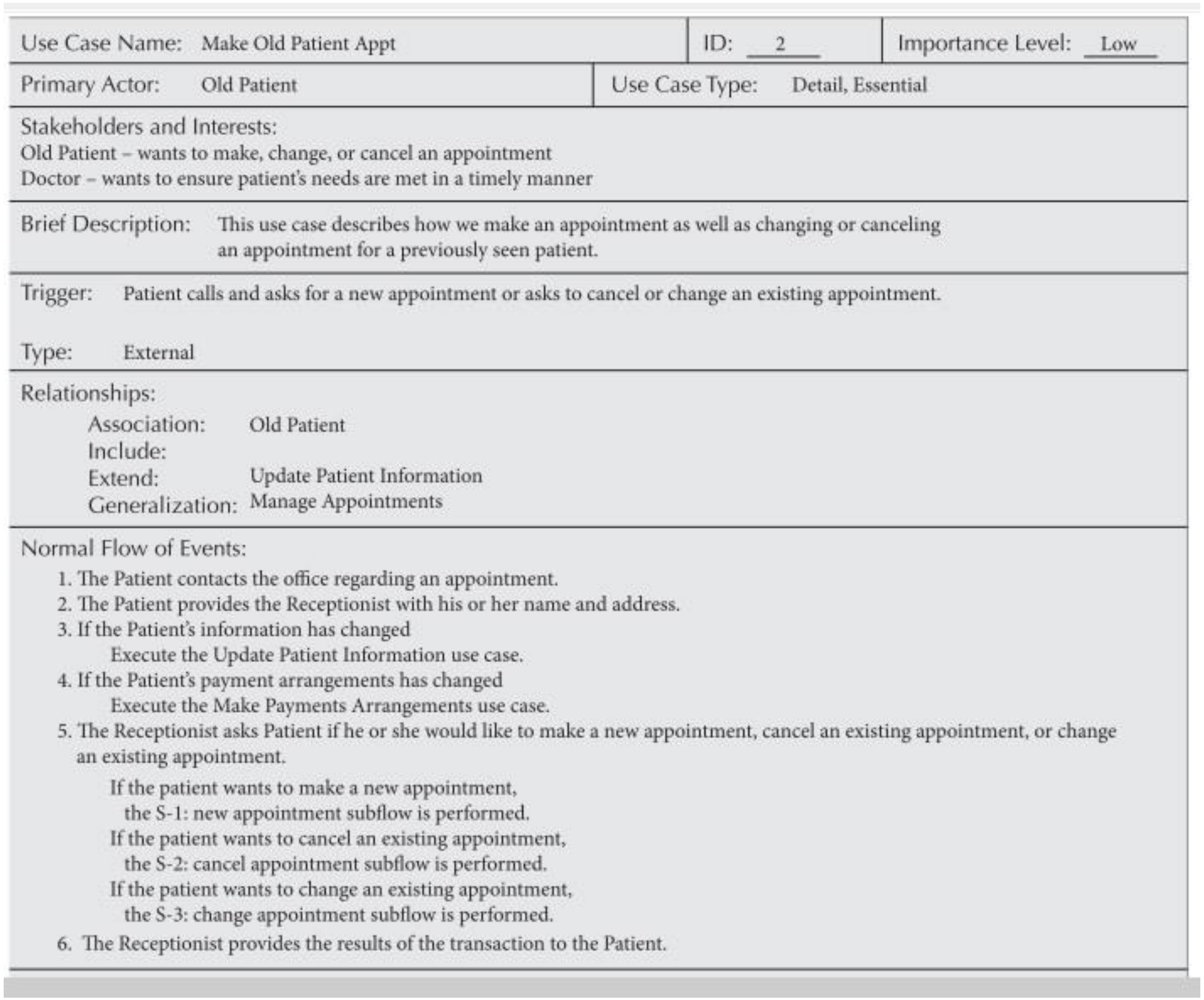
- CRC cards can be used in a role-playing exercise
- Useful in discovering additional objects, attributes, relationships, and operations.
- Effective role play depends on an explicit strategy for distributing responsibility among classes.

Role-Playing CRC Cards with Use Cases

4 steps:

1. Review use cases to pick a specific use case to role-play.
2. Identify relevant actors and objects (review each of the CRC cards).
3. Role-play scenarios.
4. Repeat Steps 1 through 3 for the remaining use cases.

A Sample Use Case Description



Use Case Name: Make Old Patient Appt | ID: 2 | Importance Level: Low

Primary Actor: Old Patient | Use Case Type: Detail, Essential

Stakeholders and Interests:
Old Patient – wants to make, change, or cancel an appointment
Doctor – wants to ensure patient's needs are met in a timely manner

Brief Description: This use case describes how we make an appointment as well as changing or canceling an appointment for a previously seen patient.

Trigger: Patient calls and asks for a new appointment or asks to cancel or change an existing appointment.

Type: External

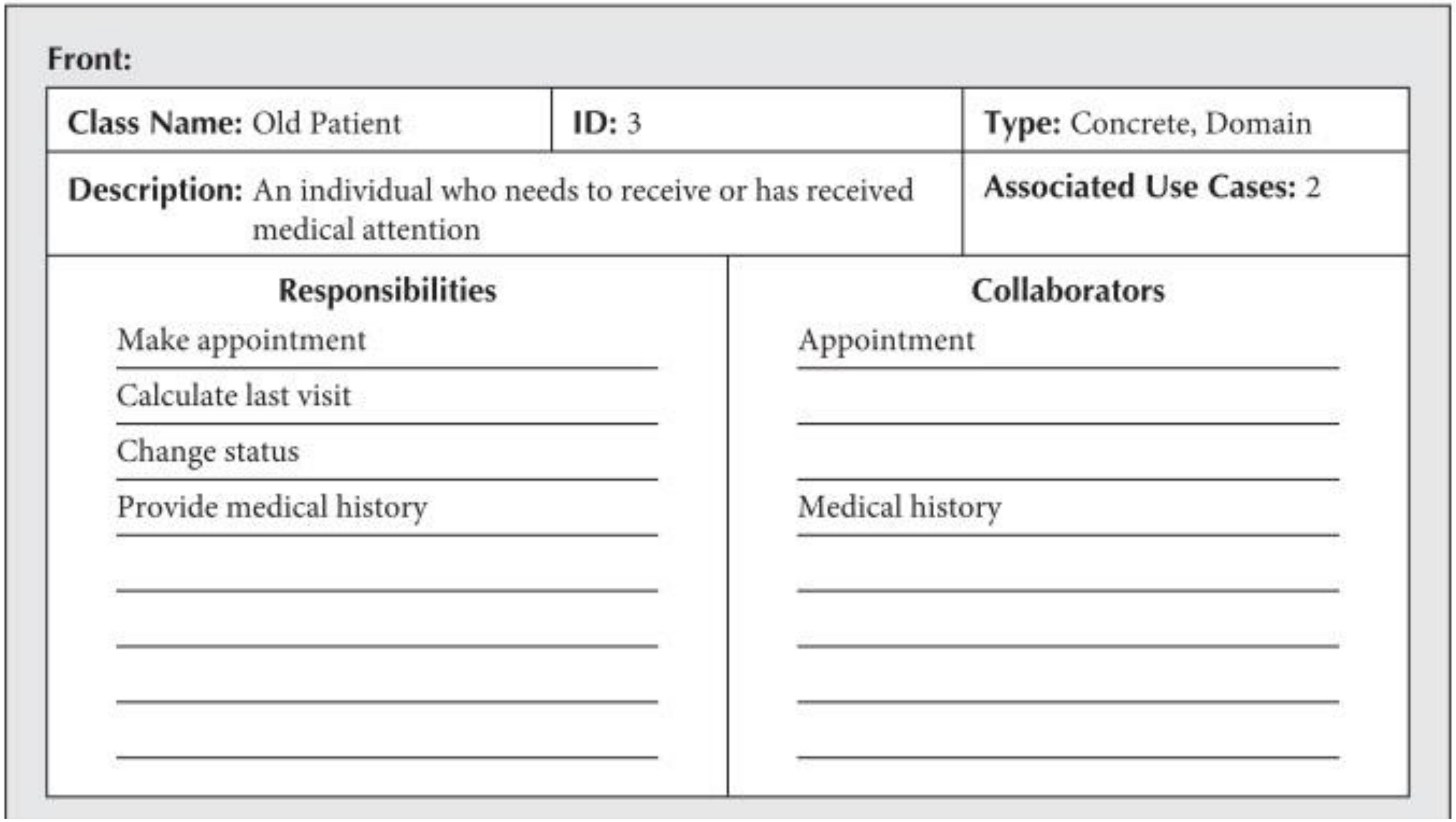
Relationships:

Association:	Old Patient
Include:	
Extend:	Update Patient Information
Generalization:	Manage Appointments

Normal Flow of Events:

1. The Patient contacts the office regarding an appointment.
2. The Patient provides the Receptionist with his or her name and address.
3. If the Patient's information has changed
   Execute the Update Patient Information use case.
4. If the Patient's payment arrangements has changed
   Execute the Make Payments Arrangements use case.
5. The Receptionist asks Patient if he or she would like to make a new appointment, cancel an existing appointment, or change an existing appointment.
   If the patient wants to make a new appointment,
   the S-1: new appointment subflow is performed.
   If the patient wants to cancel an existing appointment,
   the S-2: cancel appointment subflow is performed.
   If the patient wants to change an existing appointment,
   the S-3: change appointment subflow is performed.
6. The Receptionist provides the results of the transaction to the Patient.

A Sample CRC card (Front)

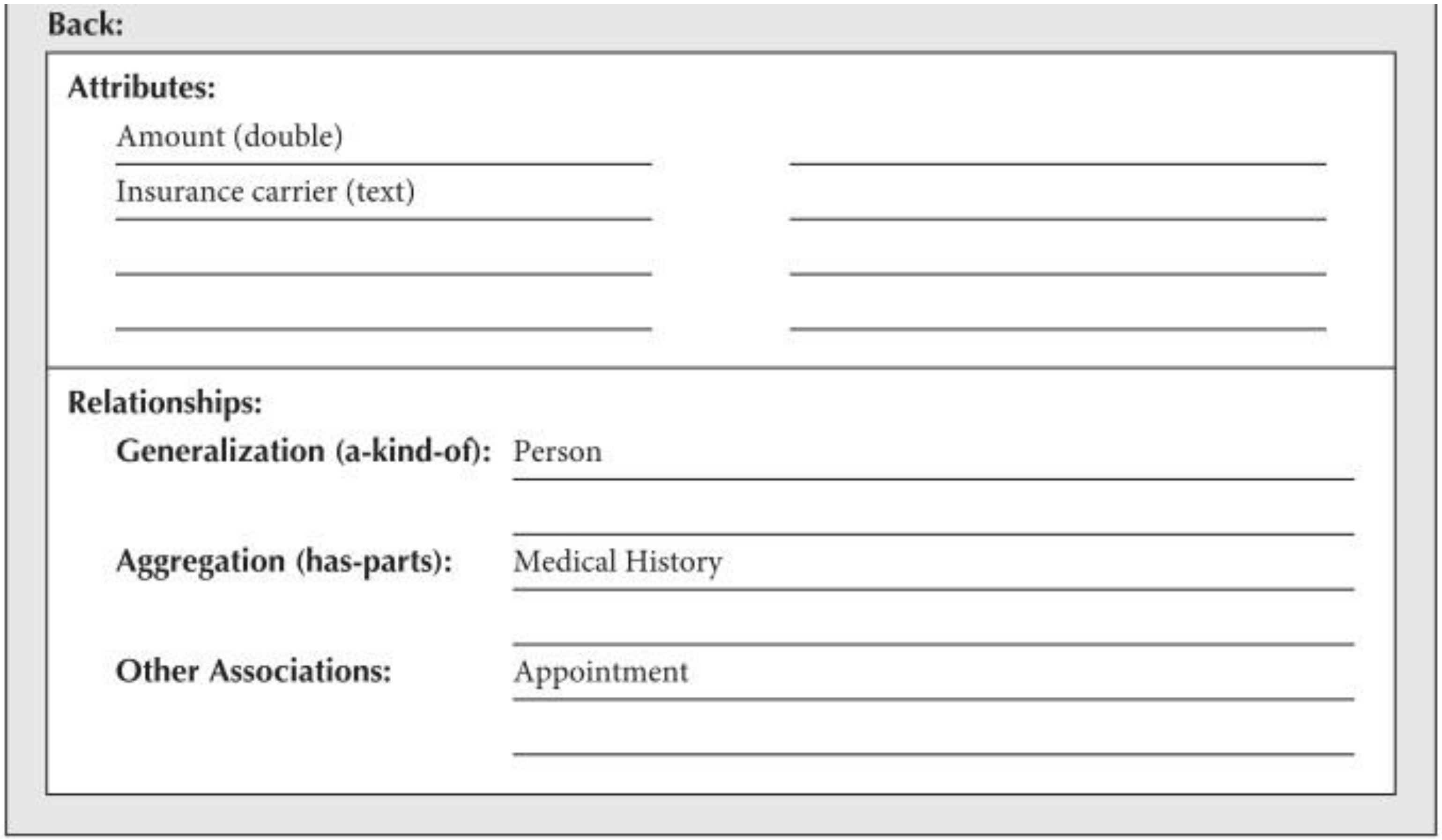


Front:

Class Name: Old Patient	**ID:** 3	**Type:** Concrete, Domain
Description: An individual who needs to receive or has received medical attention		**Associated Use Cases:** 2

Responsibilities	Collaborators
Make appointment	Appointment
Calculate last visit	
Change status	
Provide medical history	Medical history

A Sample CRC card (Back)



Back:

Attributes:

Amount (double)

Insurance carrier (text)

Relationships:

Generalization (a-kind-of):	Person
Aggregation (has-parts):	Medical History
Other Associations:	Appointment

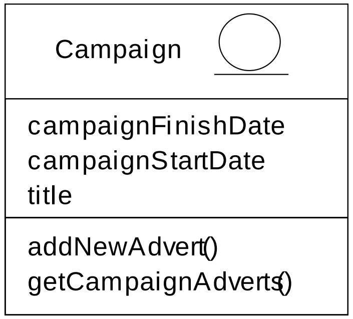
A use-case-driven process

Process to create the structural model of a problem domain:

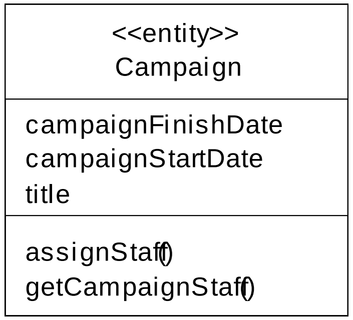
1. Create CRC Cards.
2. Review CRC Cards.
3. Role-Play the CRC Cards.
4. Create Class Diagram.
5. Review Class Diagram.
6. Incorporate Patterns.
7. Review the Model.

Assembling the Class Diagram

- 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.

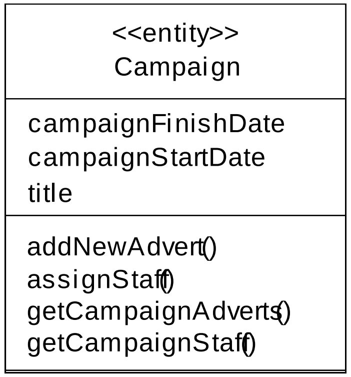


(a) Campaign class that meets the needs of `Add new advert to a campaign`

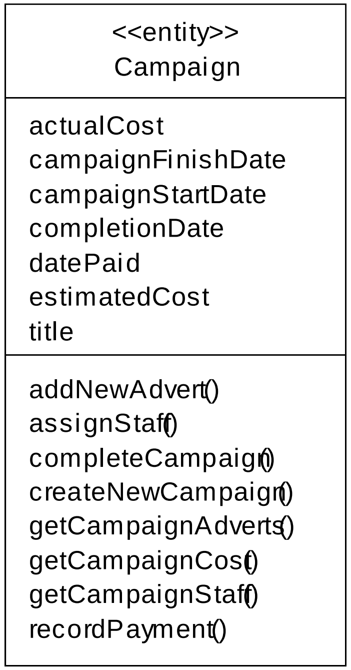


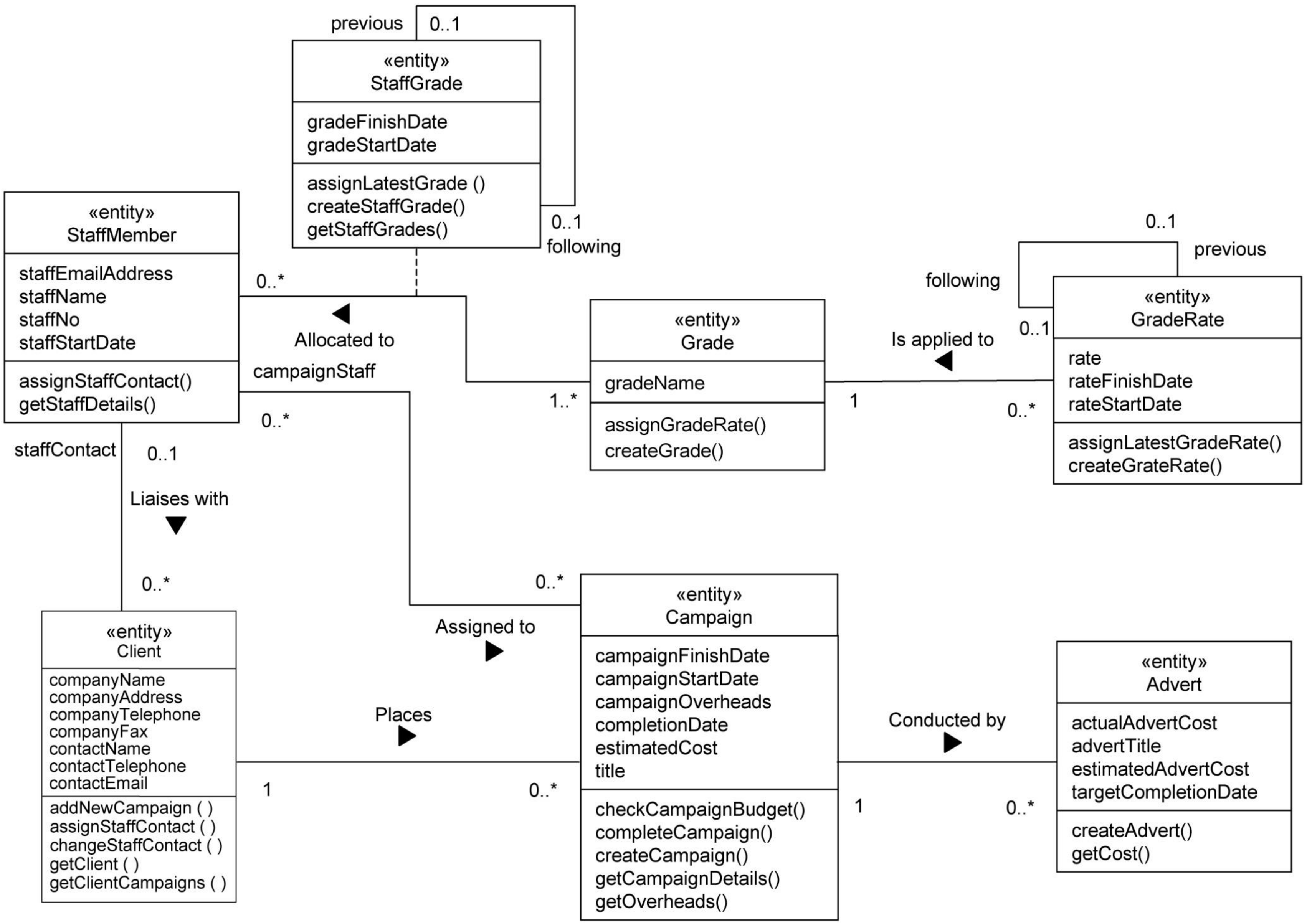
(b) Campaign class that meets the needs of `Assign staff to work on a campaign`

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



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




previous
0..1
«entity» StaffGrade
gradeFinishDate
gradeStartDate
assignLatestGrade ()
createStaffGrade()
getStaffGrades()
0..1
following
«entity» StaffMember
staffEmailAddress
staffName
staffNo
staffStartDate
assignStaffContact()
getStaffDetails()
0..*
Allocated to
campaignStaff
0..*
1..*
«entity» Grade
gradeName
assignGradeRate()
createGrade()
Is applied to
1
0..*
0..1
previous
following
0..1
«entity» GradeRate
rate
rateFinishDate
rateStartDate
assignLatestGradeRate()
createGrateRate()
staffContact
0..1
Liaises with
0..*
«entity» Client
companyName
companyAddress
companyTelephone
companyFax
contactName
contactTelephone
contactEmail
addNewCampaign ( )
assignStaffContact ( )
changeStaffContact ( )
getClient ( )
getClientCampaigns ( )
Assigned to
0..*
Places
1
0..*
«entity» Campaign
campaignFinishDate
campaignStartDate
campaignOverheads
completionDate
estimatedCost
title
checkCampaignBudget()
completeCampaign()
createCampaign()
getCampaignDetails()
getOverheads()
Conducted by
1
0..*
«entity» Advert
actualAdvertCost
advertTitle
estimatedAdvertCost
targetCompletionDate
createAdvert()
getCost()

Approach 2: 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.
- 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.

Robustness Analysis

- Aims to produce set of classes robust enough to meet 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).

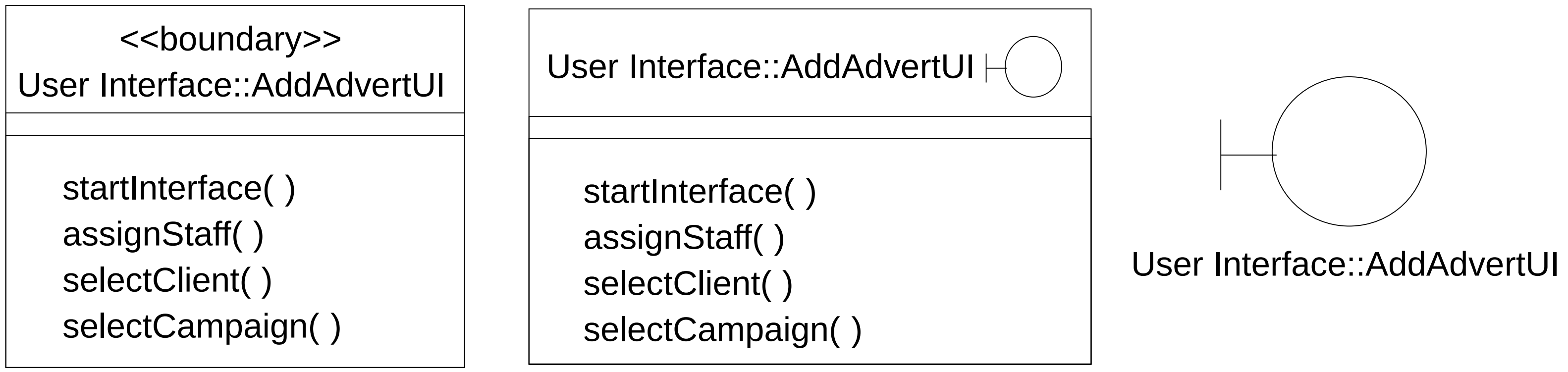
Robustness Analysis

- Robustness analysis involves:
  - Analysing the text of a use case.
  - Identifying a first-guess set of objects that will participate in the use case.
  - Classifying these objects based on their characteristics.
- Used this approach for preliminary design.
- 3 types of analysis classes: boundary, entity and control.

Boundary Class Stereotype

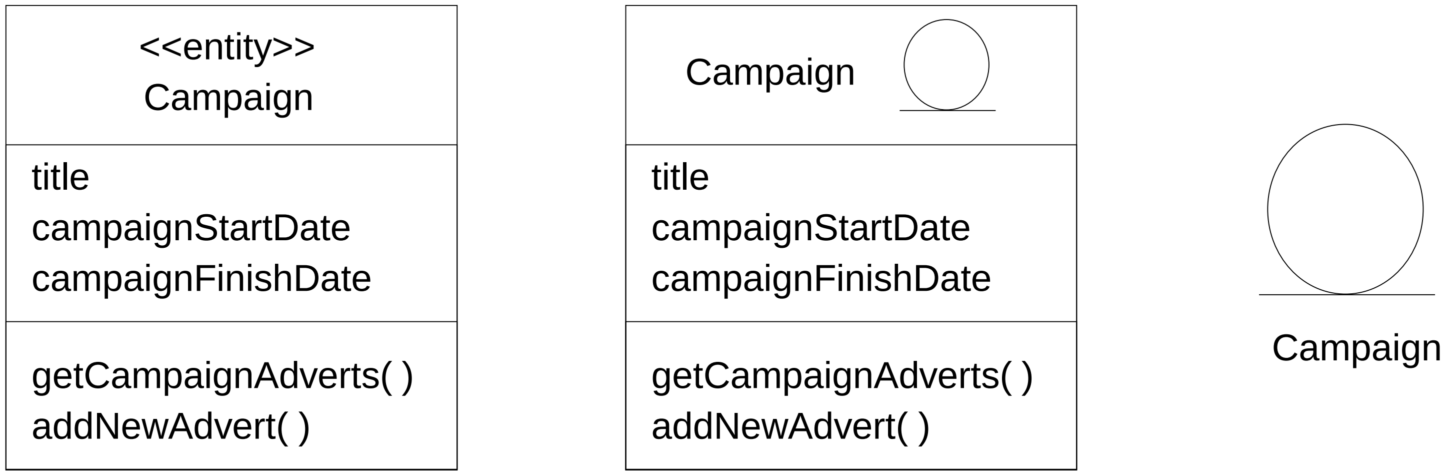
- Boundary classes represent interaction with the user - likely to be unique to the use case but inherited from a library.

- Alternative notations:



Entity Class Stereotype

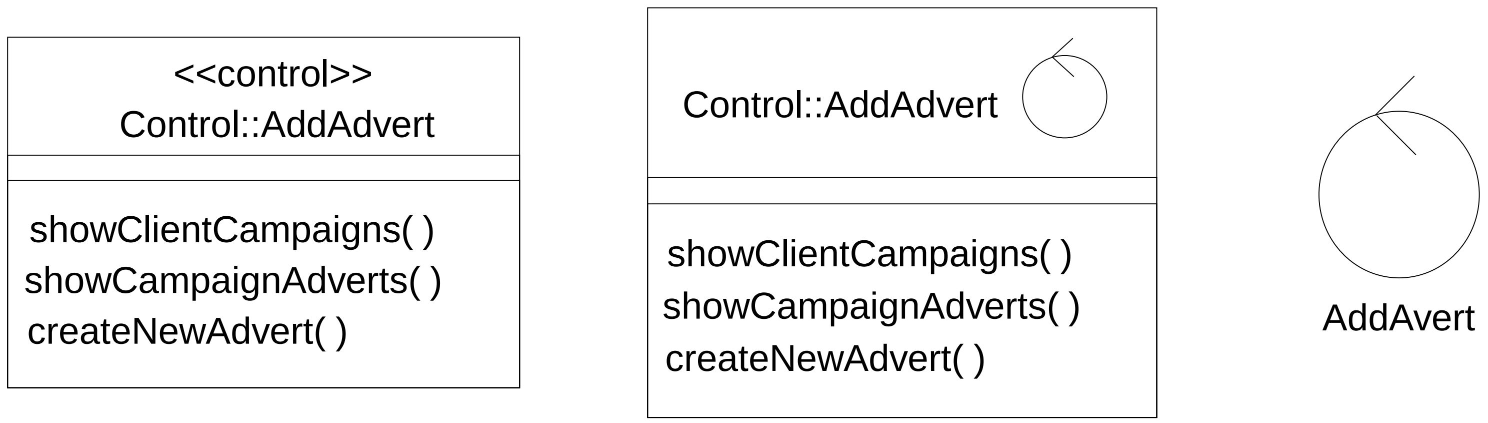
- Entity classes represent persistent data and common behaviour likely to be used in more than one application system.
- Alternative notations:



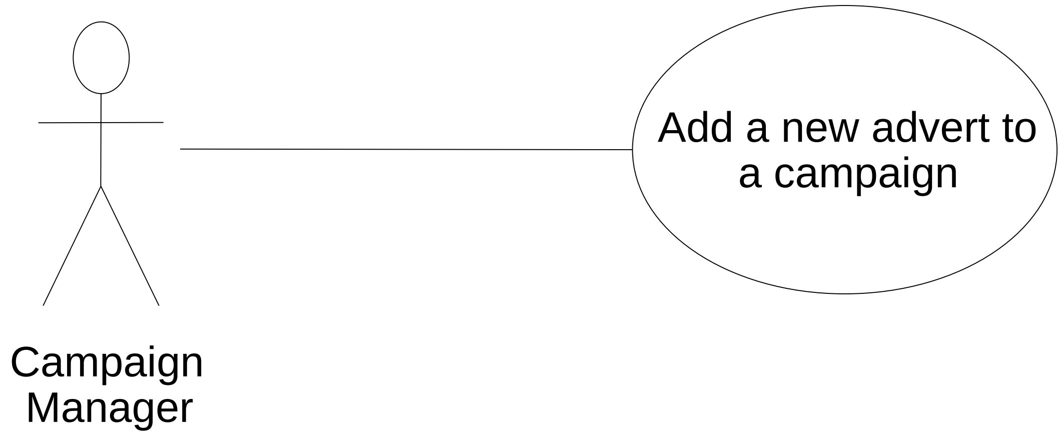
Control Class Stereotype

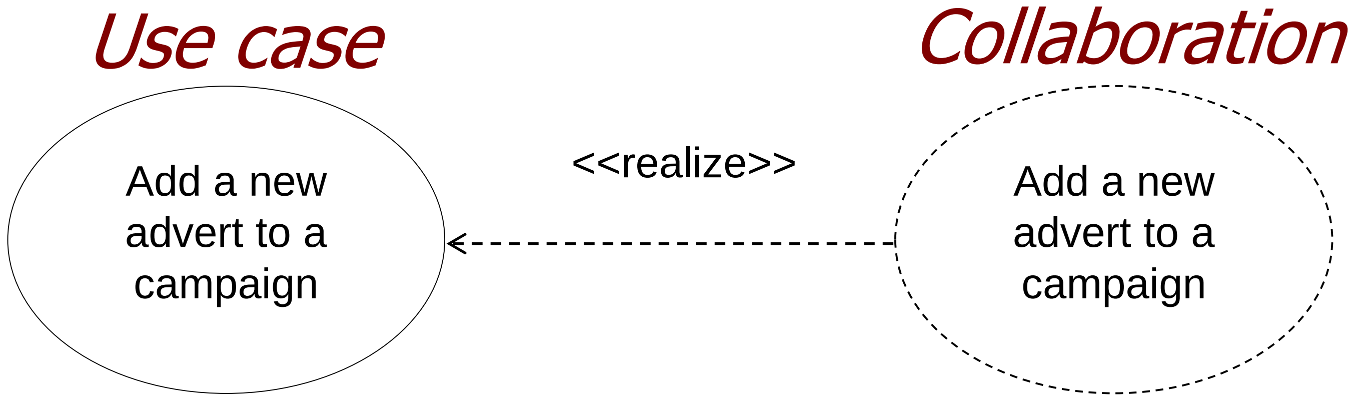
- Control classes encapsulate unique behaviour of a use case.
- Specific logic kept separate from the common behaviour of entity classes.

- Alternative notations:



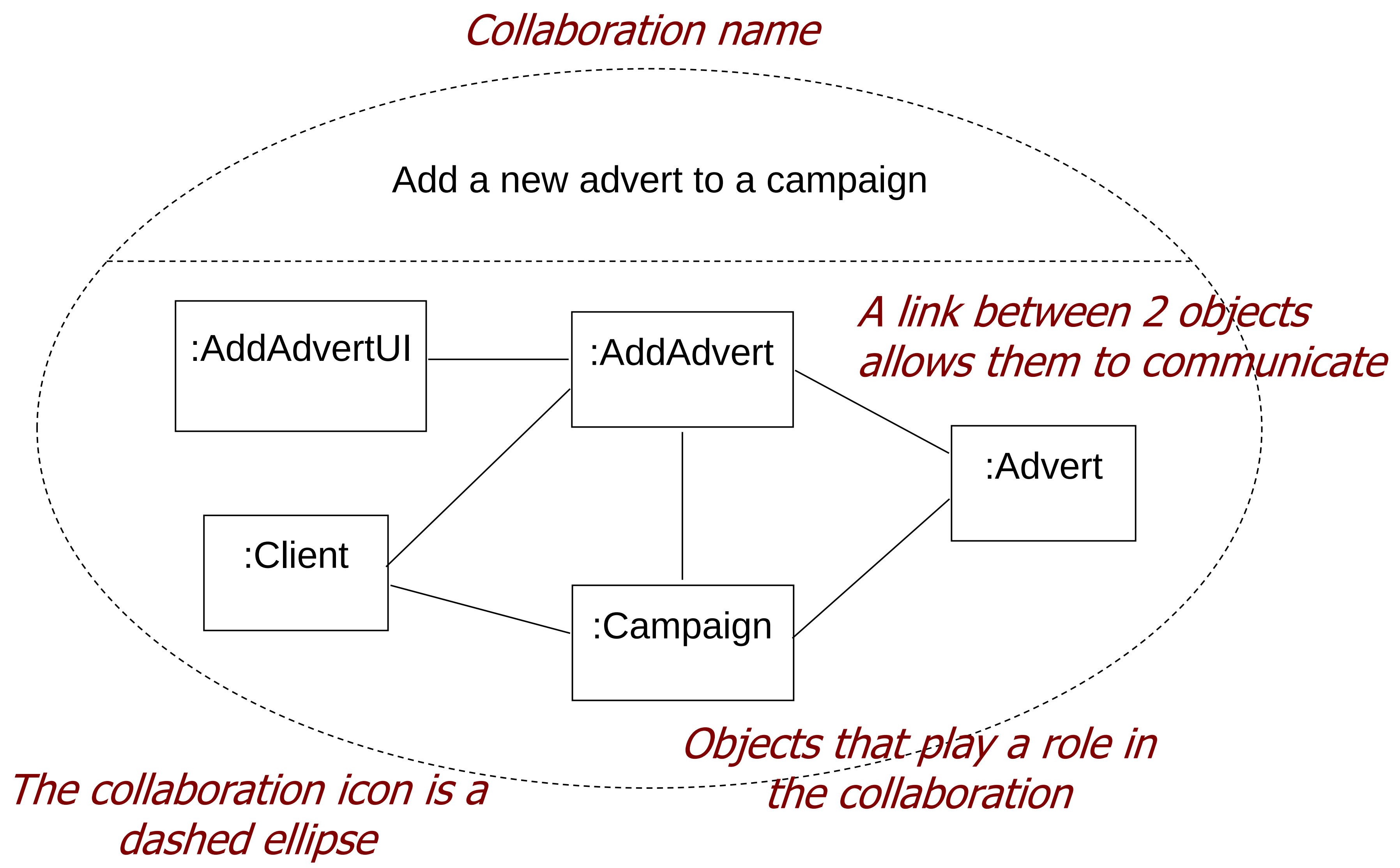
Use Case and Collaboration



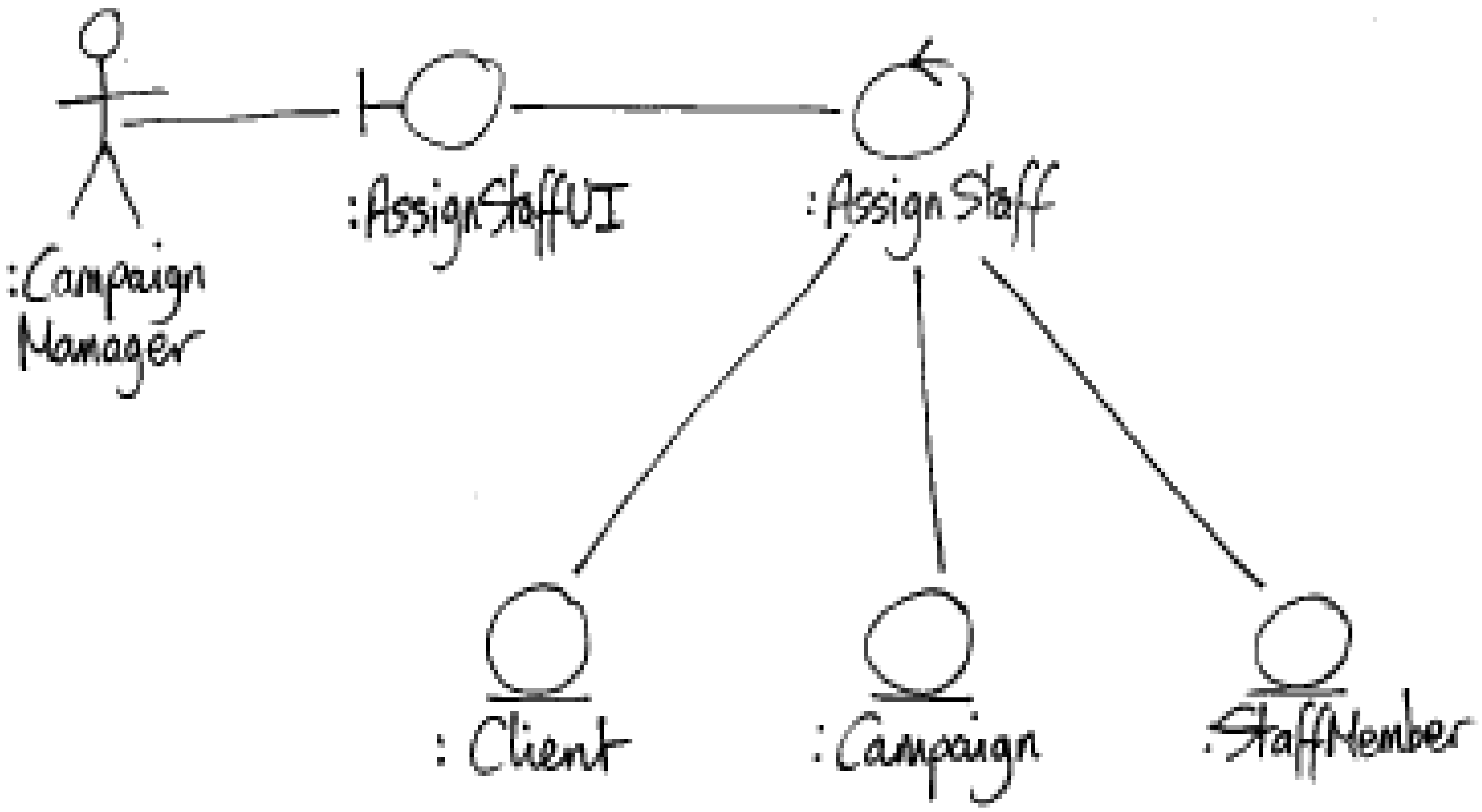

Use case
Collaboration
Add a new advert to a campaign
<<realize>>
Add a new advert to a campaign

This dependency arrow indicates that elements within the collaboration may reference elements within the use case

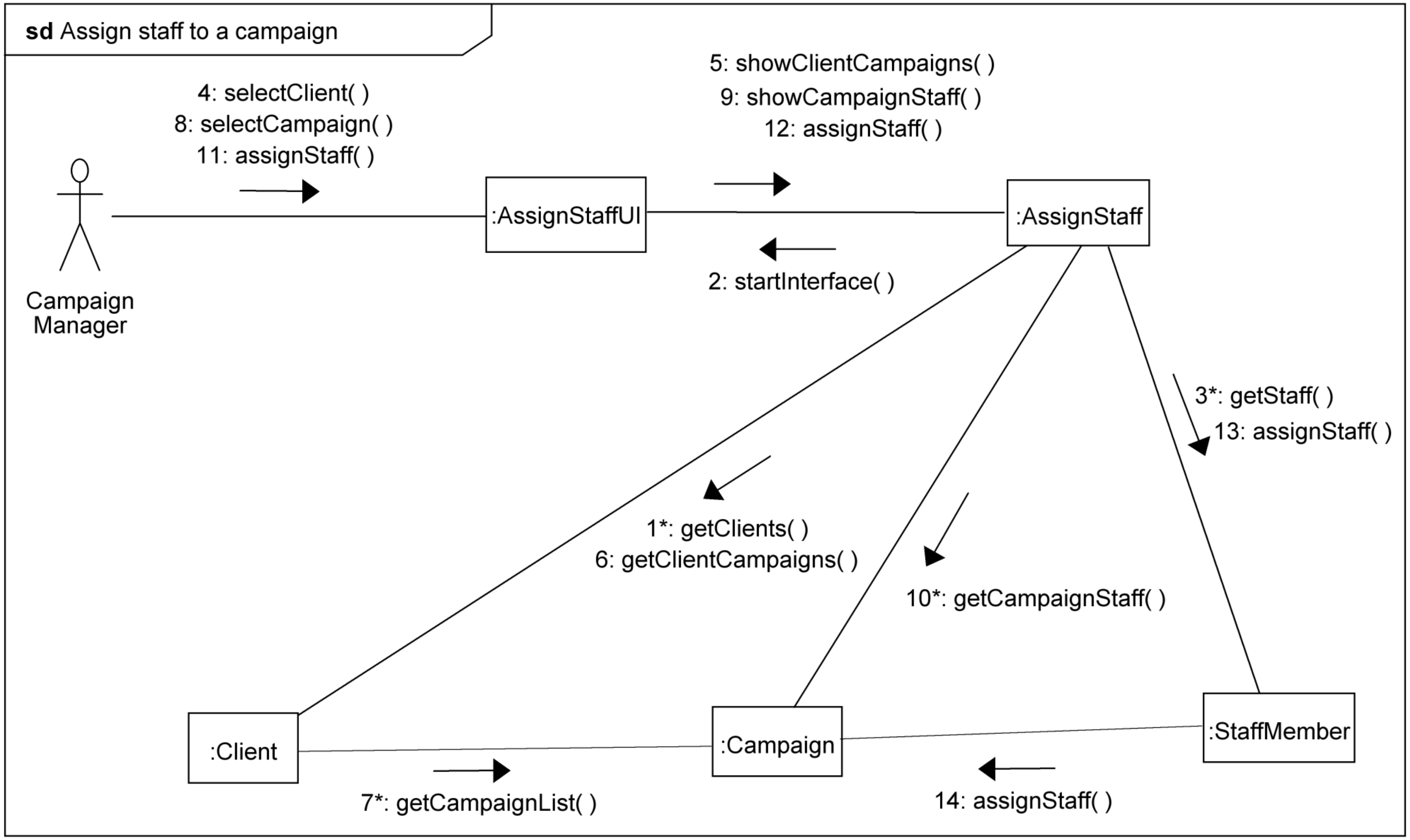
A Possible Collaboration


Collaboration name
Add a new advert to a campaign
:AddAdvertUI
:AddAdvert
:Advert
:Client
:Campaign
A link between 2 objects allows them to communicate
Objects that play a role in the collaboration
The collaboration icon is a dashed ellipse

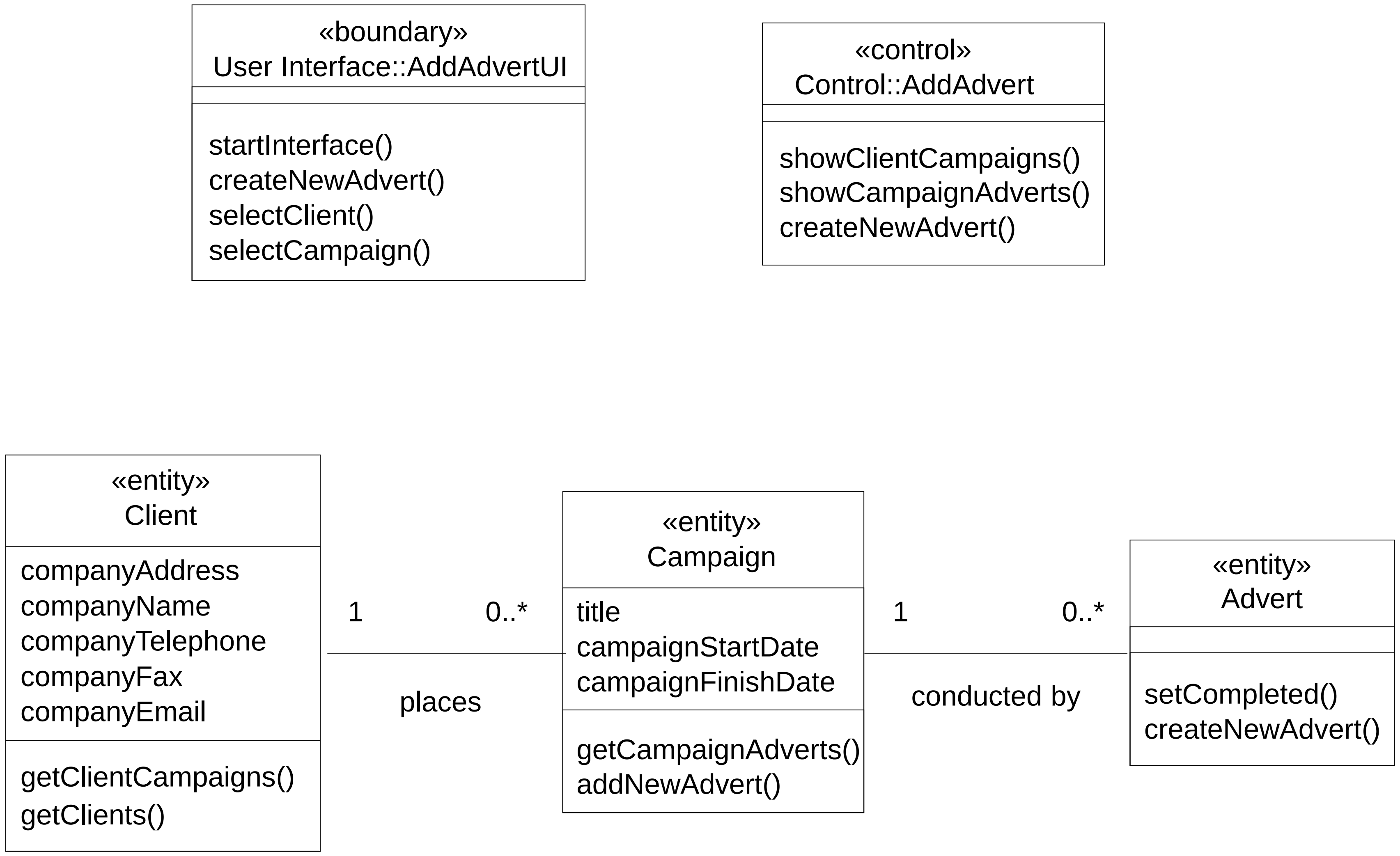
Early Draft Communication Diagram



More Developed Communication Diagram



Resulting Design 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?
  - 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?

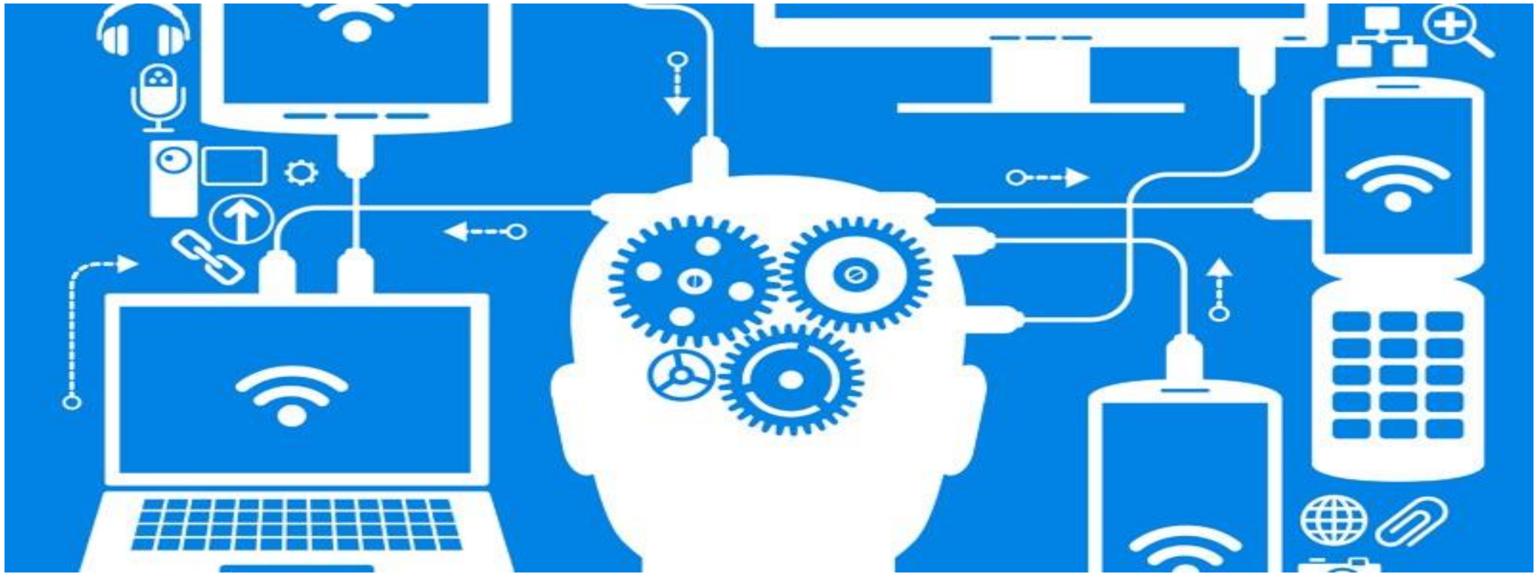
Key points

- An object diagram is essentially an instantiation of all or part of a class diagram.
- The ultimate product of realization is the software implementation of that use case.
- CRC (Class–Responsibility–Collaboration) cards are used to:
  - document the responsibilities and collaborations of a class.
  - model interaction between objects.
  - identify the classes, along with the attributes, operations, and relationships, involved with a use case.
- Robustness analysis aims to produce set of classes robust enough to meet requirements of a use case using communication diagrams

References

- Alan Dennis, Barbara Haley Wixom & David Tegarden. 2015. Systems Analysis and Design with UML, 5th edition, Wiley.
- Simon Bennett, Steve McRobb & Ray Farmer. 2010. Object Oriented Systems Analysis and Design using UML 4th Edition, McGraw-Hill.

In the next lecture..



Lecture 6: UML Activity Diagrams