Requirements Analysis 1: Requirements and Classes

Based on Chapter 7 of Bennett, McRobb and Farmer:

Object Oriented Systems Analysis and Design Using UML, (4th Edition), McGraw Hill, 2010.

In This Lecture You Will Learn:

- How an analysis model differs from requirements and design models
- What makes good analysis
- Concepts and notation of the class diagram:
  - Class and Object
  - Links, Associations and Multiplicities
  - Attributes, Operations and State

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

What an Analysis Model Does

An analysis model must also specify what designers must design:

- Unambiguous about scope and detail
- Consistent, e.g. about the names of classes, operations, etc. in different diagrams
- Complete, e.g. regarding non-functional requirements such as localization
- Correct, e.g. regarding the multiplicities of associations between classes

What an Analysis Model Does

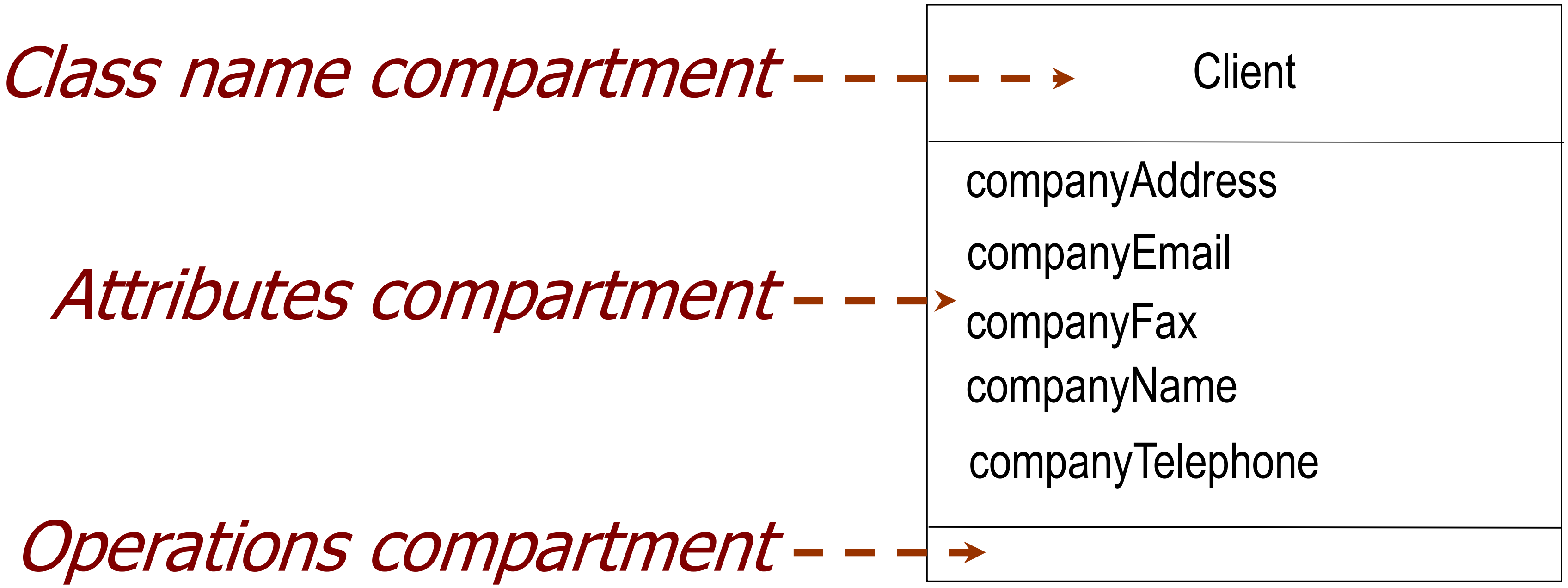
- Describes what the software should do
- Represents people, things and concepts important to understand the system
- Shows connections and interactions among these people, things and concepts
- Shows the business situation in enough detail to evaluate possible designs
- Is organized / structured so it can be useful for designing the software

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)

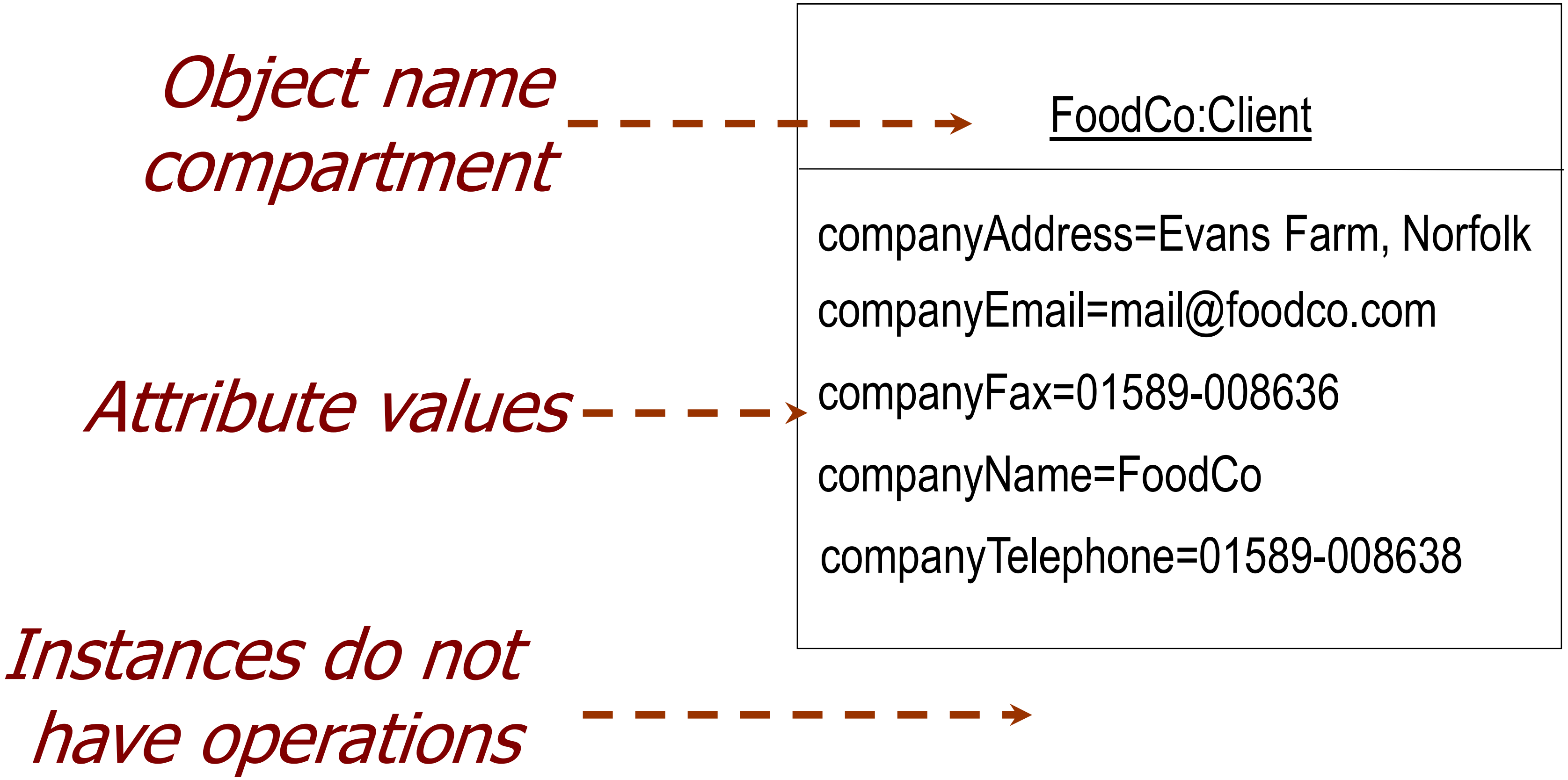
Class Diagram: Class Symbol

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



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:

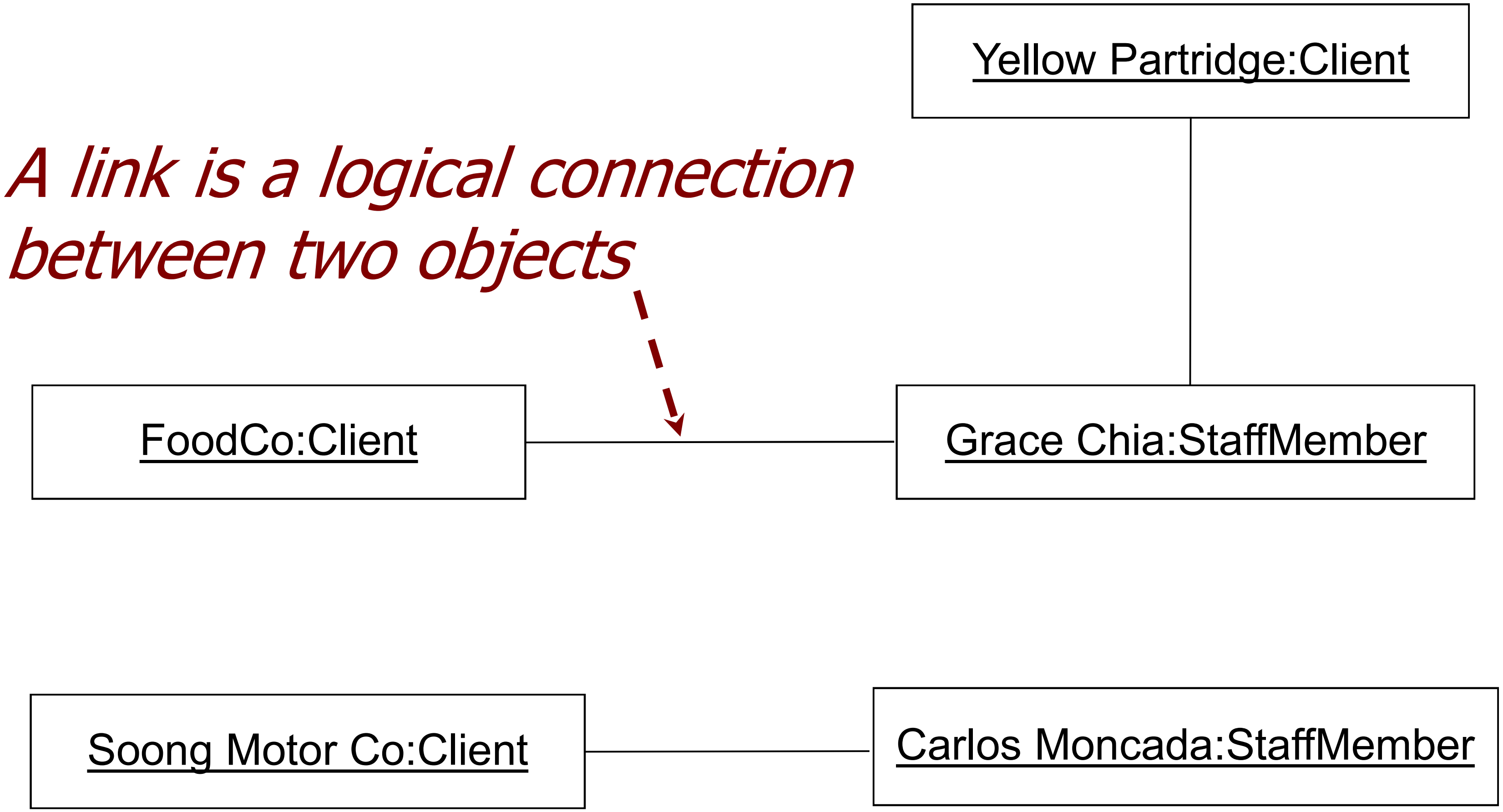
- Part of the essential description of a class
- The common structure of what the class can ‘know’
- Each object has its own *value* for each attribute in its class:
  - *Attribute=“value”*
  - companyName=FoodCo

Class Diagram: Associations

Associations represent:

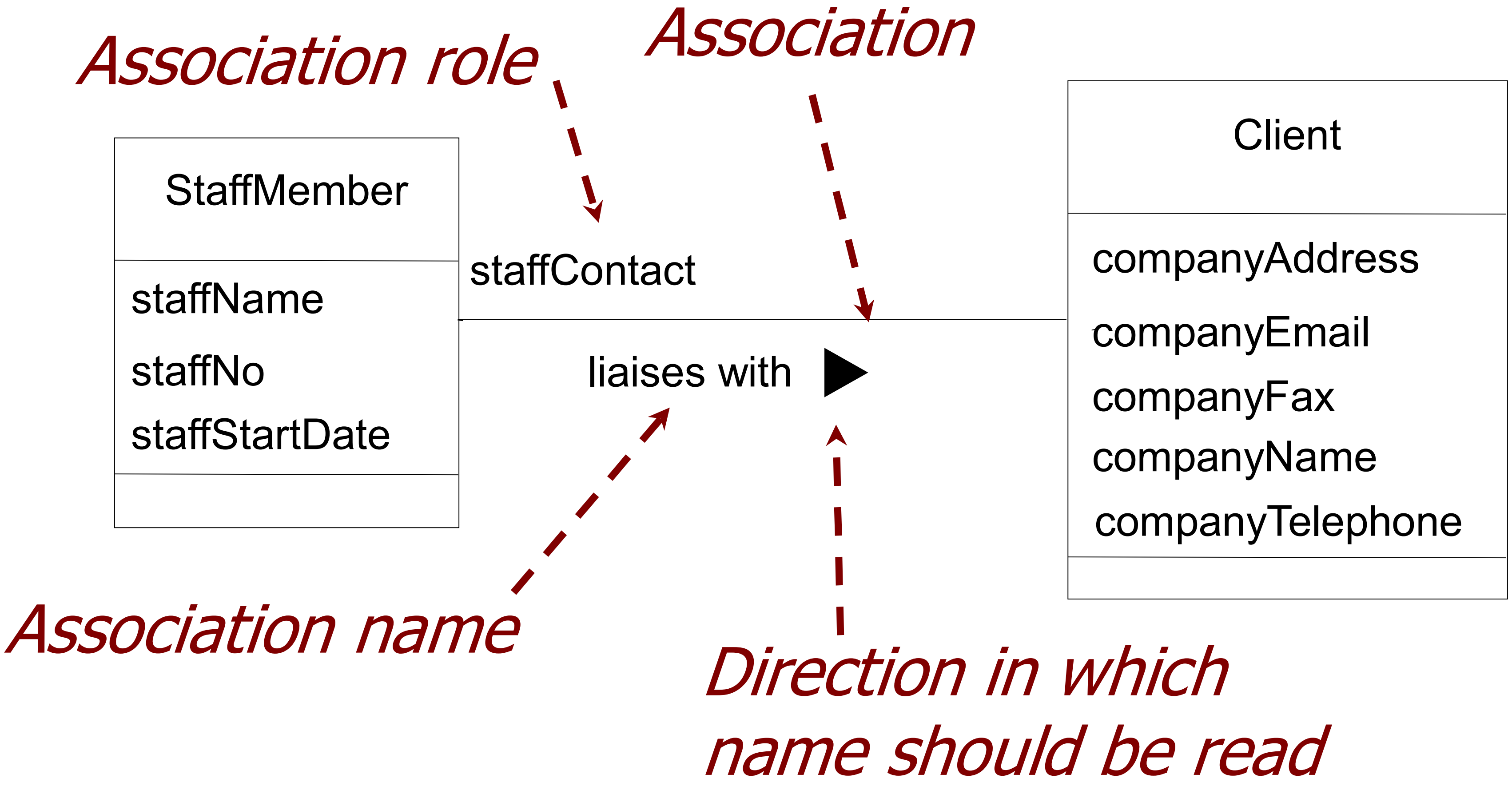
- The possibility of a logical relationship or connection between objects of one class and objects of another
  - “Grace Chia is the staff contact for FoodCo”
  - *An **employee** object is linked to a **client** object*
- If two objects are linked, their classes are said to have an association

Class Diagram: Links



A link is a logical connection between two objects

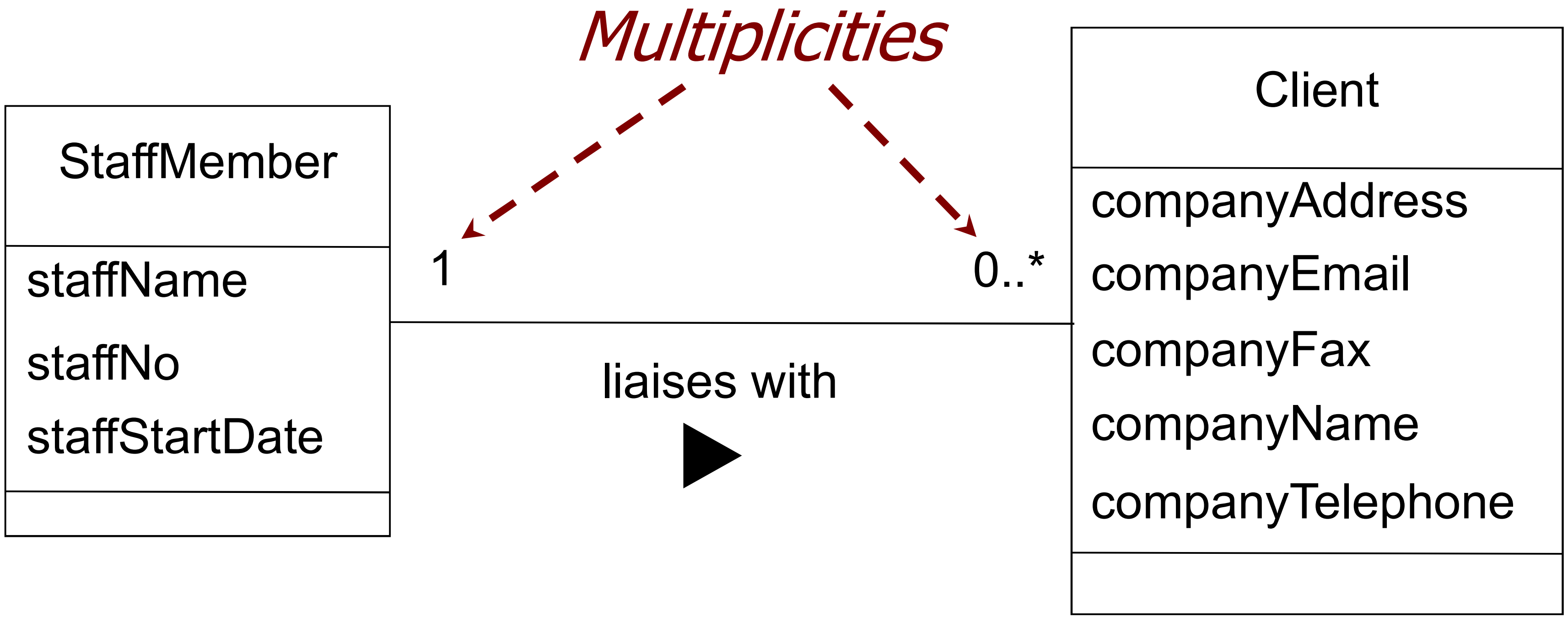
Class Diagram: Associations



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

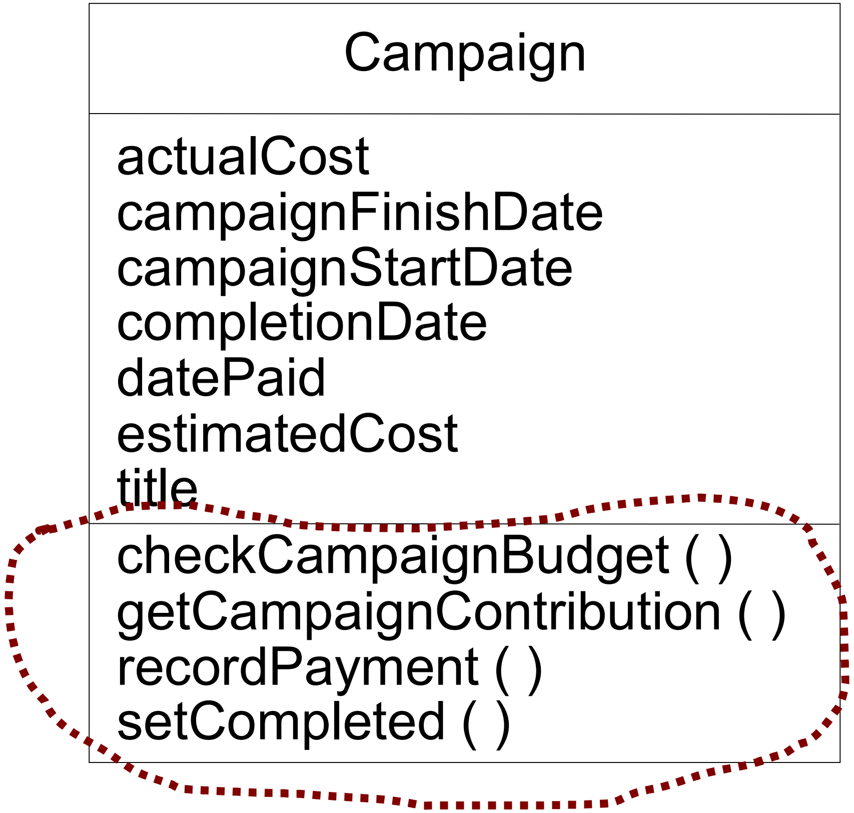
Class Diagram: Operations

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

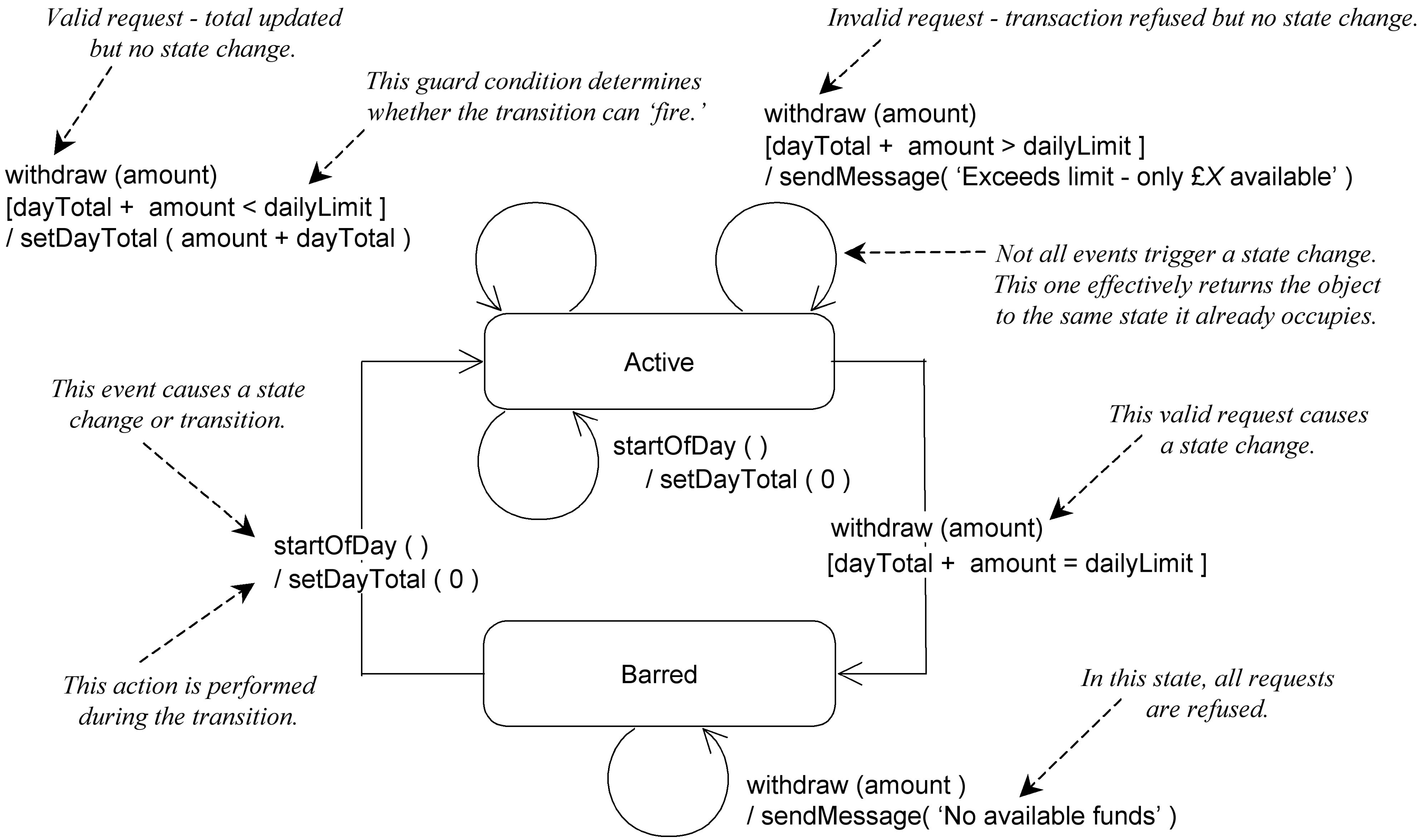
checkCampaignBudget ( )
getCampaignContribution ( )
recordPayment ( )
setCompleted ( )

Object State

- An object’s state is related to its attributes, its links and its operations
- Current state is an encapsulation of the value of attributes and links
- State constrains behaviour - it determines whether or not an operation can fire
- Executing an operation often causes a change of state

Object State

- In the ATM example on the following slide, an account object responds differently according to the current value of:
  - `dayTotal` (amount withdrawn so far today)
  - `dailyLimit` (total that can be withdrawn)
- Together, these define the object's state: `Active` or `Barred`
- The current state determines whether a `withdraw` operation can be successful



Valid request - total updated but no state change.

withdraw (amount)
[dayTotal + amount < dailyLimit]
/ setDayTotal (amount + dayTotal)

This guard condition determines whether the transition can 'fire.'

Invalid request - transaction refused but no state change.

withdraw (amount)
[dayTotal + amount > dailyLimit]
/ sendMessage('Exceeds limit - only £*X* available')

Not all events trigger a state change. This one effectively returns the object to the same state it already occupies.

Active

startOfDay ( )
/ setDayTotal (0)

This event causes a state change or transition.

startOfDay ( )
/ setDayTotal (0)

This action is performed during the transition.

This valid request causes a state change.

withdraw (amount)
[dayTotal + amount = dailyLimit]

Barred

In this state, all requests are refused.

withdraw (amount)
/ sendMessage('No available funds')

Summary

In this lecture you have learned:

- Why we analyse requirements
- Concepts represented in class diagrams
- Notation for class diagrams:
  - Classes and objects
  - Attributes and operations
  - Links and associations
- How these relate to object state

References

(For full bibliographic details, see Bennett, McRobb and Farmer)