ECE450 – Software Engineering II

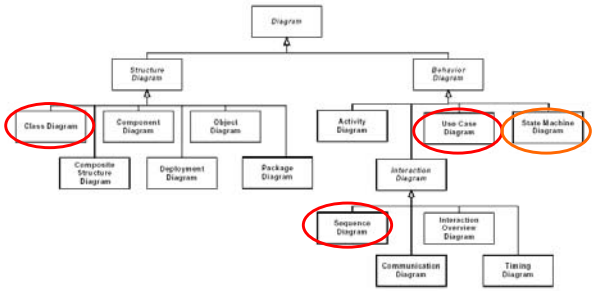
Today: **An Aside:**
The Quickest Tour through
the UML that you will ever get

Warning: Europe in 5 days

What is UML and why should I care?

- The Unified Modeling Language is an industry standard for specifying and visualizing the artifacts of software systems
  - A collection of diagrammatic languages to express everything from class structures to execution scenarios
  - A joint effort by object-oriented modeling researchers to merge their different approaches
    - James Rumbaugh, Grady Booch, Ivar Jacobson
    - UML 1.0 came out in 1997
    - Current version, UML 2.0
    - http://www.uml.org/
- If there is *one* modeling language that you need to know to get a job, this is it
  - Although frankly you may not need to *use it* once you get that job
  - If "Model-Driven Development" takes off, you *will* need this
- Easy to learn the basics, very hard to master it
  - Especially the newest version
  - For now all you need are those easy-to-learn basics

The many diagrams of UML



Class Diagrams

- Class diagrams define the structure of the classes in a system, the relationship between all classes, and the components of each class.

Class

A class is a general concept (represented as a square box). A class defines the structural attributes and behavioural characteristics of that concept. Shown as a rectangle labeled with the class name.

Association

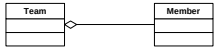
A (semantic) relationship between classes. A line that joins two classes.

Class Diagrams (cont)

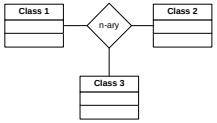
- Types of associations

Binary

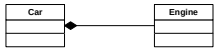
Aggregation (has-a)



n-ary



Composition (is-composed-of)

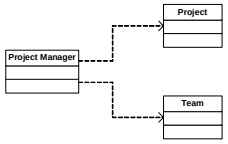


Generalization (is-a-kind-of)

Class Diagrams (cont)

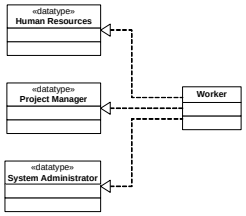
- Types of associations (cont)

Dependency



The source class depends on (uses) the target class

Realization



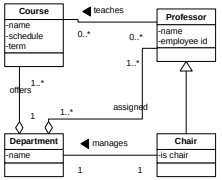
Class supports all operations of target class but not all attributes or associations.

Class Diagrams (cont)

- Attributes and operations

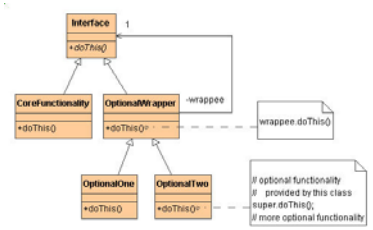
Attributes are what is known about each object of this class type. Operations are what objects of this class type do.

- Multiplicity
  - *n*, where *n* = {0, 1, *x*, *}
  - *m*..*n*, where *m*,*n* = {0, 1, *x*, *}



Class Diagrams (cont)

- Design patterns are usually expressed through their class diagrams. E.g., decorator:

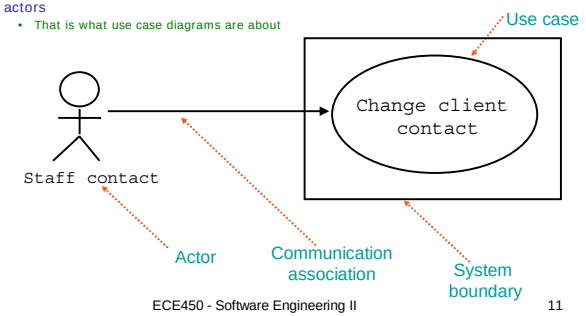


Use Case Diagrams

- Just what is a "use case"?
  - The answer to the question "What functions will the new system provide?"
    - How will people interact with it?
    - Describe the system in terms of its users and its boundary
- Normally, a use case shows:
  - A **function** that the system will provide
  - The **actors** that are involved in that function
  - A **sequence** of related actions performed by an actor and the system via a dialogue
    - The sequence usually explains the "common use" scenario, and covers some of the exceptional cases briefly
- What is an actor?
  - Anything that needs to interact with the system
    - A person
    - A role that different people may play
    - An external system

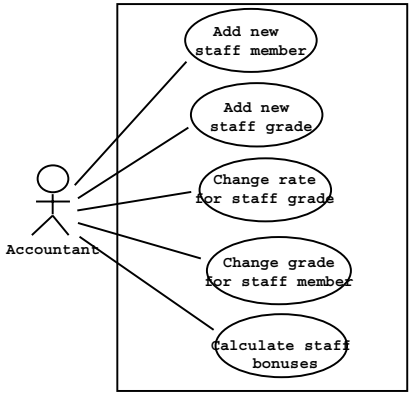
Use Case Diagrams (cont)

- A use case is not diagrammatic!
  - We normally describe use cases textually
  - But we may have diagrams that summarize the interactions between system and actors
    - That is what use case diagrams are about



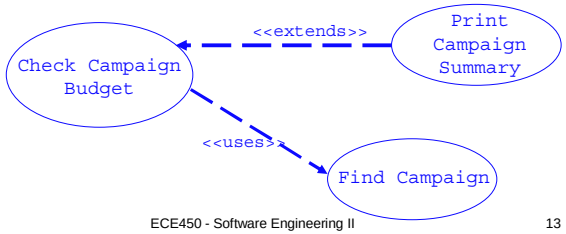
Use Case Diagrams (cont)

- An example



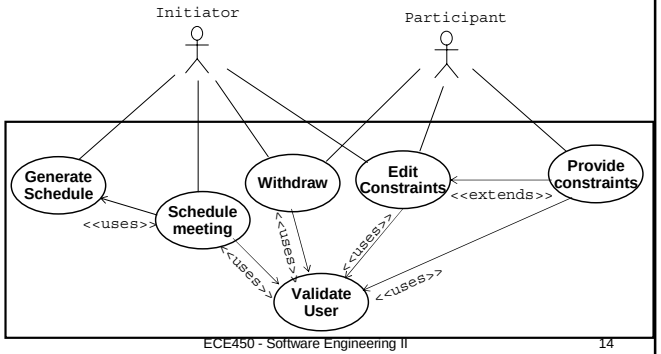
Use Case Diagrams (cont)

- <<extends>> and <<uses>>
  - <<extends>> when one case adds behaviour to a base case
    - Used to model a part of a use case that the user may see as optional system behaviour
    - Also models a separate sub-case which is executed conditionally
  - <<uses>>: one use case invokes another (like a procedure call)
    - Used to avoid describing the same flow of events several times
    - Puts the common behaviour in a use case of its own



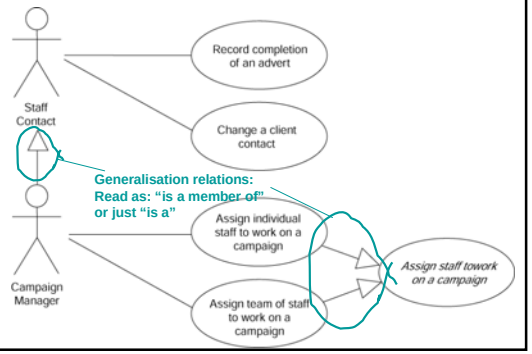
Use Case Diagrams (cont)

- Meeting scheduler example



Use Case Diagrams (cont)

- Generalizations
  - **Actor classes:** Actors inherit use cases from the class
  - **Use case classes:** Generalizations of several use cases

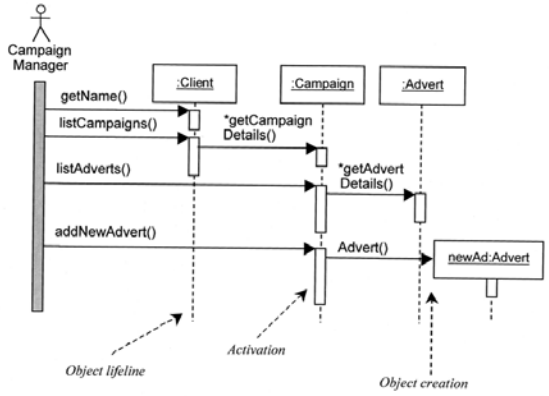


Sequence Diagrams

- Sequence diagrams provide a more detailed look of the sequence of steps executed in a use case
  - Normally used for lower-level design
  - If you wanted to specify all of your application's scenarios with sequence diagrams, you would need one for each of its features' ramifications
    - So we are usually interested in key scenarios only
- Sequence diagrams show:
  - The actors and software classes/objects that intervene in the scenario
  - The step-by-step interactions between them
    - Chronologically, from top to bottom
  - Details regarding when objects are created and activated

Sequence Diagrams (cont)

- Example



Sequence Diagrams (cont)

- This is not the full story
  - We can illustrate branching, *guards* (conditions necessary for the execution of a call), asynchronous messaging, and more
  - In UML 2.0, sequence diagrams went through a major overhaul
    - Conditionals, loops, etc.
- We don't need the full story for this course
  - These basics are enough
  - But if you want to invest time in learning more about UML, sequence diagrams are the place to start
    - Along with class diagrams, they are the most frequently used kind of model

What about the others?

- Every kind of diagram has a (sometimes slightly) different purpose
  - There is probably one that matches what you are trying to express
- On the other hand, you may rightfully accuse UML of bloating
  - Design by committee
  - Trying to be all things for all people
  - Attempts at formalizing semantics vs. attempts to maintain comprehensibility
- My advice:
  - Invest some time learning the basic diagrams
  - Try it out for a small application of your own
    - You'll learn to see when it is useful and when it is overhead
  - Do not impose it on your team
    - Use of UML should be agreed by all members