Refining the Requirements Model

In this Lecture you will Learn:

- What is meant by a *component*
- How *generalization* and *aggregation* help to develop reusable components
- How to identify generalization and *composition*
- How to model generalization and composition
- What is meant by the term *pattern*
- What types of patterns can be used in software development

Component-based Development

- Component-based development means either:
  - Assembling software from pre-existing components, or
  - Building components for others to use

Class Exercise

- What are the advantages of components?

Problems of Software Components

- Why are components hard?
  - The NIH (Not-Invented-Here) Syndrome
  - Model Organization

Class Exercise

- Explain why the NIH (Not-Invented-Here) syndrome occurs.

Component-based Development

- The contribution of object-orientation:
  - Encapsulation of internal details makes it easier to use components in systems for which they were not designed
  - Generalization hierarchies make it easier to create new specialized classes when they are needed
  - Composition and aggregation structures can be used to encapsulate components

Class Exercise

- What does object-orientation offer that helps to create reusable components?

Class Exercise

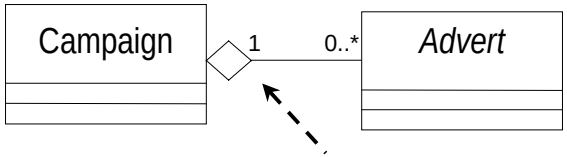
- Why is generalization important to creating reusable components?

Class Exercise

- Why is encapsulation important to creating reusable components?

Composition and Aggregation

- Special types of association, both sometimes called whole–part
- A campaign is made up of adverts:



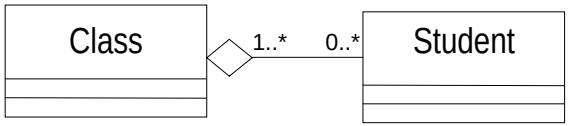
Unfilled diamond signifies aggregation

Composition and Aggregation

- Aggregation is essentially any whole–part relationship
- Semantics can be very imprecise
- Composition is ‘stronger’:
  - Each part may belong to only one whole at a time
  - When the whole is destroyed, so are all its parts

Composition and Aggregation

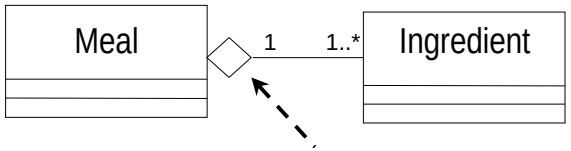
- An everyday example



- Clearly not composition
  - Students could be in several classes
  - If class is cancelled, students are not destroyed!

Composition and Aggregation

- Another everyday example

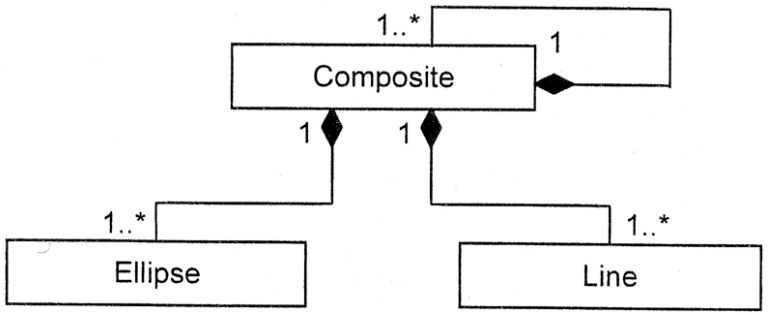


Filled diamond signifies composition

- This is (probably) composition
  - Ingredient is in only one meal at a time
  - If you drop your dinner on the floor, you probably lose the ingredients too

Example

- Composition used in class diagram to represent composite objects



Class Exercise

- Distinguish composition from aggregation.

Adding Structure

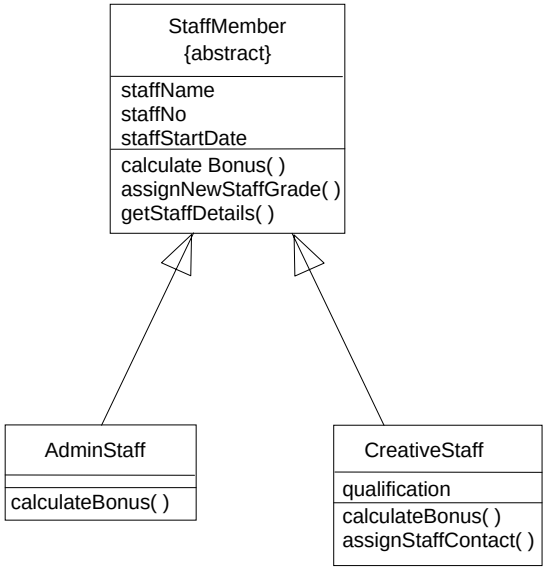
- Add generalization structures when
  - Two classes are similar in most details, but differ in some respects
  - May differ
    - In behaviour (operations or methods)
    - In data (attributes)
    - In associations with other classes

Example: Adding Structure

- Two types of staff:

Creative	Have qualifications recorded Can be client contact for campaign Bonus based on campaigns they have worked on
Admin	Qualifications are not recorded Not associated with campaigns Bonus not based on campaign profits

Example: Adding Structure



Software Development Patterns

- A Pattern is
  "describes a problem which occurs over and over again in our environment, and then describes the core of a solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice."

  Alexander et al. (1977)

Software Development Patterns

- According to one influential definition (Gabriel, 1996), a pattern comprises three elements:
  - A context in which a given problem occurs repeatedly
  - A set of forces that influence or constrain the possible solutions
  - A software configuration that allows these forces to be resolved.

Software Development Patterns

- Patterns are found at many points in the systems development lifecycle
  - **Analysis Patterns** are groups of concepts useful in modelling requirements
  - **Architectural Patterns** describe the structure of major components of a software system
  - **Design Patterns** describe the structure and interaction of smaller software components

Software Development Patterns

- Patterns have been applied widely in software development
  - **Organization Patterns** describe structures, roles and interactions in the software development organization itself

Antipattern

- An antipattern captures practice that is demonstrably bad (by documenting an attempted solution that failed), and can also provide a reworked solution that can be applied within a specific context.
- Mushroom Management is an organization antipattern

Class Exercise

- How do patterns help the software developer?