CS 494
Object-Oriented Analysis & Design

UML Class Models

Overview

- How class models are used? Perspectives
- Classes: attributes and operations
- Associations
  - Multiplicity
- Generalization and Inheritance
- Aggregation and composition

- Later: How to find classes
  - small and larger systems

Developing Class Models

- Class diagrams developed iteratively
  - Details added over time during lifecycle
  - Initially: missing names, multiplicities, other details
- Some define particular perspectives for class models:
  - Conceptual
  - Specification
  - Implementation
- Conceptual perspective
  - Represents concepts in the domain
  - Drawn with no regard for implementation (language independent)
  - Used in requirements analysis

Class Model Perspectives (cont'd)

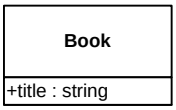
- Specification
  - Interfaces defined: a set of operations
  - But, each implementation class can include more than one interface
  - A given interface can be shared by more than one class
  - Sometimes known as a "type"
- Implementation
  - Direct code implementation of each class in the diagram
  - A blue-print for coding

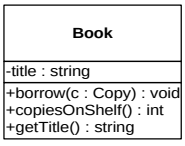
Documenting Your Objects

- Need some kind of record of your definitions
  - Your white-board?
  - A simple glossary
  - A *data dictionary* (perhaps in a CASE tool)
- What to define?
  - Attributes, operations for each class
  - Also relationships between classes
- Can you define classes of related objects?
  - Inheritance, Java interfaces

Classes in UML Diagrams

- Attributes in middle
- Operations at bottom
  - Can be suppressed. (What level of abstraction?)
- Attribute syntax:
  name : type = default
- Operation syntax:
  name (params) : return type
- Visibility
  - \+ public
  - \- private
  - # protected etc.
    nothing? Java's default-package?





Associations

- For "real-world objects" is there an association between classes?
- Classes A and B are associated if:
  - An object of class A sends a message to an object of B
  - An object of class A creates an instance of class B
  - An object of class A has an attribute of type B or collections of objects of type B
  - An object of class A receives a message with an argument that is an instance of B (maybe…)
    - Will it "use" that argument?
- Does an object of class A need to know about some object of class B?

More on Associations

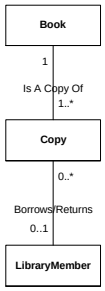
- Associations should model the reality of the domain and allow implementation
- Associations are between classes
  - A link connects two specific objects
  - Links are instances of associations
  - Note we could draw an object diagram to show objects and links
    - But often interaction diagrams are more useful for modeling objects
- Note: In practice, early in modeling, we may not name associations
- Note: One may choose to have a dynamic view associations: if at run-time two objects exchange messages, their classes must be associated

Multiplicity

- Also known as cardinality
- Objects from two classes are linked, but how many?
  - An exact number: indicated by the number
  - A range: two dots between a pair of numbers
  - An arbitrary number: indicated by * symbol
  - (Rare) A comma-separated list of ranges
- Examples:
  1 1..2 0..* 1..* * (same as 0..* but…)
- Important: If class A has association X with class B
  - The number of B's for each A is written next to class B
  - Or, follow the association past the name and then read the multiplicity
- Implementing associations depends on multiplicity

Examples of Associations

- From a Library catalog example
- One book has 1 or more copies
- One copy is linked to exactly one book
- Should there be two associations: borrows and returns?
- One copy is borrowed by either zero or one LibraryMember



Generalization and Inheritance

- You may model "inheritance" early but not implement it
  - Generalization represents a relationship at the conceptual level
  - Inheritance is an implementation technique
- Generalization is just an association between classes
  - But so common we put a "triangle" at the superclass
- Note this is a relationship between classes
  - So no multiplicities are marked. Why not?
- Inheritance may not be appropriate when it's time to implement
  - Objects should never change from one subclass to another
  - *Composition* can be used instead

Aggregation and Composition

- Again, just a specific kind of association between classes
  - An object of class A is part of an object of class B
  - A part-whole relationship
- Put a diamond on the end of the line next to the "whole"
  - Aggregation (hollow diamond): really no semantics about what this means!
  - Composition (solid diamond): a stronger relationship

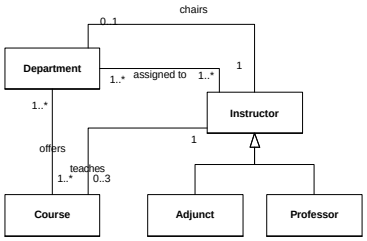
Aggregation and Composition (cont'd)

- Composition
  - The whole strongly owns the parts
  - Parts are copied (deleted, etc.) if the whole is copied (deleted, etc.)
  - A part cannot be part of more than one whole
  - Mnemonic: the stronger relationship is indicated by the stronger symbol (it's solid)
- Aggregation and composition associations are not named
- They do have multiplicities
- They can be used too often. If in doubt, use a "plain", named association.

Example 1: University Courses

- Some instructors are professors, while others have job title adjunct
- Departments offer many courses, but a course may be offered by >1 department
- Courses are taught by instructors, who may teach up to three courses
- Instructors are assigned to one (or more) departments
- One instructor also serves a department chair

Class Diagram for Univ. Courses

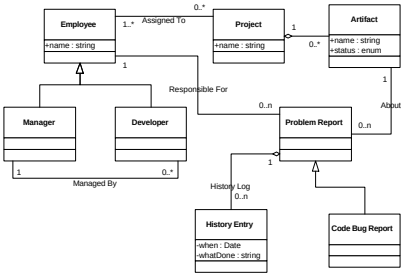


- Note this implies adjuncts can be chairs

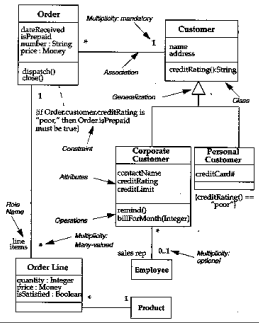
Example 2: Problem Report Tool

- A CASE tool for storing and tracking problem reports
  - Each report contains a problem description and a status
  - Each problem can be assigned to someone
  - Problem reports are made on one of the "artifacts" of a project
  - Employees are assigned to a project
  - A manager may add new artifacts and assign problem reports to team members

Class Diagram for Prob. Rep. Tool

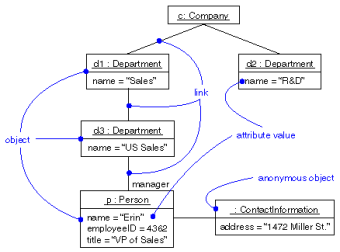


Example from Fowler



Objects, Object Diagrams

- Objects drawn like classes, but names for all instances underlined
- Objects may be "anonymous"
- Attributes are given values



Class Attributes, Operations

- Recall in Java and C++ you may have *class* attributes and *class* operations
  - keyword *static* used
  - One attribute for all members of class
  - An operation not encapsulated in each object, but "defined in" that class' scope
- In UML class diagrams, list these in the class box's compartments, but underline them

Navigability

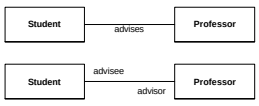
- Some call this "direction of visibility"
- Does each class really store a reference to each other?
- Do we need to decide this now? (When is "now"?)
- We can add arrows to associations to indicate this
  - What does a line with no arrows mean?

More on Associations: Navigability

- One reason for having an association between classes:
  Messages between objects of those classes
- But, often "knowledge" indicated by association is only in one direction
  - Example: In a computer system, a User needs access to his/her Password
  - From a Password object we should not be able to get back to a User!
- Note: Often ignored until design!

More on Associations: Roles

- Review:
  - Associations have an optional name
  - Name might have a "direction" indicator
- But, direction or semantics often easier to understand if we simply but a *role name* at one or both ends of the line



Dependencies

- Dependency: A <u>using</u> relationship between two classes
  - A change in the specification of one class may affect the other
  - But not necessarily the reverse
- Booch says: use dependencies not associations when one class uses another class as an argument in an operation.
- Often used for other things in UML: A general relationship between "things" in UML
  - Often use a stereotype to give more info
- Uses: binding C++ class to template; Java interfaces; a class only instantiates objects (a factory)

Stereotypes

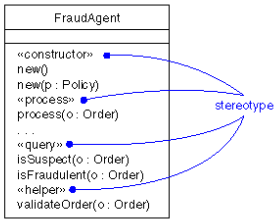
- Extends the "vocabulary" of UML
- Creates a new kind of building block
  - Derived from existing UML feature
  - But specific for current problem
- Also, some pre-defined stereotypes
- UML allows you to provide a new icon!
- Syntax: Above name add <<stereotype>> inside guillemets (French quotes)
- Again, used to provide extra info about the UML modeling construct

Stereotypes (cont'd)

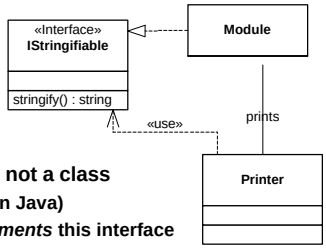
- UML predefines many:
  - Classes: <<interface>>, <<type>>, <<implementationClass>>, <<enumeration>>, <<thread>>
  - Constraints: <<precondition>> etc.
  - Dependencies: <<friend>>, <<use>>
  - Comments: <<requirement>>, <<responsibility>>
  - Packages: <<system>>, <<subsystem>> (maybe classes, too)
- Or, create your own if needed.

Class Categories

- You <u>can</u> use stereotypes to organize things by category within a class box



Stereotype Example



- IStringifiable is not a class
  - Interface (as in Java)
  - Module *implements* this interface
- Printer depends on what's in the interface

Interfaces

- Interface: specifies a set of operations that any class *implementing* or *realizing* the interface must provide
  - More than one class may realize one interface
  - One class may realize more than one interface
  - No attributes, and no associations
- Notation:
  - Use <<interface>> with a class; list operations
  - "Lollipop" notation

Interface Example Diagram

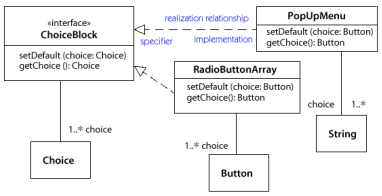


Figure 4-9. *Realization relationship*

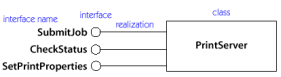


Figure 4-10. *Interface and realization icons*

Classes Realize an Interface

- "Realizes" AKA *implements*, *supports*, *matches*, etc.
- This means that class provides all the operations in the interface (and more?)
  - Remember, no implementation in interface definition
- Realization shown with dashed line, hollow arrow
  - Like dependency plus generalization
- Why have this?
  - Just factor out common functionality?
- Better "pluggability", extensibility

Tagged Values, Properties

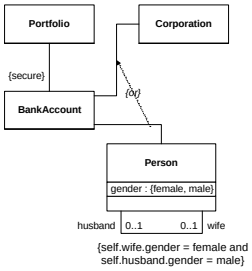
- Every modeling element in UML has its set of properties
  - Classes have: name, attributes, operations, etc.
  - What if we want to add our own? (e.g. author?)
- Just add text in curly-brackets, with name=value, and put below the element name
- Note: These tell you something about the model, not about the final system to be built!
  - Often used for code generation, version control, etc.
- Example: {abstract} classes instead of italics

Abstract Classes

- Implementation not provided for one or more operations
  - So, a subclass must extend this to provide implementations
- How to show this in UML?
  - Either italics for class name and operations
  - Or, use {abstract} property by name
- An abstract class with no attributes and all abstract operations is effectively an interface
  - But Java provides a direct implementation

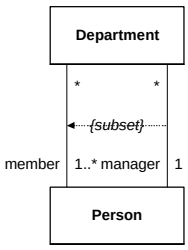
Constraints

- Conditions that restrict values, relationships,…
- Can be free text or Object Constraint Langauge (OCL) (see textbook)
- Recommendation: Use sparingly!
- This example: from *UML User Guide*, p. 82



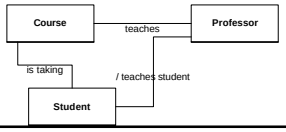
Constraints and Semantics

- Example from *UML User Guide*, p. 88
- A dependency and a constraint used
- Shows Manager must be one of Members of a Department
- One link (say, Jane-to-DeptA) is a subset of all links between Persons and DeptA

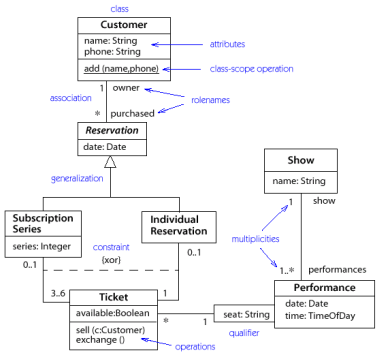


Derived Associations

- Often an association in a model be deduced from the existence of one or more other associations
- Do we show it? Is it redundant?
- Option: Draw it but mark it as derived
  - Use a slash symbol / before name
- Can use slash in front of class attributes too!



Example: Ticket Sales

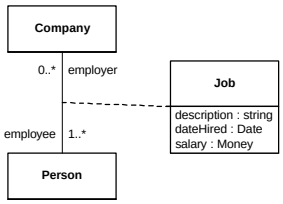


Unused slides follow

Association Classes

- Recall that qualified associations really mean that the link between two objects has an attribute
- Often associations are “first-class” things
  - They have a life-time, state, and maybe operations
  - Just like objects!
- Association classes
  - Same name as the association because...
  - They represent the same thing!

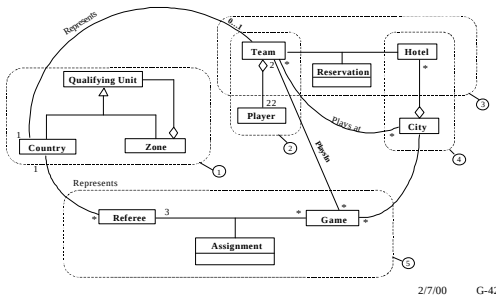
Association Class Example



World Cup Example

- We need a system to handle the World Cup. Teams represent countries and are made up of 22 players.
- Countries qualify from zones, where each zone is either a country or a group of countries.
- Each team plays a given number of games in a specific city. Referees are assigned to games. Hotel reservations are made in the city where the teams play.

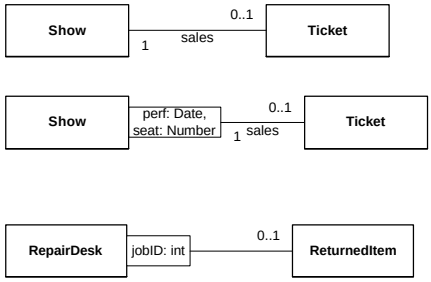
World Cup Problem: Class Model



Qualified Associations

- Equivalent to programming language idea of ***lookup, map, dictionary, associative array,*** etc.
- An object is associated with some number of other objects in a class
  - How do we identify which one we want given that association?
- The qualifier documents ***attribute(s)*** used to identify which object
  - The "key" for "lookup"
- Formally, these are attributes of the association

Qualified Association Examples:



Identifying Classes for Requirements

- From textual descriptions or requirements or use cases, how do we get classes?
- Various techniques, and practice!
  - Key Domain Abstractions:
    - Real-world entities in your problem domain
  - Noun identification
    - Not often useful (but easy to describe)
- Remember: external view of the system for requirements
  - Not system internals, not design components!

Noun Extraction

- Take some concise statement of the requirements
- Underline nouns or noun phrases that represent things
  - These are ***candidate classes***
- Object or not?
  - Inside our system scope?
  - An event, states, time-periods?
  - An attribute of another object?
  - Synonyms?
- Again, looking for "things"

Identifying Good Objects

- Don't forget from earlier:
  - attributes and operations are encapsulated in objects
  - objects have a life-cycle
- Also, don't worry about user interface
  - Think of user-commands as being encapsulated in the actors
- Consider:
  - Collections, things in a container
  - Roles
  - Organizations

Actors and Classes

- In some diagrams, actors represented as class boxes
  - With special stereotype above class name: <<actor>>
- UML allows special graphical symbol (e.g. a stick figure) to replace stereotyped classes
  - See Richter, p. 53