Inf111/CSE121: Software Tools and Methods

Fall Quarter, 2011

Susan Elliott Sim
ses@ics.uci.edu

FAQs

- Is there lab today?
  - No.
- Are discussions mandatory?
  - We don't take attendance, but we do tell you how to do the homework. It's hard to do well in this course without attending discussion.
- What textbooks do I have to buy?
  - Buy Larman (Applying UML and Patterns)
  - Rest will be available electronically

Course Home Page

- URL
  - http://www.drsusansim.org/teaching/inf111/
  - Linked from my home page
  - Linked from EEE
- Syllabus and slides will be posted there
  - In advance, when possible, but generally not

Course Overview

- Organizational details
- What this course is about

Objectives for the Course

- Modern software development methods
  - Agile, automated testing, RUP
- Increase your exposure to a number of software tools
  - Eclipse, subversion, JUnit, Rational Developer Workbench
- Give you experience working with methods
  - Ideas more important the particular tool.

- Hands on
- Interactive
- Lots of work, but you'll learn useful skills.
  - All of these tools and methods (or ones similar to them) are used in industry.
  - Good for your resume

Why do we need tools and methods?

- As teams and projects become larger, practices that worked with 1 or 2 people don't work any more
- Most students:
  - Use an IDE
  - Use the file system to manage projects
  - Print statements used for debugging
  - No install support

- Problem: Approach doesn't scale
  - More people, bigger code, different versions, multiple platforms (development and delivery)

Example: Longhorn Project

- 5000 developers (excludes non-tech staff)
- 40 million lines of code
  - 16 million added in the last three years
- Daily builds and regression testing
  - Takes 3 working days from the time you submit changes until you receive an executable back
- ~1.7 testers to every programmer
- Needs to be backwards compatible
- Installation needs to work on millions of machines

Challenges

- Logistics
  - How do you design a process that will allow 5000 people to work together at the same time?
  - How do you test so much code? For so many platforms?
- Design
  - How do you do design on a system with 40 million lines of code?
  - How do you maintain conceptual (architectural) integrity?

Course Mechanics

- Lectures
  - T Th 8:00 – 9:20am
- Discussion
  - Attend 1 per week, 2 sections (M)
  - Materials will not be distributed electronically
- Laboratories
  - Attend 1 per week, 3 sections (M)
  - Attend the section that you are registered in for grading

How to Find Me

- Email
  - Susan Elliott Sim (ses@ics.uci.edu)
  - To ensure a response, include “Inf111” or “CSE121” in the subject and send from a UCI account
- IM
  - benevolentprof on gtalk and msn (don’t send email here!)
- Twitter
  - @benevolentprof
- Office
  - DBH 5226
- Office hours
  - T Th 9:30-10:30am
  - Other times available by appointment

Textbooks

- Larman
  - UML
  - Design patterns
  - Buy this book
- van Vliet
  - 44/52 text
  - Background
  - Motivation
- Assorted Papers
  - No Silver Bullet
  - Scrum
- Syllabus contains assigned readings by topic
  - May not discuss all readings in class
  - Readings will be on the examinations

Discussion

- Discussions are for:
  - Reviewing material
  - Presenting tools
  - Discussing assignments
  - Discussing readings
  - Preparing for tests
  - Reviewing tests and assignments

Laboratories

- For getting help
- For grading in-class portion of homework
  - Work needs to be completed by the END of the lab session
- Attend the section that you are registered in

Grading

- Laboratories 60% (best 6/7)
- Midterm 15%
- Final Exam 25%

Homework Assignments

- 7 Assignments
  - One per week, except for first, midterm, and last week of quarter
  - Exception: Coding dojo
- Format of assignment
  - Laboratory portion
  - Take home portion
- Distributed on Mondays
- Due on Thursday of following week at 11pm
  - No late assignments accepted
- Submit using checkmate.ics.uci.edu

…Assignments

- Using software tools
  - Get accustomed to tools early
  - All tools installed in CS lab
  - Freeware tools can be downloaded and installed at home
  - Specific tools will be required, e.g. Rational Software Architect, not Visio
- Assignments will be submitted electronically
  - Include diagrams, tool logs, if required
- Quality is more important than length
- Express yourself clearly
  - Be concise!
  - Be specific!

…Assignments

- A forum has been created on the MessageBoard for assignments
  - Post your questions there
  - The TAs, other students, and I will post answers, suggestions, and additional information
- We will go into "silent running" during the last 24 hours before an assignment is due
  - Questions asked before the cut-off will be answered by us
  - You can always answer each others' questions
  - Not reasonable to ask TAs to stay up all night to answer questions
  - Provide some motivation to start early

Midterm and Final Exams

- Primarily based on readings
- Make-up exams only for documented medical reasons
- Will be "crowdsourced"

Policies

- Use your UCI account
  - Because of privacy
  - Needs to be activated if you are new
- If you need accommodations due to a disability, talk to me
  - see also: UCI Disability Center
- Re-grading
  - See the TA or Reader first, then me if it is not resolved to your satisfaction
  - Within 1 week, accompanied by a clear explanation of what needs to be re-considered and why
  - Entire question will be considered (grade may go up or down)

Cheating

- Letter in your UCI file
- Course grade lowered, possibly to F

- No team work on assignments
- Discussing an assignment is OK, copying the solution is not

- Things that are copied from books or Web pages need to be quoted and the source must be given

Ask Questions

Tuesday, September 27

Announcements

- Charity Scrum Course
  - https://sites.google.com/site/payitforwardcsm/
- Free Food
  - Wednesday, September 28
  - 10:30 - 11:45 am
  - Bren Hall 6011
- Please read Brooks for Thursday

UML

Unified Modeling Language

- Let's look at each of the words in the name
- Unified
  - Two important methodologists Rumbaugh and Booch decided to merge their approaches in 1994.
    - They worked together at the Rational Software Corporation
  - In 1995, another methodologist, Jacobson, joined the team
    - His work focused on use cases
  - In 1997 the Object Management Group (OMG) started the process of UML standardization

Models

- Models are abstract representations
  - Contain essential characteristics and omit non-essential details
    - "Essential" depends on the problem domain
  - There are no perfect representations
- Models can be representations of the world
  - Domain models
  - Requirements
- Models can be representations of software
  - Specifications
  - Design
  - Systems

Why make models?

- Systems are complex and hard to understand
  - The world, organizations, relationships, software
- Models can make certain aspects more clearly visible than in the real system
- What can you do with models?
  - Express your ideas and communicate with other engineers
  - Reason about the system: detect errors, predict qualities
  - Generate parts of the real system: code, schemas
  - Reverse engineer the real system to make a model

What constitutes a good model?

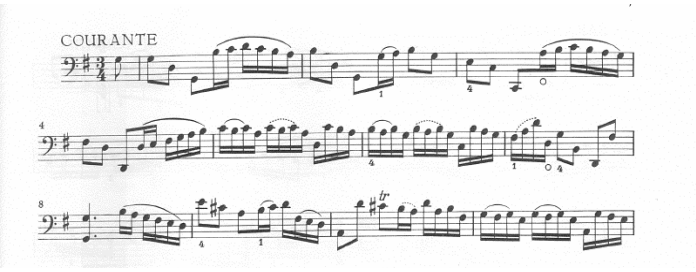
- A model should…
  - Provide abstraction
  - Render the problem in a format amenable to reasoning
    - use a standard notation
    - be understandable by clients and users
    - lead software engineers to have insights about the system
    - make the problem solvable computationally
  - Be good enough

Modeling Languages

- Natural language
  - Extremely expressive and flexible
  - Very poor at capturing the semantics of the model
  - Better used for elicitation, and to annotate models for communication
- Semi-formal notation
  - Captures structure and some semantics
  - Can perform (some) reasoning, consistency checking, animation, etc.
    - Examples: diagrams, tables, structured English, etc.
  - Mostly visual - for rapid communication with a variety of stakeholders
- Formal notation
  - very precise semantics, extensive reasoning possible
  - Every detailed models (may be more detailed than we need)

Visual Languages

- Words = symbols
- Syntax = rules for combining symbols, drawing and layout of language
  - Example: Sheet music, tic-tac-toe
  - Example: Visual Basic is a visual programming language



UML

- UML is a semi-formal visual modeling language
  - Semantics are not completely specified by standard
  - It has *extension* mechanisms
  - It has an associated textual language
    - *Object Constraint Language* (OCL)
  - Well suited for object-oriented designs

Types of UML Diagrams

Structure	Behavior

• Class diagrams
• Object diagram
• Package diagram
• Composite structure diagram
• Component diagram
• Deployment Diagram
	• Activity diagram
	• Use case diagram
	• State machine diagram
	• Interaction diagrams
	  – Sequence diagram
	  – Communication diagram
	  – Interaction overview diagram
	  – Timing diagram

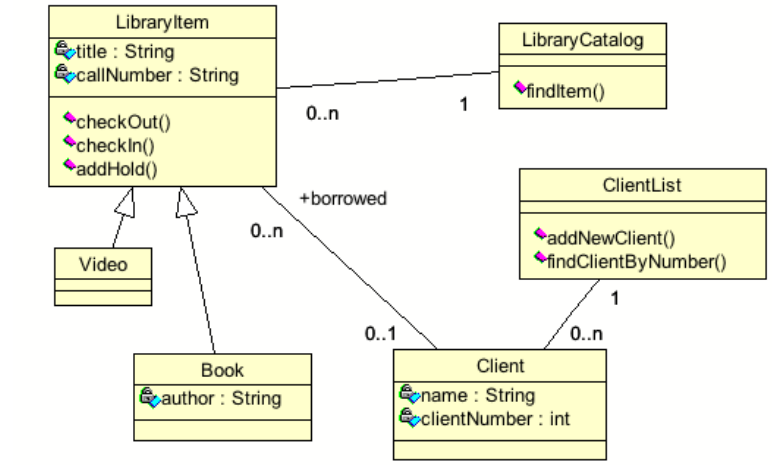
Types of UML Diagrams

Structure	Behavior

• Class diagram
• Object diagram
• Package diagram
• Composite structure diagram
• Component diagram
• Deployment diagram
	• Activity diagram
	• Use case diagram
	• State machine diagram
	• Interaction diagrams
	  – Sequence diagram
	  – Communication diagram
	  – Interaction overview diagram
	  – Timing diagram

UML- Structure Diagrams

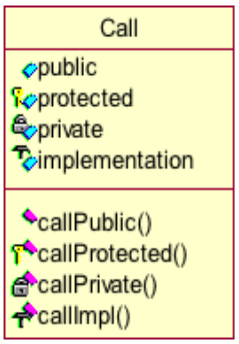
Class Diagram



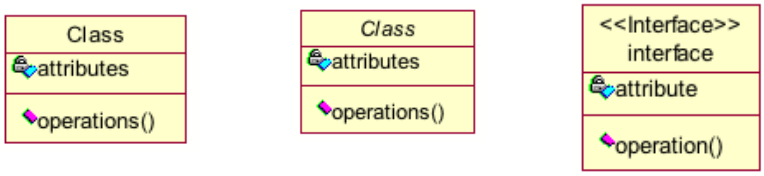
- A UML class corresponds to a Java class.

Classes, Attributes, and Operations

- Public (nothing or +)
- Protected (#)
- Private (-)



Class Diagrams



- Name
  - `Name:type`
- Attributes
  - `visibility name: type multiplicity = default {property-string}`
- Operations
  - `visibility name (parameter-list) : return-type {property-string}`
  - `direction name: type = default`

Attribute Syntax

`visibility name: type multiplicity = default {property-string}`

- optional visibility: + public, - private, # protected
- name: the name of this attribute
- optional type: data type of this attribute
- optional default: initial value of attribute
- optional property string: OCL, e.g. ordered, readonly

- Examples:

  firstName

  -middleName

  lastName: String

  -age: int = 0

  birthSign: String = "Gemini"

Property String

- Any information that cannot be expressed in the diagram notation can be included as text
  - Example: constraints
- Example:

  customerNumber: int { >=0 }
- All diagram elements can be annotated with constraints
- Can be:
  - Natural language text
  - Object Constraint Language
  - Predefined properties
    - Examples on next two slides
  - Any other text

Property Strings on Attributes

- changeable (default)
  - Value of attribute can be changed
  - May want to list legal/possible values
- addOnly
  - Can add possible values, but can't change existing ones
- frozen
  - Can't add or change

Operation Syntax

```
visibility name (parameter-list) : return-type
  {property-string}
```

- optional visibility: + public, - private, # protected
- name: the name of this operation
- optional parameters-list: parameters to this operation

```
direction name: type = default
```

  - optional direction: in, out, inout
  - name
  - optional type
  - optional default
- optional type: return type of operation
- optional property string

Operation Syntax

- Examples:

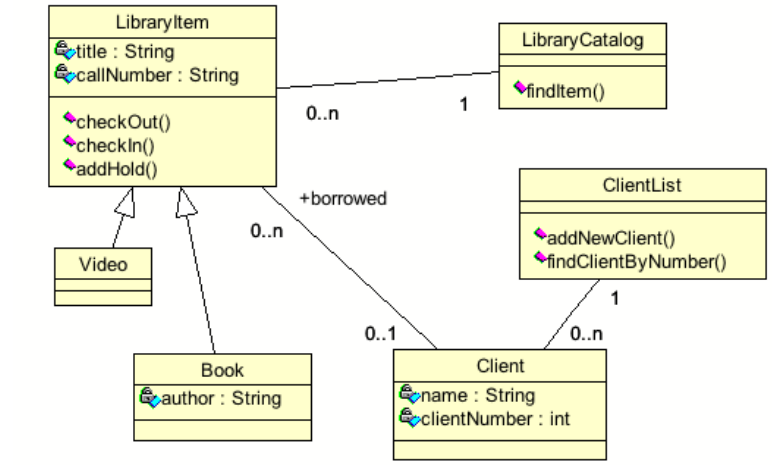
  getFirstName()
  +getMiddleName(): String
  -setLastName(name: String)
  +paintPortrait(inout c: Canvas, subject: Person)

Property Strings on Operations

- sequential
  - Only one call to a method within an instance
- concurrent
  - Multiple simultaneous calls to a method within an instance may occur
- guarded
  - Multiple simultaneous calls to a method within an instance may occur but only one at a time will be executed
- isQuery
  - Operation doesn't change the value of any attributes

Thursday, September 29

Class Diagram



Associations

- Relations between classes
- Roles
  - analogous to names of instance variables
- Multiplicities
  - 0, 1, *, 0..1, 1..*, 5..6, and so on
  - says how many objects each object knows
  - would be realized through arrays, Sets, Lists, and so on
- Navigability
  - bidirectional: each class references the other
  - unidirectional: A knows B, but B doesn't know A
  - no arrow heads: means either "bidirectional" or "not specified"

Multiplicities

- Descriptive
  - Optional (0 or more)
  - Mandatory (at least 1)
  - Single-valued (upper bound)
  - Multi-valued (upper bound of >1, usually *)
- Symbolic
  - 0..1 (zero or one, i.e. optional)
  - 1 (or another number, exactly the number specified)
  - 2..4 (range)
  - * (zero or more, no upper limit; n in Rational)

Role Name vs. Association Name

Types of Association

- Inheritance
  - Generalization
  - Realization
- Aggregation
- Composition

Some Mnemonics

- Generalization = is-a
- Composition = has-a
- Aggregation = part-of

- Examples:
  - Toaster is-a Appliance
  - Toaster has-a slot
  - Member part-of club

Inheritance

- Generalization correspond to the Java keyword "extends"
  - Generalizations are drawn with a solid line and a white triangular arrow touching the superclass.
- Realization correspond to the Java keyword "implements"
  - Realizations are drawn with a dashed line and a white triangular arrow touching the interface.
- UML itself is not restricted to single inheritance. However, you would not use multiple inheritance if you plan to implement in Java.

Inheritance

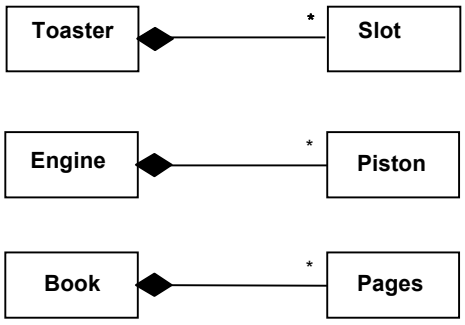
- It is common practice to arrange the diagram so that:
  - Generalization and Realization arrows point upward
  - If one class has many subclasses, the Generalization arrows overlap

Types of Associations

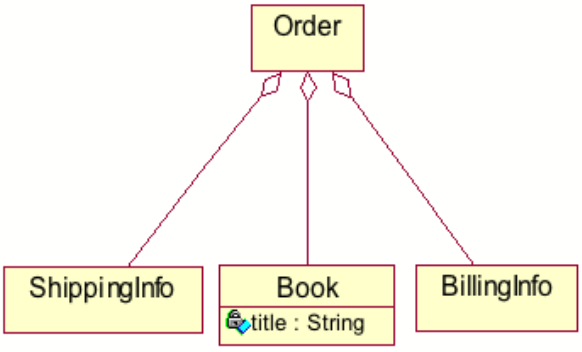
- Composition = Black diamond: parts are created with the whole and stay with exactly one whole until both are destroyed together.
- Aggregation = White diamond: parts can join the whole and later leave; one part could be part of more than one whole.
  - Some operations will be recursively applied to the parts of a whole

Association Type: Composition

- A *composition* is a strong kind of aggregation
  - if the aggregate is destroyed, then the parts are destroyed as well



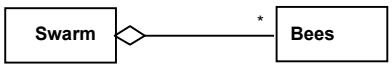
Example Aggregation



Association Type: Aggregation

•Aggregations are special associations that represent 'part-whole' relationships.

- The 'whole' side is often called the *assembly* or the *aggregate*
- This symbol is a shorthand notation association named `isPartOf`



When to use an aggregation

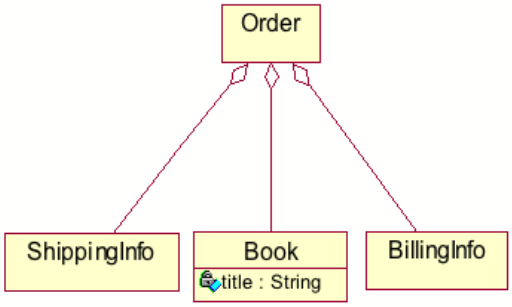
- As a general rule, you can mark an association as an aggregation if the following are true:
  - You can state that
    - the parts 'are part of' the aggregate
    - or the aggregate 'is composed of' the parts
  - When something owns or controls the aggregate, then they also own or control the parts
- Use with care

Types of Associations

- Aggregation = White diamond: parts can join the whole and later leave; one part could be part of more than one whole.
- Composition = Black diamond: parts are created with the whole and stay with exactly one whole until both are destroyed together.
  - Like aggregation but with extra requirements for managing the lifetime of internals

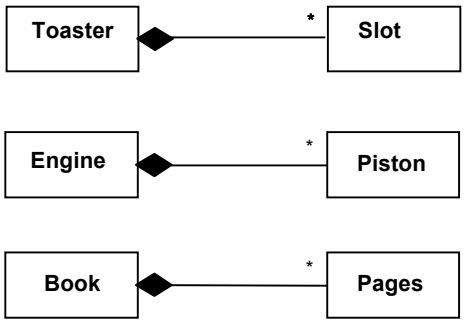
Association Type: Aggregation

- Aggregations are special associations that represent 'part-whole' relationships.
  - The 'whole' is often called the *assembly* or the *aggregate*
  - This symbol is a shorthand notation association named `isPartOf`



Association Type: Composition

- A *composition* is a strong kind of aggregation
  - if the aggregate is destroyed, then the parts are destroyed as well

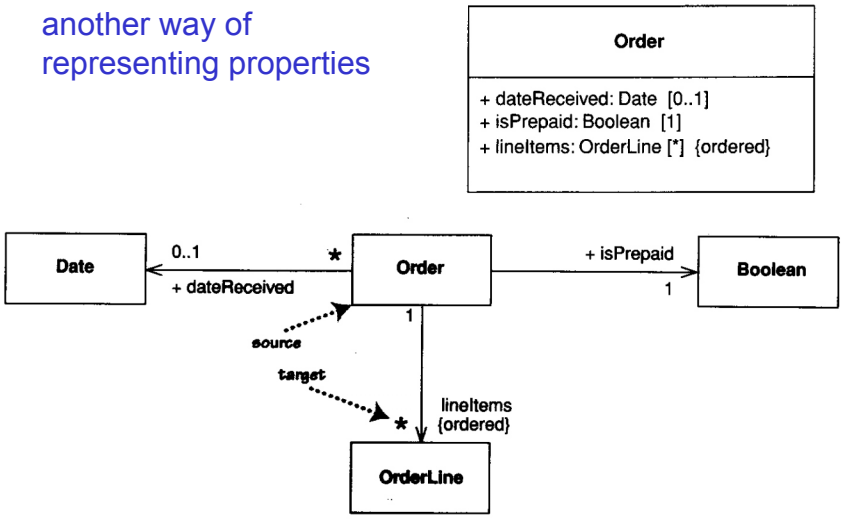


When to use a composition

- An association is a composition if the whole has sole responsibility for the memory associated with a part
  - In Java, the whole must prevent garbage collection of the part
  - In C/C++, the whole allocs/frees memory for the part
- As a general rule, you can mark an association as a composition if the following are true:
  - You can state that
    - the parts 'are part of' the whole
    - or the whole 'is composed of' the parts
  - When something owns or controls the whole, then they also own or control the parts
- Use with care

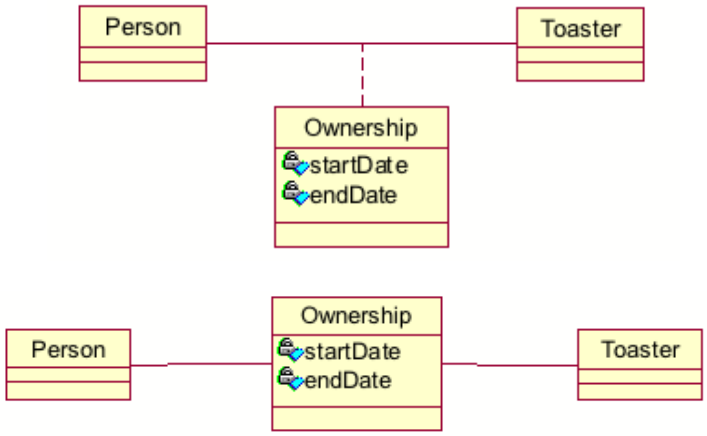
Associations and Properties

- Associations are another way of representing properties

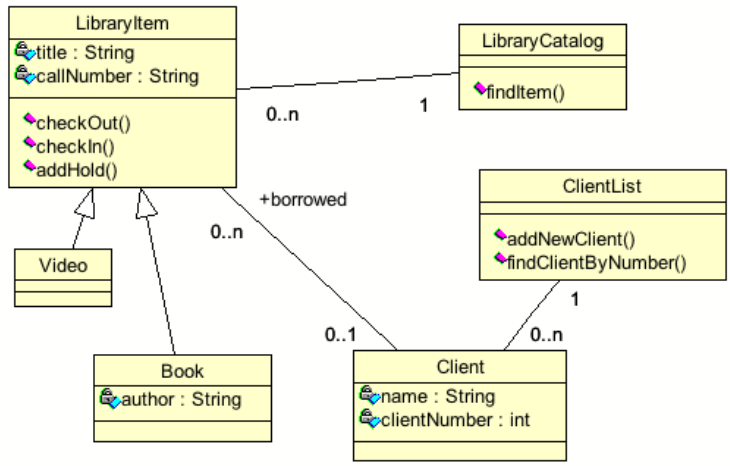


Association Classes

- When you want to add properties or operations to an association, use an association class.
  - Frequently simpler to promote associate class to full class



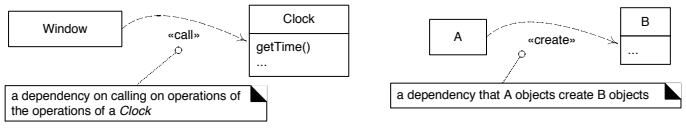
Class Diagram

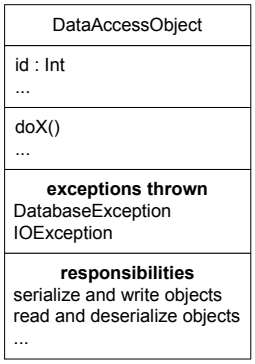


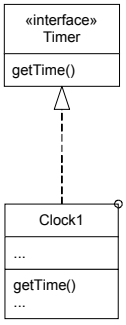
More in Larman

- Keywords («guillemets»)
  - Stereotypes
  - Examples: calls, interface, permit,…
- Responsibilities
  - Another compartment in class
  - Items prefixed by --
- Template classes

Examples







Hints for Class Diagrams

- Remember: models are for communication
- Remember: include only important stuff
- How do I find classes, attributes and so on?
  - Classes often correspond to nouns
  - Associations often correspond to verbs
- A class should
  - Represent a coherent concept
    - Principle: Low Coupling, High Cohesion
  - Have a small, well-defined set of responsibilities
  - Be named with a singular noun that says what each instance of the class is
  - Have no more than 10-20 operations

Hints for Class Diagrams

- Class diagrams should
  - have a single purpose
  - have a title that expresses the purpose
  - show only things that are relevant for this purpose
- Avoid
  - cyclical dependencies, if possible
  - generalization hierarchies with more than 5 levels
  - crossing edges

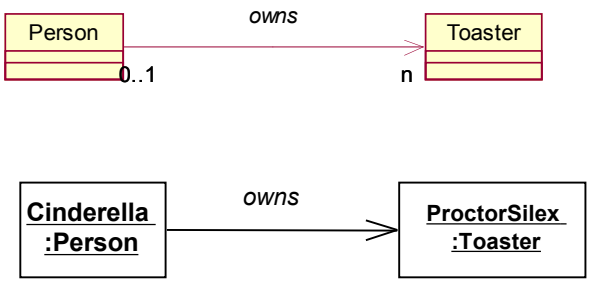
Hints for Class Diagrams

- Use colors judiciously
  - to highlight and group things
  - unless you have to print it in black-and-white!
- Lay out classes in a meaningful way
  - similar classes close to each other
  - top: closer to the user, bottom: closer to the data structures

Object Diagrams

- Show instantiation or specification of classes
- Associated with a particular use or instance of the model
- Differences between Classes and Objects
  - Name:class is underlined
  - Attributes and operations included as needed
  - Fields have data added
- Useful for showing interactions between interfaces, abstract classes, etc.
  - Where functionality is not clear until instantiation

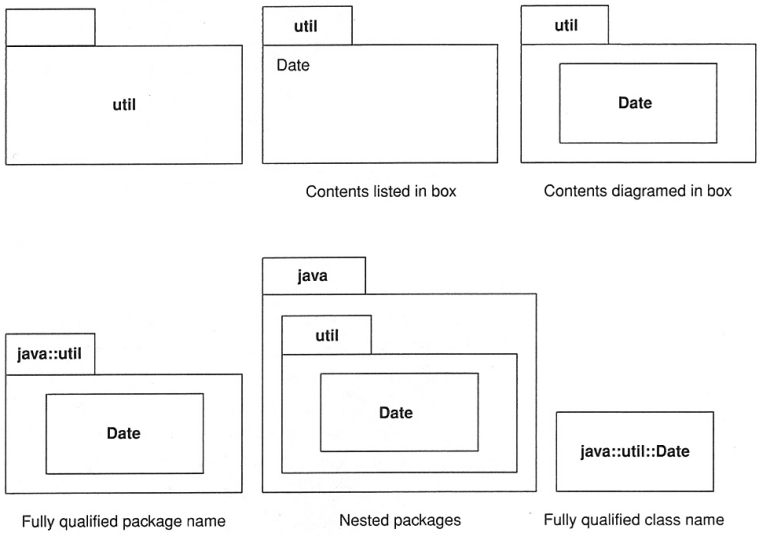
Example: Class to Object Diagrams



Package Diagrams

- Package is a grouping construct
  - Most commonly used for class diagrams, but can be used with any UML diagram or elements
  - Used to create a hierarchy or higher level of abstraction
  - Corresponds to package in Java
- Each package represents a namespace
  - Like Java, can have classes with same name in different packages

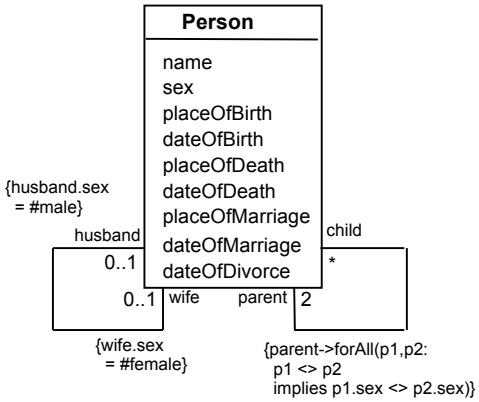
Representing Packages



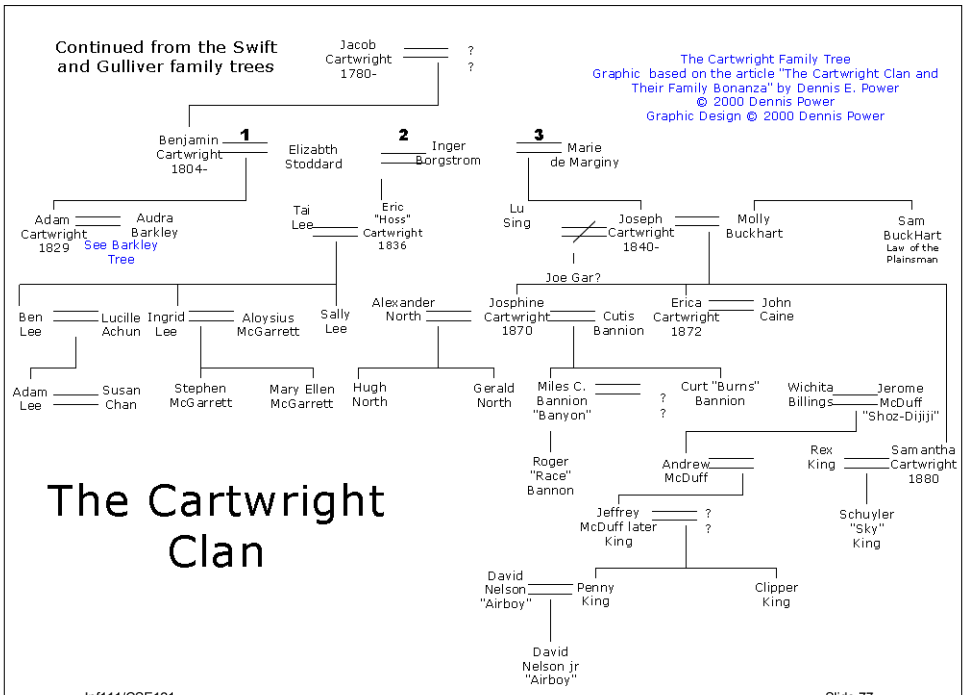
Design Example: Genealogy

- Shows relationships between family members
  - Often depicted as a family tree
    - A family tree is a visual language

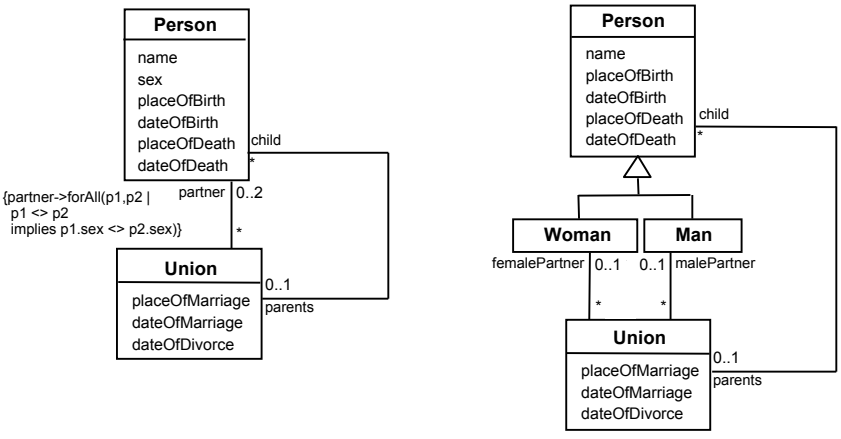
Class Diagram for Genealogy



The Cartwright Family



Possible Refinements



Thoughts on These Solutions

- Can't represent same-sex marriage (or same-sex unions)
- A person is only allowed to have two parents
  - Step-parents
  - Fostering
  - Biotechnology-enabled parenting: sperm donor, egg donor (with or without DNA), surrogate mother
- Sex is an M/F attribute
- Modeling is always political