

## Component Level Class Design

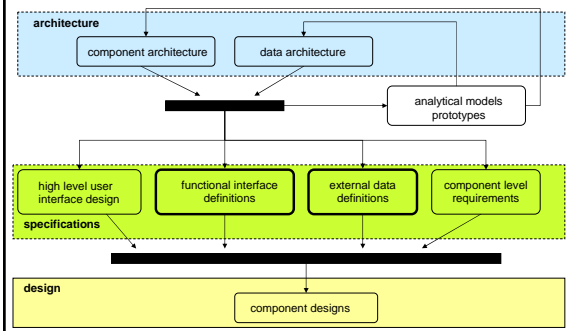
- Specifications, Design, and Components
- Packages and Classes
  - patterns and reasons for class creation
  - elements of good class design
  - classes in non object oriented languages
  - class packages
  - elements of good packaging
- Diagramming Classes & Objects
  - UML class models
  - UML object models

10/11/2007

Class Design

2

## Model Hierarchy/Succession



10/11/2007

Class Design

3

## What is a “component”?

- A modular, deployable, and replaceable part of a system, that encapsulates implementation, and exposes a set of interfaces.
  - it is a defined piece of a larger system
  - it can be added or removed from that system (not necessarily a **Field Replaceable Unit**)
  - it contributes to the working of the system
  - its inner mechanisms may be hidden
  - its functionality is defined by an interface

10/11/2007

Class Design

4

## Component Specifications

- a step between requirements & design
  - a component specific list of requirements
  - the basis for the component design
- functional specifications
  - written from the user's point of view
  - enumerate component capabilities, interfaces
- technical specifications
  - written to guide the implementer
  - capture key design decisions or suggestions

10/11/2007

Class Design

5

## Design Patterns

- Much is written about design patterns
  - architectural, class structure, algorithmic
  - good solutions to recurring problems
  - I encourage you to study about them
- view them as Chess gambits or Go joseki
  - using them will improve your game
  - studying them will improve your understanding
  - but in the end, they are merely tactical aids
    - they can't analyze the board for you
    - they aren't a substitute for strategy

10/11/2007

Class Design

6

## When to create a new class

- provide needed objects
  - obvious objects from the problem domain
- provide better behaved objects
  - kinder, gentler versions of real objects
- compartmentalize complexity
  - bring all related code into a single place
  - simplify interface seen by rest of system
- make applications more stable & portable
  - isolating implementation specifics in a class
  - abstraction protects app from future evolution

10/11/2007

Class Design

7

## Characteristics of a good class

- it is well abstracted ●
- it is cohesive ●
- it exhibits good information hiding ●
- Other principles are tests of goodness ●
  - Open/Closed principle ●
    - open for extension, closed for modification ●
  - Liskov Substitution principle ●
    - derived sub-class can substitute for its parent ●
  - Dependency Inversion Principle ●
    - depend on abstraction – not implementation ●

10/11/2007

Class Design

8

## OO Languages and Design

- OO languages provide valuable features
  - mechanisms to support class inheritance
  - mechanisms to encourage information hiding
  - explicit support for interface polymorphism
  - automatic object instantiation
- these help us design better software
  - organizing our designs into modular classes
  - consciously decide what is public/private
  - encourage us to reuse common components

10/11/2007

Class Design

9

## Classes in non-OO languages

- the basic principles of good design
  - apply to all software: C, FORTRAN, asm, perl
- any module, in any language, should
  - implement a general and intuitive “class”
  - export a well abstracted interface to that class
  - be cohesive with respect to that class
  - employ good information hiding
  - be usable, w/o change, for many purposes
  - be organized/grouped with related modules

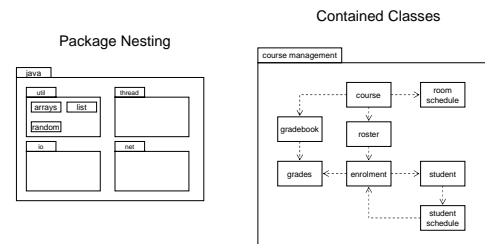
“Program into your language, not in it.” ●

10/11/2007

Class Design

10

## UML Package Contents



10/11/2007

Class Design

11

## (Class Packages)

- some classes make sense in isolation
  - stacks, queues, strings, input files
- some classes naturally come in groups
  - courses, rosters, programs, grades
- a package is a collection of classes
  - that is aggregated together into a group
  - that are added and removed as a group
- some OO languages support packages
  - may not correspond to install-time packages

10/11/2007

Class Design

12

## Class Packaging Principles

- Release/Reuse Equivalency Principle
  - “the granule of reuse is the granule of release”
  - (keep your packages cohesive)
  - if someone only needs classes A and B, don't force him to take the unrelated class C as well.
- Common Closure Principle ●
  - “classes that change together travel together”
  - (avoid strong inter-package coupling)
  - if class B depends on the implementation of class A, deliver both of them in a single package.

10/11/2007

Class Design

13

## UML Class Models

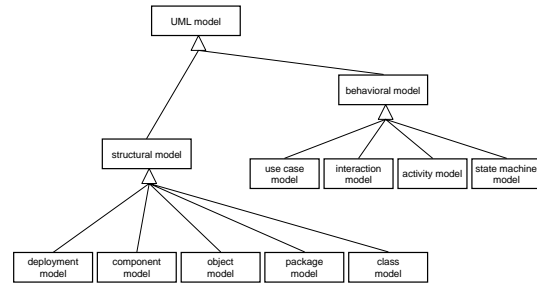
- describe classes and static relationships
  - these are not models of run-time objects!
- boxes represent classes
  - may have 3 parts: **name**, attributes, methods
- lines represent class relationships
  - inheritance (source is derived from target)
  - associations (source refers to target)
  - aggregations (multiple instances of target)
  - compositions (source is sum of the targets)
  - dependency (source uses target)

10/11/2007

Class Design

14

## UML Class Inheritance



10/11/2007

Class Design

15

## UML Class Properties

class name	visibility <b>name</b> : type [#] = default { properties }
class properties	+ ... public EXAMPLES: dueDate - ... private assgName: string + ... package # ... protected + grade: int {readonly} - students: string[1..100]
class methods	

- specification may be complete
  - all properties listed w/complete declarations
- specification may be partial
  - list only properties/information “of interest”
  - list only properties different from parent class
  - may list no types (or even properties) at all

10/11/2007

Class Design

16

## UML Class Methods

class name	visibility <b>name</b> (parameters) : return type { properties }
class properties	EXAMPLES: addStudent deposit( amount : dollars ) cancelJob( reason ) : boolean
class methods	

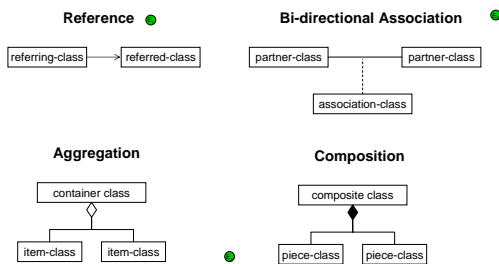
- specification may be complete
  - all methods listed w/complete declarations
  - parameters can be defined as **in**, **out**, **inout**
- specification may be partial
  - list only methods “of interest”
    - simple get/set methods are routinely ignored
  - specify only non-obvious return types
  - specify only key parameters, non-obvious types

10/11/2007

Class Design

17

## UML Class Associations

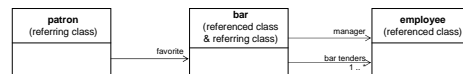


10/11/2007

Class Design

18

## Labeling UML Associations

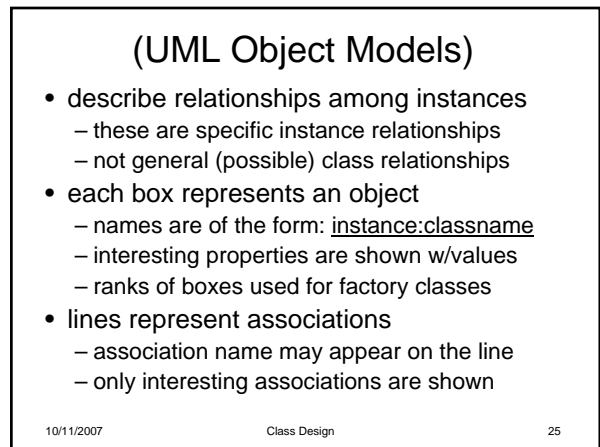
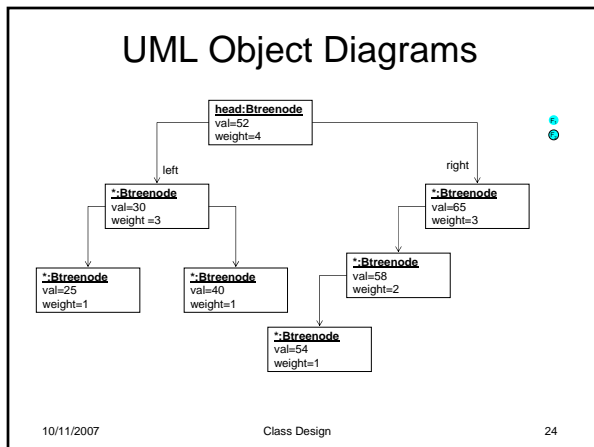
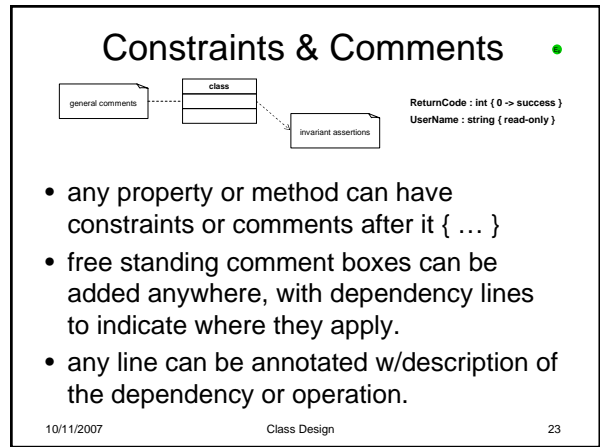
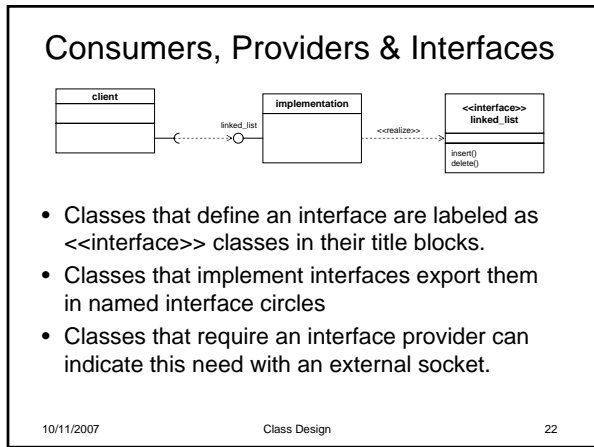
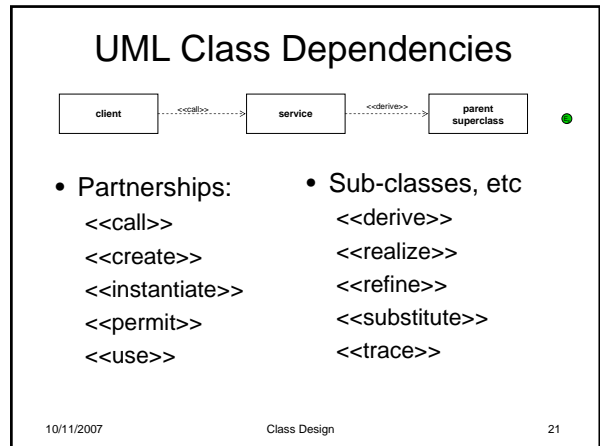
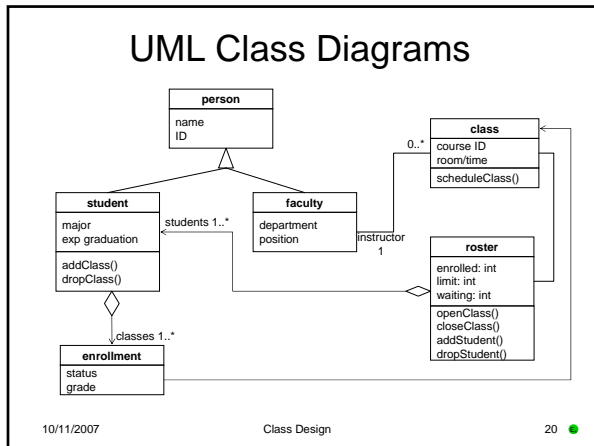


- names & counts are association-specific
  - these are **class** diagrams, **not objects**
  - classes may share multiple associations
- labels go on target end of association
  - name by which **this** object is known
  - number of **this** object that can be referred to
  - this becomes an issue for bidirectional associations
- you can also label the association line itself
  - to explain, or to distinguish among multiple associations

10/11/2007

Class Design

19



## For the next lecture

- McConnell ch 7, 9
  - routine design and development
- McConnell, ch 18
  - table driven methods
- UML Interaction Diagrams
  - diagramming messages and calls
- Ambler: UML State Diagrams
  - representing finite state machines
- Wikipedia: selected algorithmic patterns

10/11/2007

Class Design

26

## Supplementary Slides

10/11/2007

Class Design

27

## Further Reading

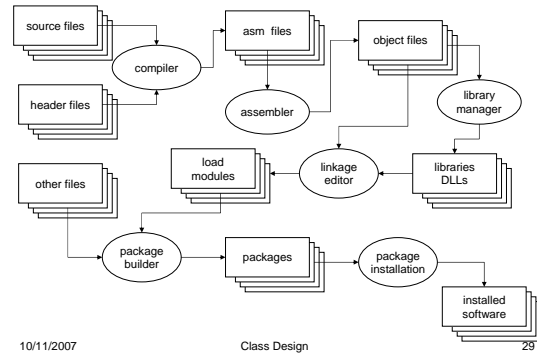
- Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, Design patterns: Elements of reusable object-oriented software, Addison-Wesley, 1995
- Mary Shaw, David Garlan, Software Architectures: Perspectives on an emerging discipline, Prentice-Hall, 1996

10/11/2007

Class Design

28

## Whence all these components?



10/11/2007

Class Design

29

## UML Package Models

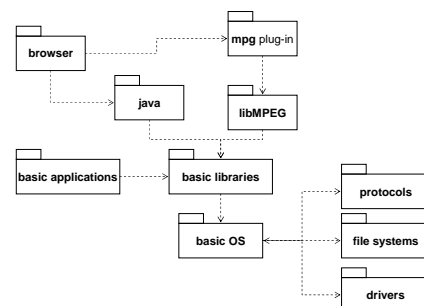
- describe package contents/relationships
- package is a collection of related classes
  - each could be described by a class diagram
  - contained within a single, large, package box
- tab-folders represent packages
  - with the package name at the top
- dashed lines represent dependencies
  - source package uses the target package

10/11/2007

Class Design

30

## UML Package Dependencies



10/11/2007

Class Design

31