System Modeling

- types of system models
- UML structural models
  - general conventions
  - UML class diagrams
    - inheritance, properties, methods, constraints
    - associations, dependencies, interfaces
  - UML package diagrams
  - UML object diagrams
  - UML component/deployment diagrams
- Data Navigation Diagrams

System Views

- Complex systems are hard to comprehend
  - more information than the mind can hold
- We have to decompose them
  - into hierarchies of systems and subsystems
    PC = { case, power supply, motherboard, devices}
    motherboard = { processor, bus, memory, support chips }
    bus = { addressing, data transport, arbitration }
  - into relatively independent systems
    skeletal, muscular, circulatory, neural, digestive …
  - into components on orthogonal axes
    pragmatic, moral, legal, aesthetic, political, …
- A system modeling language must be able to
  - describe hierarchies of models
  - describe different types (and aspects) of models

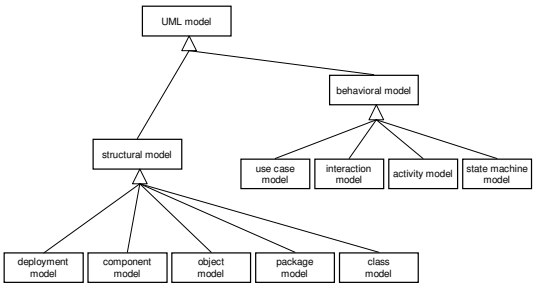
UML General Conventions

- Squares represent logical things
  - classes, objects, packages, components
  - box ornaments determine the type of thing
  - name of thing appears at top, inside the square
- Arrows represent relationships
  - communication solid, dependency dashed
  - arrow heads determine type and direction
  - relationships (and end connectors) can be named
- Circles represent interfaces
  - they can be named
- 3D boxes represent physical containers
- UML diagrams can be nested and composed

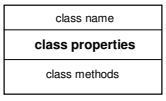
UML Class Models

- 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)

UML Class Inheritance



UML Class Properties



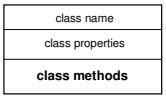
visability **name:** type [#] = default { properties }

\+ … public
\- … private
~ … package
# … protected

EXAMPLES: dueDate
assgtName: string
+ grade: int {readonly}
~ students: string[1..100]

- 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

UML Class Methods

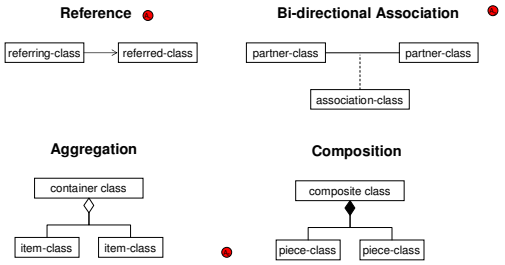


visability **name** (*parameters*) **:** return type { properties }

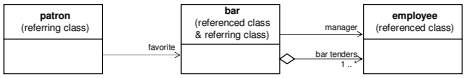
EXAMPLES: addStudent
deposit(amount : dollars)
cancelJob(reason) : boolean

- 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

UML Class Associations

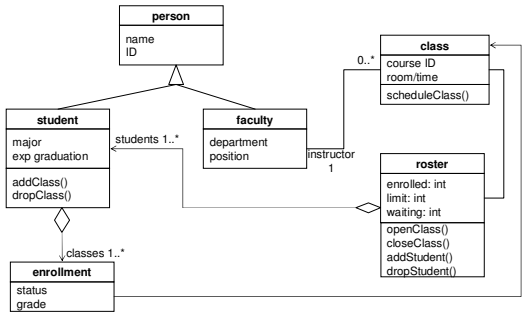


Labeling UML Associations



- 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

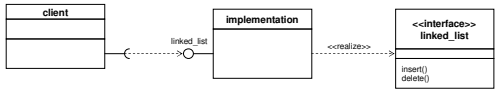
UML Class Diagrams



UML Class Dependencies

- Partnerships:
  - <<call>>
  - <<create>>
  - <<instantiate>>
  - <<permit>>
  - <<use>>
- Sub-classes, etc
  - <<derive>>
  - <<realize>>
  - <<refine>>
  - <<substitute>>
  - <<trace>>

Consumers, Providers & Interfaces



- Classes that define an interface are labeled as <<interface>> classes in their title blocks.
- Classes that implement interfaces export them in named interface circles
- Classes that require an interface provider can indicate this need with an external socket.

Constraints & Comments

- any property or method can have constraints or comments after it { … }
- free standing comment boxes can be added anywhere, with dependency lines to indicate where they apply.
- any line can be annotated w/description of the dependency or operation.

For Next Lecture

- Kampe: Why/how we model s/w systems
  - overview of analytical model goals/techniques
- Ambler: Principles of Agile Modeling
  - a manifesto for minimalist modeling

Supplementary Slides

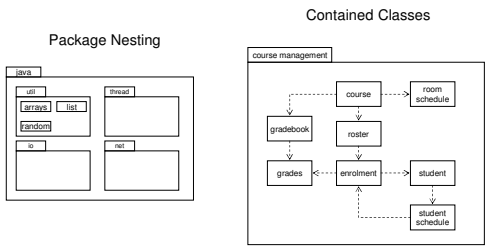
Class Packages

- 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
  - not the same as install-time packages

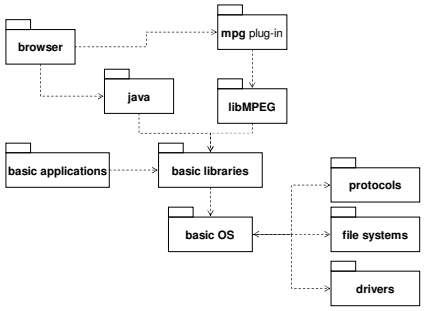
UML Package Models

- 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

UML Package Contents



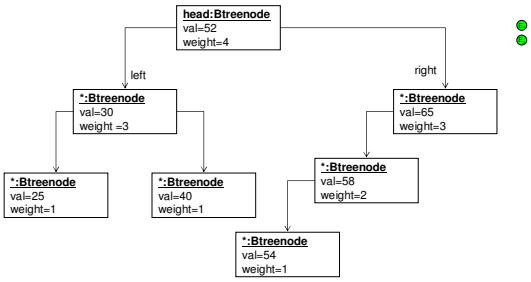
UML Package Dependencies



UML Object Models

- describe relationships among instances
  - these are specific instance relationships
  - not general (possible) class relationships
- each box represents an object
  - names are of the form: instance:classname
  - interesting properties are shown w/values
  - ranks of boxes used for factory classes
- lines represent associations
  - association name may appear on the line
  - only interesting associations are shown

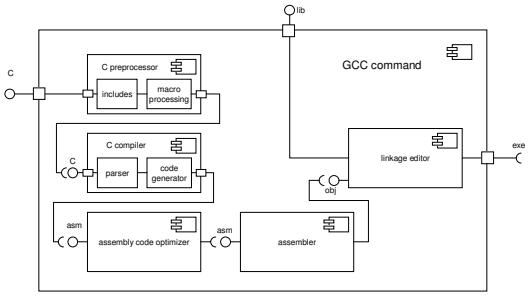
UML Object Diagrams



UML Component Models

- Class models are based internal structure
  - use, derivation, inheritance, dependency, etc.
- Component models take a black box view
  - system is composed of multiple black boxes
  - they are interconnected with one-another
  - some connectors support standard interfaces
- Component models focus entirely on
  - the independent components
  - their interconnections and interfaces

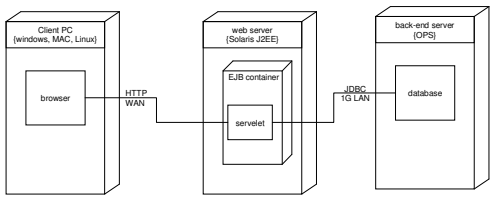
Component Models



Component Deployment Models

- Most UML structural models are logical
  - classes and software components
- We must also model physical systems
  - hardware components
  - physical interconnections between them
- And the deployment of logical functionality
  - which software runs on which hardware
    - distributed and client/server applications
  - which software runs in which container
    - application servers, virtual machines, etc.

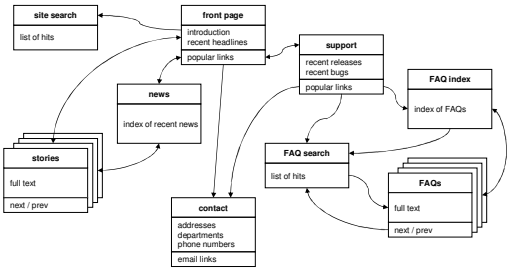
Component Deployment Models



non-UML structural modeling

- UML was designed for OO software
  - it is not suitable for all structural modeling
- Consider web content
  - structured pages of content
  - cross referenced w/links
  - navigable with standard menus
  - context sensitive searching
- How would you model this structure?

Content Navigation Model



Choice of Representation

- different models show different details
  - component … functions and relationships
  - deployment … of functionality to boxes
  - class/package … class content/relationships
  - activity … flow of control
  - swim lane … multi-thread flow control
  - state … state/transition models
  - data flow … data processing
- a single system can have many models
  - UML models are meant for hierarchical use