Class 9 CS540

Gregg Vesonder

Stevens Institute of Technology

Roadmap- Class 9

- Nits from last class
- Log Book volunteer
- OO Analysis and Design
- Quiz review
- Reading: BY 4, pp346-353, B13(next week)
- Reading next class: Brooks, Chapter 13

Calendar -Key Dates

- November 7th - second test
- November 21st - log books due
- December 12th - final exam

Clarifications

- None - Hobbits and Orcs?
- What's a module -
  - not flowchart driven - one begins with difficult design decisions or with design decisions that are likely to change.
  - Each module hides a decision from the others -- data hiding, sequence hiding, preparation to call a routine.
- Parnas - UML = Undefined Modeling Languages :-)
  - "a picture may be worth a thousand words, but in software it will take a thousand words to describe what it really means"

Hobbits and Orcs

- There are 3 orcs and 3 hobbits on the left bank of a river, and they all have to get over to the right bank. The only way across is by using a boat that can carry up to two creatures. The problem is, if there are ever more orcs than hobbits on one side, the orcs will eat the hobbits. Hobbits will not eat orcs under any circumstances. Of course at least one creature must be in the boat as it goes from side to side.

Logbook

- Your Entry
- Star (Language) Wars

Design

- Five elements affect the quality of the design: abstraction, modularity, information hiding, complexity and system structure (call graphs)
- Cohesion and coupling
- Patterns
- Vetting - informal reviews, formal analysis

Some Preliminaries

- Object (state (variables), behavior (methods))
  - Instance, instance variables, instantiated, encapsulation
- Message - everything an object can do is represented by its message interface
- Class - software blueprint including common elements of objects that need not be repeated
  - Class variables
- Inheritance
  - States and methods, override
- Data containing instances and function containing classes

OO

- OO analysis and design
- European : American :: modeling : reuse
- Class represents what an object is about & relationships
- Relationships: generalization - specialization
- Object is, state + behavior
- What are objects, from requirements: nouns, roles, … The behavior of the objects are verbs

- Difficulty, users think in tasks not objects, case studies bridge this
- CRC - informal
- The UML
- Design Patterns: MVC & wrappers (legacy)
- Metrics really matter
- The great developer and application divide - 90s

OO Analysis and Design

- OO Analysis = Requirements analysis + Domain class selection
  - Product = Complete requirements document + domain class model + basic sequence diagrams
  - Domain classes obtained via use cases -> sequence diagrams and brainstorming/editing process
  - Use domain classes to organize requirements
- OO Design = all other activities except coding
  - Product = complete detailed design ready for coding

Eric Braude, Software Design, John Wiley, 2004.

"Schools" of OO

- European school, influenced by the Scandinavian school of Programming, OO analysis and design is modeling real world objects both animate and inanimate
- American school, OO focuses on data abstraction and component reuse - identifying reusable components and building an inheritance hierarchy.
  - "What matters is not how closely we model today's reality but how extensible and reusable our software is"

OO viewpoints

- Modeling (European) viewpoint - conceptual model of some part of a real or imaginary world.
  - Each object has identity, is unique
  - Objects have substance, properties that hold and can be discovered
  - Objects are implementations of abstract data types
    - Mutable state, variables of abstract data type
    - Operators to modify or inspect the state
      - Only way to access object
      - Interface to object
  - Object = identity + variables + operators or
  - Object = identity + state + behavior

OO Viewpoints - 2

- Philosophical view - objects as existential abstractions, the unifying notion underlying all computation
  - Beginning and end to objects
  - Eternal objects, e.g., integers
    - Not instantiated, cannot be changed
- Software Engineering view - data abstractions encapsulating data and operations
  - Object based languages encapsulates abstract data types in modules whereas
  - Object oriented also includes inheritance

OO Viewpoints - 3

- Implementation viewpoint
  - Continuous structure in memory, a record of data and code elements
- Formal viewpoint
  - Object viewed as a state machine with a finite set of states and a finite set of state functions. State functions map old states and inputs to new states and inputs
- While modeling conceptual viewpoint is stressed
- Tensions between a problem oriented (analysis) vs. solution oriented viewpoint (design)

Objects

- Characterized by a set of attributes or properties
  - Attributes originate from Entity-Relationship Modeling
    - In ERM attributes represent intrinsic properties that do not depend on other entities
    - Shared properties among objects are denoted as relationships
- OO modeling uses attributes to denote any field in the underlying data structure
  - State includes intrinsic and shared properties
    - State includes set of structural attributes and operations = behavioral attributes.
  - Identity is an attribute

More on Objects

- Programming level;
  - Objects having same set of of attributes belong to the same class
  - Individual objects of the class are called instances, when they are created they are instantiated
  - Objects not only encapsulate state but also behavior - the way it is acted upon and acts upon other objects
  - Behavior of an object is described as the services provided by that object
  - Services are invoked by sending messages from the requestor to the object acted upon
  - Client server model of objects, client object requests services from server object, services also are referred to as responsibilities

OO Analysis

- Not considered with instances - concerned with object types, classes
- Major goal- identify set of objects (classes) with their attributes (states) and their services (behavior)

Relations Between Objects

Relationship	Example
Specialization/generalization, isa	Table isa furniture
Whole-part, has	Table has tabletop
Member-of, has	Library has member

More on Relations

- Generalization-specialization can be expressed as a hierarchy
  - Single inheritance, tree
  - Multiple inheritance, directed acyclic graph
  - Define common attributes at a higher level and let descendants inherit the attributes
  - Object hierarchy can be viewed as a type hierarchy, chair and table are subtypes of furniture

More on Relations - 2

- Part of relationship aggregates components into a whole
  - It is a transitive relationship
- Member-of relationship represents the relation of a set and its neighbors
  - It is not transitive

OO Analysis and Design Schemes

- Common notations of the schemes:
  - Class diagram - static depiction of objects as nodes and their relations as edges
  - State diagram - models dynamic behavior of single objects using a variant of a finite state machine representation. Nodes in state diagram represent state of object, edges possible transitions between states
  - Interaction diagrams - model sequence of messages in an interaction among objects
    - Sequence diagrams emphasize time orderings
    - Collaboration diagrams emphasize objects and their relationships relevant to a particular interaction

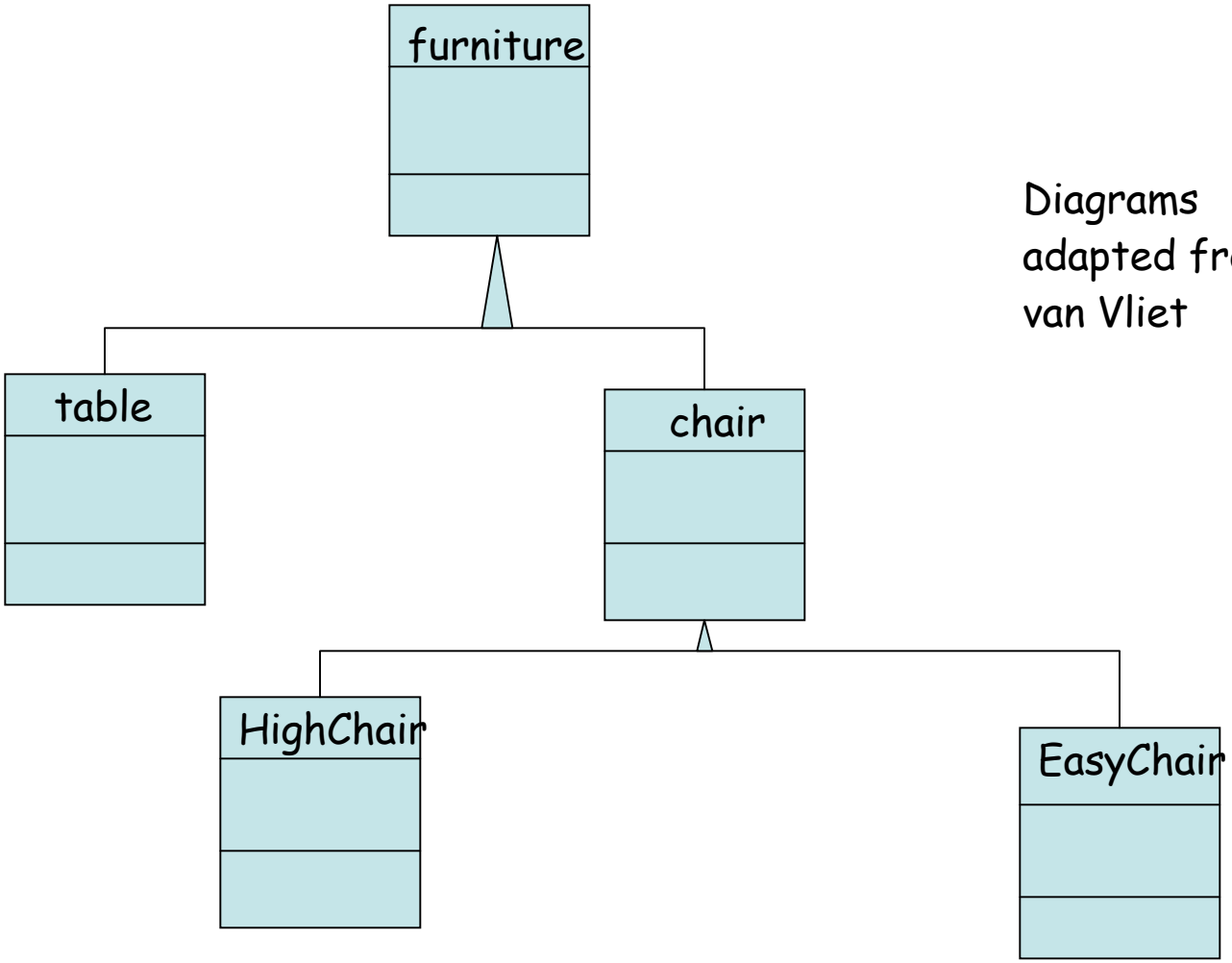
OO Analysis and Design Assumptions

- Assume a stable problem statement
  - Not strong on elicitation (but not weak either)
  - A collection of use cases are one view of the software architecture of a system

The Class Diagram

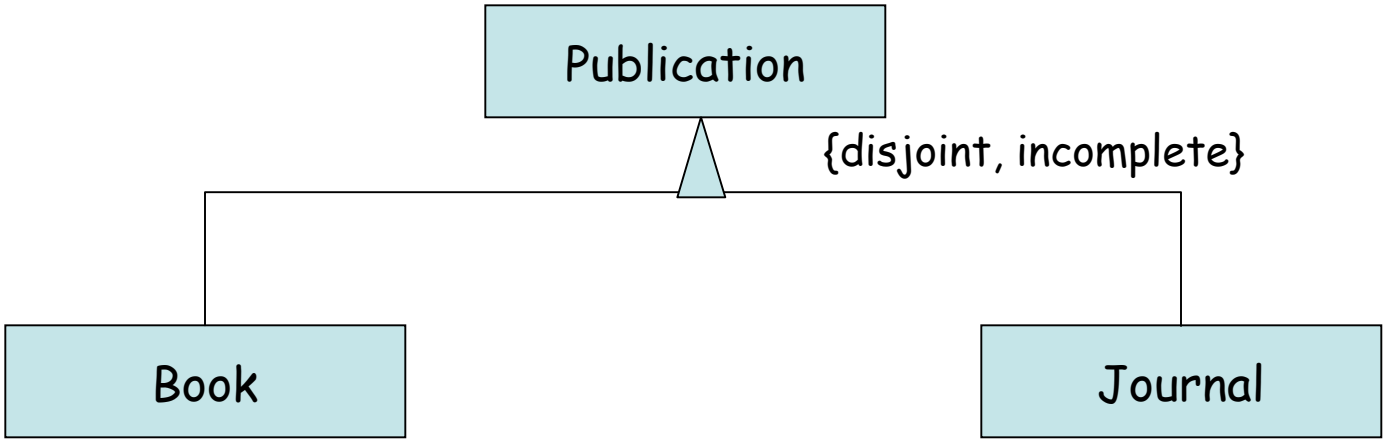
- Shows the static structure which are elements of the model, usually classes, and relationships between them
- Class diagram also represents subclass-superclass hierarchy - a hollow triangle indicates a generalization specialization relationship
- Class boxes have 3 parts: name of object, list of attributes and list of services
- UML permits indicating semantic constraints between classes in braces{}
  - Disjoint constraint indicates that a descendant may not be descended from more than one of the superclasses
  - Overlapping constraints indicate that it may

Object Hierarchy

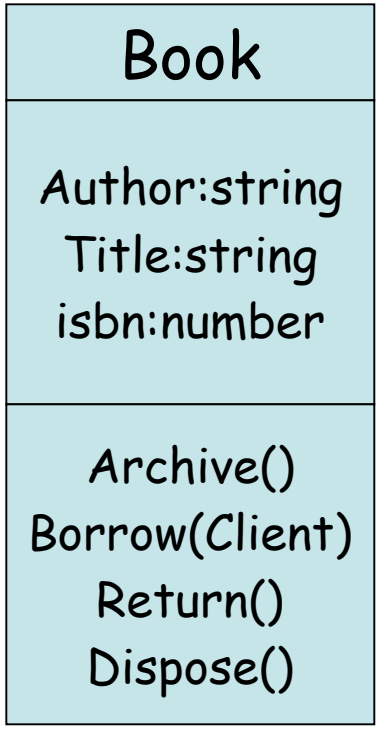


Diagrams adapted from van Vliet

UML Class Diagram



Class Box


Book
Author:string
Title:string
isbn:number
Archive()
Borrow(Client)
Return()
Dispose()

More on Class Diagrams

- UML Class diagram variants can become increasingly elaborate indicating relationships between classes in addition to superclass-subclass
- Can show relationships such as composition, a strong variant of aggregation indicating that a part object may belong to only one whole object and that part is expected to live and die with the whole object

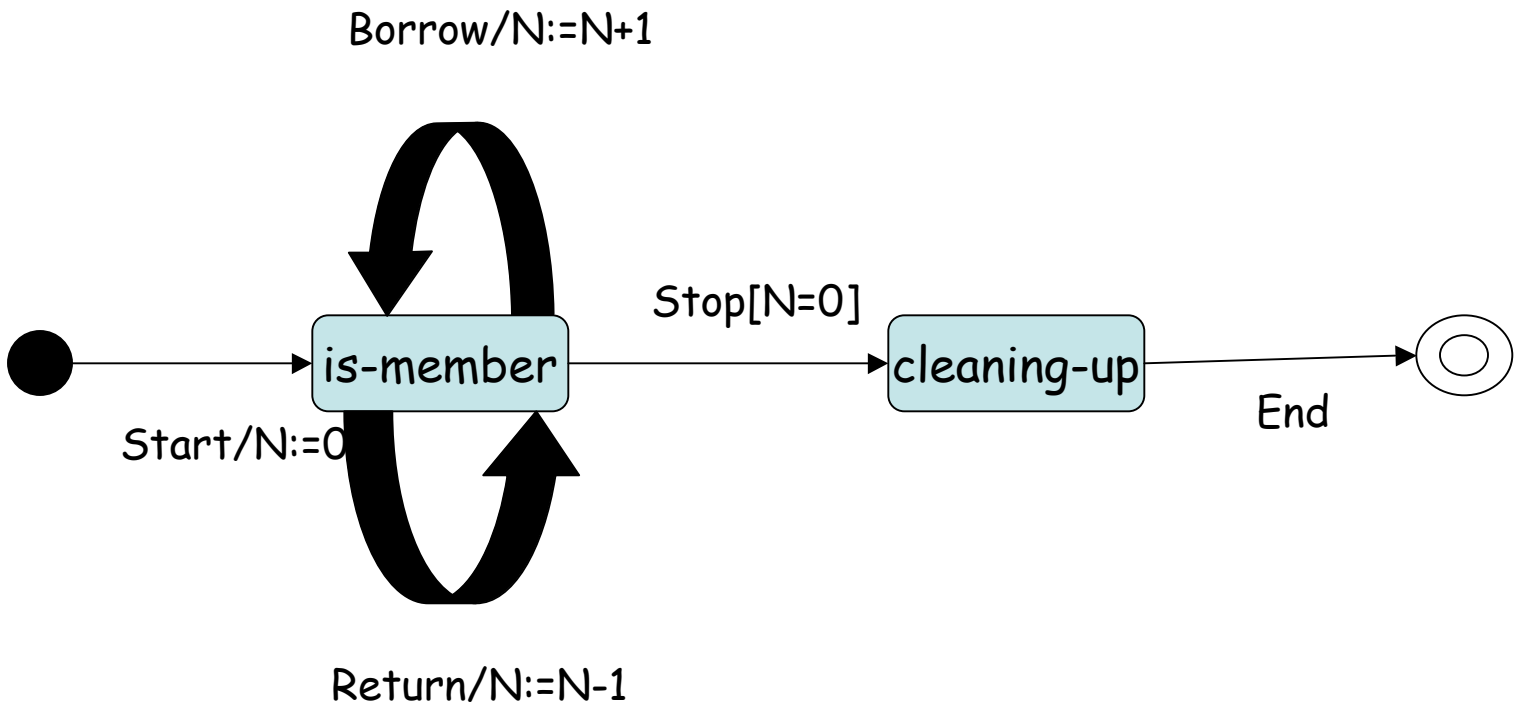
The State Diagram

- Model the life cycle of the object
  - Augment finite state machine with local variables
    - State transitions can change values of the variables
    - Variables can be tested to determine the next state
  - Input events trigger transitions (usually linked to verbs in the requirements)
- Finite state diagrams can become unwieldy
  - Part of the model may be compressed into one state and then expanded
- aka statechart, state diagram is UML terminology

State Diagram -2

- Depict sequence of states an object progresses through
- Known as extended FSMs because they have local variables and output actions may be associated with both transitions and states
- The states can be arranged hierarchically(see Fig 12.8 in book)
- "Statesmanship"
  - A state is some condition in the life of an object
  - A change in state causes a transition to fire
  - Transition labeled by transition string
    - String may include procedural expression after the '/'
  - States can be nested for ease of exposition

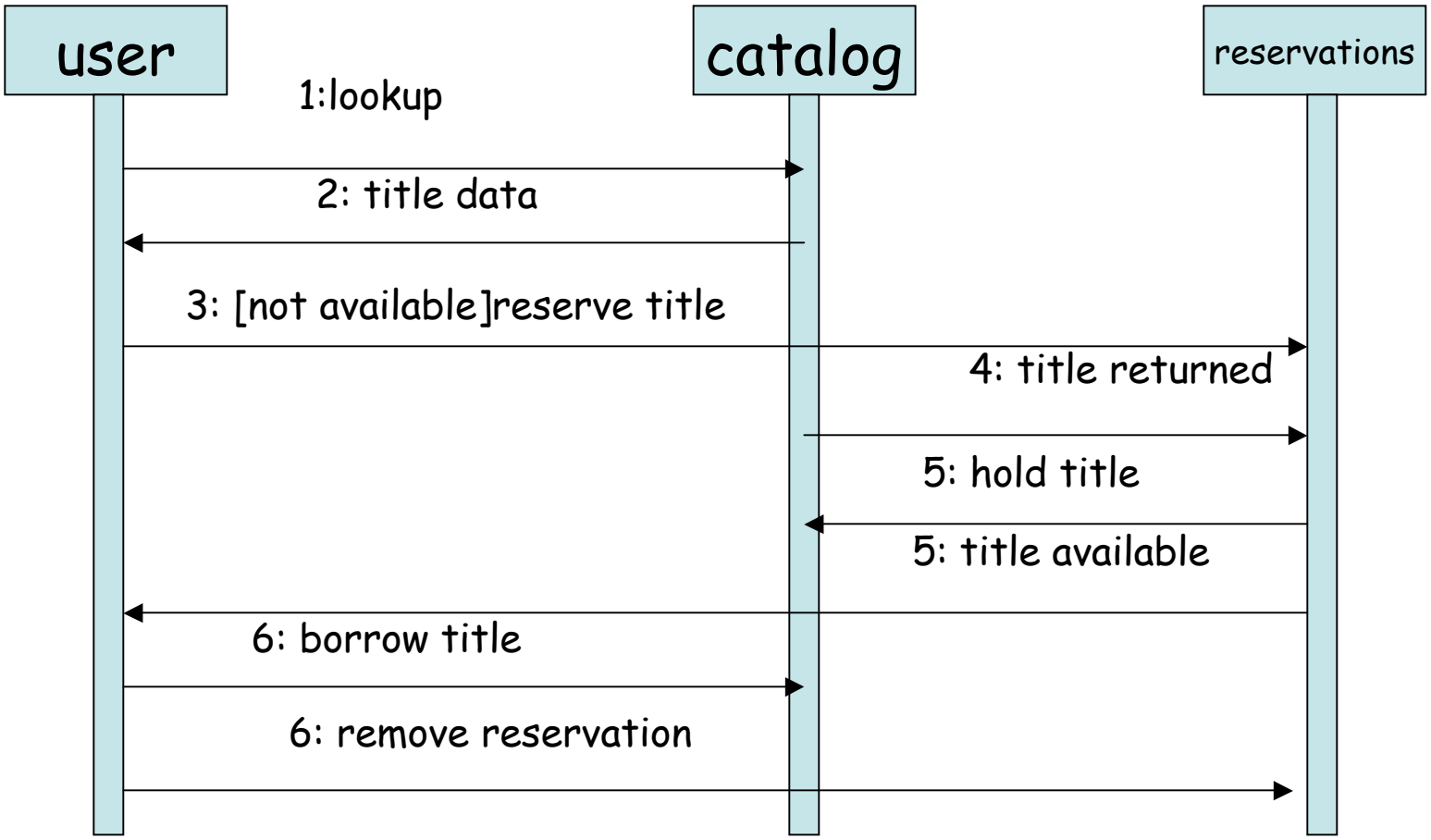
State Diagram



Sequence Diagram

- Shows time ordering of messages
  - aka interaction diagram, event trace diagram
  - Horizontal dimension shows objects that participate in interaction
  - Object is a vertical dashed line, its lifeline
  - Vertical dimension depicts time sequencing
  - Messages are shown as labeled arcs between objects
- Diagrams can become quite elaborate showing asynchronous and synchronous message passing, indicate iteration, …

Sequence Diagram



Collaboration Diagram

- Directed graph - nodes are objects and edges depict communication among objects
- Collaboration diagrams emphasize objects and their relationships relative to a specific interaction
  - Sequence numbers are mandatory, impossible to follow otherwise

Use Case Diagram

- Scenario based analysis we have encountered before
  - Jacobson, "… a scenario that begins with some user of the system initiating some transaction or sequence of interrelated events"
- Multiple ways to document use cases
  - Narrative text
  - State-transition diagrams
- Use case diagram is an overview of a set of use cases with each use case depicted as an ellipse
  - Rectangle indicates system boundary
  - Actor that initiates is shown as stick figure

CRC Cards

- Class Responsibility Collaborator cards
- Documents collaborative decisions - usually done in a group, but useful individually
- Very helpful in early stages of software development
- Nice for small to medium size projects
- One of the techniques to use, very useful - a winner!

CRC Card

Class Name: Superclasses: Subclasses:	
Responsibilities	Collaborators

Analysis and Design Methods

- Approach:
  - Identify the objects
  - Determine attributes and services
  - Determine relationships between objects
- Variant of a class diagram provides static map of objects and their interrelationships
- State chart is a dynamic model in addition to scenario analysis through use cases

Identify the Objects

- Look for important concepts from the application domain
  - Domain specific entities are prime candidates for objects
    - Real world objects - books
    - Roles played - customer
    - Organizational units - department
    - Locations
    - Devices
- Look at existing classifications and assembly (whole, parts relationship)
  - Sometimes listing most of the nouns in the requirements specification or the problem statement
  - Eliminate from the noun list implementation constructs
  - Vague terms replaced by concrete terms or eliminated
  - Eliminate synonymous terms
  - Demote some terms to attributes
- Some information is from statement, some from tacit knowledge
- Diagram starts with relationships and evolves to more detailed descriptions (cardinality constraints and inheritance)

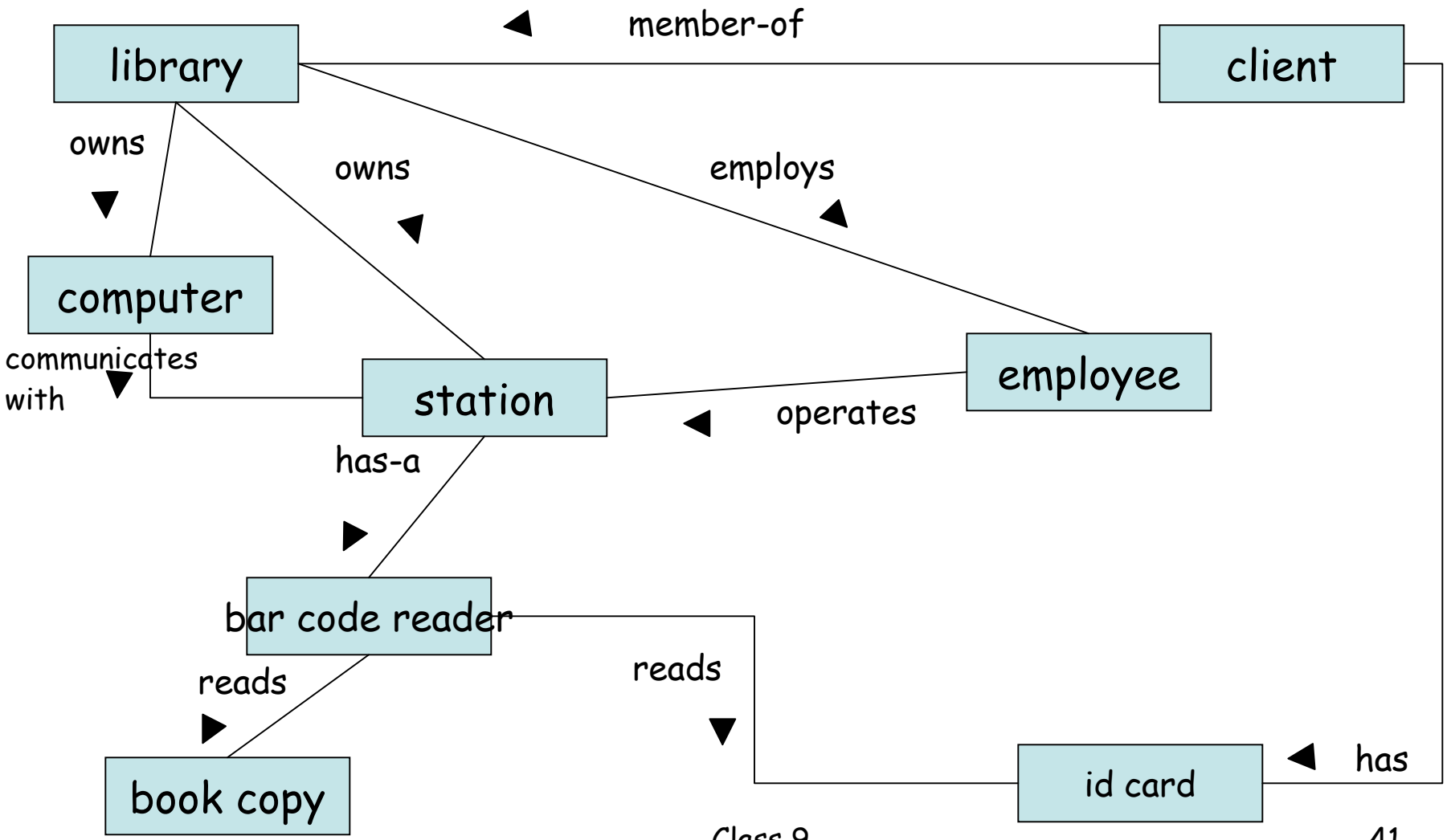
Identify Attributes and Services

- Describe an instance of the object
  - The state of the object
  - Consider characteristics that distinguish instances but are common properties of the objects
    - Look for atomic rather than composite attributes - compute from atomic to composite if necessary
  - Services are related to life cycle and are usually verbs in the description, e.g., book is acquired, borrowed, returned, retired
    - Concern state of the object
    - Usage scenarios aid in discovery

Identify Relationships

- Services are one way objects can be related
  - OO flavor comes from whole-part, gen/spec relationships
  - Consider similarities of objects as basis for specification of a more general object
  - Publication is an abstract object, one that has no instances
    - Attributes and services defined at publication level constitute a common interface for its descendants
  - Generalization/Specialization relationship can lift services to higher levels in the hierarchy and they are represented by virtual functions - services for which a default implementation is provided and can be redefined by specializations

Initial Object Model



Comments on OO Analysis and Design

- Instances of an object should have common attributes if not repartition or reconsider
- Over evolution/time, object hierarchy should remain stable but attributes and services may change
- OO can be considered a middle-out design method!
  - Set of objects constitute the middle design level
- OO could prosper if there were a future where collections of domain specific classes become available - domain libraries - DIFFICULT

RUP

- Rational Unified Process and it is a process framework (customizable) - uses risk management throughout
  - When you start using RUP you need a development case - requires experience
- Product of IBM
- Designed for reuse
- Based on 6 best practices:
  - Develop iteratively
  - Manage requirements
  - Use component architectures
  - Model visually (UML)
  - Continuously verify quality
  - Manage change

RUP-2

- Huge Knowledge Base 30+ books from business modeling to deployment
- Role has these Activities and is responsible for this Artifact
- SPEM - Software Process Engineering Metamodel, http://www.omg.org
- ROSE/XDE for analysis/design, WayPointer (Jacobson, http://www.jaczone.com) for process, ClearCase for configuration management (and more IBM tools tacked on all the time)

RUP's 4 Phases

- Inception - initial evaluation, go/no go to elaboration
- Elaboration - develops primary use cases and software to explore architecture. At end:
  - Good sense of requirements
  - Resolved major risks
  - Skeleton working system on which to base development
- Construction - build it
- Transition - deploy, train users, …

Design Patterns

- Portland Design Repository - http://c2.com/ppr/
- Motivation came from a "real" architect, Christopher Alexander-
  - "Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem in such a way that you can use this solution over a million times without ever doing it the same way twice."
  - Referring to buildings and towns, e.g. couple's realm, children's realm, sleeping to the east, …

Real Architecture Example

- Couple's Realm contains Fireplace
  - Connects to Bed Alcove, Dressing Room, Porch/Balcony, Still Pond, Bathing Room and its Child centers
- Bed Alcove
  - Connects to Couple's Realm Main Area, Bathing Room and child center
  - "after a long day, read a book, do a lot of sketching, sheltering, lots of natural light, …
- http://SimplyBuilding.net

Categories of Patterns

- Creational - abstract instantiation, strive for independence - problem is creating a complex object.
  - Used to control creation of objects, including families of objects. Often replace constructors.
- Structural Patterns - how classes and objects are composed to form larger structures--uses inheritance - problem is representing a complex structure
  - Lists, collections and trees with convenient interfaces
- Behavioral- algorithms and assignment of responsibility, describe objects and communication - problem is representing behavior
  - -application behavior options at runtime
- Pros, more flexibility
- Cons, increased complexity and potential performance issues

OO Metrics

Source	Metric	OO Construct
Traditional	Cyclomatic Complexity	Method
Traditional	Lines of Code	Method
Traditional	Comment Percentage	Method
OO	Weighted Methods per Class (WMC)	Class/Method
OO	Response for a Class (RFC)	Class/Message
OO	Lack of Cohesion of Methods (LCOM)	Class/Cohesion
OO	Coupling between Objects (CBO)	Coupling
OO	Depth of Inheritance Tree(DIT)	Inheritance
OO	Number of Children(NOC)	Inheritance

OO Metrics

- WMC (Weighted Methods per Class) is a measure of size of class, assumes larger classes are less desirable - usually a count of the number of methods
- RFC (Response For a Class) is number of methods in class + number of methods called by each of these class methods where each method is counted once
- LCOM (Lack of Cohesion of Methods)is number of disjoint sets of methods of a class, any 2 methods in the same set share at least one local state variable, preferred value is 0, cohesion metric
- CBO (Coupling Between Objects) is coupling metric, 2 classes are coupled if a method of one class uses a method or state variable of the other class, high values = tight bindings, undesirable
  - Gradations
  - Law of Demeter - methods of a class should only depend on top level structure of own class
- DIT (Depth of Inheritance Tree) is the distance of the class from the root of the tree. Language dependent
  - Strive for inheritance trees of medium height, not narrow and deep, not shallow and broad
- NOC (Number of Children)is number of immediate descendants of a class, large number suggests improper abstraction

"Much has been written about the best way to develop software applications, much of it with a bit of truth, but there is no 'best' way" - BY p111

OO: Hype or Answer?

- OOA and OOD are similar with OOD adding implementation specific classes, OOA should be problem oriented, OOD should be solution oriented
- Transition to OO takes time
- Issues: handling of real time requirements, less mature, measuring progress is hard, no good cost models, scalability and interoperability with non OO systems pose problems
- Traditional functional models are easier to understand by the customer
  - Users do not think in objects they think in tasks - use cases as a way to bridge this
- BUT…

Context

- Testing
- Architecture and Design
- Techniques
- Reliability
- OO
- Last third focuses on how it is stitched together and how it is represented (CHI)

Thought Problems

- You are beginning an OO project, what analysis style would you use, e.g., the UML, *CRC cards,* your choice. Do you think it would depend on the size of the project?
- What so you think are relevant criteria for deciding if you should use OO methodology?

So Far

- Software Process Models, Software Project Planning (woosh!), Requirements, Estimation, Risk Analysis, Multics case study, Architecture Reviews, Questionnaire Design
- Software Quality Assurance
- Configuration Management and Testing
- Architecture and Design
- Software Engineering skills: Problem Solving, meeting, stat, … (and finished Arch and Design)
- This Time: OO
- Next Time: Lightweight Methodologies, XP, Test

Test Review

- Brooks 6-12
- Quality
- Configuration Management
- Testing
- Architecture
- Design
- OO Analysis and Design
- Statistics, problem solving, meeting skills, negotiation
- Bernstein paper

References

- van Vliet
- http://satc.gsfc.nasa.gov/support/STC_APR98/apply_oo/apply_oo.html - OO metrics
- Braude, E.J. Software Engineering: An Object-Oriented Perspective, Wiley, 2001, ISBN: 0-471-32208-3
- http://java.sun.com/docs/books/tutorial/java/concepts/