

Object Modeling

Unified Modeling Language (UML) — Object Modeling

Goals of Chapter

Understanding object concept & identification

Understand Object Modeling with UML

- Models
- Notations
- Guides

Learn basics of creating

- Collaboration Diagrams
- Static–Structure Diagrams

Outline

Object Concept & Identification

Models & Diagrams

Notations & Guidelines

Review of Student Examples

Summary

Why Model Objects?

Why Model Objects? (cont.)

OO emphasizes concept of “**Responsibility-based Collaboration**”

- Modularity of behavior & structure
- Components working together

How is computer or car built?

- Any piece of hardware

Need to understand structure of system

- What is responsibility of each object
- How do objects work together

What is an Object?

An Object is¹:

- Anything that is visible or tangible and is stable in form.

e.g., A particular:

Airplane	Army
Computer	City
Person	Planet

- Anything that may be apprehended intellectually: *objects of thought*.

e.g., A particular:

Flight Software	Navigation System
Operating System	Calendar
Database	Schedule

- A person or thing with reference to the impression made on the mind or the feeling or emotion elicited in an observer: *an object of pity*.
- A thing, person, or matter to which thought or action is directed.

e.g., A particular target

¹ Webster's Encyclopedic Unabridged Dictionary of the English Language, Portland House, 1989.

**I have this box here that will solve your problem.
What do you want to know about it?**

What is an Object? (cont.)

Object–Oriented Definition:

- A self-contained thing which is characterized by:
 - ◆ Unique Identity
 - ◆ Structure
 - ◆ Responsibilities²
 - Its state
 - Its dynamic behavior
 - Operations it can perform
 - Requests it makes of other objects³
 - ◆ Qualities
 - Quantitative (e.g., size, speed)
 - Qualitative (e.g., “user friendly”, re-usable)

² Equivalent to the pre-OO concepts of functionality & dynamic behavior, or the concept capabilities.

³ Each object collaborates (works) with other objects in performance of its responsibilities.

What is an Object? (cont.)

UML Definition (*Glossary*, p B-12):

- *An entity with a well-defined boundary and identity that encapsulates state and behavior*

where:

- ◆ State is represented by
 - Attributes
 - Relations
 - ◆ Behavior is represented by
 - Operations
 - Methods
 - State Machines
- An Object is an instance of a *Class*

What is an Object? (cont.)

An Object may be viewed from:

- External Perspective (Specification/Interface)
- Internal Perspective (Implementation)

An Object may be:

- *Active*
 - Able to act on its own power
 - Owns a thread and can initiate control activity (*Glossary*, p. B-2)
- *Passive*
 - Able to act only when powered by an active object

An Object may be:

Complex

- Composed of 1 or more “simpler” Objects

Atomic

- No component objects

What is an Object? (cont.)

Objects exist for a period of time

Objects may be:

Persistent

- Exist before start of execution of our system and/or
- Exist beyond end of execution of our system

Transitory

- Exist only during execution of our system
- Object is created or destroyed (or both) during execution of our system

Identifying & Designing Objects

Inputs

Abbott's Approach

Guides

Inputs

- System Analysis/Design Model for current build
 - Context
 - Use-case(s)
- Problem description
- Domain knowledge
- Patterns
 - ◆ Larman’s GRASP
 - ◆ *Gang of Four* (“GoF”)
 - Gamma, et al
 - ◆ *Gang of Five* (“GoV”)
 - Siemens Corp
 - Buschmann, et al
 - ◆ others

Abbott's Approach⁴

Steps:

- Underline every **noun, pronoun, and noun phrase** in Problem Description
- Create a list of all “nouns”
- Classify each “noun” as 1 of:

Proper Noun	Unit of Measure
Direct Reference	Mass Noun
Common Noun	Countable Noun
- Classify each “noun” as belonging to problem or solution space
 - Does it have to be represented in solution?
- Assign meaningful legal identifier to each “noun”
 - If 2 “nouns” refer to same thing, use same identifier

⁴ Abbott, R. J. (1983), “Program Design by Informal English Descriptions”, CACM, November, Volume 26, Number 71, pp882-895.

Noun Classification Notes

Class	Definition/Notes	Examples
Common Nouns	Name of class of persons, places, or things Note: Can be used in sentence with "Limiting Modifiers"	Apple, Building, F16, Computer a, an, every, this, that, my (etc.)
Proper Nouns	Name of a specific person, place, or thing	
Direct Reference	Reference to previously identified or known person, place, or thing without necessarily using name	the President, this building, the F16
Mass/Abstract Noun	Name of a quality, activity, or substance	water, traffic, software, cooking, usefulness, programming, completeness
Countable Noun	Name of a thing that can be enumerated	(1) partridge, (2) turtledoves, (50) systems
Unit Of Measure	Means of referring to a quantity	lines (of code), group (of sensors), bushel (of leaves)

Implications of Noun Classification

Proper Nouns ⇒ **Objects**

Direct References ⇒ **Objects**

Common Nouns ⇒ **Class (Type) of Objects**

- Values of type are designators of objects

Countable Nouns ⇒ **Class (Type) of Objects**

- Values of type are integer values

Mass Nouns ⇒ **Class (Type) of Objects**

- Values of type are measure of quality, activity, or substance
- Need to identify **Unit of Measure**

Implications of Noun Classification (cont.)

Note:

- Some nouns can be either
 - Common nouns
 - Countable nouns
 - Mass nouns

e.g. “Sensors” could be common noun or countable noun
- Must look at context to identify appropriate treatment

Identifying Operations

Operations derived from:

- Verbs in informal description
- Attributes of objects
 - Create an operation to retrieve attribute information about object

Identifying Operations from Verbs

Steps:

- Underline every verb, verb phrase, descriptive expression, predicate, & attribute in informal description

- Create list of all “verbs”

- Associate each “verb”, as appropriate, with single type or object

- Classify each “verb” as belonging to problem or solution space
 - Does it have to be represented in solution?

- Assign meaningful legal identifier to each “Verb”
 - If 2 “verbs” refer to same operation, use 1 identifier

Operation Classification Notes

Class	Definition/Notes	Examples
Verb/Verb Phrase	Identify an action performed on object(s) Usually “Constructor/Destructor/Modifier” operation	“Create element” “Add value to list” “Delete object”
Predicate	Designates a property or relation that can be considered to be True or False (to be verb phrases) Usually {query} functional operation	“Is Empty” “Is Full” “Exits”
Descriptive Expression	Characterizes/Identifies a component of an object(s) or a calculation Usually {query} functional operation	“Find the Nth element of the List” “Compute the sum”
Attribute of object	Identifies a property, or characteristic of an object/type Usually {query} functional operation	“Upper Limit” of Sensor “Length” of List

Identifying Attributes

Steps

- Associate objects with classes
 - If no class for object, create new class
 - Do not use language-dependent predefined classes
- Identify adjectives & adjective phrases in informal description
 - Some are values of common & mass nouns (classes)
 - ◆ Sometimes this results in identification of new class
 - e.g. must assume the sensor is broken ... Initially, all sensors are disabled.
disabled & broken ⇒ there exist **states** of sensor (a new class, mass noun)

Identifying Attributes (cont.)

- Adjectives & adjective phrases in informal description define attributes: (cont.)
 - Some are characteristics of objects
 - ◆ e.g. lower and upper limits of a given sensor
⇒ lower limit & upper limit are characteristics of each sensor
 - ◆ Generally, these attributes applies to all objects of the class
- Review RA results for requirements & constraints on objects/classes
 - Determine whether requirement/constraint applies to object or class
 - Be careful to maintain appropriate level of detail
- Assign meaningful legal identifier to each attribute/value
 - if 2 attributes/values refer to same thing, use 1 identifier

Identifying Attributes of Operations

Steps

- Identify adverb & adverb phrases in informal description
 - Gives information about:
 - ◆ Timing relations
 - ◆ Sequence of control
 - ◆ Repetition & number of iterations
 - May result in identification additional objects to maintain info
- Review RA results for requirements & constraints on operations
 - Be careful to maintain appropriate level of detail

Object Decomposition Guide

Design component objects by modularizing structure & behavior of parent (system, component, or object)

- Use both description of parent's context and behavior

Generally > 1 set of objects can be used to implement parent object

- Set of objects implements parent object if:
 - Implements behavior expected of parent
 - Meets at least worst acceptable levels of all critical operational characteristics
- Analyze trade-offs among alternative designs which satisfy above conditions with respect to operational characteristics

How to Identify & Create Parts

Represent “real world” parts of parent object needed for abstraction

e.g. Pen’s cap, inkwell, point

- May not be needed if
 - *Information* about part
 - ◆ Is not set or requested by parent's expected behavior
 - ◆ Can be adequately represented as attribute of parent or other parts
e.g. Pen’s ink level
 - *And behavior* performed by part
 - ◆ Is not required as determined by parent's expected behavior
 - ◆ Can be adequately represented as behavior of parent or other parts
- May still be desirable for supporting later enhancement
 - ◆ At cost of adding complexity
- Parts of passive parent object must be passive
 - If it seems parts should be active, re-evaluate assessment of parent as passive

How to Identify & Create Parts

Maintain attributes that were identified for parent object

– “refinement”

- May not be needed if information is
 - Already maintained by some other component
 - Calculated based on information obtained from other objects
 - ◆ Either component or non-component
- Attribute components are always passive

Active Parents in Particular

Create 1 object to manage/monitor each external object which parent interacts with

- Not needed if
 - This is only responsibility of parent object
 - A part with this responsibility has already been identified
- May combine management/monitor responsibility for multiple externals in 1 object; but generally separate
 - Convenient if managing/monitoring of multiple externals are highly coupled
 - May raise maintenance and enhancement cost
- Most likely an active component

Active Parent Objects (cont.)

Create ≥ 1 objects to decouple objects that

- Need to be responsive to actors (or other components/objects)
 - Analyze operational characteristics specified for each monitor/manager object
 - May defer queuing until time to decide implementation of objects & interactions (i.e. Detailed Design)
- Should not know object they are talking to
- Other design goals?

Create ≥ 1 active objects for periodic & aperiodic behavior not directly related to manager/monitor objects

- Not needed if
 - This is only responsibility of parent object
 - A part with this responsibility has already been identified

Other Relations

Collection–Member / Container–Contents

- Identify sets, groups, queues, stacks, etc. that must be maintained
- Identify physical containers that must be represented
 - e.g.** jar of jelly beans

Collaborations

- Objects that must communicate with each other
 - May not need new relation if Already identified parts or collection–member relation

Outline

Object Concept & Identification

Models & Diagrams

Notations & Guidelines

Review of Student Examples

Summary

Structural Models

Also known as *static* models

Emphasizes the structure of objects in a system, including their classes, interfaces, attributes and relations

Viewed using a combination of

- Static Structure Diagrams
 - Class Diagrams
 - Object Diagrams
- Implementation Diagrams
 - Component Diagrams
 - Deployment Diagrams
- "Basic" Collaboration Diagrams
- Text forms & tables⁵

⁵ Format of these forms and tables is not defined by UML.

Static–Structure Diagrams

Focuses

- On how system is built
 - Things that exist
 - e.g.** classes, types, & objects
 - Their internal structure
 - Their relations
- Not on temporal behavior
 - Contains occurrences of things that have temporal behavior

2 view styles

- *Class Diagrams*
- *Object Diagrams*

Class Diagrams

Is

- Graphic view of static–structural model
- Graph of
 - Static declarative model elements
 - ◆ Classifier elements
 - ◆ Packages
 - ◆ Instances
 - e.g.** objects & links
 - Connected by their various static relations

Notation Guide says

- “Perhaps a better name would be ‘static structural diagram’ but ‘class diagram’ is shorter & well established” [UML p259]
 - Seems to contradict goals of “ready to use” & well–defined semantics

Class Diagrams (cont.)

Individual class diagrams do not divide static–structure model

- Separate diagrams are for graphical convenience

Class Diagrams may be organized into packages

- Possible organizations:
 - Single hierarchy of packages
 - ◆ Group related classes in package
 - ◆ Diagram(s) in package show definition of class(es) in that package
 - e.g. Each diagram show attributes, operations & relations for classes defined in package
 - Separate hierarchies of packages
 - ◆ Classes in 1 hierarchy
 - ◆ Diagrams in another hierarchy

Object Diagrams

Is

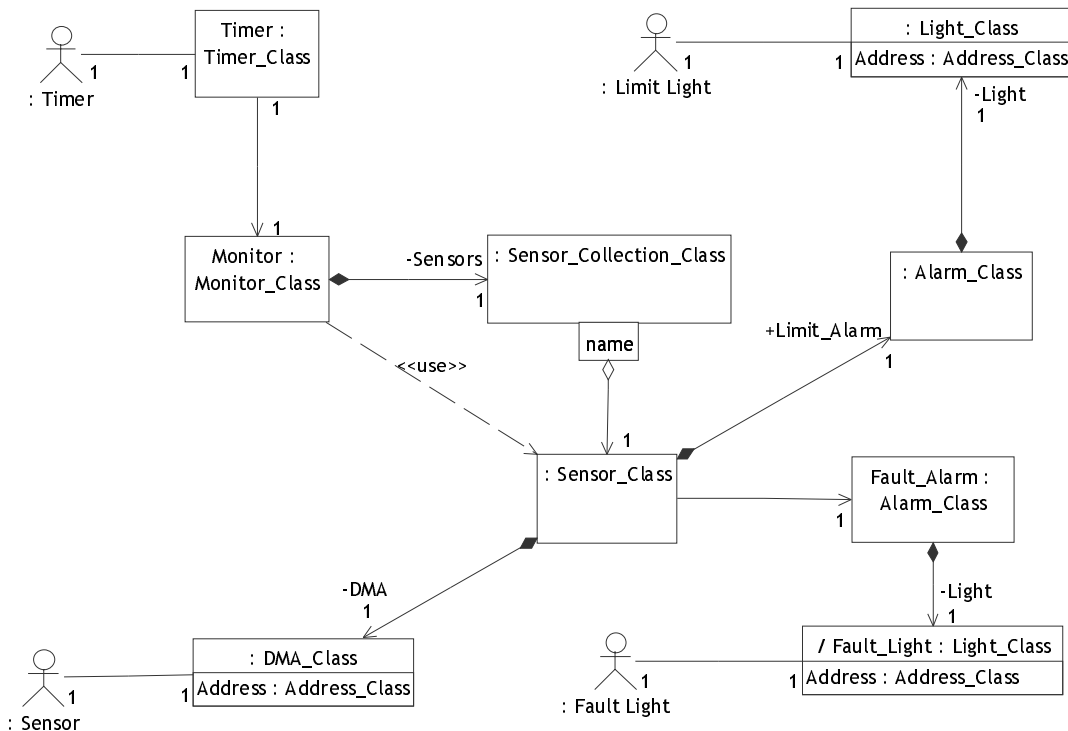
- Graph of instances, including objects and data values
 - Shows instances of Classes
 - Their specific attributes & relations
- Instance of a class diagram
 - Shows a snapshot of the detailed state of a system at a point in time
- “Class diagram with objects and no classes” [UML p260]
 - Should say “static-structure diagram with objects and no classes”

Mainly used to show examples of data structures

- Often before & after an operation or use-case

Tools need not support a separate format for object diagrams

Object Diagrams Example



Collaboration Diagrams

Shows

- Interaction organized around
 - Participating Objects/Roles
 - Links among objects/roles
 - Not shown in Sequence Diagram

- Sequence & concurrent threads of messages
 - Must be determined from sequence numbers
 - More obvious in Sequence Diagram
 - ◆ Time is separate dimension in Sequence Diagrams

Collaboration Diagrams (cont.)

Forms

Collaboration Diagram

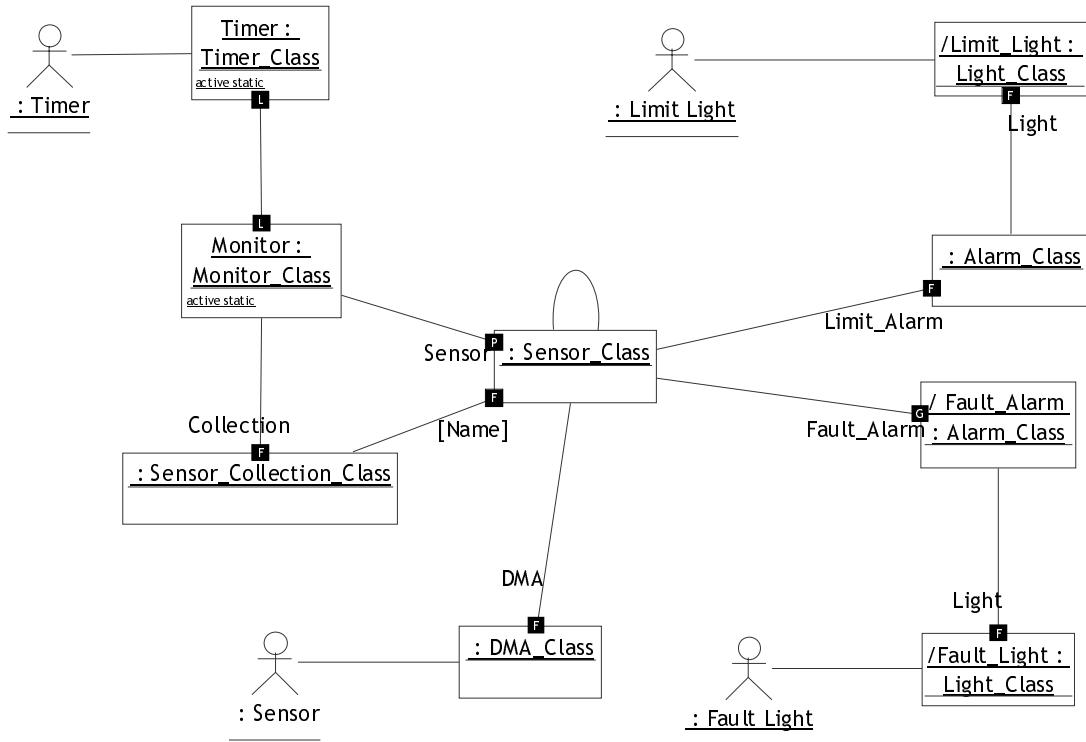
- Structural description of its participants without messages
 - Shows roles of objects & their relations
- Context in which interaction occurs

Interaction Diagram

- Pattern of messages exchanged among objects to accomplish a specific purpose
 - A *collaboration* with messages
 - ◆ Each set of messages applied to a collaboration results another interaction

Collaboration Diagrams

Example



Outline

Object Concept & Identification

Models & Diagrams

Notations & Guidelines

Review of Student Examples

Summary

Notations & Guidelines

"Basic" Collaboration Diagrams

Object Diagrams

Classifier Roles & Objects

Instances of classifiers that participate in collaboration

- Instance of class that is focus of diagram
 - When describing
 - ◆ Implementation of operation (“*method*”)
 - ◆ Class context
 - When describing a use–case, no one object is the focus
- Collaborating objects
 - Including
 - ◆ Instances of associated classes
 - ◆ Parameters (Arguments)
 - ◆ Local or Global Variables
 - ◆ Associations

Classifier Roles & Objects (cont.)

- Collaborating objects (cont.)
 - Not including attributes
 - ◆ If need to show messages to attribute, then
 - Don't use attribute notation in class diagram
 - Model attribute with composition
 - ✦ From attributed class
 - ✦ To attribute class

Collaboration objects may exist

- Prior to start of collaboration
- After end of collaboration
- Only during collaboration

Classifier Roles & Objects Representation

Shown as

- Class rectangle
- With
 - At least 1 of
 - ◆ Object name
 - ◆ Class role name
 - ◆ Classifier name
 - Classifier Stereotype (optional)
 - Multiplicity indicator (“*”) in upper right-hand corner (optional)

*

Object Name / Class Role Name :
Classifier Name [,Classifier Name]
{property list}

Representation (cont.)

- Property list (optional)
 - ◆ Concurrency
 - active
 - ◆ Life-time
 - new If created during execution of collaboration
 - destroyed If destroyed during execution of collaboration
 - transient If created & destroyed during execution of collaboration

In Specification– & Instance–Level Diagrams

Each object plays 1 or more *role(s)* within collaboration

- Each role represents type of an object that will play role
 - e.g.** Subset of all interfaces & relations required for collaboration
- Each role may be
 - Relative to class, operation, use–case realized by collaboration
 - ◆ Called *classifier role*
 - Relative to particular relation with associated object
 - ◆ Called *association role*

Specification–Level Diagrams

- Describes roles that will be filled
- Only classifier role in class rectangle

Instance–Level Diagrams

- Actual objects are bound to roles
- Include names of objects

Active Objects

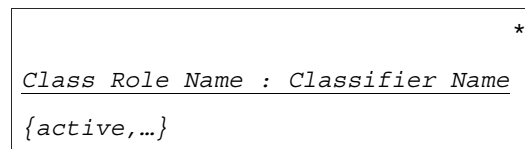
Encompass their own thread of control

- Can change their state spontaneously
- Act **concurrent** to other active objects
- All forms of message synchronization with other objects are allowed

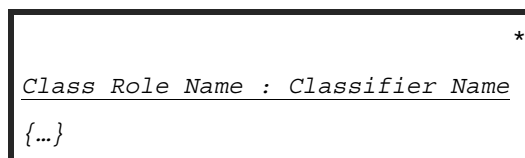
Representation

- Standard icon with either:

– {active} property



– Thick border



Multiobject

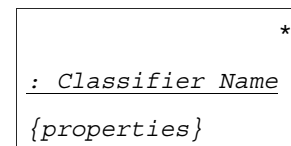
Set of instances of same class (superclass)

- On the many end of an association

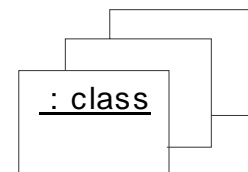
Representation either

- Standard icon with

– Multiplicity indicator (“*”) in upper right-hand corner



- Triple-Stacked Standard icon



Links & Association Roles

A *link* is

- A relation between instances of classes
- Instance of a class relation
 - Associations
 - Aggregation
 - Dependency
- A communication path between objects
 - Links may be directional
 - i.e.** Indicates 1-way navigability

Also called an *association role*

Representation

Shown by

- Path connecting instances of classes
- Name of Association (optional)
 - Underlined
- Arrowhead on link if directional
 - Points from object that has visibility to other object
- Role descriptions for participants (optional)
 - Name of role
 - Constraints
 - Multiplicity
- Qualifiers (optional)

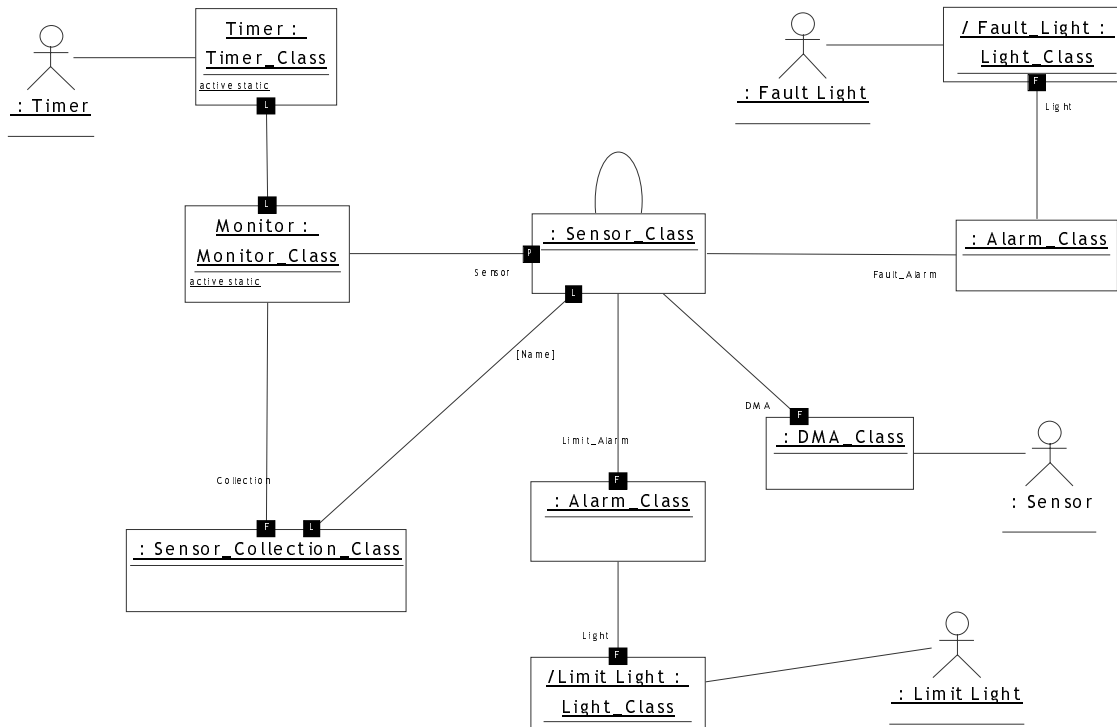
Stereotypes⁶

Specify how instance on one end is known to instance on other end

Name	Indicates instance of class is visible to other instance
Association	Classes are related by association (default)
Global	Instance is defined in global scope relative to other object
Local	Instance is local variable of method i.e., composition relation between classes or local variable in operation
Parameter	Instance is method parameter (inc. return result)
Self	Instance is both client & server

⁶ Rose uses first letter of constraint in box attached to end of link. If box is hollow, then instance can be "shared" between objects. Rose doesn't support Association constraint. It divides UML's Local constraint into "F" (for "Field") for a component of the class and "L" for local variables; but you don't have to use the "F" option. A Self constraint is shown by special looped link.

Example



Notations & Guidelines

"Basic" Collaboration Diagrams

Object Diagrams

Object

Represents particular instance of class

- Has identity & attribute values

Notation:

- Rectangle with 2 compartments

– Top compartment

- ◆ Name of object, role, & its class

➤ Underlined

➤ Syntax:

objectname / rolename : classname

– Bottom compartment (optional)

- ◆ List of attributes & their values

➤ Syntax:

attributename : type = value

➤ “: type” is optional

```
Triangle: Polygon Class
```

```
Center = (0,0)
```

```
Vertices = ((0,0), (4,0), (4,3))
```

```
Border_Color = black
```

```
Fill_Color = white
```

Object (Cont.)

- Object name may be omitted
 - Must have “: *classname*”
 - Represents anonymous object of specified class
 - ◆ Object’s identity given by role in relations
- “: *classname*” may be suppressed
- Class name can be prefixed by pathname of enclosing package separated by double colons
 - e.g.** Triangle : Graphics::Shapes::Polygon_Class
- May list multiple classes separated by “,”
 - Only one implementation class
 - Multiple types permitted

Object (cont.)

- May show object’s stereotype
 - Stereotype for object must match stereotype for its class
- Can show object is in particular state
 - Syntax:
 - objectname* : *classname* '[' *statename-list* ']
 - List is comma-separated names of concurrent states

Same notation also represents role within collaboration

- Roles have instance-like characteristics

Composite Object

Represents high-level object made of tightly-bound parts

Is instance of composite class

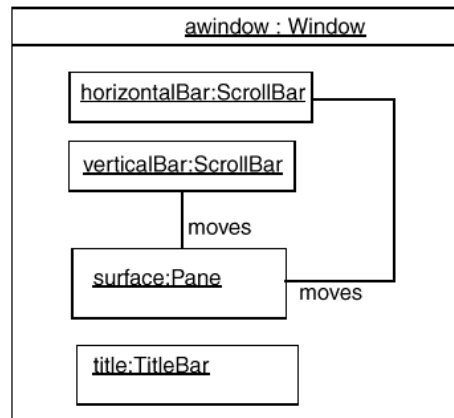
- Implies composition between class & its parts

Is similar to collaboration

- But, defined completely by composition in static model

Notation

- Object symbol
 - Top compartment
 - ◆ Name of composite object & class
 - Bottom compartment
 - ◆ Object parts
 - ◆ No list of attributes



Relationship

UML Definition:⁷

- Semantic connection among model elements

Static–Model relationships are:

- Association
 - Aggregation
 - ◆ Composition
- Dependency
- Instance
- Link

⁷ From an English Language perspective, it would be better if UML used the word relation.

Dependency

Definition

- Relationship between 2 (or more) modeling elements in which change to one modeling element will affect other modeling element
 - Independent element(s) call “supplier(s)”
 - Dependent element(s) called “client(s)”
- Implementation or functioning of one or more elements requires other

Typically

- “Using” relation
- No direct, permanent, physical connection

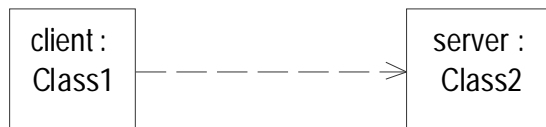
Examples:

- *Person uses Machine*
- *Baker uses Batter*
- *Package_A accesses Package_B's classes*

Dependency (cont.)

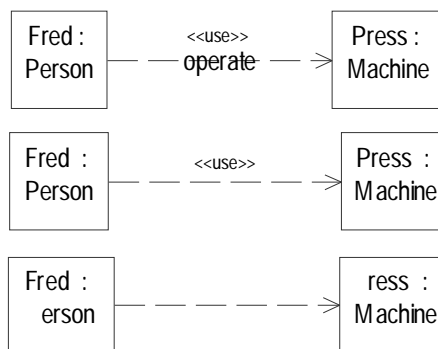
Representation

- Dashed arrow between 2+ modeling elements
- Optional stereotype
- Optional name



Examples:

- *Fred uses the Machine Press*



Stereotypes

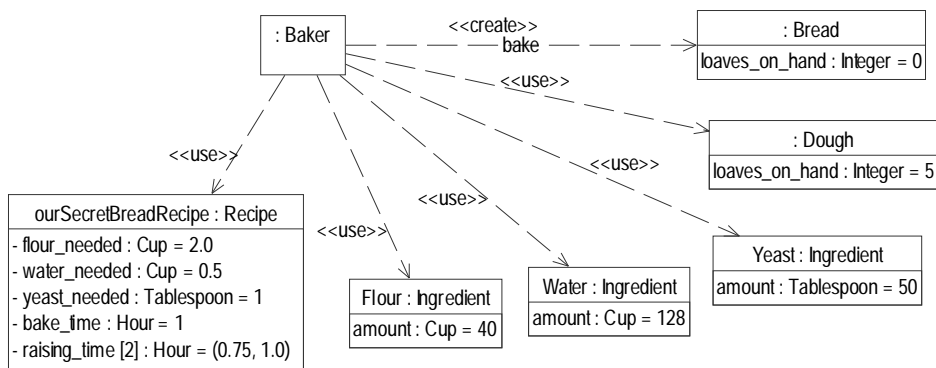
Stereotype	Description
access	Grants "access" permission for "client" package to reference public elements of other "server" package
derive	Element is computable from other element(s)
import	Grants "import" permission for "client" package to reference public elements of other "server" package Public elements of supplier added to client package
refine	Historical connection between elements with mapping (possibly incomplete) between them Mapping can be described in attached note More specific stereotypes have been proposed for different kinds of refinement
trace	Historical connection between elements that represent same elements at different levels of abstraction
use call, create, instantiate, send	Client requires presence of server for its correct implementation More specific stereotypes describe specific kinds of usage
become ⁸	A "flow" relation where source & target represent same instance at different points in time, possibly with new values for attributes, different state, different roles
copy ⁸	A "flow" relation where source & target are different instances with same values for attributes, state, & roles; but distinct identities

⁸ Generally only used in Activity or Implementation diagrams.

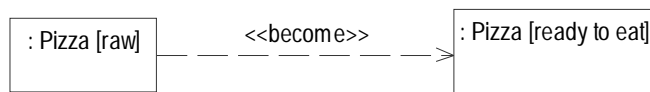
Examples

Our Baker *uses* our Secret Recipe to make the Dough from Flour, Water, & Yeast

Our Baker *uses* Dough & Recipe to bake Bread



An uncooked pizza *becomes* ready to eat once baked



Link & Association

Definitions

- A *link* is
 - Physical or conceptual connection between instances
 - Instance of an *association*
- An *association* is
 - Semantic relation between two or more classifiers
 - ◆ Defines relation among instances of classifiers
 - ◆ Describes set of potential links
 - In same way that a class describes set of potential objects

Examples:

- Association
 - Instances of class *Person* *Works-for* instances of class *Company*
- Link
 - *Fred Works-for IBM*

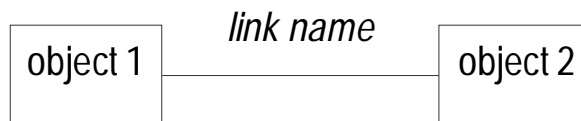
Binary Link

Most links are binary relations (involve 2 objects)

- But may involve more than 2 objects (called “*N*-ary links” — see below)

Representation

- Path connecting 2 objects
- Link name may be omitted if obvious
- Direction to read link is determined by direction of link
 - Can show direction arrow (a triangle) next to name



Example

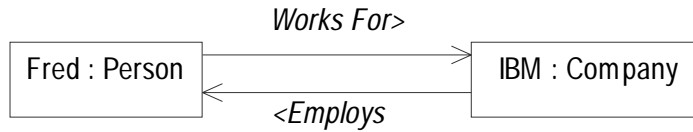


Binary Link (cont.)

1 association often represents 2 relations that are complementary

e.g.

- Fred works for IBM
- IBM employs Bob



When name association pick name of one relation⁹

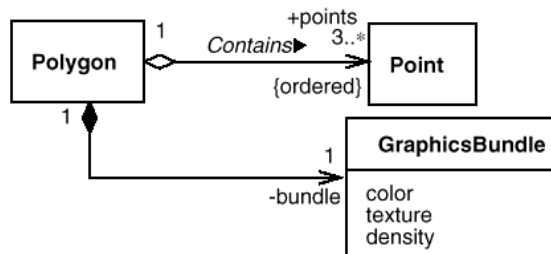


⁹ We'll see later how we can represent only 1 of the 2 relations.

Binary Link (cont.)

Ends of link is where most of interesting information is kept

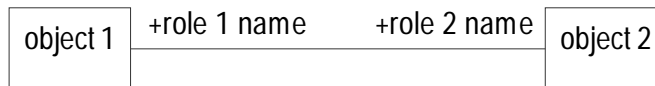
- Multiplicity
 - Not common for links
- Ordering
- Qualifier
- Navigability
- Aggregation Indicator
- Role Name
- Interface Specifier
- Changeability Property
- Visibility



Roles

Each object involved in association is said to play a *role*

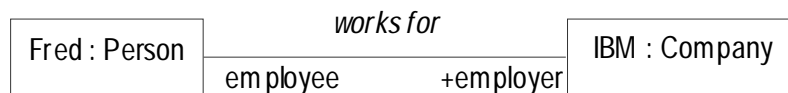
Representation



- Labeled with role played placed on end of link
 - Can omitted if role is obvious

Example

- Fred in *Works-For* association with IBM is an *Employee* of the IBM & the IBM is the *Employer*



Multiplicity

Association may relate instance of class to multiple instances of other class

- *Multiplicity* defines how many instances of class are related
- If many instances of class, then relation may be *{ordered}* and *{sorted}*

Notation

- Comma-separated sequence of integer intervals

- Interval represents range of integers

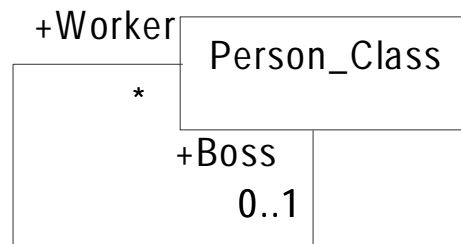
Integer value

lower-bound .. upper-bound

- Star character (*) may be used for upper-bound or integer value

- ♦ Denoting unlimited upper bound

- Each bound can be expression that evaluates to integer value

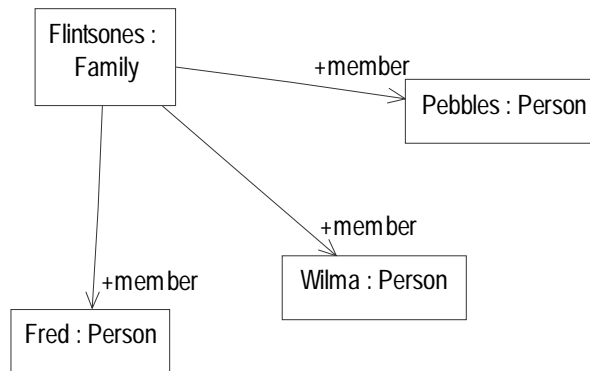


Link Multiplicity

"Multiplicity" is uncommon for link end

- Multiplicity commonly shown as "N-ary link" or separate links

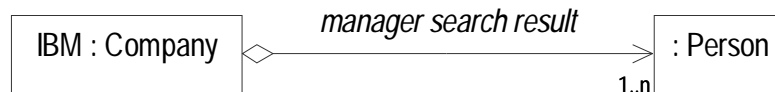
– Example



- Sometimes useful for sets

– Especially if dynamic

– Example



Aggregation

Definition

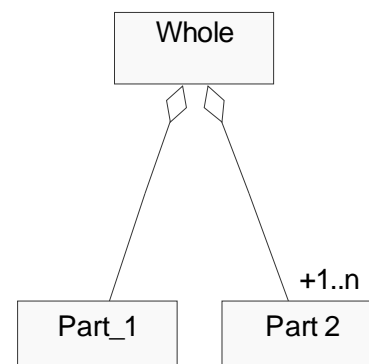
- Object is “part-of” one another object
 - “Part” object play role of *component*
 - Other object plays role of *assembly*
- Also known as “whole–part” relation

Representation

- Diamond on “whole” end of link

2 forms of aggregation

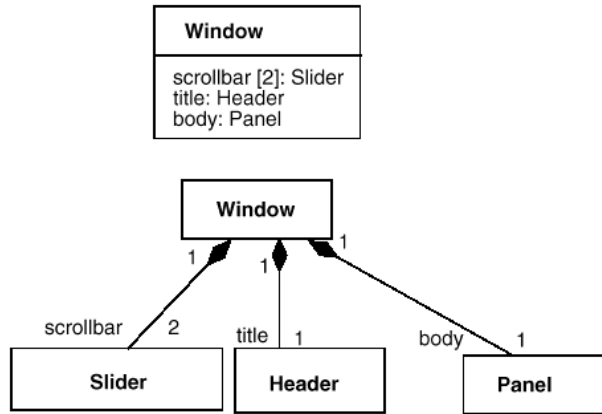
- *Composition*
- *Ordinary*



Composition

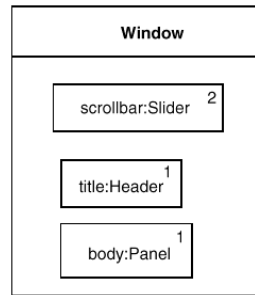
“Ownership”

- Lifetime of instance part coincident with composite instance
- “Whole” is responsible for creation & destruction of parts
- Part owned by 1 composite instance at time
 - Can be part of different composites @ different times



Implies propagation semantics

- i.e.** some of behavior of whole is propagated to its parts
- e.g.** if whole is copied or terminated then so are parts

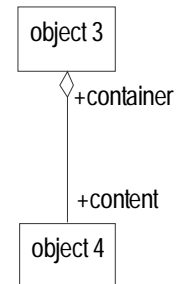


True “whole–part” relation

Ordinary Aggregation

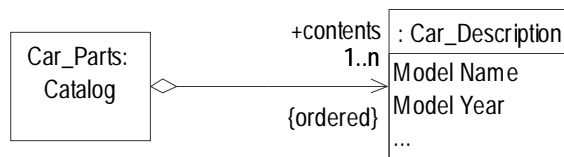
“Weak ownership”

- “Part” may be included in several aggregates
- Its container may change over time
- Does not imply propagation semantics
- e.g.** parts aren’t terminated when one of its aggregates is terminated



Represents

- “Container/ Content”
- “Collection Member”



Navigability

Each end of association or link can be shown to be navigable

Notation:

- Arrowhead pointing to class



Presentation Options:

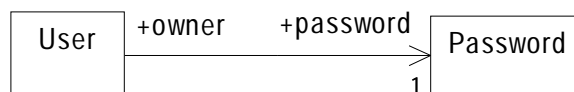
- Show all arrows
 - No arrows => no navigation
- Suppress all arrows
 - No inference about navigability
- Suppress arrows when both directions
 - Show arrow when 1 direction
 - Can't distinguish both directions from no navigation
 - ◆ But, no navigation is rare

Navigability (cont.)

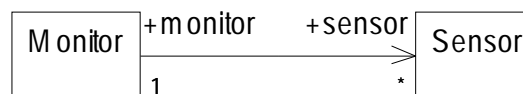
Must indicate navigability when want to limit navigation

Examples

- User has a Password
 - Don't want to find user that owns password



- Monitor of Sensors
 - Sensors don't know Monitor



- Committee has Members
 - Should be able to navigate either way



Association End Properties

{ changeable }

- No restrictions on changing link
- Default

{ addonly }

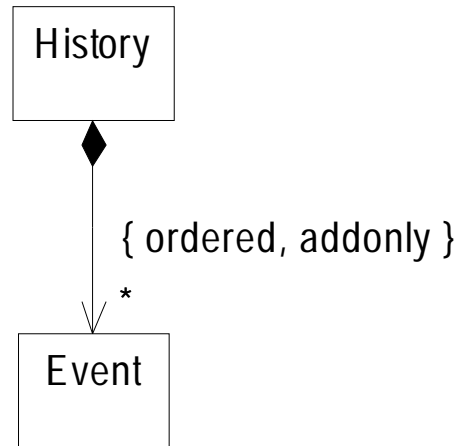
- Only applies if multiplicity > 1
- Additional links may be added but not removed or changed

{ frozen }

- Can't change link after initialized

{ ordered }

- Associated objects are ordered



Association/Link Qualifiers

Qualifier is

- Attribute or list of attributes
 - Owned by association
 - Values partition associated objects at many end of association into sets
 - ◆ Usually reduces multiplicity to 1–1 or 0–1 relation
 - i.e.** Each value uniquely identifies 1 associated object
 - ◆ Sometimes still 1–many relation
 - i.e.** Each value identifies a set of associated object

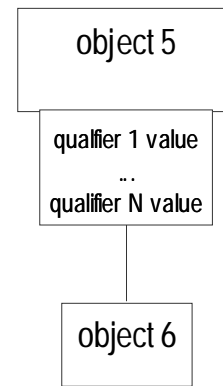
Examples

- File name distinguishes files in a directory
- Employee number distinguishes *employee* of Company
- Position qualifies Players on a Team
- Ticker Symbol distinguishes Companies listed on Stock Exchange

Link Qualifiers (cont.)

Representation

- Rectangle attached to object at end of link
 - Source end of link
 - Each end of link
 - ◆ Rare
- Label
 - Value of qualifier attribute



Example



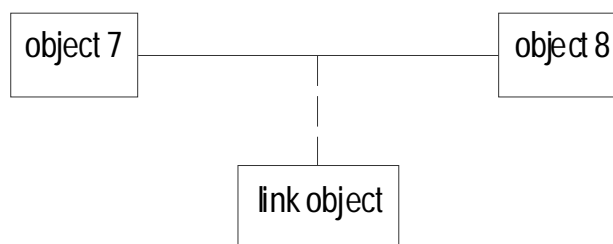
Link Object

Is

- An link that also has object features
 - i.e.** attributes, & links
- Or object that has link properties

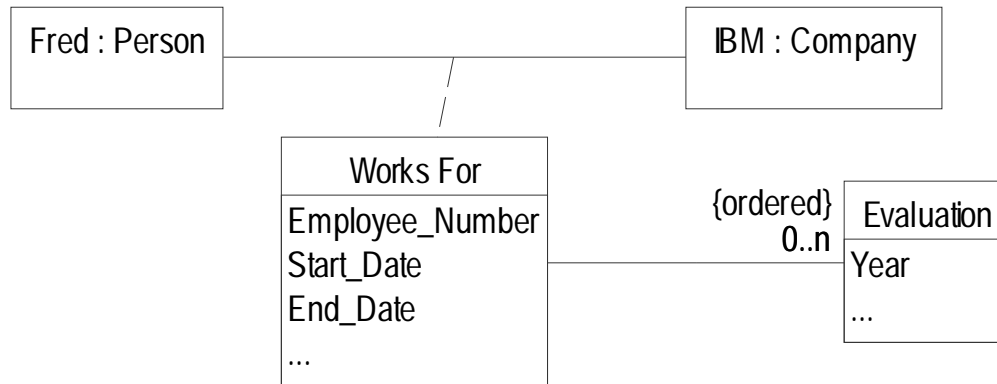
Representation:

- Object attached to link via dashed line
- Object name & association name should be identical
- Single model element even though drawn as association & class



Association Class (cont.)**Example**

- Fred *Works For* IBM from start date to end date and has yearly evaluations.

**Outline**

Object Concept & Identification

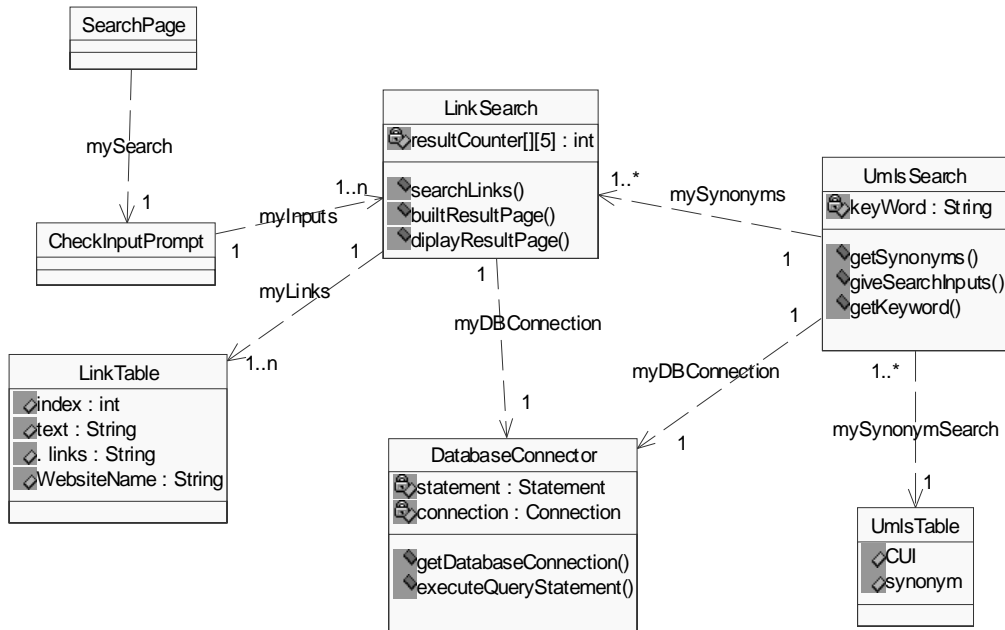
Models & Diagrams

Notations & Guidelines

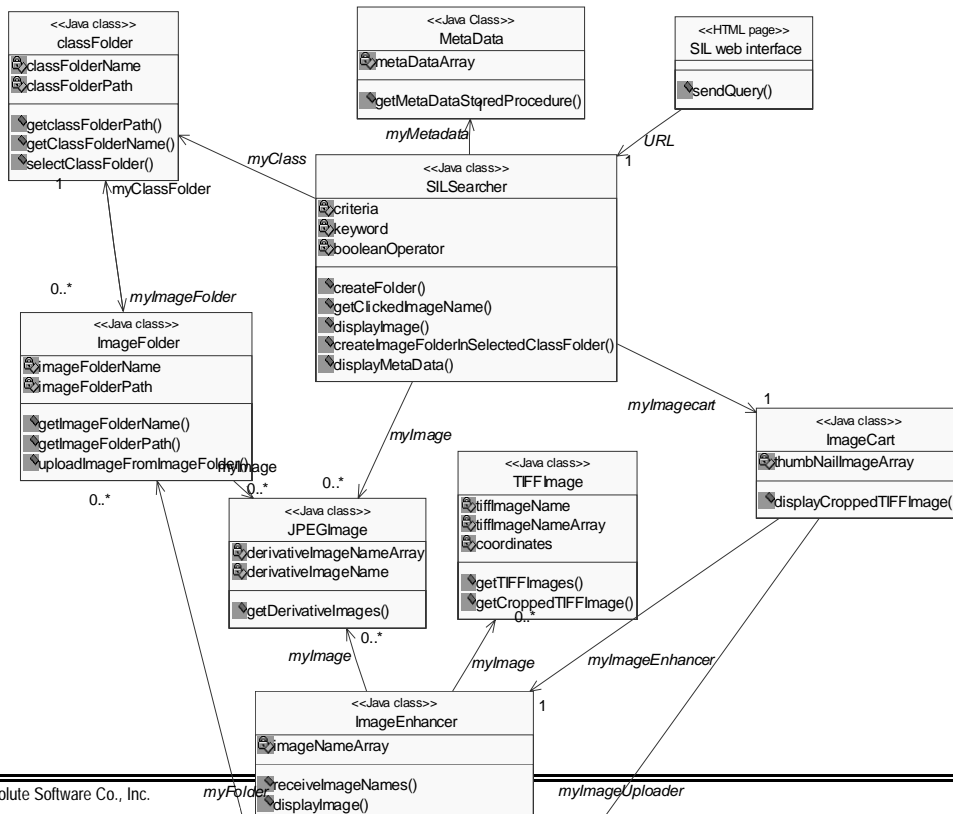
Review of Student Examples

Summary

Team 3 SSAD Figure 16: User Subsystem



Team 17 SSAD Figure 22 (Part 1)



Outline

Object Concept & Identification

Models & Diagrams

Notations & Guidelines

Review of Student Examples

Summary

Final Observations

Object modeling is critical for efficient development of system

UML defines multiple notations for describing static structure of system

- Emphasize slightly different information
 - Use one the most effectively conveys the information you need to communicate
 - Tool support may drive certain selections
- Usable at both different levels of abstraction
 - Must clearly delineated

Goal is to precisely represent your abstraction so to facilitate

- Communication
- Analysis