

Unified Modelling Language



DIAGRAMS
class diagrams

Our goal:

- Build a system that satisfies our current requirements (asap)

Build a system that will be easy to maintain and adapt to future requirements

UML Diagrams

- Graphical representation of a set of objects
- used to visualize the system
- no complex system can be understood in its entirety, diagrams permit focus on aspects of the system
- make the **system** approachable

Modeling static parts of the system: structural diagrams



- Class diagram: static design view
- object diagram: set of objects and their relationships. Used to illustrate data structures. Static snapshot of the system.
- component diagram: set of components and their relationships.
- deployment diagram: view of the architecture

Modeling dynamic parts of system: behavior diagrams

- Use case diagram
 - sequence diagram: time ordering of messages
 - collaboration diagram: emp structural organization
 - statechart diagram
 - activity diagram: flow
- Interaction diagrams**



Class Diagrams

attributes & operations

Class Diagram

- picture of the classes, and the associations between the classes, in the system

7

What are classes

- A class describes a set of objects with same behavior
- tangible things: *book, copy, course*
- roles: *library member, student*
- events: *request to borrow a book*

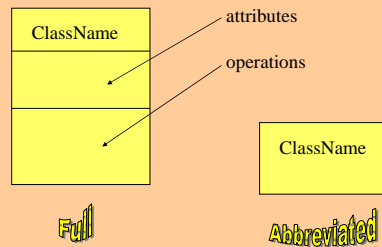
8

Noun identification

- technique to identify classes in our system
- underline all nouns and noun phrases
- discard candidates that are inappropriate:
 - redundant
 - vague: might be a problem in the use case
 - an event without state, behavior or identity
 - outside the scope of the system
 - something simple with no interesting behavior

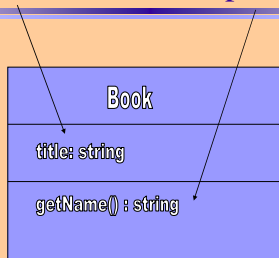
9

class symbol



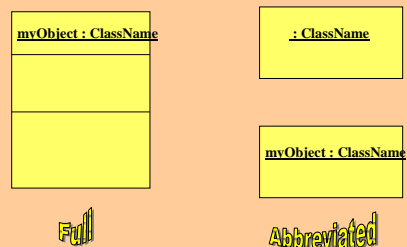
10

data attributes and operations



11

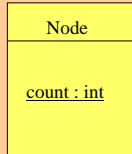
object symbol



12

Owner scope (static)

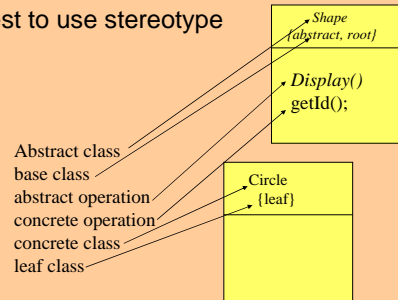
- Specifies whether the feature appears in each instance of the classifier or in all instances
- modeled by underlining



13

Abstract, root, leaf classes

- easiest to use stereotype



14

Interface

- A collection of operations that are used to specify a service of a class or component
- state the desired behavior of an abstraction independent of an implementation of the abstraction
- Supported by Java and Corba IDL
- Similar to ABC pattern

15

Interface (cont)

- Every interface must have a unique name; can be simple or path; can be modeled by a circle with a name:



ISpell

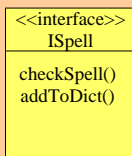


Network::Router

16

Interface (cont)

- May not include any attributes
- Can be rendered as a stereotyped class

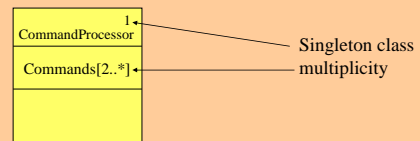


Abstract classes may have data attributes but Interfaces may not; thus, they are UML different

17

Multiplicity

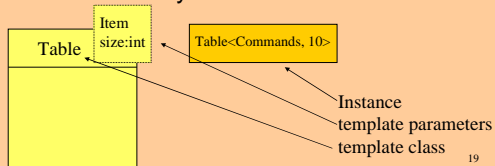
- Can restrict class instances



18

Template classes

- Parameterized element in C++ and Ada: slots for **classes**, **objects** and **values**
- each template defines a family of classes
- most commonly used for containers



19

Accessibility (visibility)

- prefix the name with:
 - + means public
 - - means private
 - # means protected
- this notation applies to classes, packages, subsystems

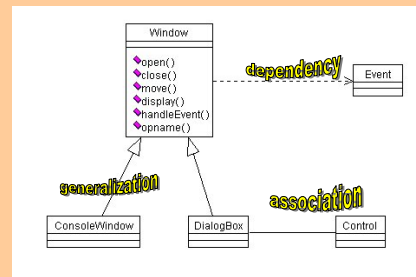
20

Relationships

- Connections between classes
- 4 important OO relationships:
 - dependency $\cdots\rightarrow$
 - generalization \longrightarrow
 - association \longrightarrow
 - realization \longrightarrow

21

Example of 3 relationships



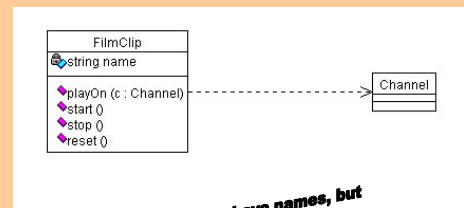
22

dependency

- change in one class may effect another class that uses it; but not necessarily in reverse
- rendered as a dashed line directed to the thing being depended upon
- indicates that one class uses another class as an argument in the signature of an operation

23

Dependency



Dependencies can have names, but it's more common to use stereotypes

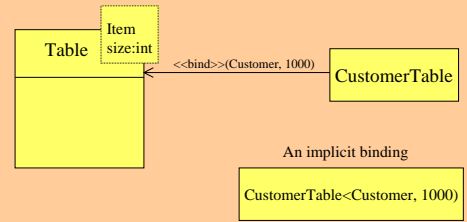
24

Stereotypes

- Some stereotypes for dependency relationships:
 - (1) **bind**: source instantiates the target **template** using the given parameters
 - (2) **friend**: source has special visibility granted by the target
 - (3) **derive**: source may be computed from target
 - (4) **instanceOf**: source is instance of target
 - (5) **instantiate**: source creates instances of target
 - (6) **use**: semantics of the source depends on the semantics of the public part of the target

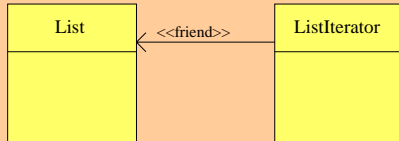
25

Bind stereotype



26

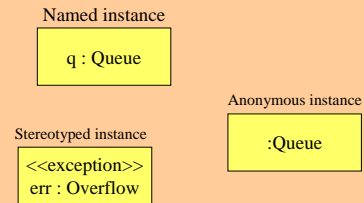
Friend stereotype



27

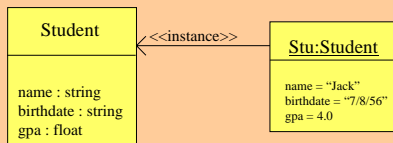
Instance

- Instance & object are largely synonymous
- concrete manifestation of an abstraction



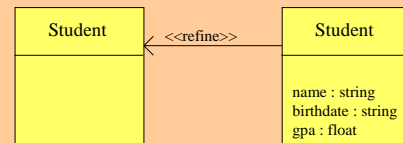
28

Instance relation: stereotype



29

refine stereotype



30

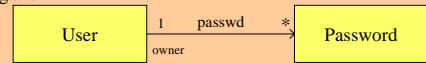
The Association Relationship

- Structural relationship
- objects of one thing are connected to objects of another
- Navigation: unless otherwise specified, navigation is bi-directional
- in some cases, may want to limit navigation to one direction

31

One-way navigation

4 adornments:
name, role,
multiplicity &
aggregation



Given a user, you may want to find his/her password;
but given a password, you don't want to be able to
find the user!

32

Aggregation

- Simple aggregation is conceptual; distinguishes the whole from the part.
- Simple aggregation does not change the meaning of navigation across the association between whole & parts

33

Composition

- Adds some important semantics to aggregation
- composition is a form of aggregation with strong ownership; the whole **owns** its parts
- if a whole object is copied or deleted, its composite parts are copied or deleted: navigation goes from whole to part!
- in a composite aggregation, an object may be a part of only one composite at a time.

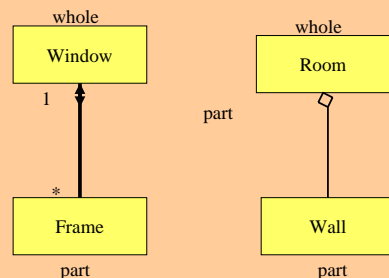
34

Aggregation vs. Composition

- In an aggregation, a part may be shared by several wholes; e.g., in a house object, a wall object may be shared by more than one room object
- in composition, an object is part of only one composite at a time: in a windowing system, a Frame belongs to exactly one window
- In composition, the whole is responsible for the disposition of the parts

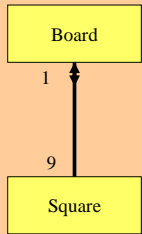
35

Composition vs. Aggregation



36

Composition

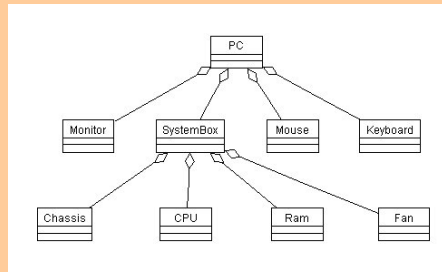


Aggregation and composition are special cases of association; thus, association can always be used instead of aggregation or composition

In C++:
Aggregation is like a reference or a pointer;
composition is like containing a value

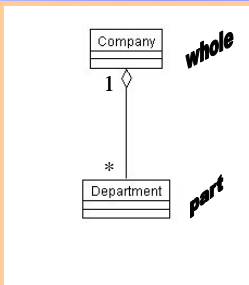
37

aggregation



38

aggregation



39

Generalization

- Relationship between a general thing and a more specific kind of that thing
- aka is-a or is-a-kind-of
- means that objects of the child may be used anywhere the parent may appear, **but not in reverse!**
- A child **may** inherit the attributes of the parent

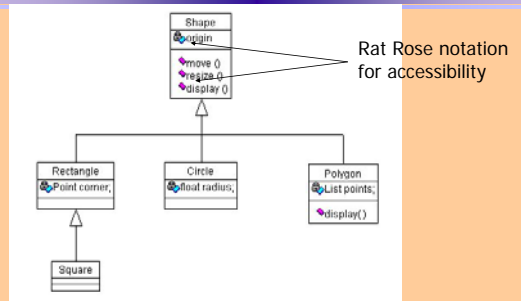
40

Generalization

- A class may have 0 or more parents
- child w/ no parents is a **base class**
- a class with no children is a leaf
- a class with a single parent: **single inheritance**
- a class with more than one parent uses **multiple inheritance**

41

Generalization



42

Associations link classes

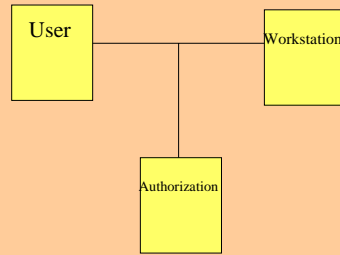
- *associations link class instances*
- associations are often impl as pointers
- binary: relates a pair of classes
- indicates the **sending of a message**



navigability: the direction of the message

43

Can model associations as classes



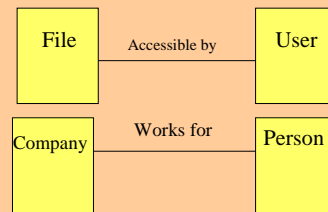
44

Associations may have 4 adornments

- **Name**: describe the nature of the relationship
- **role** that a class plays in an association
- **multiplicity** - it's a structural relationship. This indicates "how many" objects may participate in the association
- **aggregation**: whole/part relationship. A whole that consists of many parts

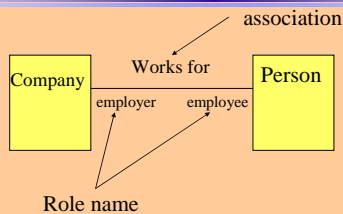
45

The name adornment



46

Role



47

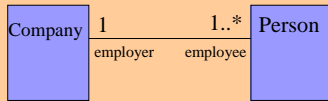
Multiplicity

- indicates the number of items involved in an association
- must indicate the number at each end of the association



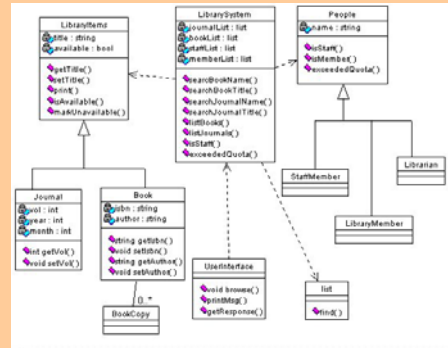
48

Multiplicity



49

A Class diagram for Library Application



50

Tips

- Initially, don't think about types of relationships or adornments
- dwell with the model
- think about the present and the future
- remember the goal: Build a system!

51