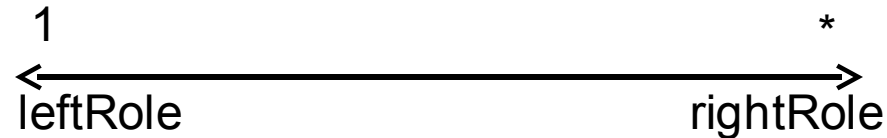


Associations



- Relations between classes
- Roles
 - analogous to names of instance variables
- Multiplicities
 - 0, 1, *, 0..1, 1..*, 5..6, and so on
 - says how many objects each object knows
 - would be realized through arrays, Sets, Lists, and so on
- Navigability
 - bidirectional: each class references the other
 - unidirectional: A knows B, but B doesn't know A
 - no arrow heads: means either "bidirectional" or "not specified"

Naming Conventions for Associations

- If you have an association with a role called "client":
 - there may be an instance variable "client"
 - there may be operations "getClient()", "setClient()"
 - encapsulation principle
- If there is only one association to class "Something":
 - role name redundant
 - role often assumed to be called "something"

Associations: Conceptually

- Responsibility
 - A knows B
 - A talks to B
 - A is responsible for updating B
 - A can get data from B
 - A is owner of B
 - A can do things with B
- A link between objects of the two classes

Generalizations



- Another relation among classes: “is a kind of”
- B generalizes A
- A inherits from B
- Substitutability
 - A can be substituted for B
 - A does everything that B does, and possibly more
- Polymorphism possible
 - A has the same specification as B
 - A implements unspecified features differently
 - Different replies to the same request possible
- Multiple inheritance possible

Background on Relations (Review)

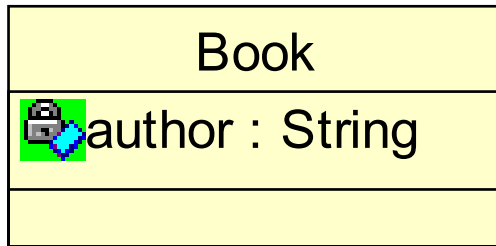
- A relation between A and B is a subset of $A \times B$
- Example: Divides relation on $\{2..6\}$
divides = $\{ (2,2), (2,4), (2,6), (3,3), (3,6), (4,4), (5,5), (6,6) \}$
- Well-known relations from mathematics
 - divides
 - =
 - $<, >$
 - functions are special relations
 - A function $A \rightarrow B$ is a relation between A and B so that each element of A occurs in exactly one pair

Background on Relations

- Properties of relations
 - reflexive: $a=a$
 - symmetric: $a=b \Rightarrow b=a$
 - transitive: $a>b \text{ and } b>c \Rightarrow a>c$
- UML relations
 - associations
 - may be symmetric: if bidirectional
 - generalizations
 - are transitive:
 X inherits from Y
 Y inherits from Z
implies: X inherits from Z
 - are never symmetric:
 X inherits from Y *implies* Y does not inherit from X

Attributes and Associations

- ...are exchangeable!



is the same as:



Attributes

- Are used instead of associations:
 - for frequently used types (int, String, ...)
 - to denote value semantics (copied not referenced)
 - in Java: primitive types
 - in C++: members that are not pointers or references
 - to save space
- Syntax:
visibility name: type
- Everything's optional
- Visibility
 - + public - private ~ package # protected
- List attributes only if they help in understanding class
 - should become private in implementation model