UML Class Diagrams

These lecture slides are copyright (C) Marty Stepp, 2007. They may not be rehosted, sold, or modified without expressed permission from the author. All rights reserved.

UML class diagrams

- What is a UML class diagram?
  - **UML class diagram**: a picture of the classes in an OO system, their fields and methods, and connections between the classes that interact or inherit from each other

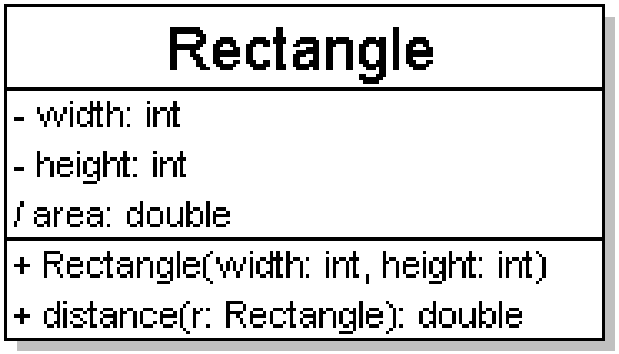
- What are some things that are not represented in a UML class diagram?
  - details of how the classes interact with each other
  - algorithmic details; how a particular behavior is implemented

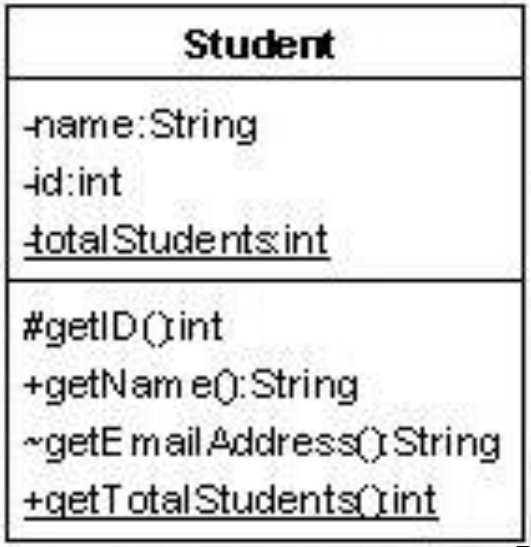
Diagram of one class

- class name in top of box
  - write <<interface>> on top of interfaces' names
  - use *italics* for an *abstract class* name

- attributes (optional)
  - should include all fields of the object

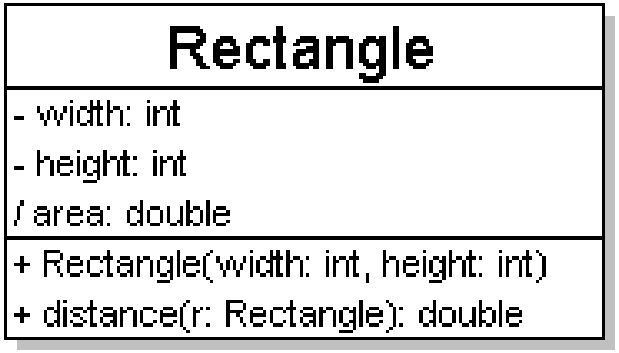
- operations / methods (optional)
  - may omit trivial (get/set) methods
    - but don't omit any methods from an interface!
  - should not include inherited methods

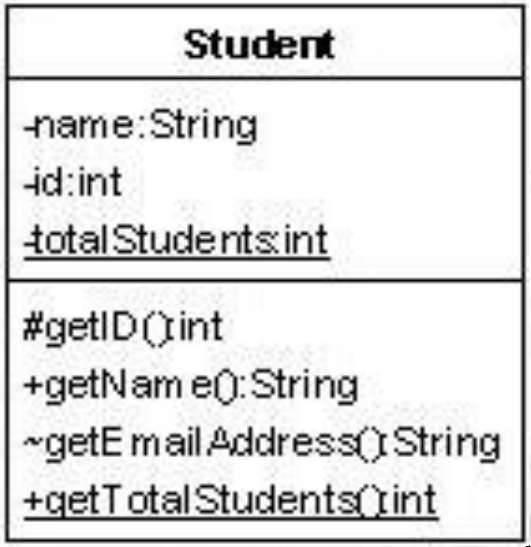




Class attributes

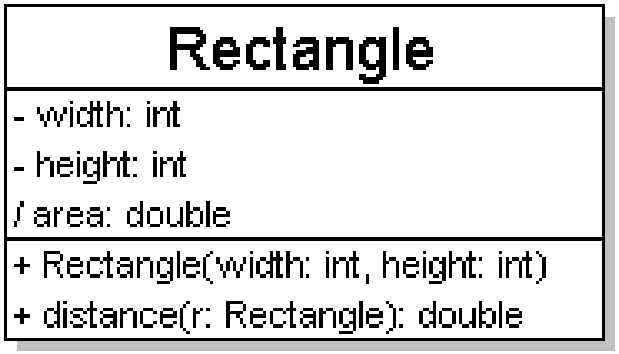
- attributes (fields, instance variables)
  - *visibility name* : *type* [*count*] = *default_value*
  - visibility: + public
    # protected
    - private
    ~ package (default)
    / derived
  - underline static attributes
  - **derived attribute**: not stored, but can be computed from other attribute values
  - attribute example:
    - balance : double = 0.00

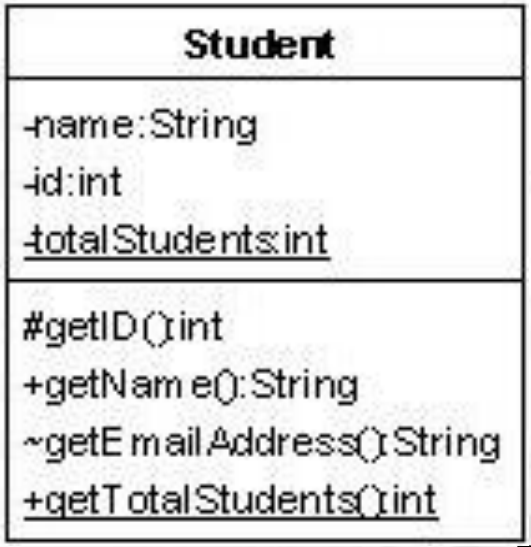




Class operations / methods

- operations / methods
  - *visibility name* (*parameters*) : *return_type*
  - visibility: + public
    # protected
    - private
    ~ package (default)
  - underline static methods
  - parameter types listed as (name: type)
  - omit *return_type* on constructors and when return type is void
  - method example:
    + distance(p1: Point, p2: Point): double





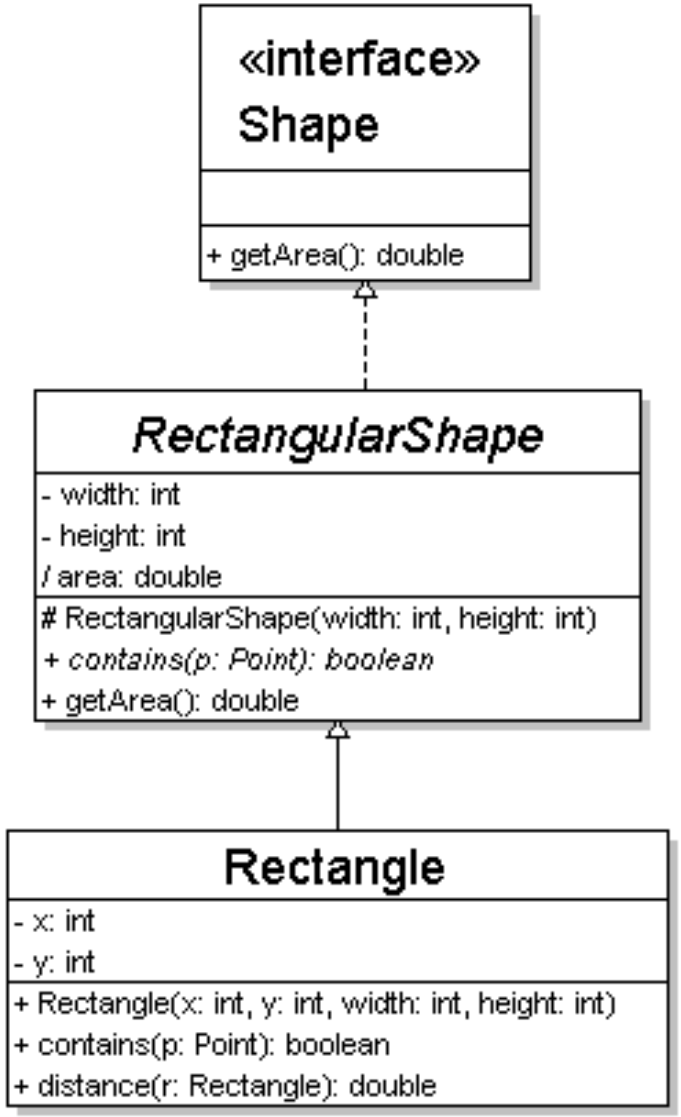
Relationships btwn. classes

- **generalization**: an inheritance relationship
  - inheritance between classes
  - interface implementation

- **association**: a usage relationship
  - dependency
  - aggregation
  - composition

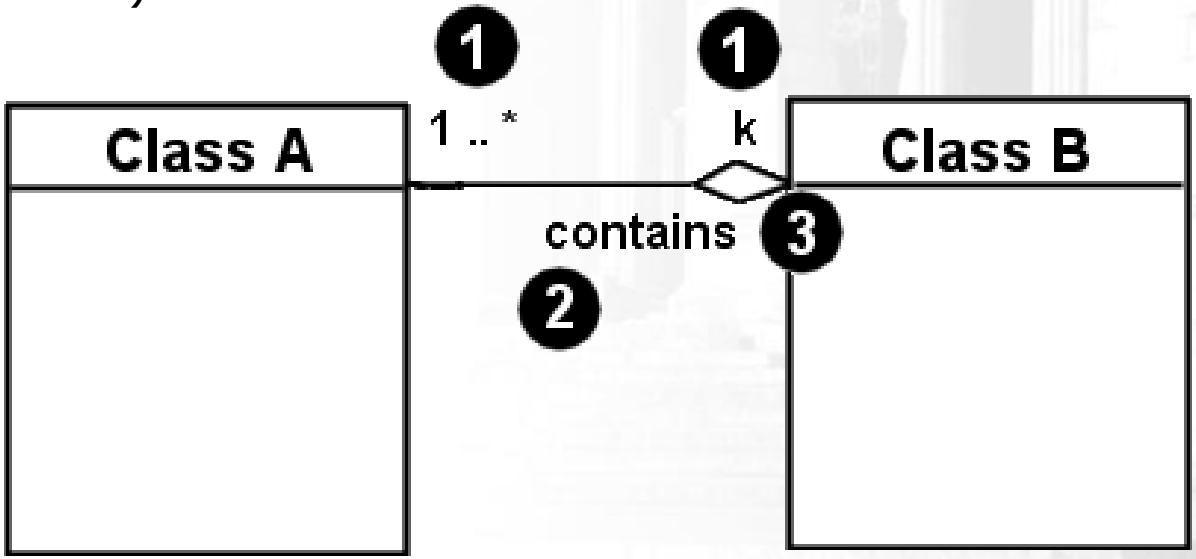
Generalization relationships

- generalization (inheritance) relationships
  - hierarchies drawn top-down with arrows pointing upward to parent
  - line/arrow styles differ, based on whether parent is a(n):
    - class:
      solid line, black arrow
    - abstract class:
      solid line, white arrow
    - interface:
      dashed line, white arrow
  - we often don't draw trivial / obvious generalization relationships, such as drawing the Object class as a parent



Associational relationships

- associational (usage) relationships
  1. multiplicity (how many are used)
     - * ⇒ 0, 1, or more
     - 1 ⇒ 1 exactly
     - 2..4 ⇒ between 2 and 4, inclusive
     - 3..* ⇒ 3 or more
  2. name (what relationship the objects have)
  3. navigability (direction)

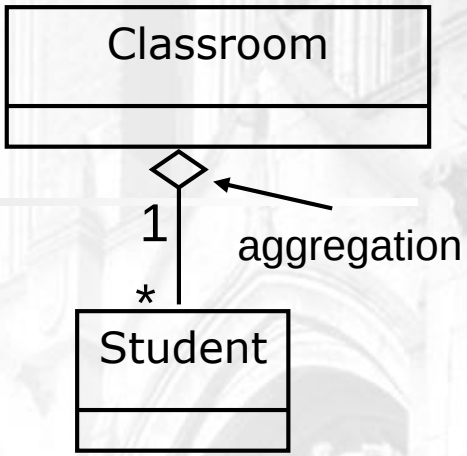


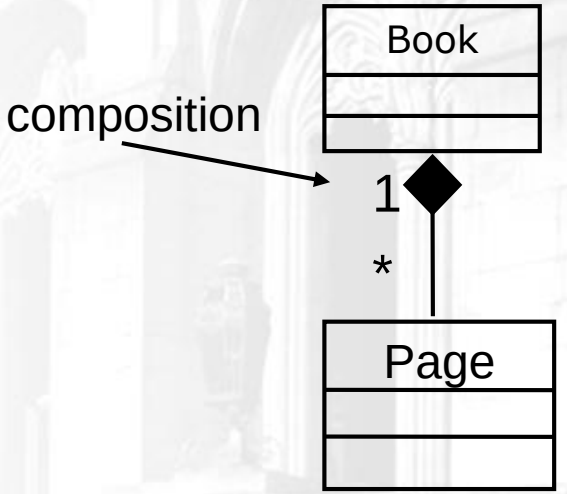
Association types

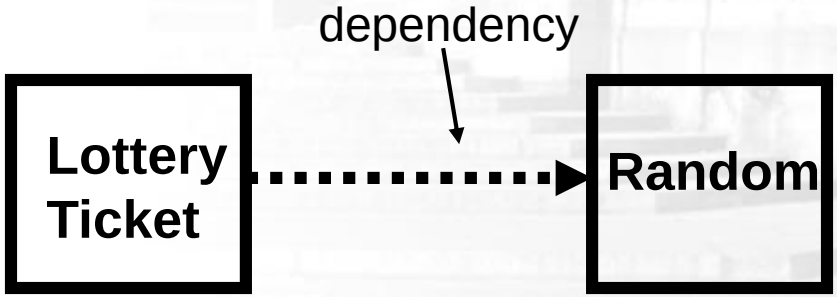
- **aggregation**: "is part of"
  - symbolized by a clear white diamond

- **composition**: "is entirely made of"
  - stronger version of aggregation
  - the parts live and die with the whole
  - symbolized by a black diamond

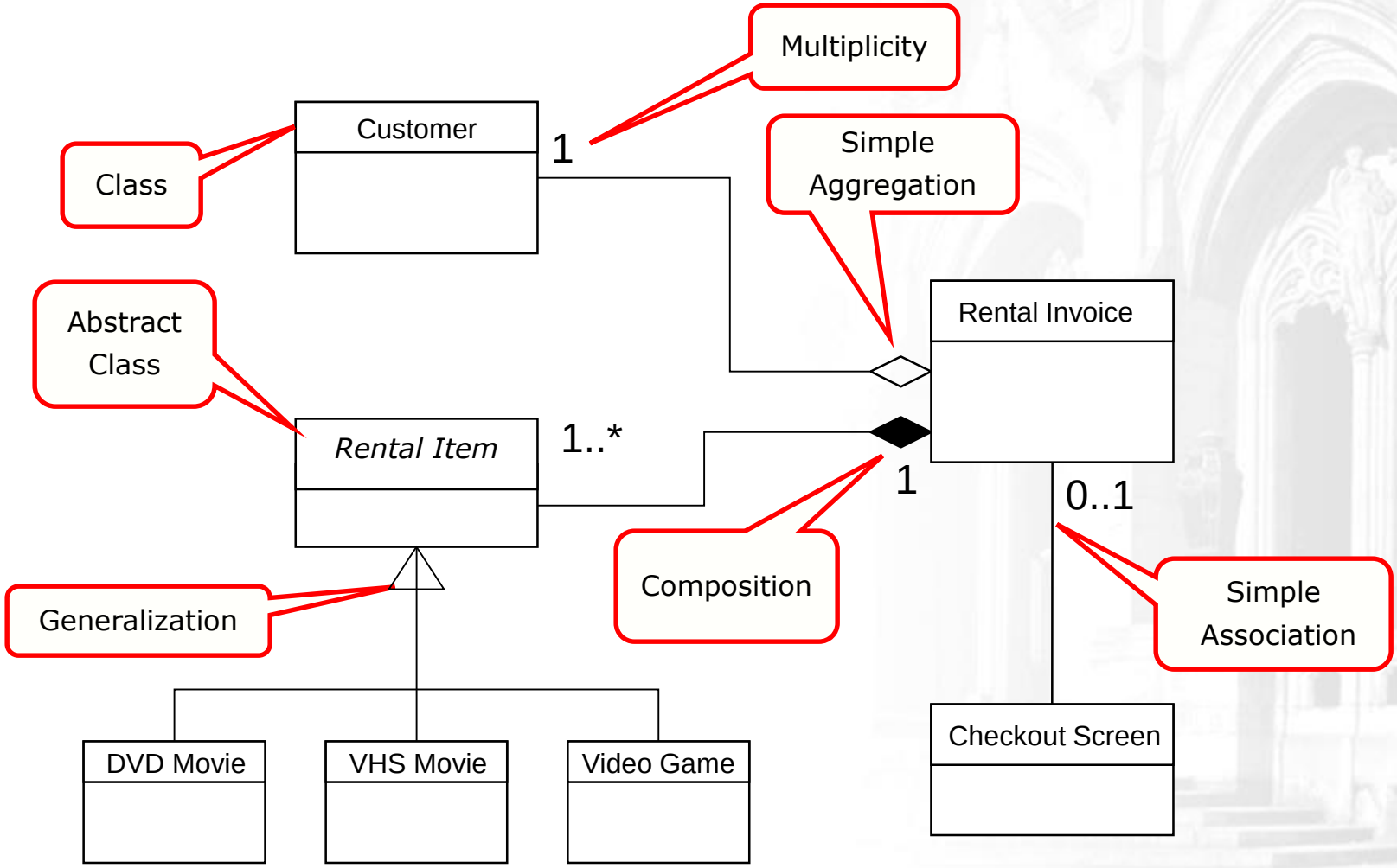
- **dependency**: "uses temporarily"
  - symbolized by dotted line
  - often is an implementation detail, not an intrinsic part of that object's state

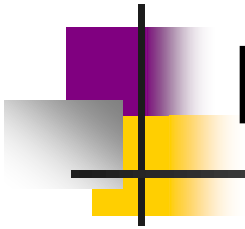






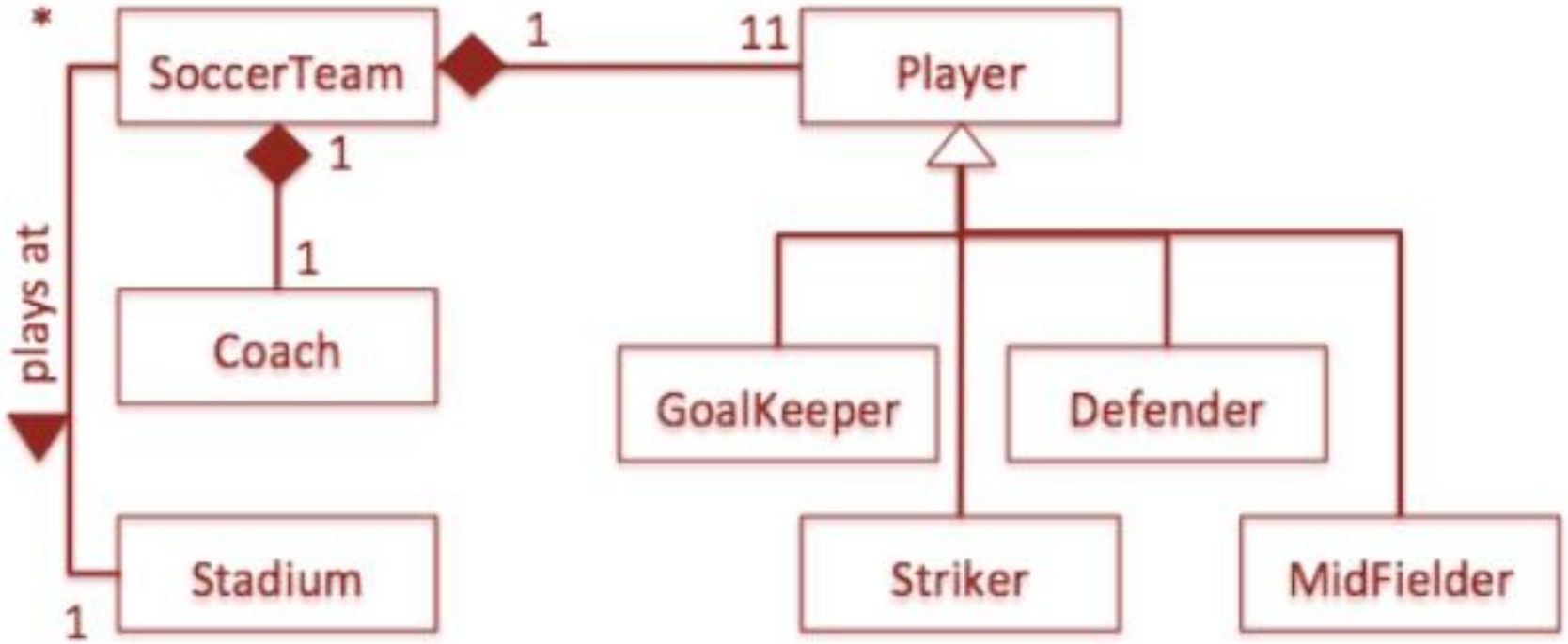
Class diagram example





Exercise

- Draw the relations between the following classes in the UML notation. Do NOT try to put any methods or fields in classes, simply use boxes with names. Make sure to put multiplicities. Also put labels on arrows if needed. Note that a soccer team has eleven players and one coach. A player can play for only one team. A coach can train only one team.
- SoccerTeam, Coach, Player, GoalKeeper, Striker, Defender, Midfielder, Stadium.


*
SoccerTeam
1
11
Player
1
1
plays at
Coach
GoalKeeper
Defender
Stadium
1
Striker
MidFielder