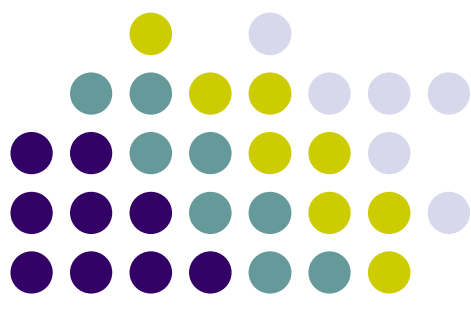


ICS 52: Introduction to Software Engineering

Instructor: Prof. Dan Frost
TA: Derek
dpfister@uci.edu
Oct. 29, 2008





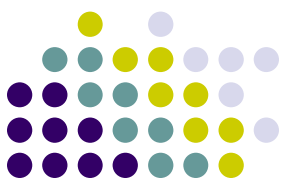
Is there anything besides Code in Software?

- What have we discussed already?

Types of Architectural Modeling

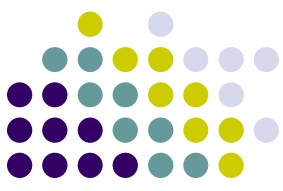


- xADL 2.0 (Developed by UCI)
- Others (ACME, etc.)
- Most Popular:
 - UML
 - Developed by a company called Rational Software in 1994
- From wikipedia:
 - “UML is often criticized as being gratuitously large and complex[4]. It contains many diagrams and constructs that are redundant or infrequently used. This criticism is more frequently directed at UML 2.0 than UML 1.0, since newer revisions include more [design-by-committee](#) compromises[[citation needed](#)].”



Types of Diagrams

- UML pretty extensive
- Some types of diagrams used more often than others
- Structure Diagrams
- Behavior Diagrams
- Interaction Diagrams

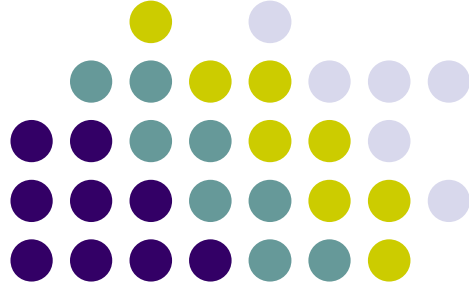


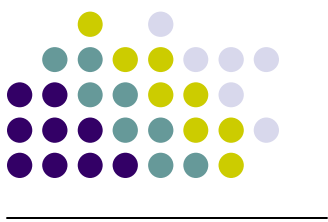
UML Modeling

- General purpose programming language to create an abstract model of a system
- Modeling
 - Functional model
 - Use cases diagram (behavior diagram)
 - Object model
 - Class diagrams (what we are going to talk about, structure diagram)
 - Dynamic model
 - Sequence diagrams, state machines etc. (interaction diagram)

Class Diagrams

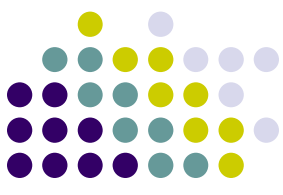
- *What is it?*
- *Why do we need it?*
- *What are its main components?*





Class Diagrams

- **What?** *a class diagram is a diagram showing a collection of classes and interfaces, along with the collaborations and relationships among classes and interfaces.*
- **Why?**
 - A class diagram is a pictorial representation of the detailed system design.
 - It shows the *static* structure of the things that exist, their internal structure, and their relationships to other things.
 - Class diagrams are referenced time and again by the developers while implementing the system.



Class Diagrams

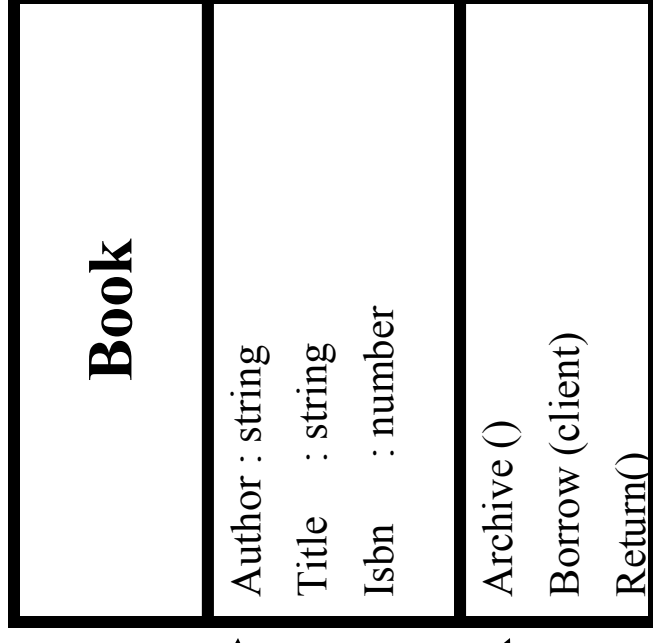
- ***Main Components:***

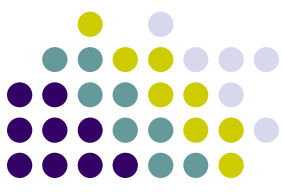
- Class
 - Class name
 - Attribute
 - Operation
- Relationships
 - Generalization
 - Association
 - Aggregation

Class Diagrams: class



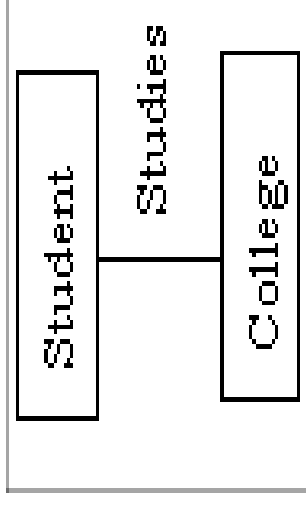
- Classes can have three parts
 - Name
 - Attributes (properties)
 - Operations (behavior)
- Classes can show visibility and types.



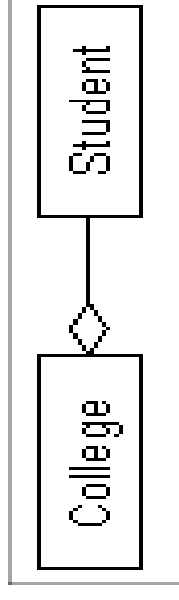


Class Diagram: Relationships

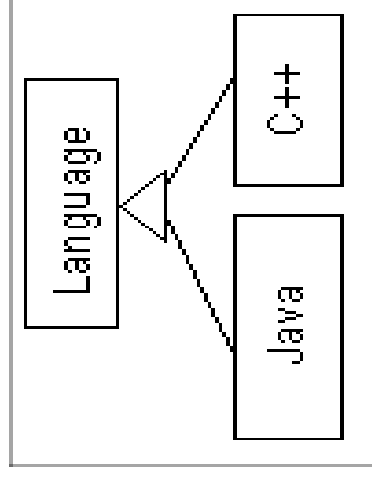
- Association

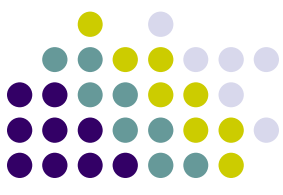


- Aggregation

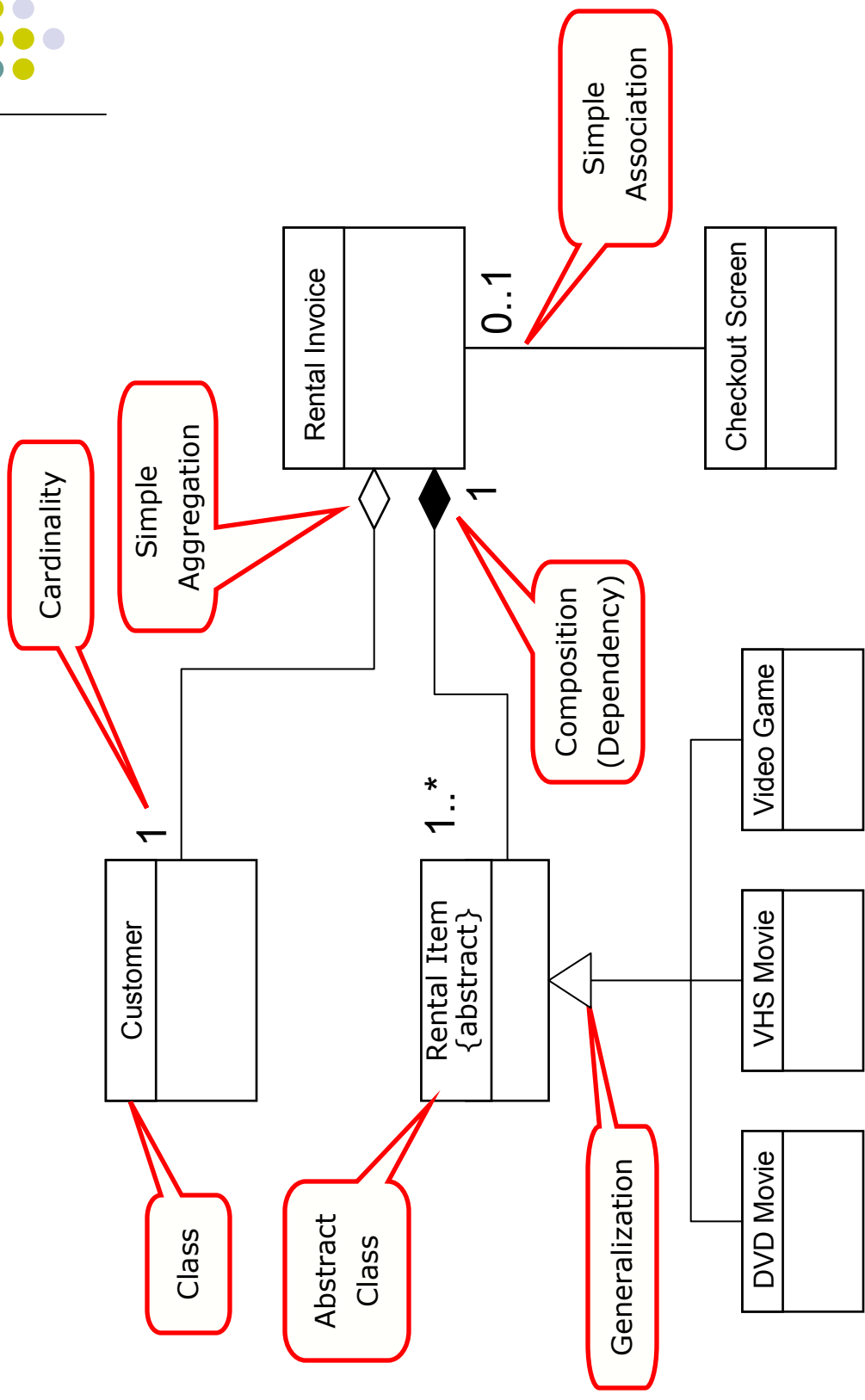


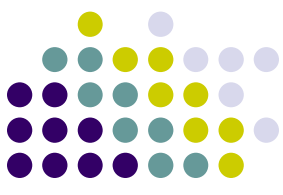
- generalization



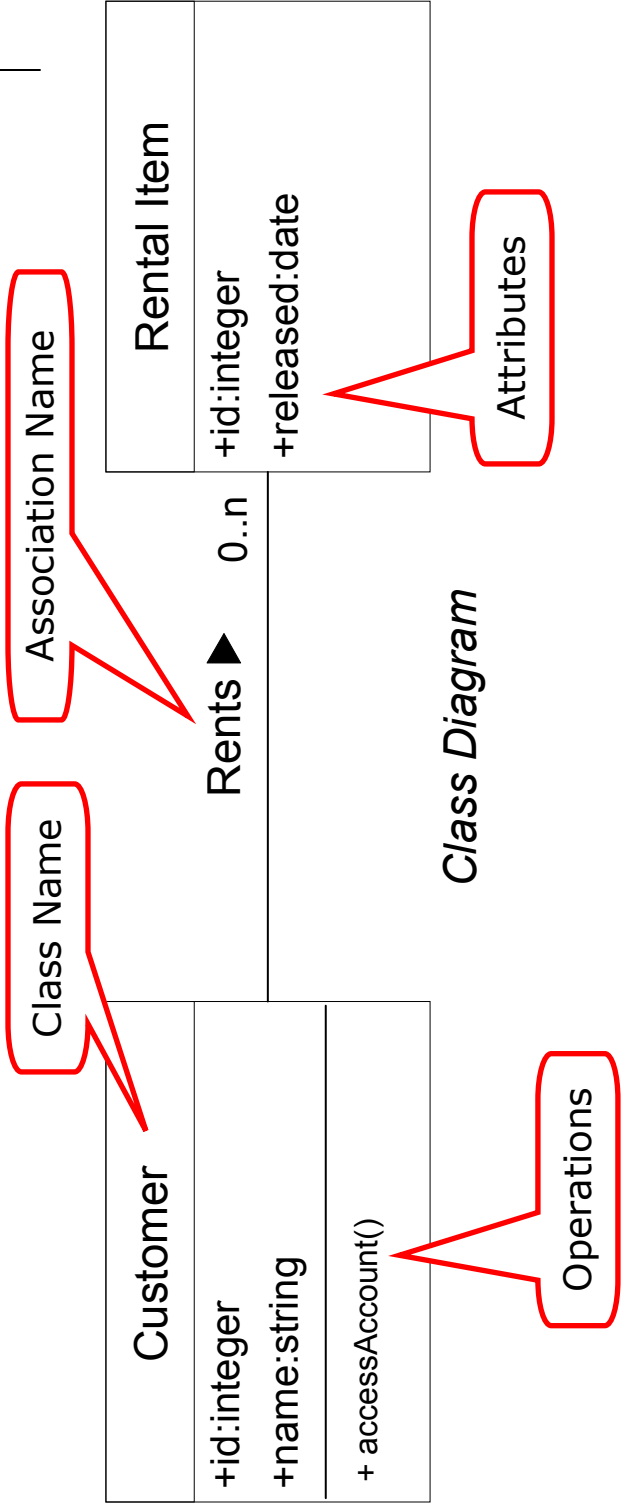


Class Diagrams



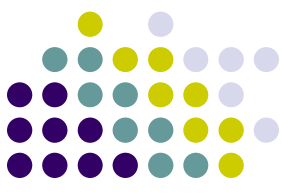


Class Diagrams

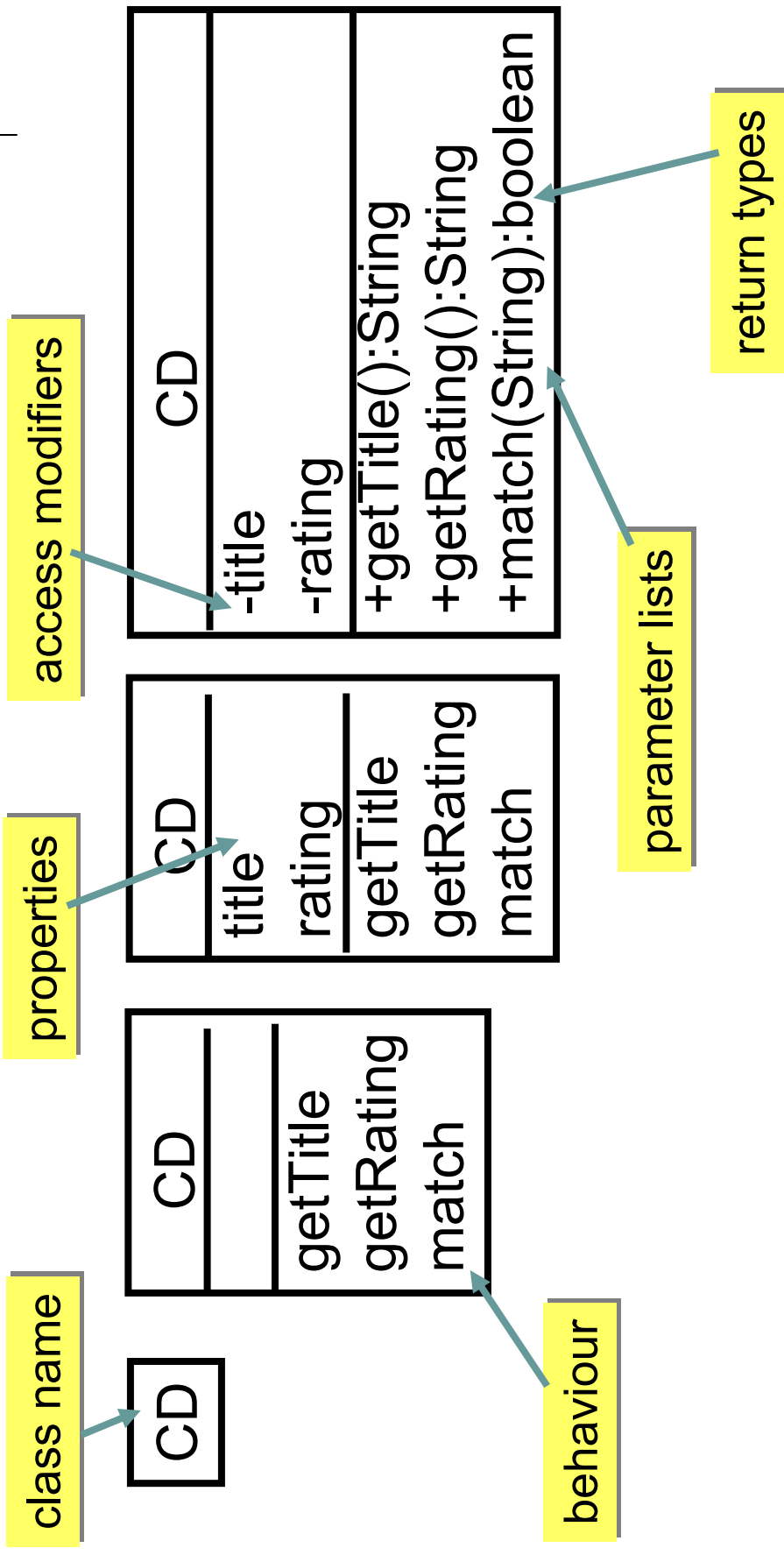




- Attributes are the data
- Operations are the functions of your classes



Class Diagrams

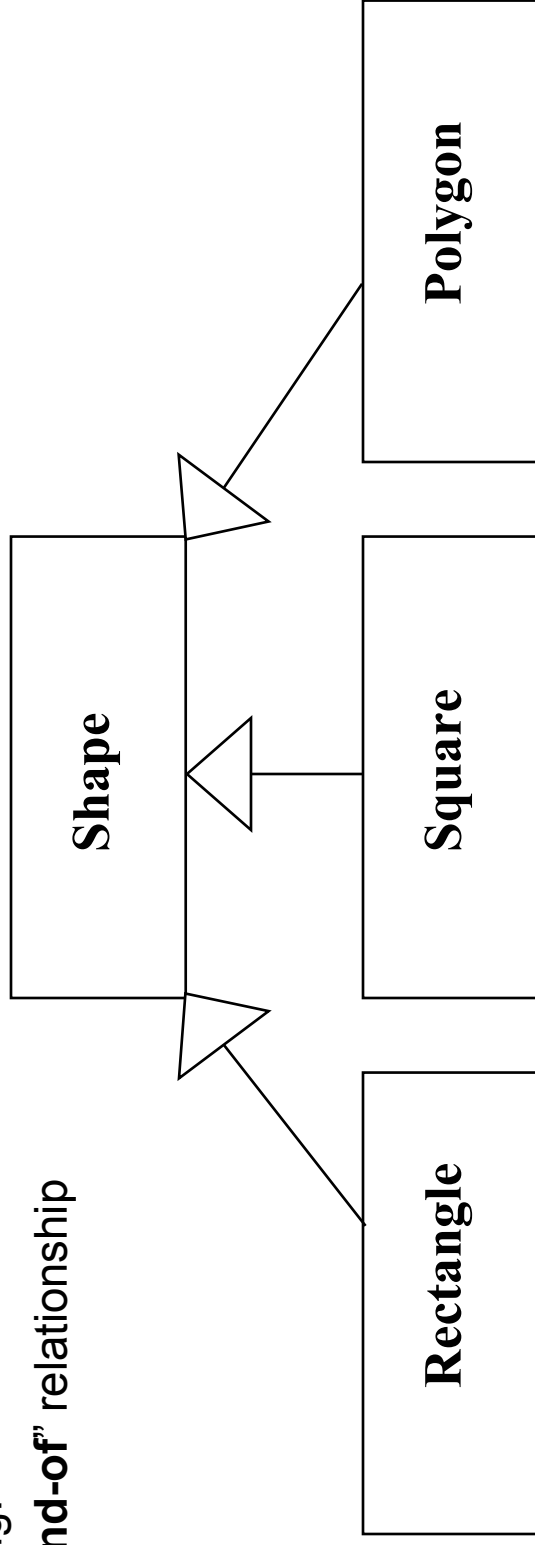


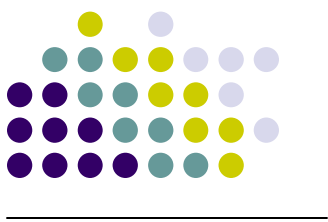
Class Diagrams - Relationships



Generalization

- A relationship between a general thing and a more specific kind of that thing.
- “**is-a-kind-of**” relationship

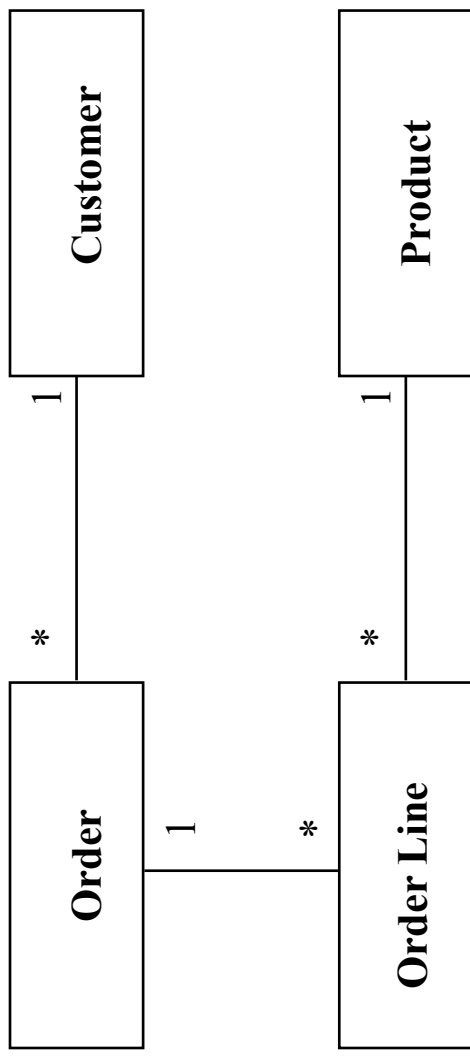




Class Diagrams - Association

Association

- A structural relationship that specifies that objects of one thing are connected to objects of other

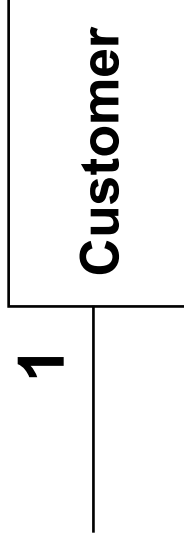


- An Order has to come from a single Customer.
- A Customer may make several Orders over time.
- Each of these Orders has several Order Lines, each of which refers to a single Product.

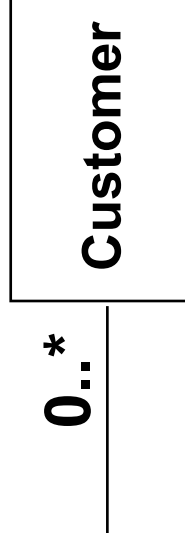


Class Diagrams – Association – Cardinality

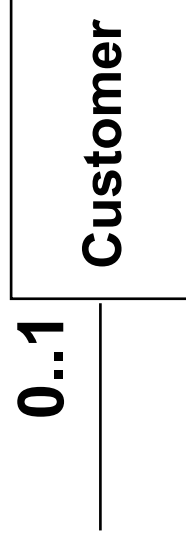
-how many objects participate in a relationship. Also known as **multiplicity**.



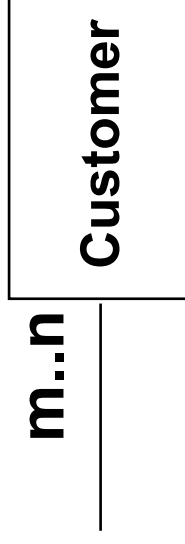
Exactly one



Many (zero or more)



Optional (zero or one)

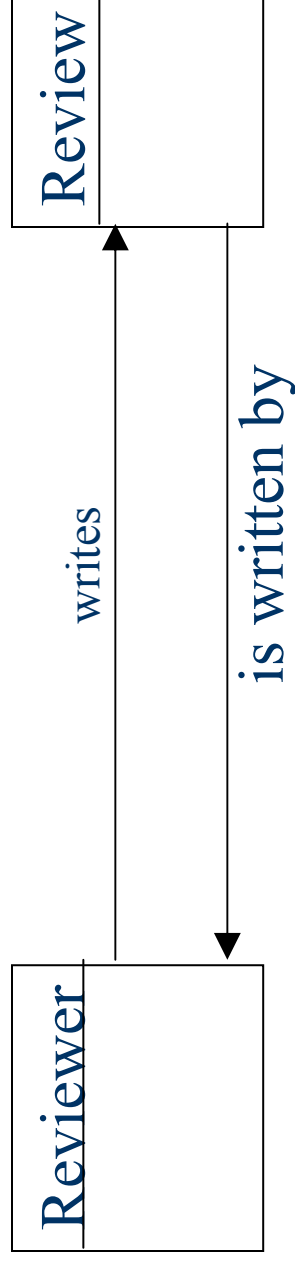
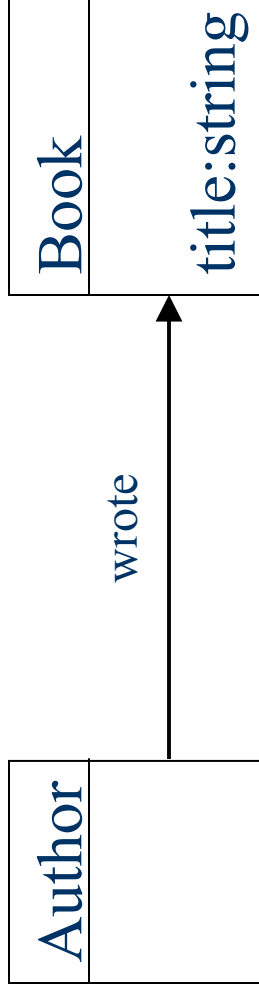


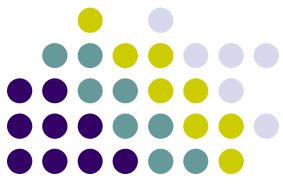
Numerically specified e.g 3..6
or 2..5 or 1,000..*



Class Diagrams — Association — Roles

- An association role can appear on top of the link between two classes with a triangle indicating the direction of the role
- An association role changes accordingly to the viewpoint selected





Object Composition

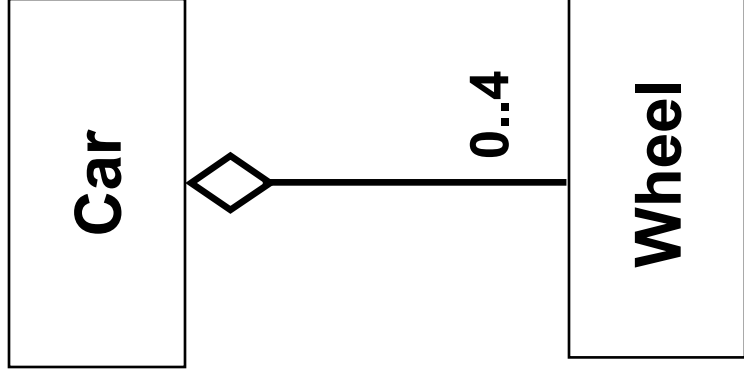
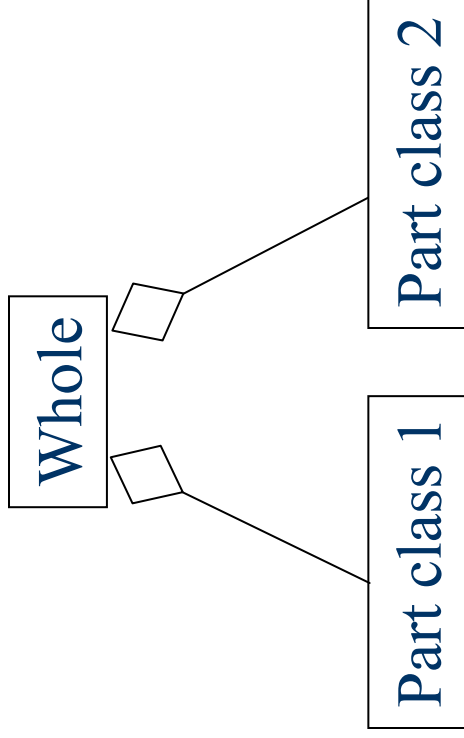
- Way to combine simple objects into more complex objects
 - C example:

```
struct point {  
    double x;  
    double y;  
} first;
```
- Different from generalization
 - Think of (void *) and (int *)
- Different kinds of composition
- Ownership: When composition is destroyed, should objects belonging to it be destroyed as well?
- If not, its called “aggregation”



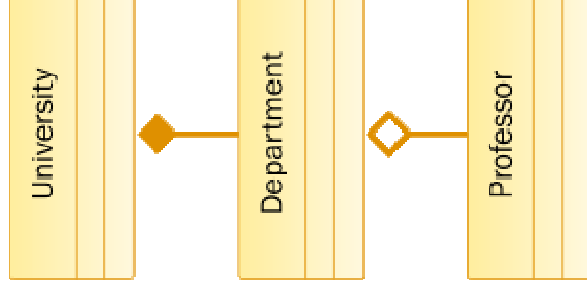
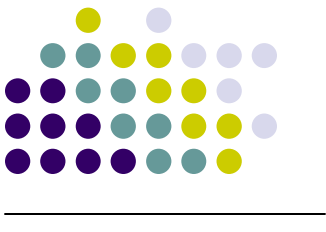
Class Diagrams - Aggregation

- *Aggregation* is a whole/part relationship

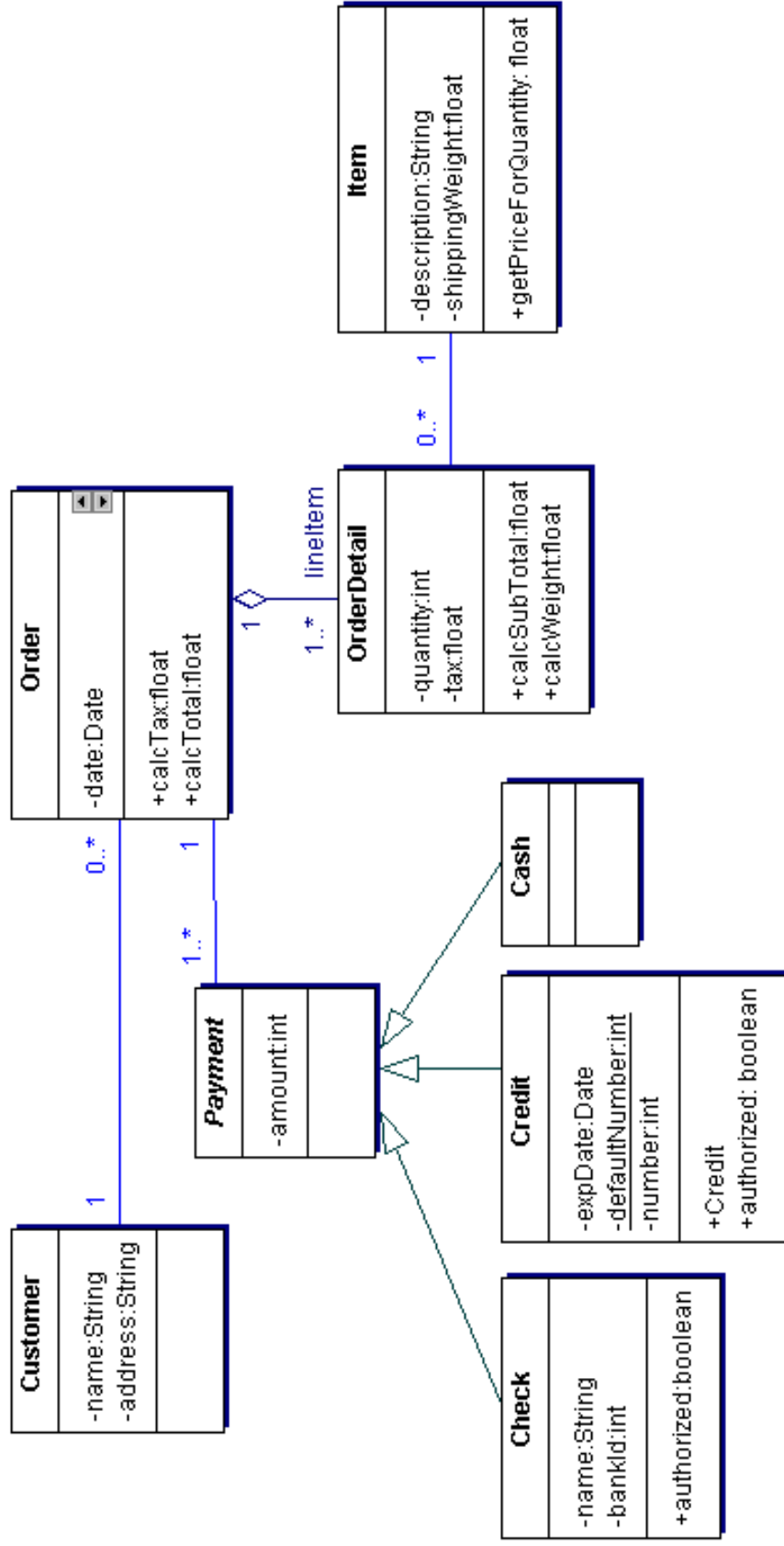


What's with the diamond?

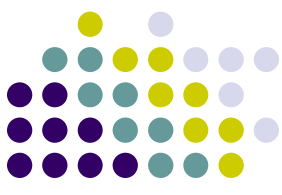
- Diamond specifies if composition is aggregation or not.
- Example
 - A university owns departments
 - Each department has a number of professors
 - If department closes, departments cease to exist.
 - Not so with professors
- Hence, university is a *composition* (solid diamond) of departments but department is an *aggregation* (hollow diamond) of professors.



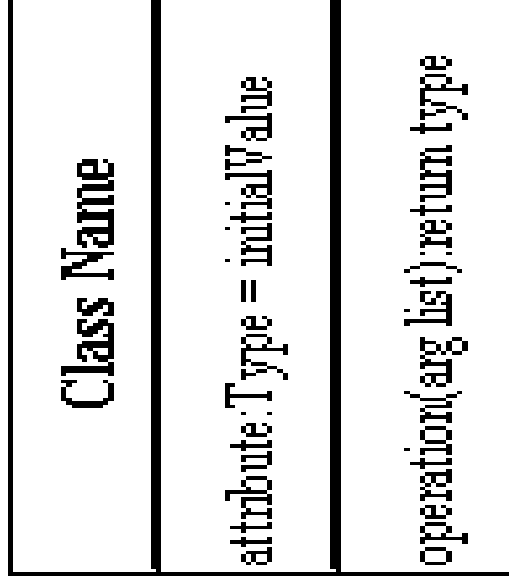
Example



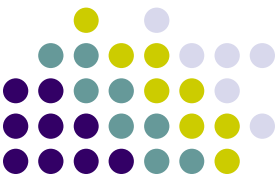
Basic Class Diagram Symbols and Notations



- **Classes**
 - Illustrate classes with rectangles divided into compartments.
 - Place the name of the class in the first partition (centered, bolded, and capitalized)
 - list the attributes in the second partition
 - write operations into the third.



Note: Please only list the most important attributes and operations in hw

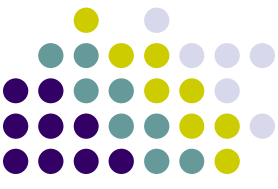


- **Visibility**

- Use visibility markers to signify who can access the information contained within a class.
- Private visibility hides information from anything outside the class partition.
- Public visibility allows all other classes to view the marked information.
- Protected visibility allows child classes to access information they inherited from a parent class.

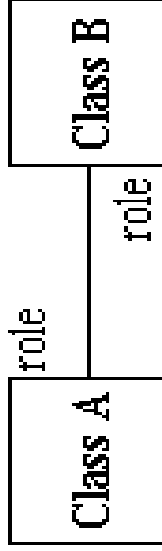
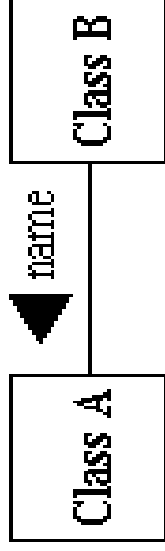
Class Name
- attribute
- attribute
+ operation
+ operation
+ operation

+ <i>public</i>
- <i>private</i>
<i>protected</i>

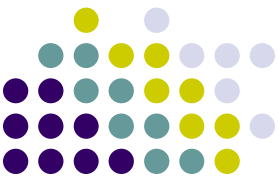


● Associations

- Associations represent static relationships between classes.
- Place association names above, on, or below the association line.
- Use a filled arrow to indicate the direction of the relationship.
- Place roles near the end of an association. Roles represent the way the two classes see each other.

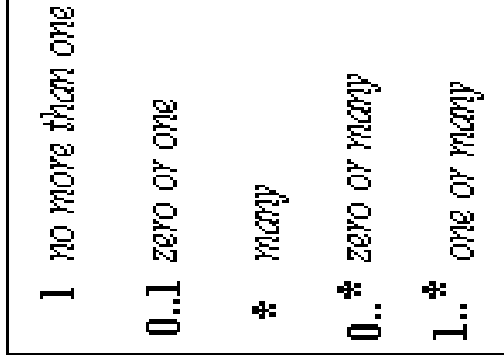
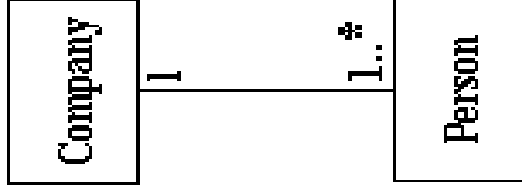


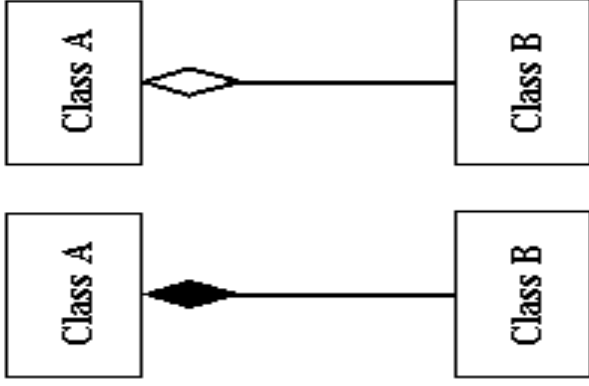
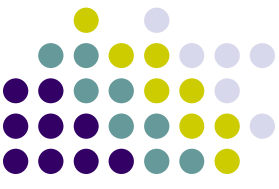
Note: It's uncommon to name both the association and the class roles.



● Multiplicity (Cardinality)

- Place multiplicity notations near the ends of an association.
- These symbols indicate the number of instances of one class linked to *one* instance of the other class.
- For example, one company will have one or more employees, but each employee works for one company only.



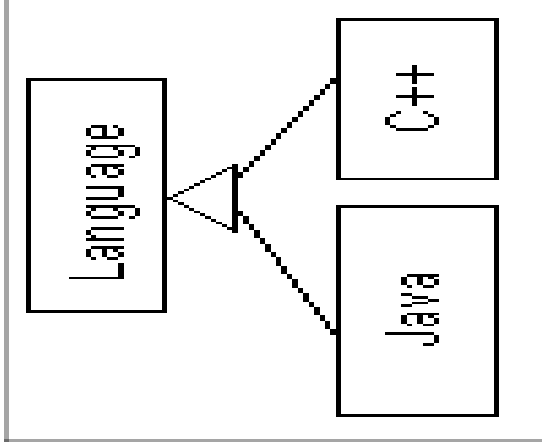


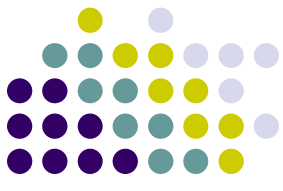
- **Aggregation and Composition**
 - Use a hollow diamond to represent a simple aggregation relationship, in which the "whole" class plays a more important role than the "part" class, but the two classes are not dependent on each other.
 - Composition is a stronger form of aggregation where the whole and parts have coincident lifetimes, and it is very common for the whole to manage the lifecycle of its parts. Illustrate composition with a filled diamond.
 - The diamond end in both a composition and aggregation relationship points toward the "whole" class or the aggregate.



● Generalization

- Generalization is another name for inheritance or an "is a" relationship. It refers to a relationship between two classes where one class is a specialized version of another.
- For example, Honda is a type of car. So the class Honda would have a generalization relationship with the class car.
- In real life coding examples, the difference between inheritance and aggregation can be confusing. If you have an aggregation relationship, the aggregate (the whole) can access only the PUBLIC functions of the part class. On the other hand, inheritance allows the inheriting class to access both the PUBLIC and PROTECTED functions of the superclass.





Requirements for A System

- The school administration enters, for an upcoming term, the courses, classrooms, and available times, along with additional information such as the teacher(s) assigned to each course.
- A course is characterized primarily by its one or two teachers and the maximum number of students.
- Each classroom, lab, and lecture hall has a fixed capacity, as well as other characteristics.
- Teachers can specify, for each course, their time and room preferences.
- Students can specify, for the upcoming term, courses they would like to take. Schoocr will try not to schedule to meet at the same time courses that many students want to take in the same term.
- Schoocr creates a schedule that is based on all the data that has been entered.
- Schoocr can produce a number of statistical and informative reports.

Example: Online Component of VSS



- Review of Requirements:
 - Users login with their username and password
 - Can renew and rate movies
 - Users enter email
 - Can't have late fees, can't renew movie if it is already reserved