COURSE NAME

OBJECT ORIENTED ANALYSIS AND DESIGN

CSC 2210

(UNDERGRADUATE)

CHAPTER 3

CLASS DIAGRAM

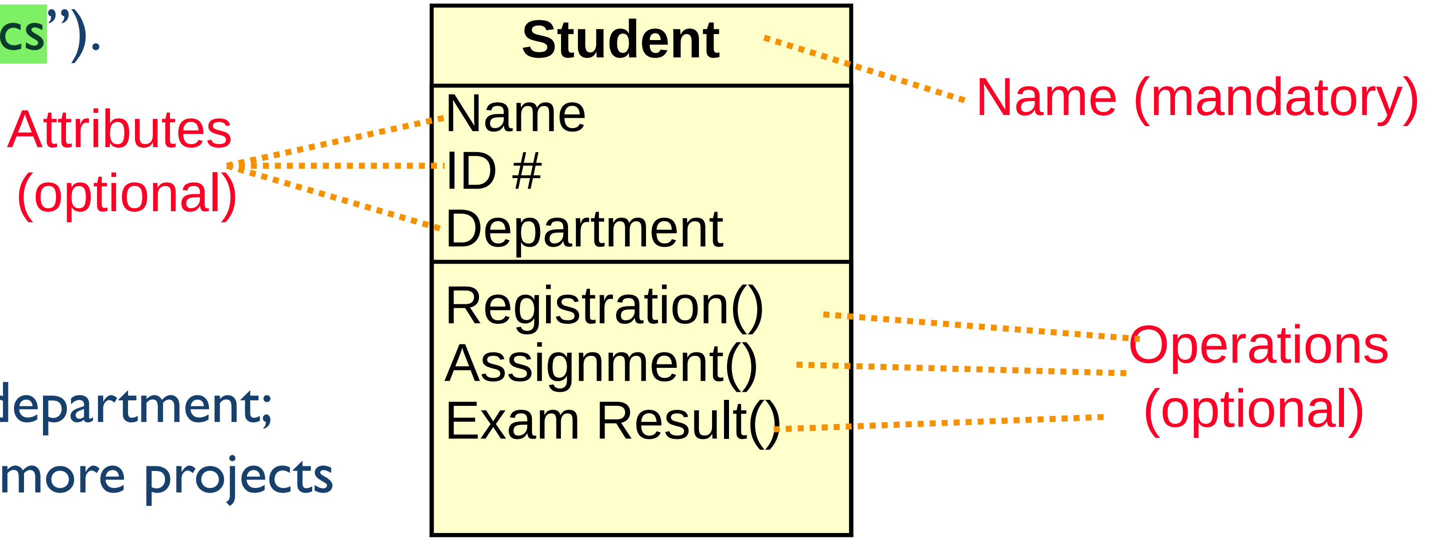
VICTOR STANY ROZARIO
ASSISTANT PROFESSOR
DEPARTMENT OF COMPUTER SCIENCE, AIUB
Web: https://cs.aiub.edu/profile/stany

RG Google Scholar LinkedIn

WHAT IS CLASS?

- A class describes a group of objects with
  - similar properties (attributes),
  - common behavior (operations),
  - common relationships to other objects,
  - and common meaning ("semantics").

- Examples
  - Employee:
    - has a name, employee id, and department;
    - an employee works in one or more projects



CLASS DIAGRAM

- The Class diagram represents classes, their component parts, and the way in which classes of objects are related to one another.
- The Class diagram includes:
  - Attributes describe the appearance and knowledge of a class of objects (e.g., birds have wings).
  - Operations define the behavior that a class of objects can manifest (e.g., dog is barking).
  - Stereotypes help you understand this type of object in the context of other classes of objects with similar roles within the system's design.
  - Properties provide a way to track the maintenance and status of the class definition (e.g., properties of a class attribute can be public/private/protected)
- Classes are used to represent objects. Objects can be anything having properties and responsibility.

DRAWING CLASS DIAGRAM : ATTRIBUTE

- An attribute describes a piece of information that an object owns or knows about itself.
- Attribute visibility:

- Public (+) visibility allows access to objects of all other classes. Default accesses specifier.
- Private (-) visibility limits access to within the class itself except friend function.
  For example, only operations of the class have access to a private attribute.
- Protected (#) visibility allows access by subclasses. In case of generalizations (inheritance), subclasses must have access to the attributes and operations of the super-class or they cannot be inherited.
- Package (~) visibility allows access to other objects in the same package.

DRAWING CLASS DIAGRAM : ATTRIBUTE

visibility / attribute name : data type = default value {constraints}

- Visibility (+, -, #, ~): *Required before code generation.* The programming language will typically specify the valid options. For example, the minus sign represents the visibility "private" meaning only members of the class that defines the attribute may see the attribute.
- Slash (/): The derived attribute indicator is *Optional*. Derived values may be computed or figured out using other data and a set of rules or formulas. Consequently, there are more design decisions that need to be addressed regarding the handling of this data. Often this flag is used as a placeholder until the design decisions resolve the handling of the data (C = A* B).
- Attribute name: *Required.* Must be unique within the class. It is recommended that attribute name should be a meaningful name with the value it stores, i.e., at least three character long.

DRAWING CLASS DIAGRAM : ATTRIBUTE

- Data type: *Required.* During analysis, the data type should reflect how the client sees the data. You could think of this as the external view. During design, the data type will need to represent the programming language data type for the environment in which the class will be coded. These two pieces of information can give the programmer some very specific insights for the coding of get and set methods to support access to the attribute value.

- Assignment operator and default value: *Optional.* Default values (usually zero) serve two valuable purposes.

  - First, default values can provide significant ease-of-use improvements for the client.
  - Second and more importantly, they protect the integrity of the system from being corrupted by missing or invalid values.
  - A common example is the tendency to let numeric attributes default to zero. If the application ever attempts to divide using this value, you will have to handle resulting errors that could have been avoided easily with the use of a default.

DRAWING CLASS DIAGRAM : ATTRIBUTE

- Constraints: Constraints express all the rules required to guarantee the integrity of this piece of information. Any time another object tries to alter the attribute value, it must pass the rules established in the constraints. The constraints are typically implemented/enforced in any method that attempts to set the attribute value, e.g. {ID is assign by the system}
- Static Class level attribute (underlined attribute declaration): *Optional.* Denotes that all objects of the class share a single value for the attribute. (This is called a *static* value in Java.)

DRAWING CLASS DIAGRAM : OPERATION

- Objects have behaviors, things they can do and things that can be done to them. These behaviors are modeled as operations.
- Operations require a name, arguments, and sometimes a return.
- Arguments, or input parameters, are simply attributes, so they are specified using the attribute notation (name, data type, constraints, and default), although it is very common to use the abbreviated form of name and data type only.

DRAWING CLASS DIAGRAM : OPERATION

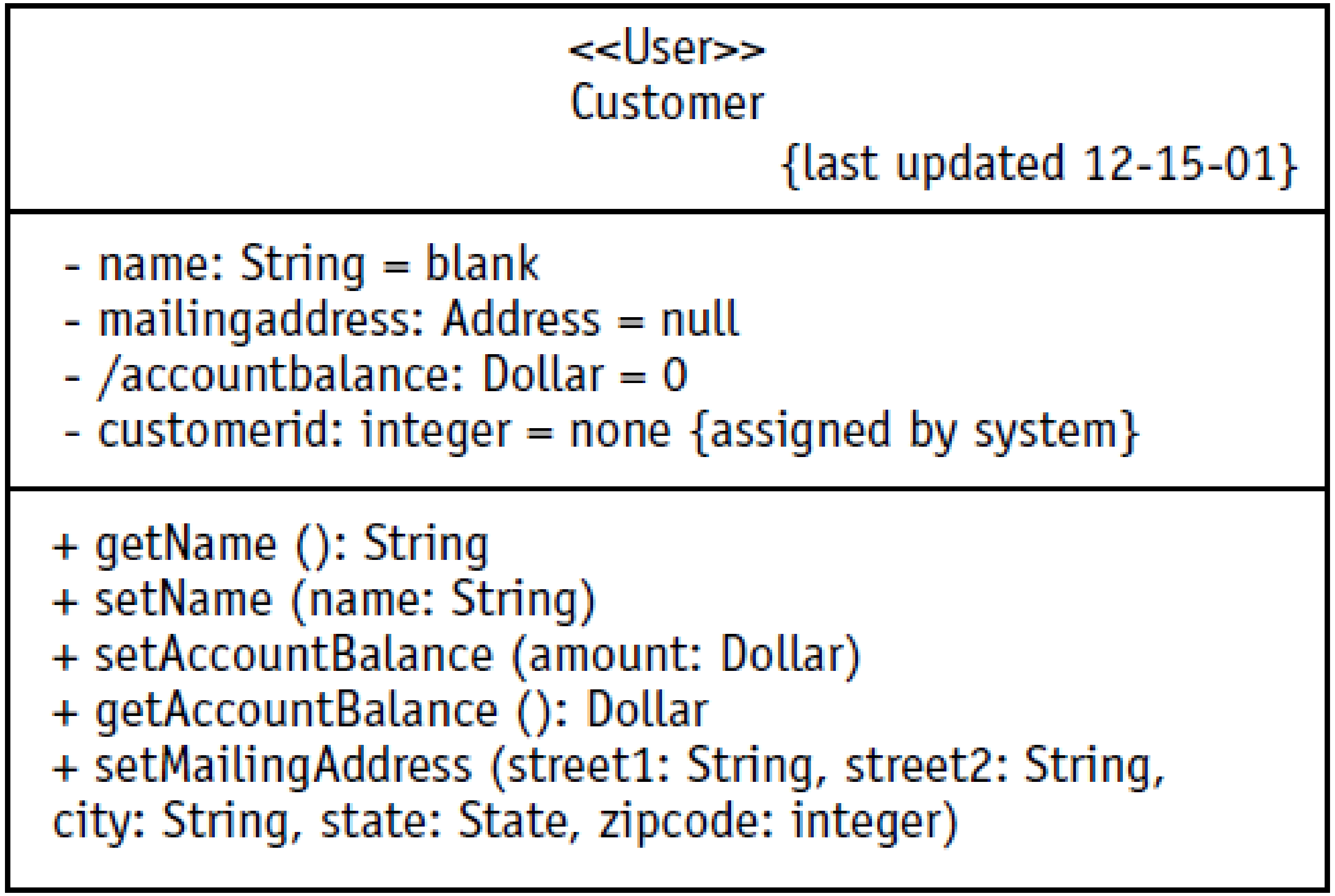
visibility operationName (argname : data type {constraints}, ...) : return data type {constraints}

- Visibility (+, -, #, ~): *Required before code generation.* The visibility values are defined by the programming language, but typically include public (+), private (-), protected (#), and package (~).
- Operation name: *Required*. Does not have to be unique, but the combination of name and parameters does need to be unique within a class.
- Arguments/parameters: Parameter is the list of arguments. Any number of arguments is allowed in the parameter. Each argument requires an identifier and a data type. Constraints may be used to define the valid set of values that may be passed in the argument. But constraints are not supported in many tools and will not be reflected in the code for the operation, at least not at this point.

DRAWING CLASS DIAGRAM : OPERATION

- Argument name: *Required for each parameter, but parameters are optional*. Any number of arguments is allowed.
- Argument data type: *Required for each parameter, but parameters are optional.*
- Constraints: *Optional*. In general, constraints express rules that must be enforced in the execution of the operation. In the case of parameters, they express criteria that the values must satisfy before they may be used by the operation. You can think of them as operation level pre-conditions.
- Return data type: *Required for a return value, but return values are optional.* The UML only allows for the type, not the name, which is consistent with most programming languages. There may only be one return data type, which again is consistent with most programming languages.

DRAWING CLASS DIAGRAM



```
<<User>>
Customer
                {last updated 12-15-01}
--------------------------------------------------
- name: String = blank
- mailingaddress: Address = null
- /accountbalance: Dollar = 0
- customerid: integer = none {assigned by system}
--------------------------------------------------
+ getName (): String
+ setName (name: String)
+ setAccountBalance (amount: Dollar)
+ getAccountBalance (): Dollar
+ setMailingAddress (street1: String, street2: String,
city: String, state: State, zipcode: integer)
```

Name Compartment

Attribute Compartment

visibility / attribute name : data type = default value {constraints}

Operation Compartment

visibility operationName (argname : data type {constraints}, ...) : return data type {constraints}

Figure 9-2 Complete class specification with all three compartments

CLASS DIAGRAM RELATIONSHIPS

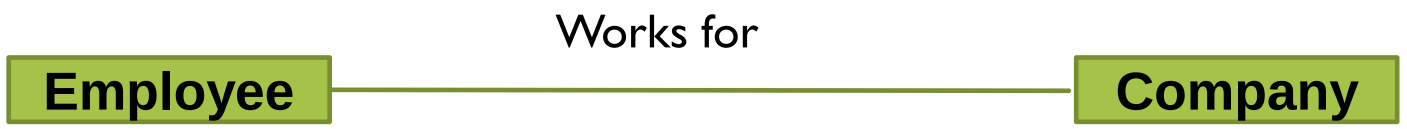
- **Association** is just a formal term for a type of relationship that this type of object may participate in. Associations may come in many variations, including simple, aggregate and composite, and Reflexive (e.g., relationships among objects of the same classes).
- Association Class encapsulates information about relationship
- **Inheritance** allows you to organize the class definitions to simplify and facilitate their implementation.

❑ Although other diagrams are necessary, but their primary purpose is to support the construction and testing of the Class diagram.

❑ Whenever another diagram reveals new or modified information about a class, the Class diagram must be updated to include the new information. If this new information is not passed on to the Class diagram, it will not be reflected in your code.

ASSOCIATIONS

- A semantic relationship between two or more classes that specifies connections among their instances.
- A structural relationship, specifying that objects of one class are connected to objects of a second class.
- Example: "An Employee works for a Company"

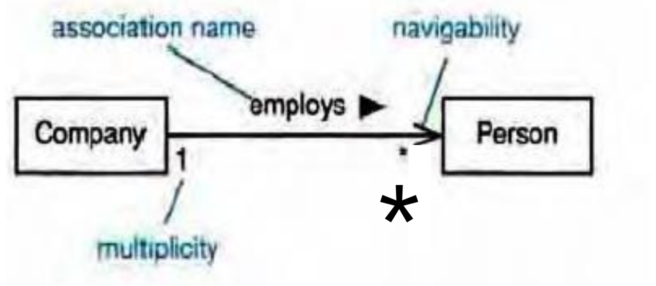


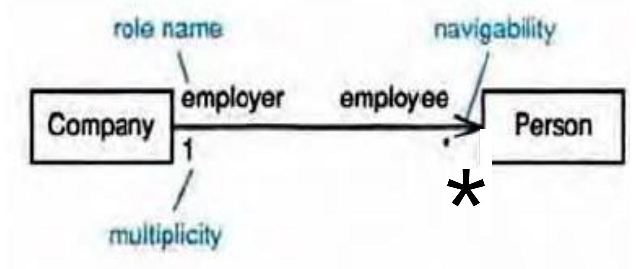
- An association between two classes indicates that objects at one end of an association "recognize" objects at the other end and may send messages to them.
- This property will help us discover less trivial associations using interaction diagrams.
- Qualified association in class diagram indicates a qualifier to be used for Indexing

ASSOCIATIONS

Associations may optionally have the following:

- **Association name**
  - Small black arrowhead to indicate the direction in which the name should be read
  - Should be a verb or verb phrase;
- **Role names**
  - on one or both association ends;
  - should be a noun or noun phrase describing the semantics of the role;
- **Multiplicity**
  - The number of objects that can participate in an instantiated relation
- **Navigability**
  - The direction of Arrows

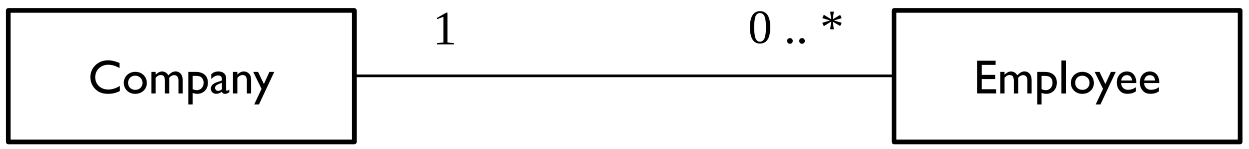




ASSOCIATION MULTIPLICITY

- Association multiplicity limits the number of objects that can participate in an association.
- Provides a lower and upper bound on the number of instances (at opposite end)
- Indicates whether an association is mandatory

Ask questions about the associations:



- Can a company exist without any employee?
  - If yes, then the association is optional at the Employee end - zero or more (0..*)
  - If no, then it is not optional - one or more (1..*)
  - If it must have only one employee - exactly one (1)
- What about the other end of the association? Can an employee work for more than one company?
  - If no, the correct multiplicity is 1.

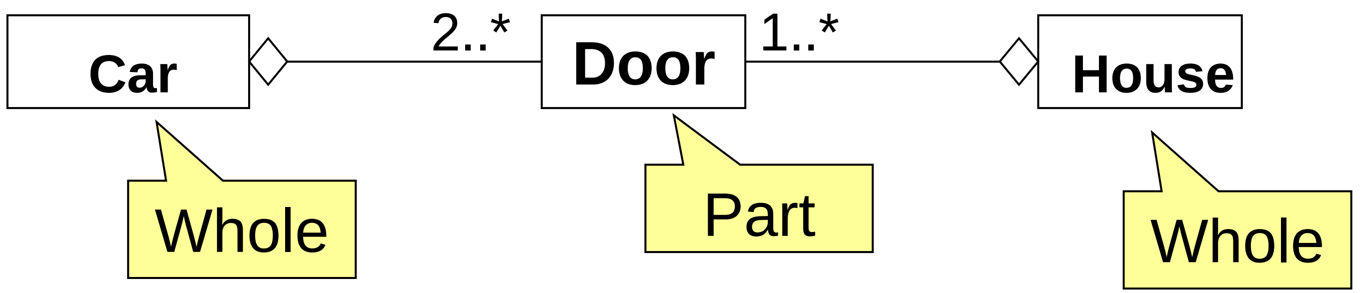
MULTIPLICITY INDICATOR

Exactly one	1
Many	*
Zero or more	0..*
One or more	1..*
Zero or one (optional association)	0..1
Specified range	2..4
Multiple, disjoint ranges	2, 4..6, 8

AGGREGATION

- A special form of association that models a whole- part relationship between an aggregate (the whole) and its parts (shared class i.e., **Part** by **Whole** class)

- Models a "is a part-part of" "has-a" relationship.



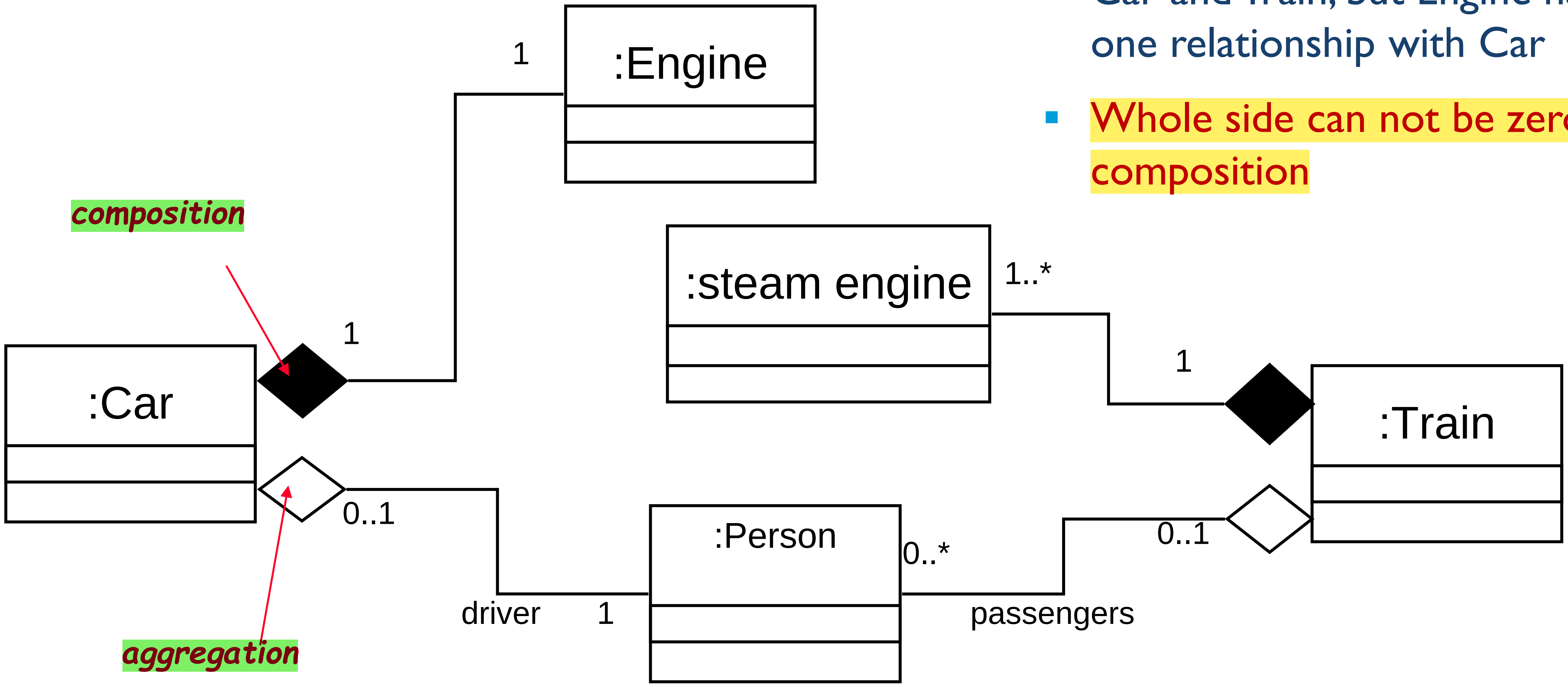
AGGREGATION

- Is the phrase "part of" used to describe the relationship?
  - A door is "part of" a car
- Are some operations on the whole automatically applied to its parts?
  - Move the car, move the door.
- Are some attribute values propagated from the whole to all or some of its parts?
  - The car is blue; therefore, the door is blue.
- Is there an intrinsic asymmetry to the relationship where one class is subordinate to the other (inheritance) ?
  - A door **is** part of a car. A car **is not** part of a door.

COMPOSITION

- A strong form of aggregation
  - The whole is the sole (only) owner of its part.
    - The part object may belong to only one whole
  - The lifetime of the part is dependent upon the whole.
    - The composite must manage the creation and destruction of its parts.
    - if the whole is removed from the model, so is the part.
    - the whole is responsible for the disposition of its parts

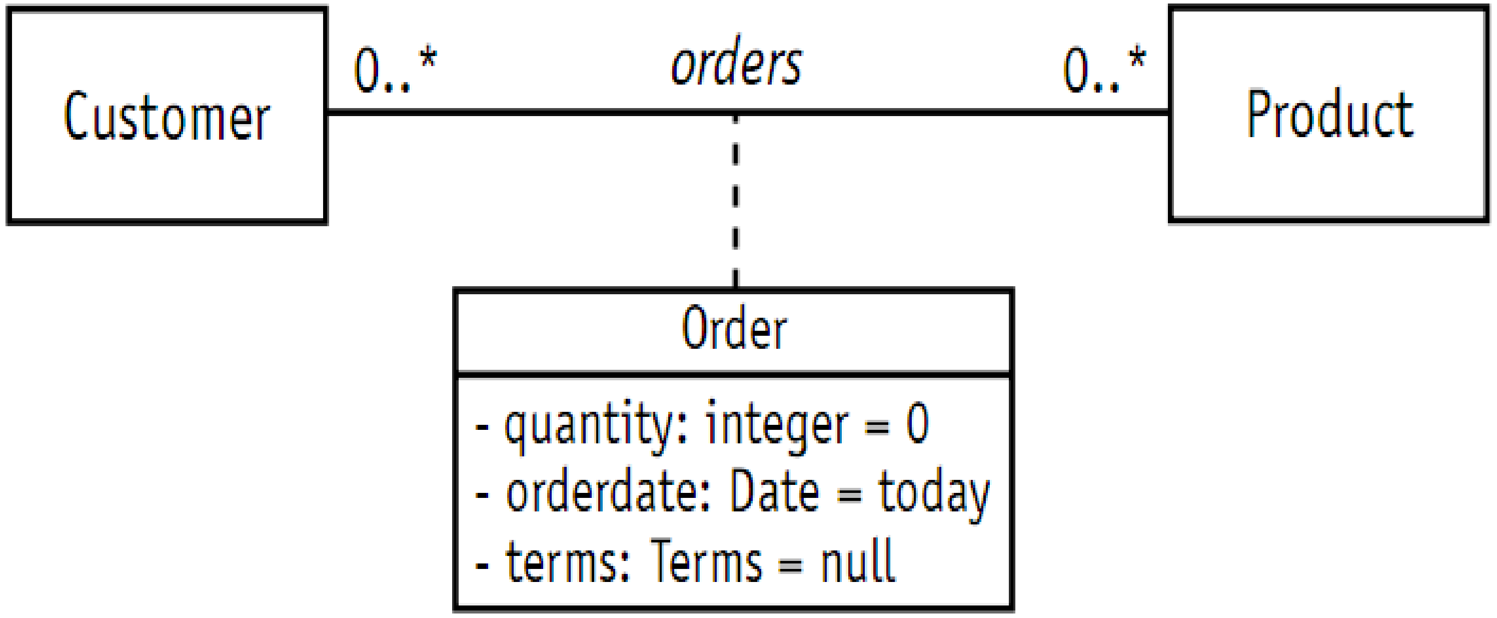
AGGREGATION & COMPOSITION



- Person has relationships with both Car and Train, but Engine has only one relationship with Car
- Whole side can not be zero in composition

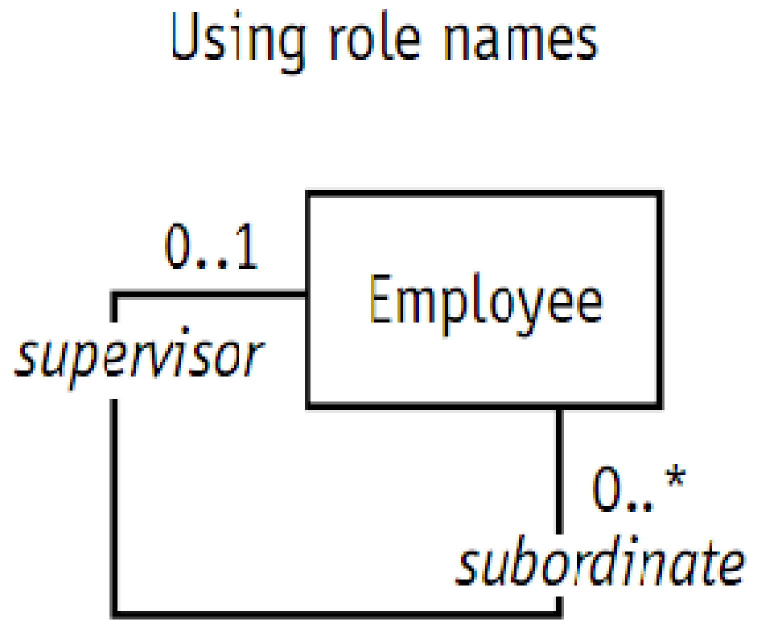
ASSOCIATION CLASS

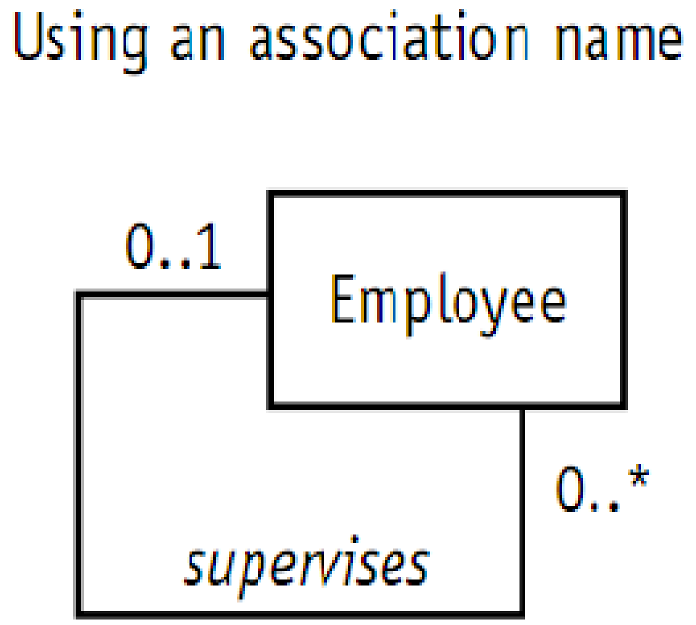
ENCAPSULATES INFORMATION ABOUT AN ASSOCIATION



REFLEXIVE ASSOCIATION

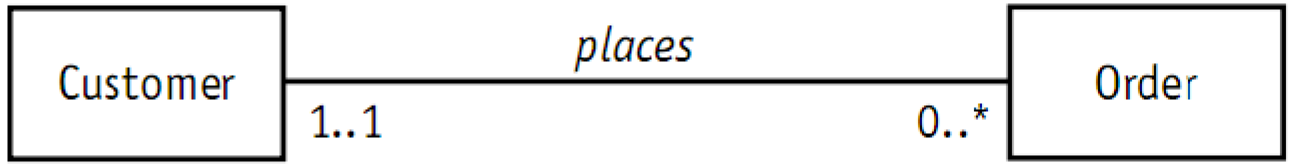
Reflexive association is a fancy expression that says objects in the same class can be related to one another. The entire association notation you've learned so far remains exactly the same, except that both ends of the association line point to the same class.



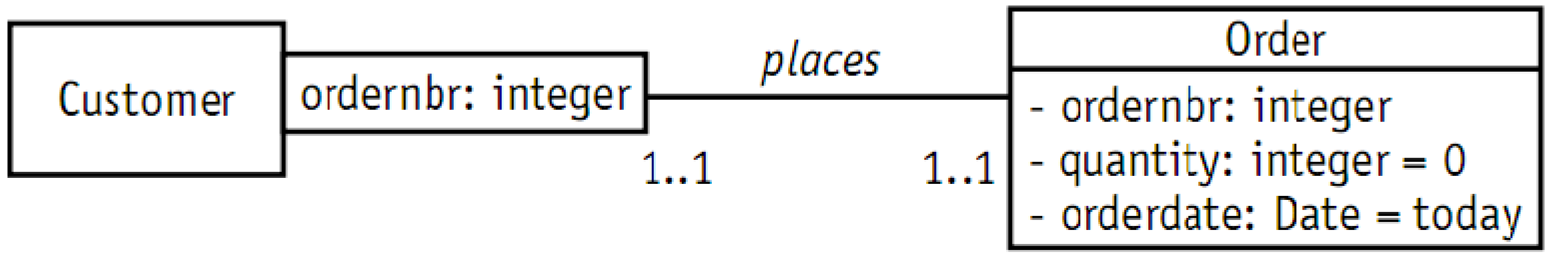


QUALIFIED ASSOCIATION

Without a qualifier



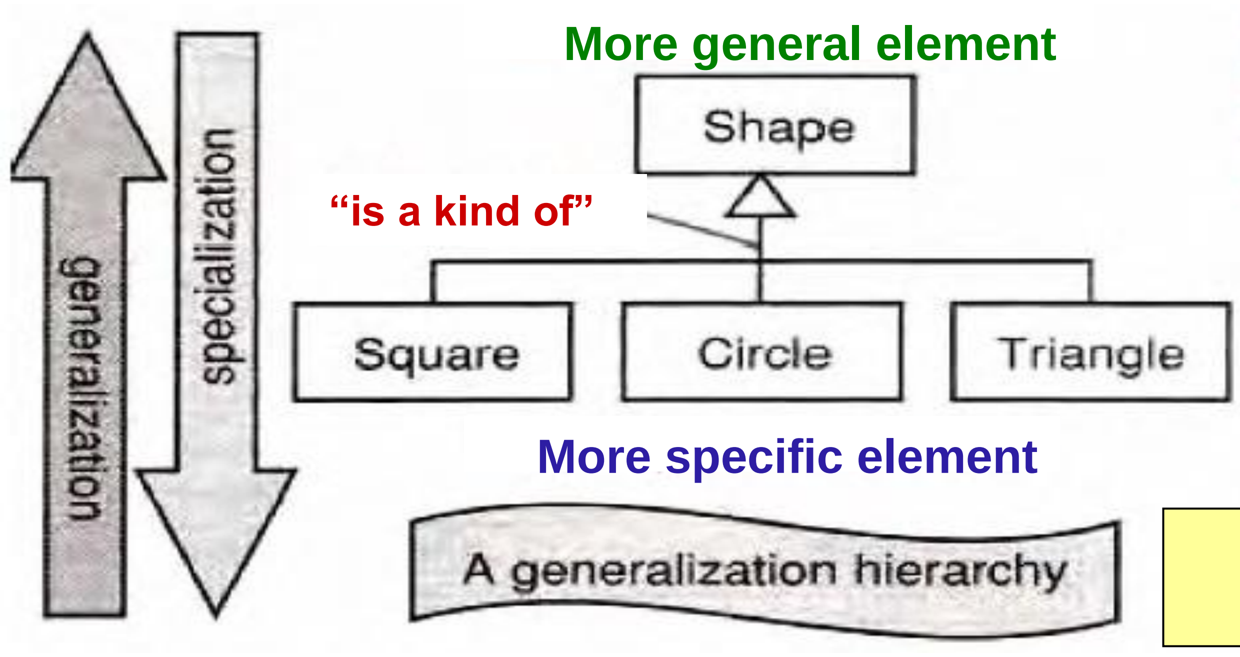
With a qualifier



GENERALIZATION

- Generalization is a relationship between a more general thing and a more specific thing:
  - The more specific thing is consistent in every way with the more general thing.
  - The substitutability principle states that you can substitute the more specific thing anywhere the more general thing is expected.
  - Generalization is a bottom-up process
- Generalization hierarchies may be created by generalizing from specific things or by specializing from general things.
- "is kind of" or "is type of" relationship.

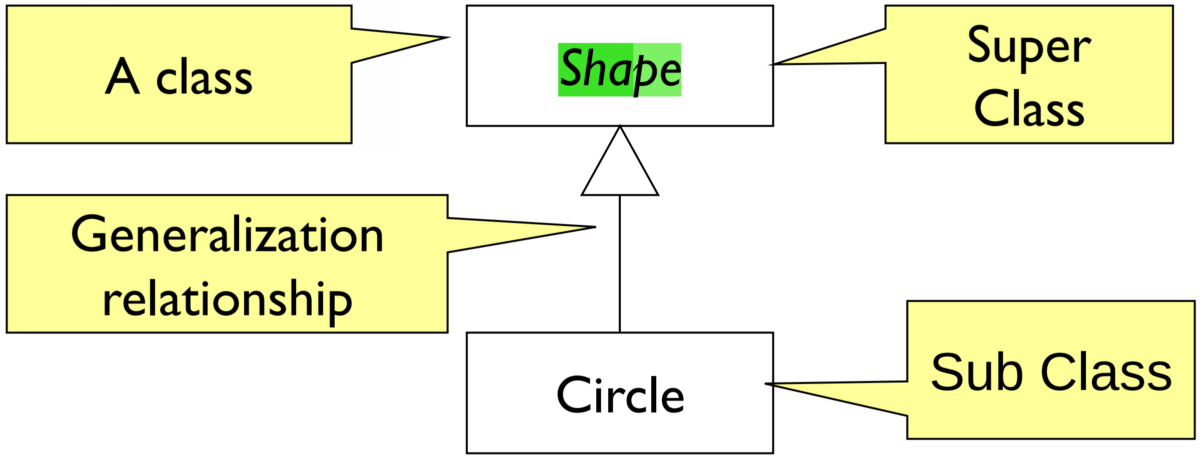
GENERALIZATION



Parent
Superclass
Ancestor
Base Class

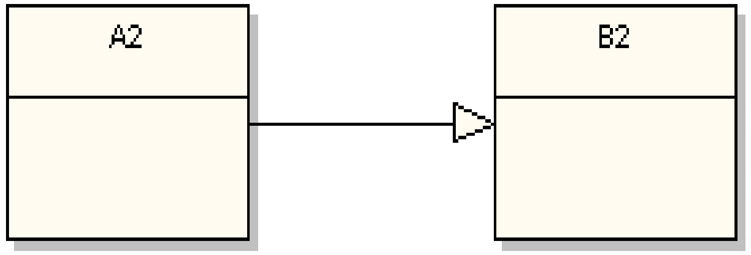
Child
Subclass
Descendant
Leaf / Derived

- **{abstract}** is a tagged value that indicates that the class is abstract (no object).
- The name of an abstract class should be italicized

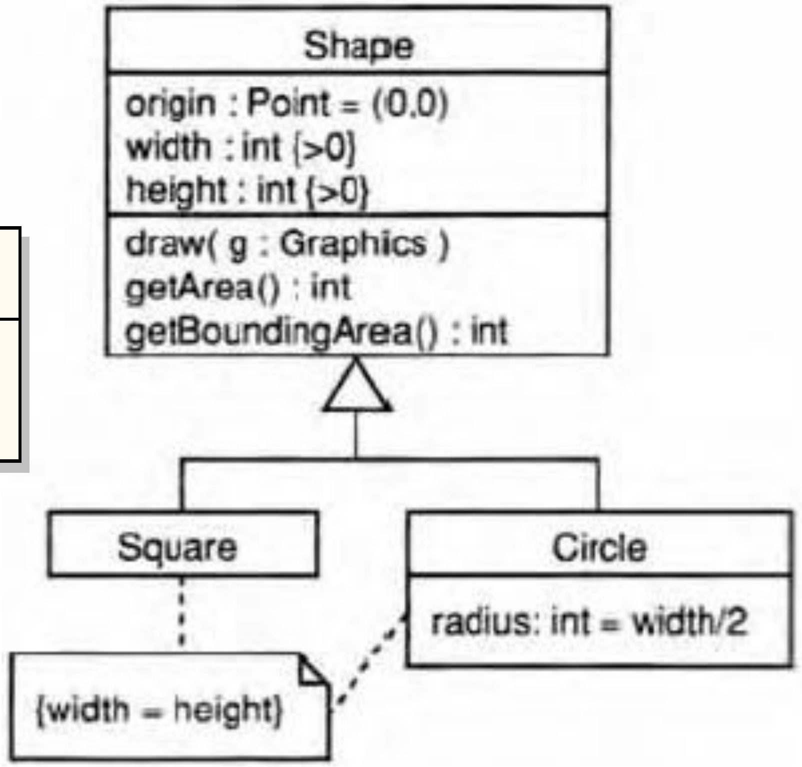


INHERITANCE

- A sub-class inherits from its super-class
  - Attributes
  - Operations
  - Relationships
- A sub-class may
  - Add attributes and operations
  - Add relationships
  - Refine (override) inherited operations



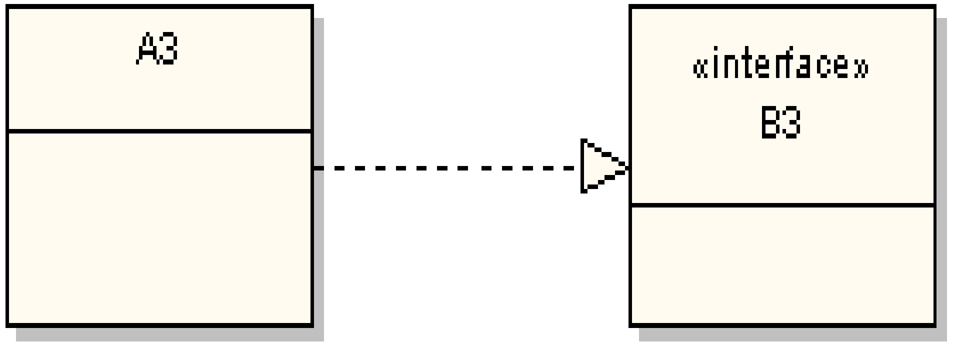
```
class A2 extends B2
```



- Super classes may be declared {abstract}, meaning they have no instances
- A generalization relationship **may not** be used to model interface implementation (realization)

REALIZATION

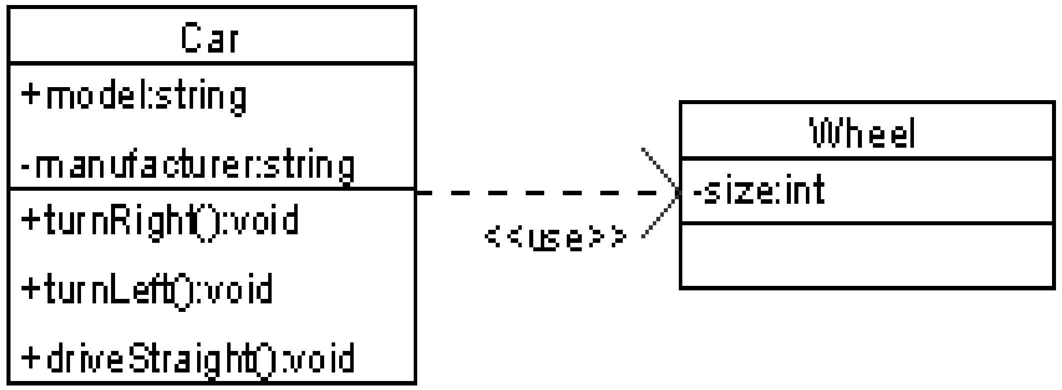
- A realization relationship indicates that one class implements a behavior specified by another class (an interface).
- An interface can be realized by many classes. In another words, a class may realize many interfaces.



```
class A3 implements B3
```

DEPENDENCY

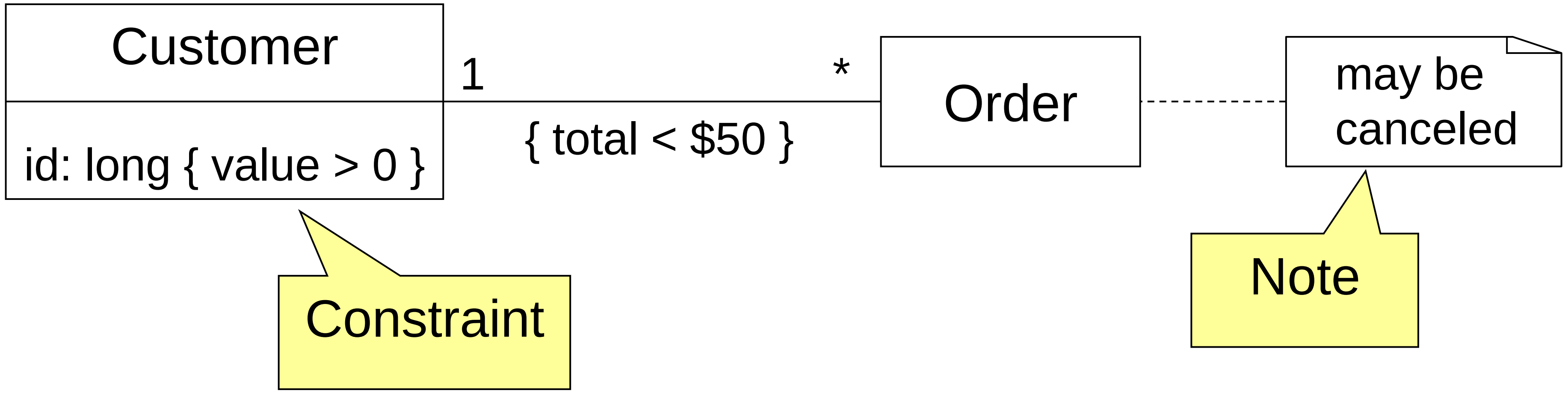
- A dependency indicates a semantic relation between two or more classes in which a change in one may force changes in the other.



Car depends on wheel

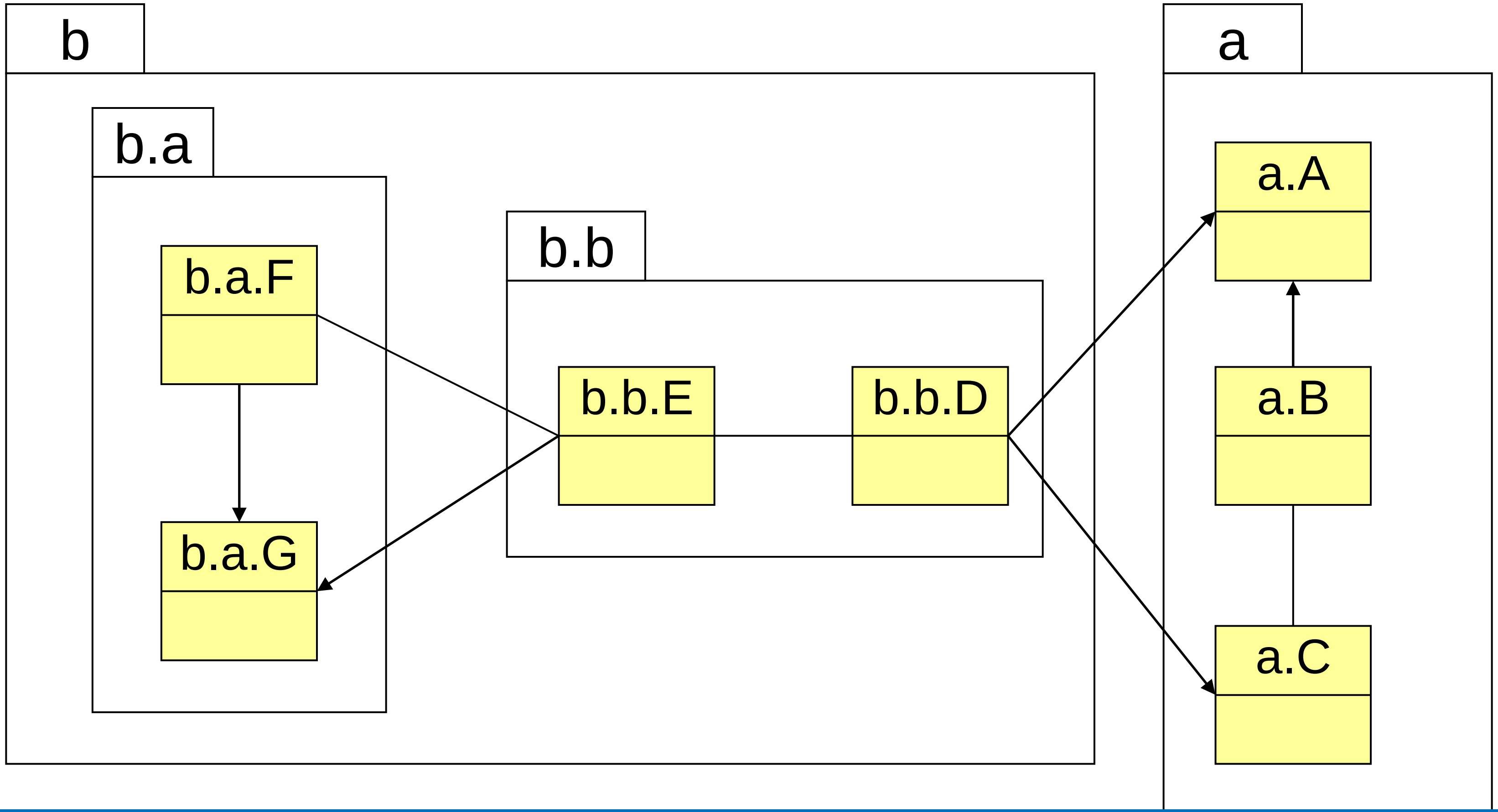
CONSTRAINT RULES AND NOTES

- **Constraints** and **notes** annotate among other things, e.g., associations, attributes, operations, and classes.
- Constraints are semantic restrictions noted as Boolean expressions.
- UML offers many pre-defined constraints.



LOGICAL DISTRIBUTION OF CLASSES

- Add package information to class diagram



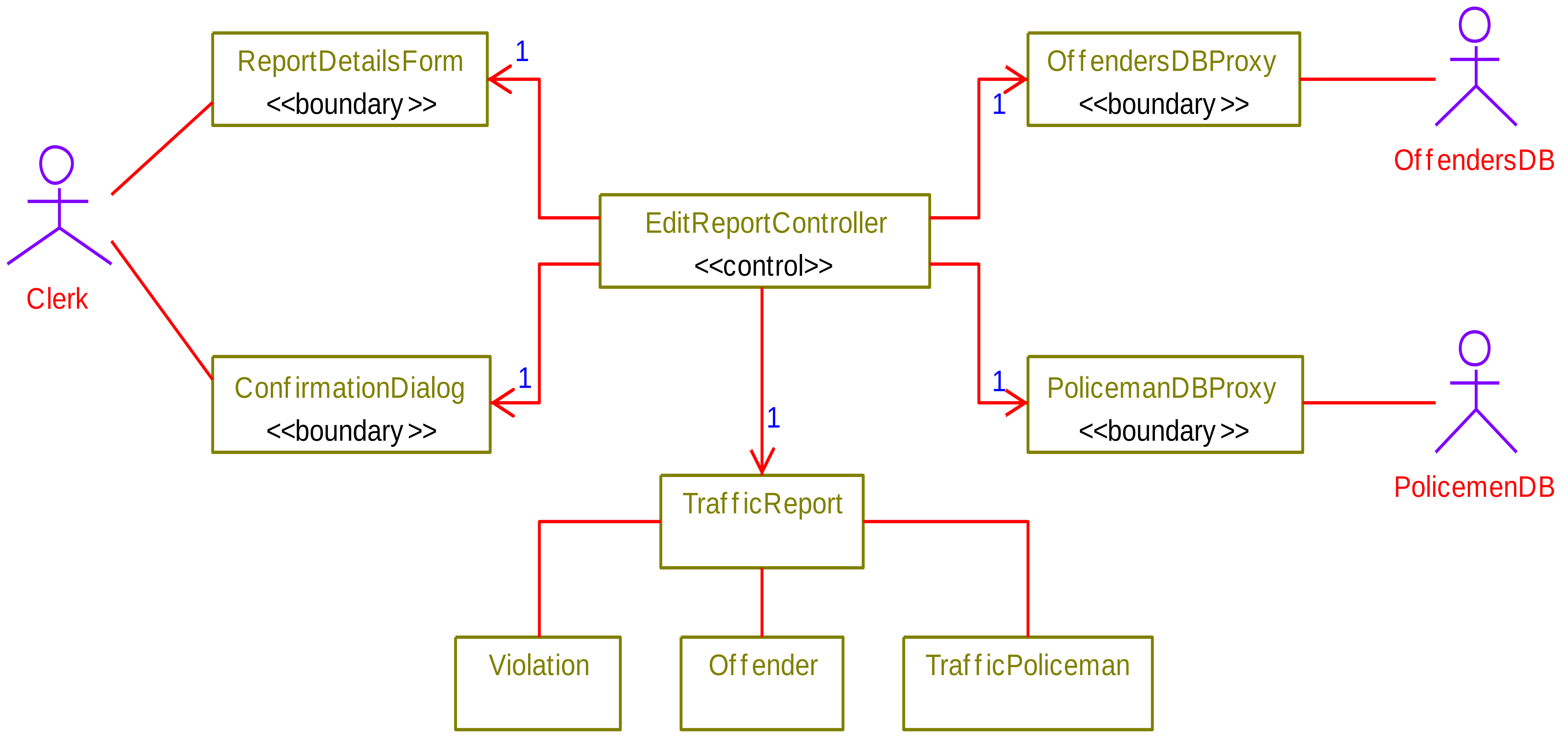
CLASSES CATEGORIZATION

- Boundary Classes
  - Models the interaction between the system's surroundings and its inner workings
  - User interface classes, Concentrate on what information is presented to the user, don't concentrate on user interface details
  - System / Device interface classes, concentrate on what protocols must be defined. don't concentrate on how the protocols are implemented
- Entity Classes
  - Models the key concepts of the system
  - Usually models information that is persistent
  - Contains the logic that solves the system problem
  - Can be used in multiple behaviors

CLASSES CATEGORIZATION

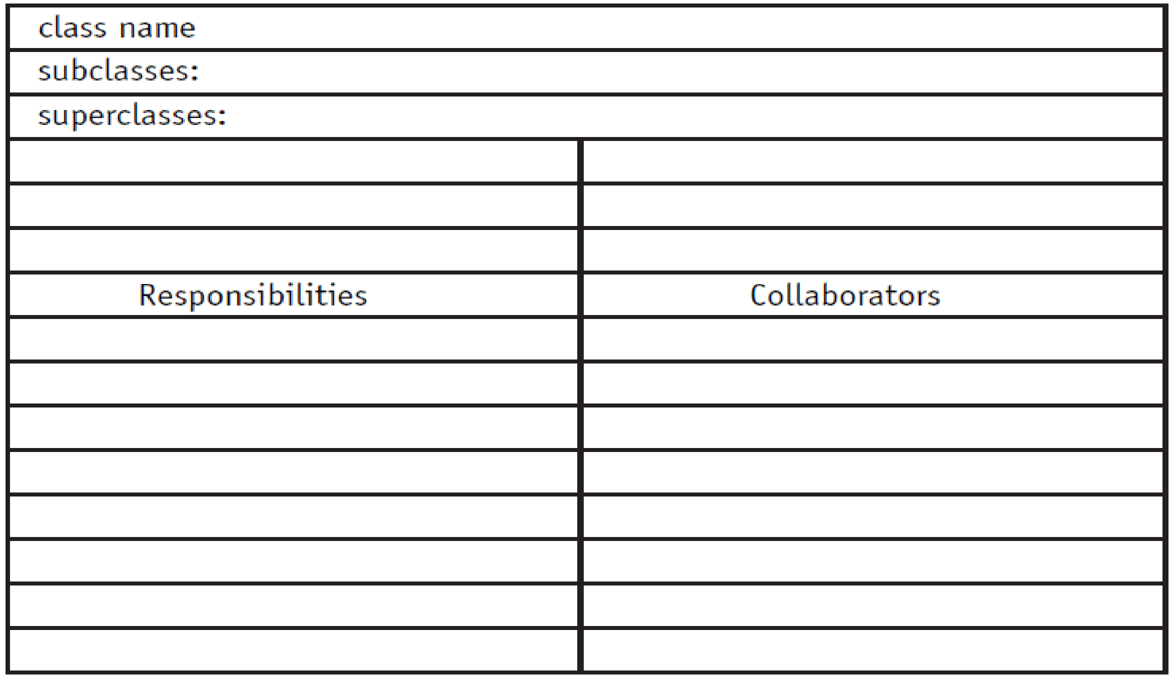
- Control Classes
  - Controls and coordinates the behavior of the system
  - A control class should tell other classes to do something and should never do anything except for delegating (directing) the work to other classes
  - Control classes separate boundary and entity classes

CLASSES CATEGORIZATION



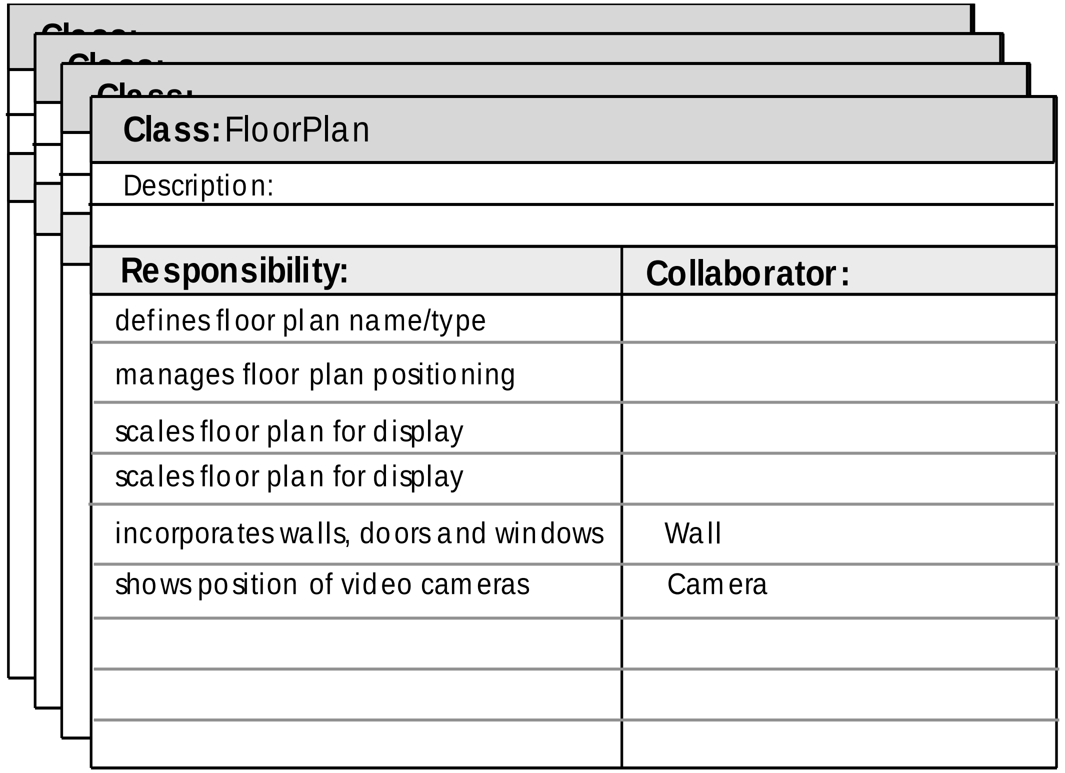
CRC CARD

- Class Responsibility Collaboration
- CRC goals: provide the simplest possible conceptual introduction to OO design



class name	
subclasses:	
superclasses:	
Responsibilities	Collaborators

Figure 2-2 A CRC card sample



Class: FloorPlan

Description:

Responsibility:	**Collaborator:**
defines floor plan name/type	
manages floor plan positioning	
scales floor plan for display	
scales floor plan for display	
incorporates walls, doors and windows	Wall
shows position of video cameras	Camera

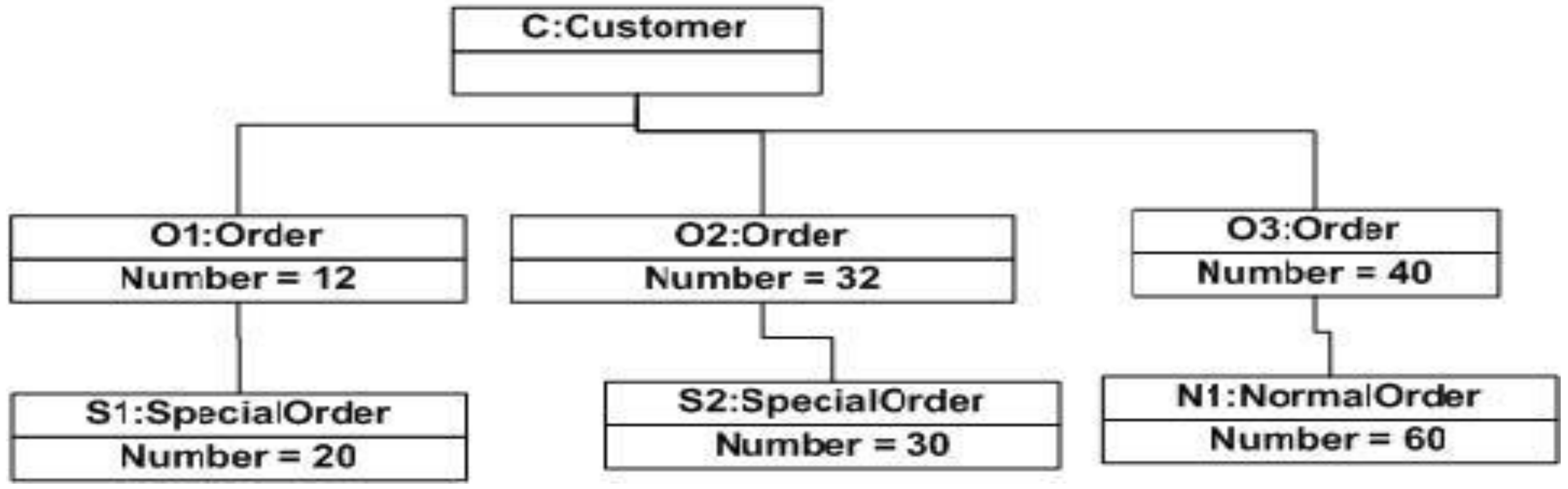
CRC CARD

- A CRC card is a 3-x-5" or 4-x-6" lined index card.
- The physical nature of the cards emphasizes the division of responsibility across objects.
- The physical size of the cards also helps to establish limits for the size and complexity of the classes.
- The CRC card technique does not use the UML, instead it is used to discover information about classes that is then placed into a UML Class diagram.
- The body of the card is divided in half.
  - The left column/half lists the responsibilities of the class
  - The right column/half lists the other objects that it works with, the collaborators, to fulfill each responsibility.

CLASS VS. OBJECT DIAGRAM

- The class defines the *rules*; the objects express the *facts*. The class defines what *can be;* the object describes *what is.*
- If the Class diagram says, "*This is the way things should be*" but the Object diagram graphically demonstrates that "*it just ain't so*" then you have a very specific problem to track down.
- The Object diagram can confirm that everything is working as it should.

Object diagram of an order management system



REFERENCES

❑ Booch, G., Rumbaugh, J. & Jacobson, I. (2005). *The unified modeling language user guide*. Pearson Education India.

CASE STUDIES

- Case 1: Draw the Class notation for Class Named **'Book'**.

The private attributes of the class are:	Public methods of this class are:
bookID (Numeric; assigned by system, default value: none), bookName (String), authorName(String) , bookCopy(Numeric), totalBook(Static and numeric)	setBookID(Numeric; assigned by system, default value: none), setBookName(String), setAuthorName(String) , addBookCopy(Numeric ,default value: zero), deleteBookCopy(Numeric), showBookInfo (void)

CASE STUDIES

- Case 2: Draw the Class diagram for the following scenario.

  Student may attend any number of courses
  Every course may have any number of students
  Instructors teach courses
  For every course there is at least one instructor
  Every instructor may teach zero or more courses
  A school has zero or more students
  Each student may be a registered member of one or more school
  A school has one or more departments
  Each department belongs to exactly one school
  Every instructor is assigned to one or more departments
  Each department has one or more instructors
  For every department there is exactly one instructor acting as the department chair

CASE STUDIES

- Case 3: Draw the Class diagram for the following scenario.

A building is owned by one person
A person may own more than one building
Each building is either a house or an apartment
An apartment contains two or more Rental Units
It is possible to buy a house, and rent or vacate a Rental Unit

CASE STUDIES

- Case 4: Draw the Class diagram for the following scenario.

A company has one or more divisions
A division has one or more departments
A company can have one or more persons working as employees
A person is associated with the company as an employee
A department can have one or more persons assigned to it
A person is assigned to one department

CASE STUDIES

- Case 5: Draw the Class diagram for the following scenario.

The library contains books and journals. It may have several copies of a given book. Some of the books can be reserved for short-term loans and others for long-term. Member of library can borrow books. There is a special type of member- Member of Staff, who can borrow the journals. The system must keep track of when books and journals are borrowed.

CASE STUDIES

- **Case 6:** Draw the Class diagram for the following scenario.

A nonprofit organization depends on a number of different types of persons for its successful operation. Three types of persons are of greatest interest: employees, volunteers, and donors. Employees and volunteers work on various projects organized by the organization. One project consists of many sections. A volunteer works under the supervision of an employee. An employee is responsible to write reports on the project at the completion of the project. To become a recognized donor, a donor must have donated one or more items, and an item may have no donors, or one or more donors. Projects and events are also sponsored by sponsors. Sometimes a single project or event is sponsored by multiple sponsors. An event is made of various activities.