Object-Oriented Design

CSCE 247 - Lecture 16 - 03/20/2019

Objectives for Today

- Introduce object-oriented design.
  - Design the system based on interactions between entities.
- UML Class Diagrams
  - Visualization of the static structure of the classes and their relationships.

Common Problems

- The requirements are wrong.
  - Incomplete, ambiguous, inconsistent
  - Developer and customer had different interpretations.
- Requirements drift
  - Requirements tend to change often.
  - Leads to late design changes.
- The result - **continual change**
  - Functionality changes often.
  - Many of these changes come late in the project.
  - Many changes during maintenance.

The Solution

- **Good:** Rigorous requirements and planning stages.
  - Make sure stakeholders and developers are on the same page.
- **Better:** Structure the system to accommodate change.
  - Isolate parts that are likely to change.
  - Modularize so changes are contained.
  - Attempt to not compromise the system structure during change.

The Object-Oriented Solution

The problem domain is relatively consistent.

- Creating ID Cards
  - Assemble data based on selected options, place in correct position on card layout.
- Autopilot System
  - Get the plane from point A to point B using available control options.
- Word Processor
  - Style text using user-selected options, render the document as it would appear once printed.

The Object-Oriented Solution

Changes: functionality, data representation.

- Creating ID Cards
  - Type of information and where it is placed changes.
  - New types of ID may need to be added.
- Autopilot System
  - Hardware interfaces need to adapt to new airplanes.
  - Operation options may evolve over time.
- Word Processor
  - New style options and templates added over time.
  - New document types supported (HTML, XML, etc.)

The OO Approach:

Structure the system based on the abstract concepts of the problem domain, not the concrete instantiations.

What is OO Design?

OO design is a way of thinking about a problem based on abstractions of concepts (entities) that exist in the real world.

OO design is not the same as programming in an OO language.

- Can reason about entities and relationships even when programming in C, Fortran, etc.
- OO languages do not ensure OO design.

Viewpoints of OO Analysis

Static Models:

- Describe the structure of the entities in the system.
  - Individual entities (attributes and operations).
  - Relationships between entities (association and inheritance).
  - Clustering of entities into logical subsystems.

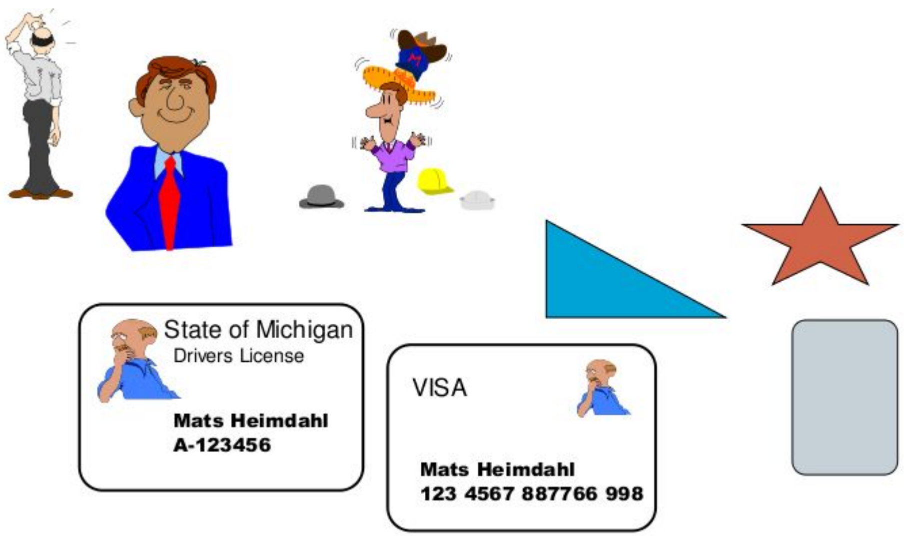
Dynamic (Behavioral) Models:

- Describe sequences of interactions between object instantiations during execution.
  - Show changes to attributes.
  - Model the control aspects of the system.

The OO Solution

- The design should be organized as a collection of objects that model concepts in the problem domain.
  - Concrete concepts in the real world
    - A driver’s license, an aircraft, a document…
  - Logical concepts
    - A scheduling policy, conflict resolution rules...
- What defines an object:
  - Data representation
    - Characteristics that define an object (attributes).
  - Functionality
    - What the object can do (operations).

Card Entities



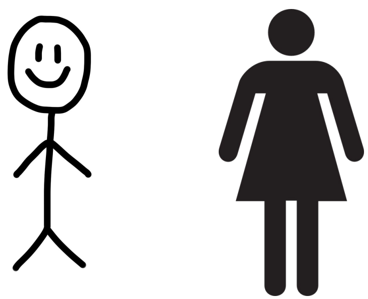
You are building a system that can print different types of card (ID, license, credit cards).

What are some of the entities that make up this problem domain?

How do these entities relate?

Attributes and Operations

Person Objects



abstracts to

Attributes

- Name
- Age
- Height
- Weight
- Address
- Role

Operations

- Edit Information
- Change Role

Card Objects

Attributes

- Owner
- Layout
- ID Number
- Expiration Date

Operations

- Issue
- Edit Information
- Renew
- Retract

Objects vs Classes

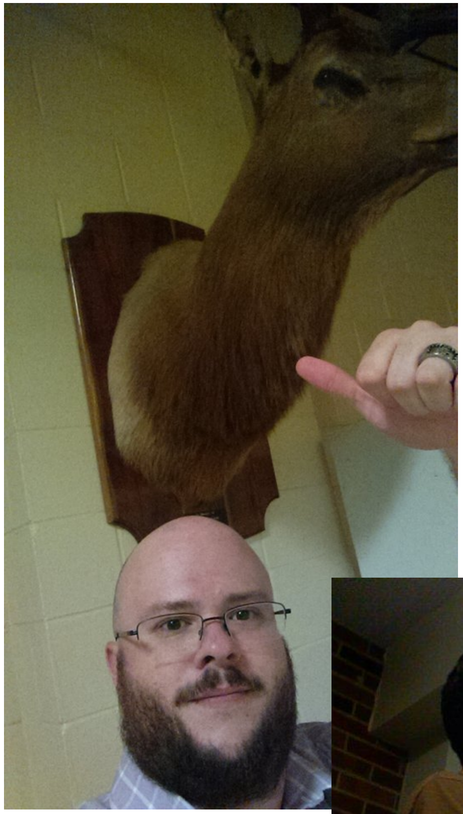
- Objects are concrete entities that make sense in the application domain:
  - Greg Gay
  - Greg’s credit card
  - Greg’s driver's license
- All objects have an identity and are distinguishable
  - Greg’s credit card vs Jason’s credit card
- Not an object:
  - Person
  - Driver’s License

Classes

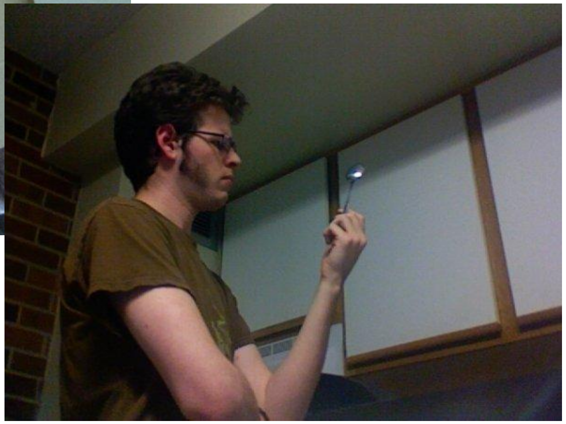
- Describes a **type** of object.
  - **Objects are instances of classes.**
  - Each instance has the same attributes and behaviors, the same relationships to other classes, and common meaning.
  - Each instance may have different values for those attributes.
- Person instances:
  - Greg Gay, Jason Biatek
- Credit Card instances:
  - Greg's credit card, Jason's credit card

Objects Characteristics

- Objects have a classification.
  - Objects are instances of classes.
  - Each instance has the same structure and behavior.
- Objects have identity.
  - Discrete and distinguishable entities.



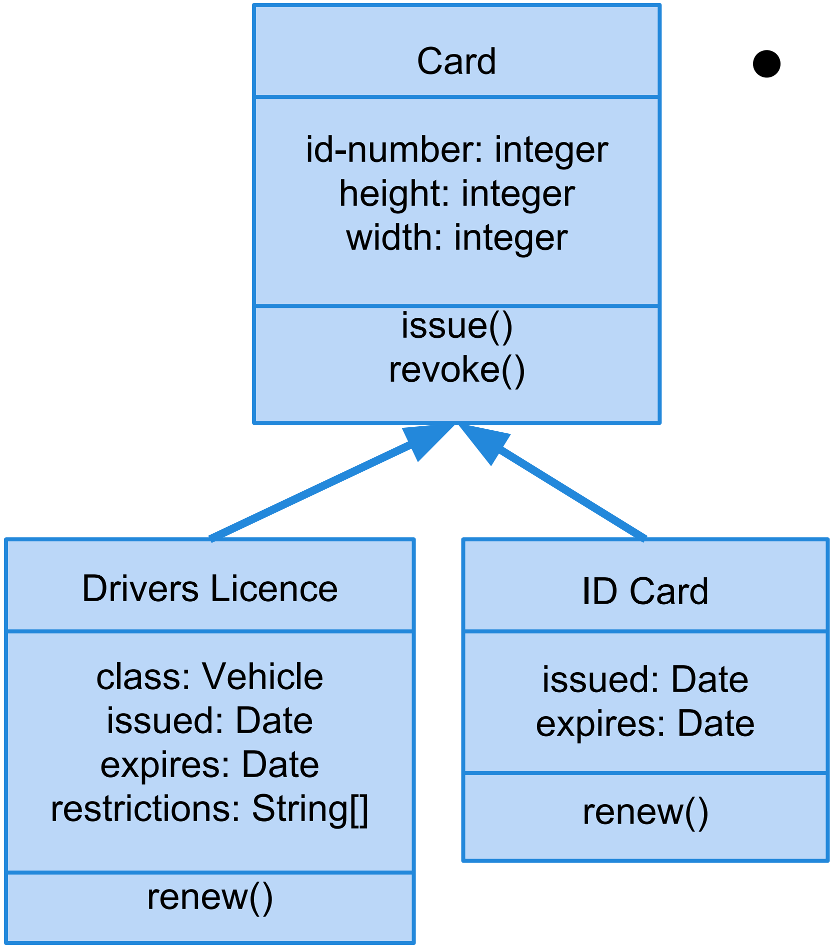
!=



Objects vs Classes

- Classes are used in **static** views of a domain or system.
  - Classes are defined in the source code.
  - When we design the system structure, we don’t care about Greg. We care about what defines any abstract Person.
- Objects are used in **dynamic** views of a domain or system.
  - Objects represent the system state during runtime.
  - When the system is running, we care about Greg’s state and behavior, not an abstract Person.

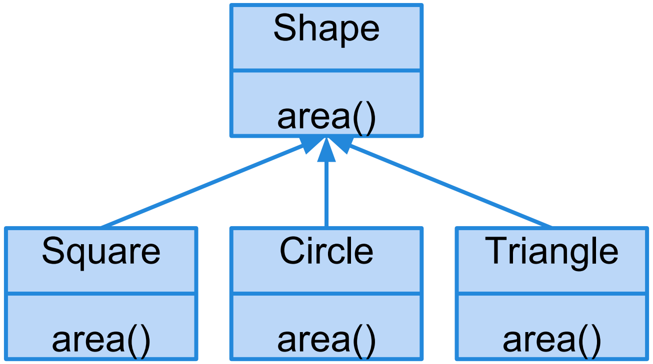
Inheritance



- Child classes share attributes and operations based on a hierarchical relationship.
  - Allows the creation of specialized subclasses without reimplementing functionality or including attributes and operations where they aren't needed.
  - Objects instantiated from a child are instances of that class and of the parent class.

Polymorphism

- The same operation may behave differently when used on different classes.
    - Specifically, we can *redefine operations* in each related class.
- Because Shape defines an area() method, we know all children offer that method.
    - But, we can redefine that method in each child to offer the right answer.



Because objects are instances of both their class and their parent class:

```
void getArea(Shape s){
        System.out.println(s.area());
}
```

Gives the right answer if a square, circle, triangle, etc is passed in.

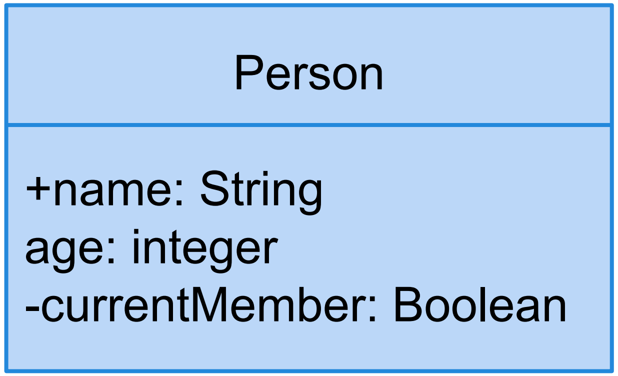
Class Diagrams

Visualize system structure: classes and how they relate.

Class Diagrams

Class Diagram:

Used to describe class with attributes.



Attributes are variables

- That describe the instantiated object.
- That are used by objects to perform operations.

Include the data type, and (optionally) a symbol to indicate visibility:

- + (public), - (private), # (protected), ~ (package-level)

Operations

Operations are transformations that can be applied to or performed by an instance of a class.

Card
height: integer thickness: integer -id-number: integer
issue() revoke()

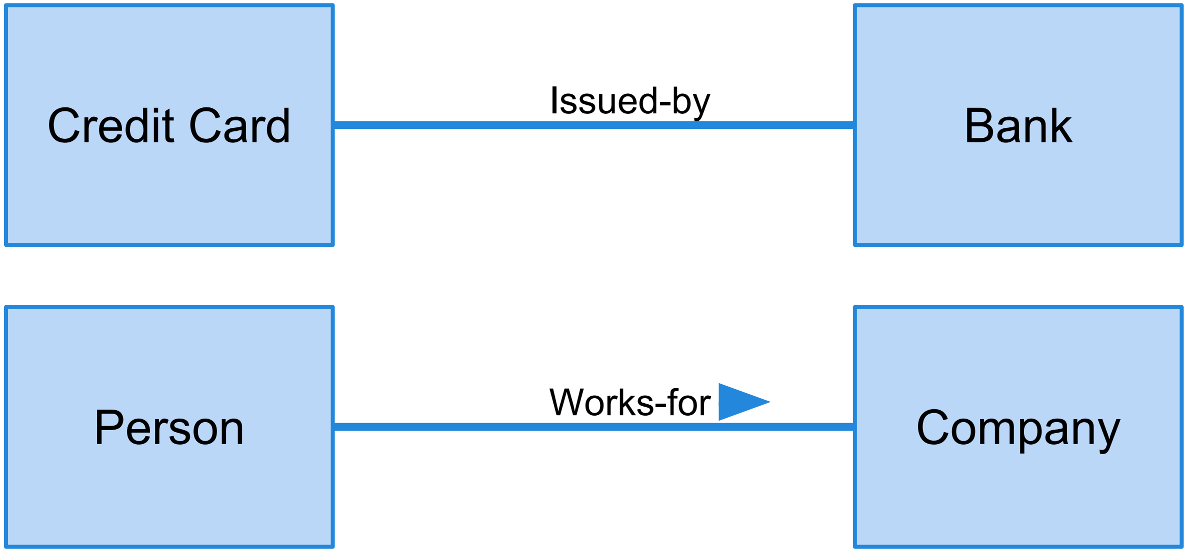
Operations may have arguments.

Shape
height: integer width: integer
rotate(angle: integer) move(x: integer, y: integer)

Associations

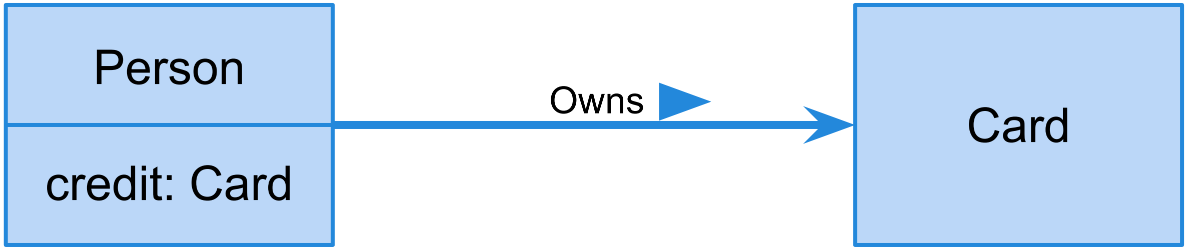
A conceptual connection between classes.

- A credit card is issued by a bank.
- A person works for a company.

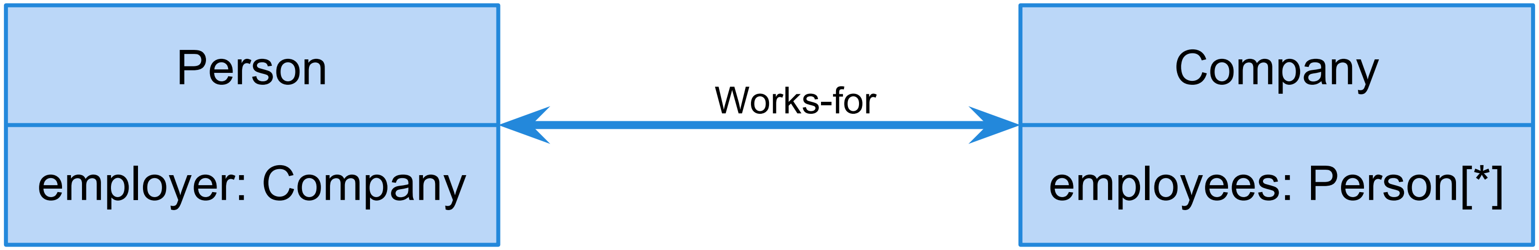


Associations Can Have Direction

Direction on an association indicates control.
Which object owns and calls on the other?



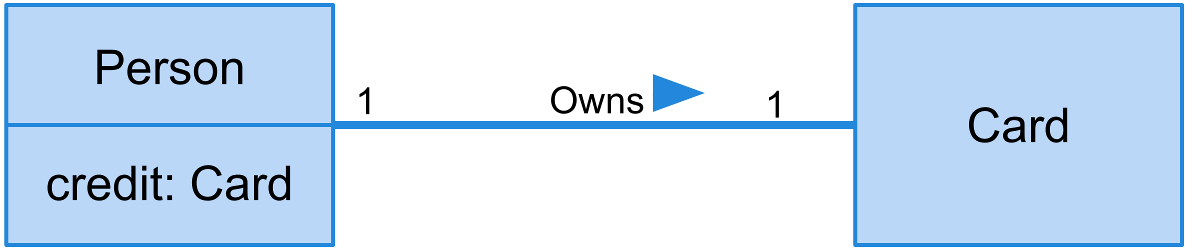
Associations can be bidirectional.



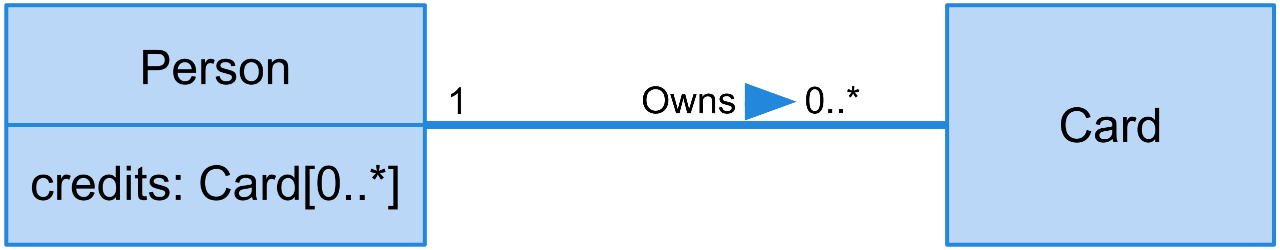
Associations Have Multiplicity

Associations should be labeled with how many instances of a class are expected on each side.

- One Person owns one Card



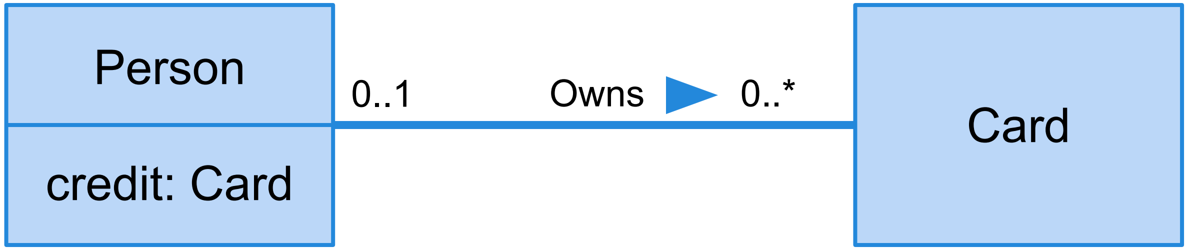
- One Person can own zero or more cards



Multiplicity

Defined with a lower and upper bound.

- One Person can own zero or more Cards.
- Each Card is owned by zero to one Person.



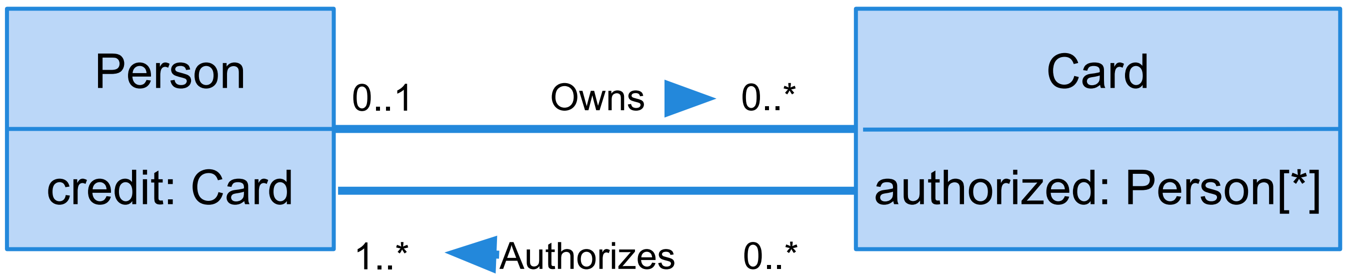
Common terms that imply multiplicity:

- **Optional:** implies lower bound of 0.
- **Mandatory**: implies lower bound of 1 or more.
- **Single-Valued**: implies upper bound of 1.
- **Multivalued**: implies an upper bound > 1 (often *).

Multiple Associations

Can have multiple associations between objects, each with their own multiplicities.

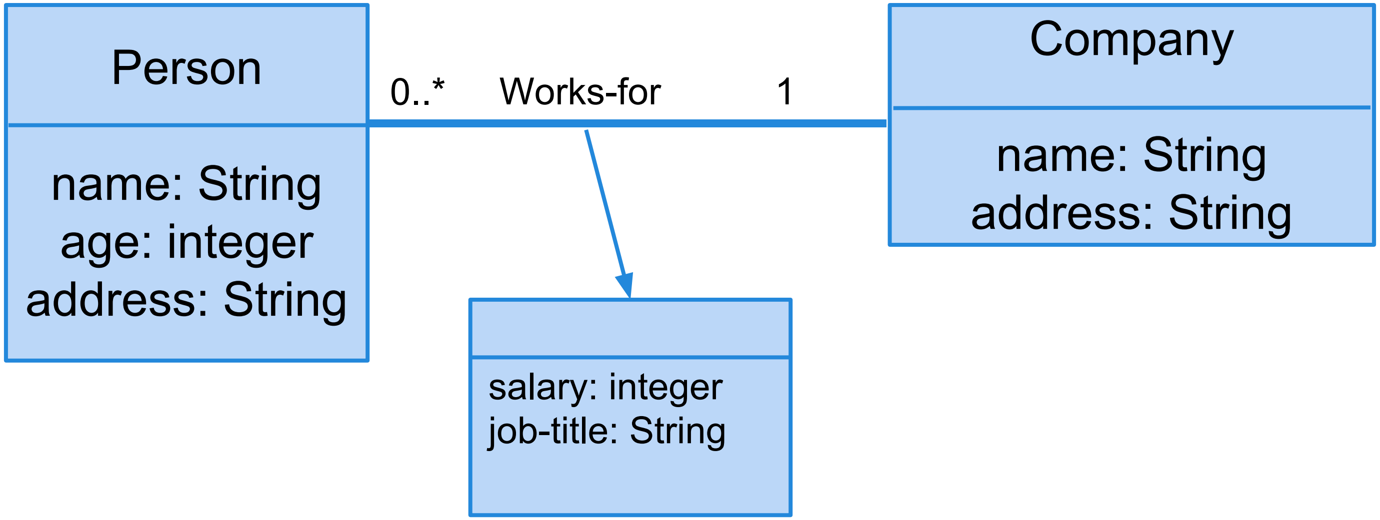
- One Person can own zero or more Cards.
- Each Card is owned by zero to one Person.
- Each Card has one or more authorized users.
- One Person can be authorized to use zero or more Cards.



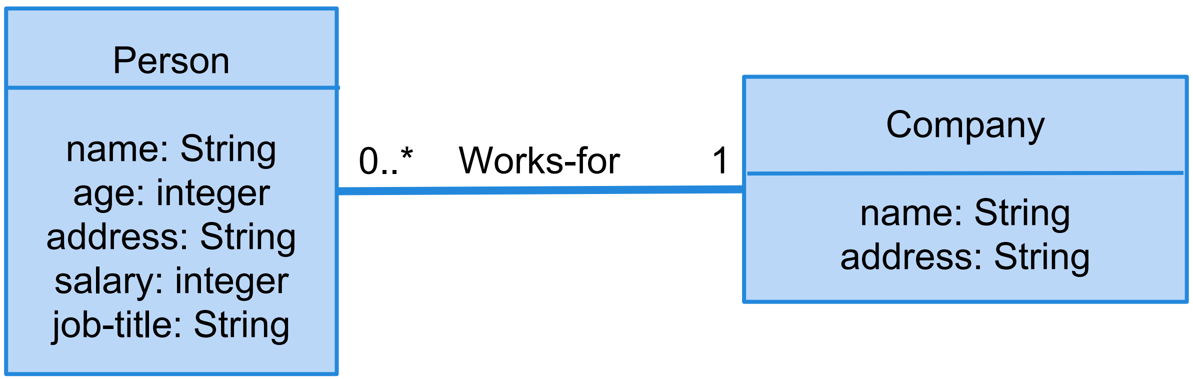
Link Attributes

Associations can have attributes just like classes can have attributes.

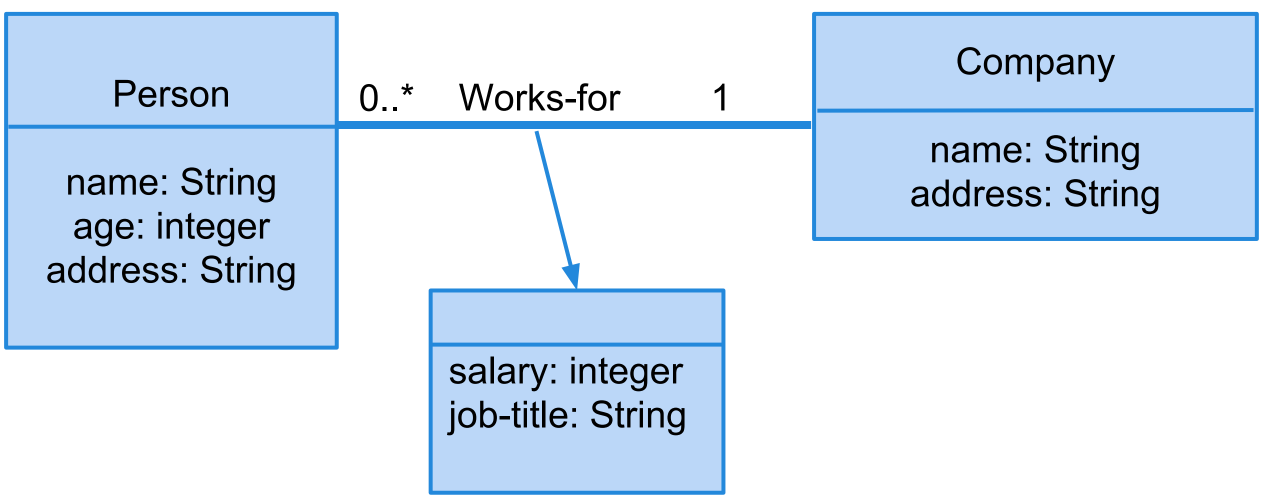
- How do you represent salary and job title?



Folding Link Attributes into Classes



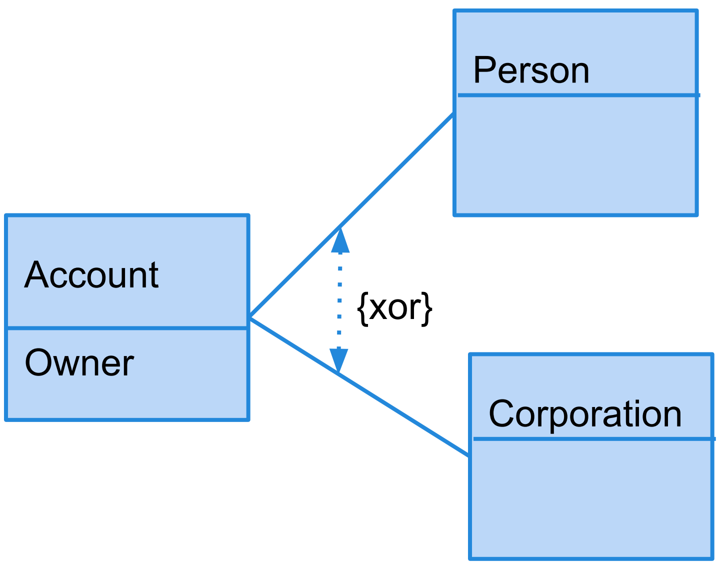
Why not this?



Association Constraints

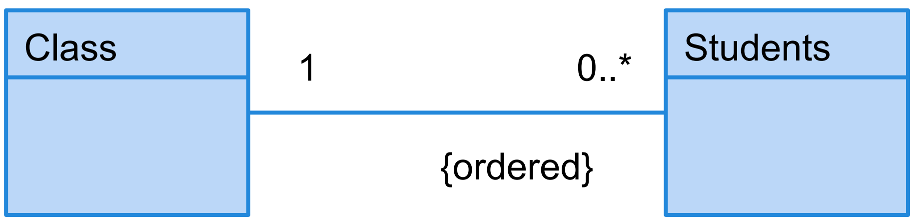
General Constraints:

On one association or between multiple. Plain English. Use dotted line to show dependency.



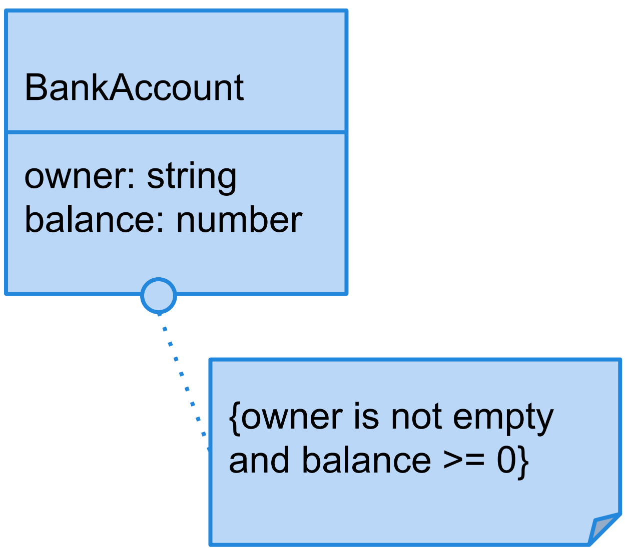
Ordering:

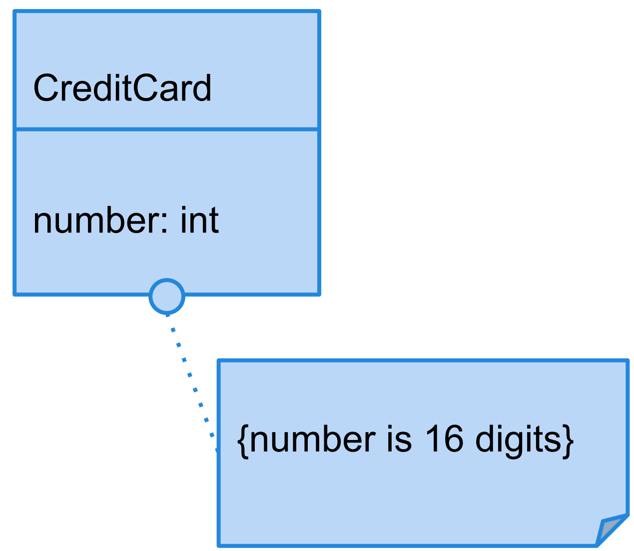
On one association. Implies that objects on the “many” side must be ordered.



Attribute Constraints

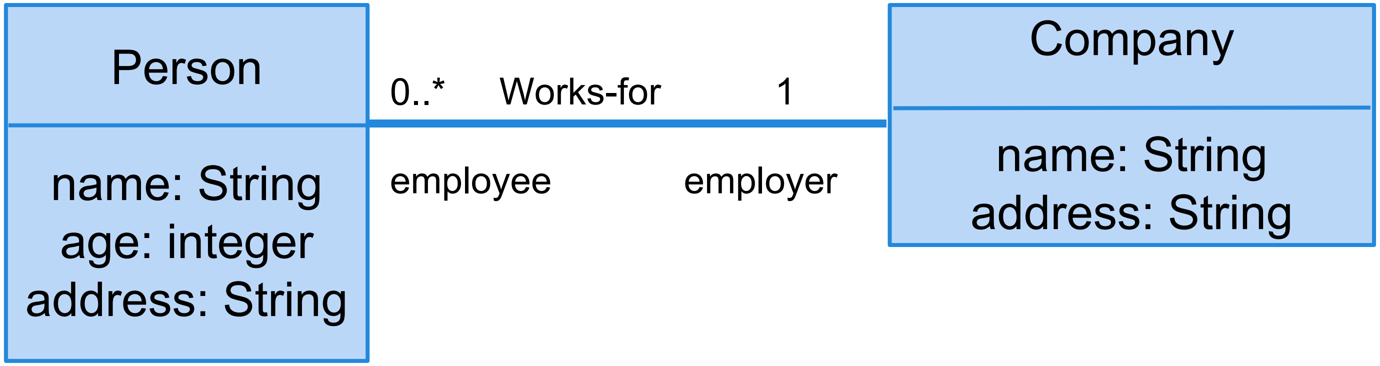
General Constraints: Plain English. Can be constraints on an attribute or on multiple related attributes.





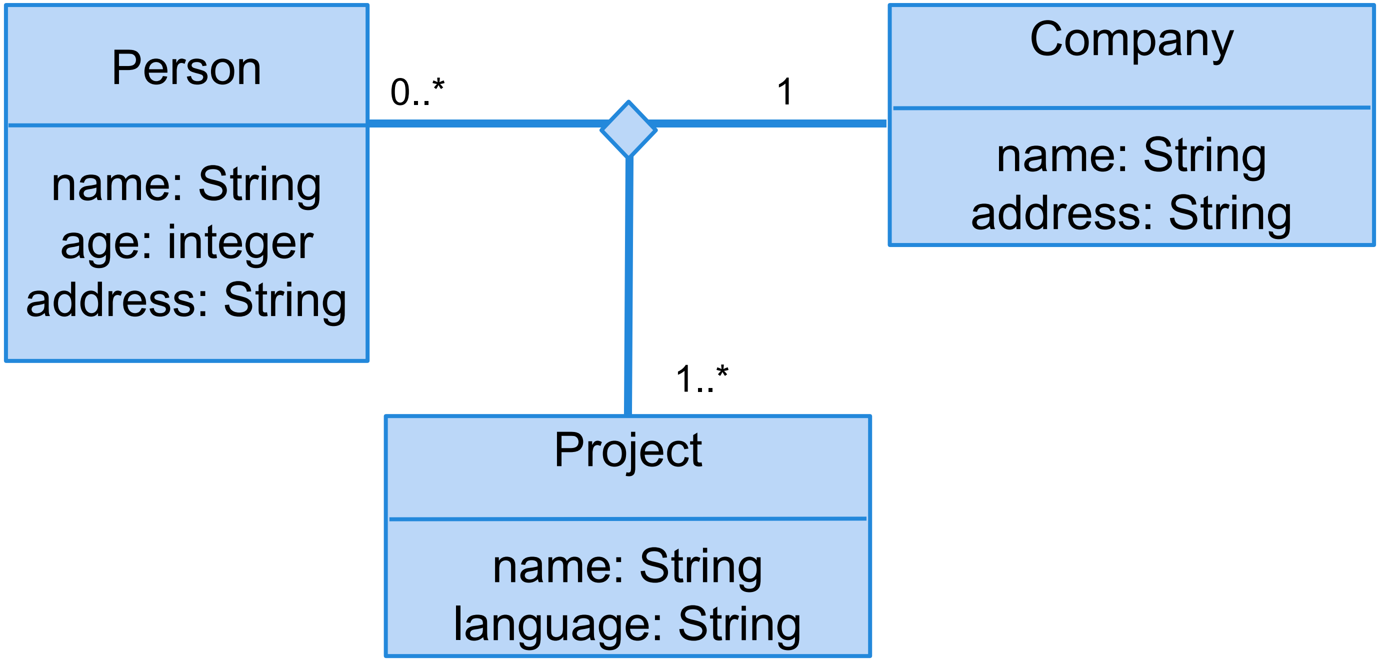
Role Names

Attach names to the ends of an association to clarify its meaning.



Higher Order Associations

Associations can be between more than two classes.



Aggregation

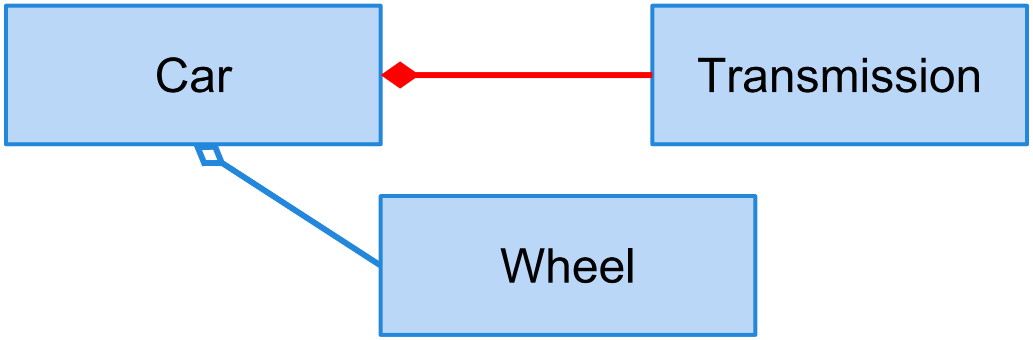
A special type of association. Indicates membership.

- A sentence is part of a paragraph.
  - (A paragraph consists of sentences.)
- A paragraph is part of a document.
  - (A document consists of paragraphs.)

Composition

A **stronger** type of aggregation.

- Aggregation indicates membership. Member objects can exist outside of the owner.
- Composition indicates dependence. The instance is destroyed if its owner is destroyed.

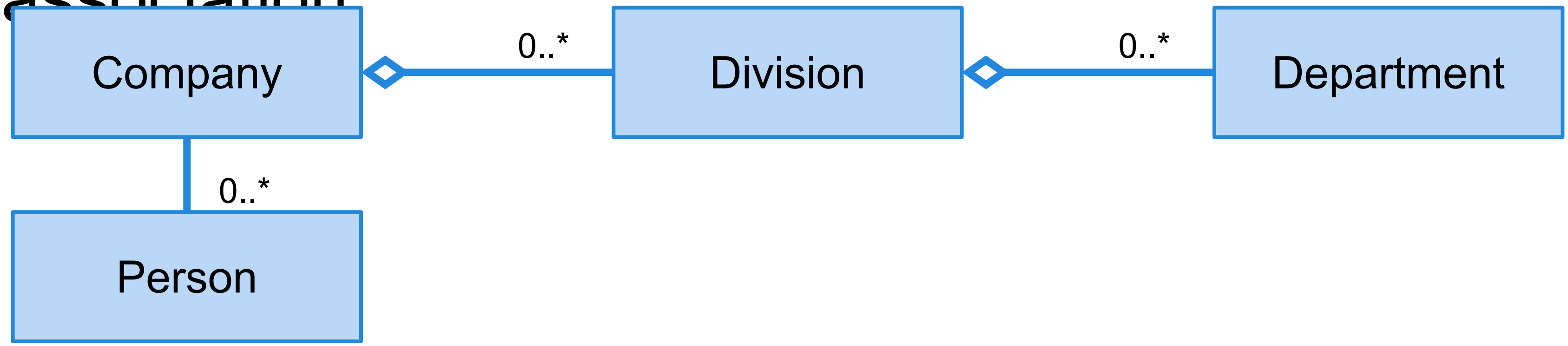


Aggregation vs Association

When should you use a plain association versus an aggregation?

- Can you use the phrase “is made of”?
- Are operations automatically applied to the parts?

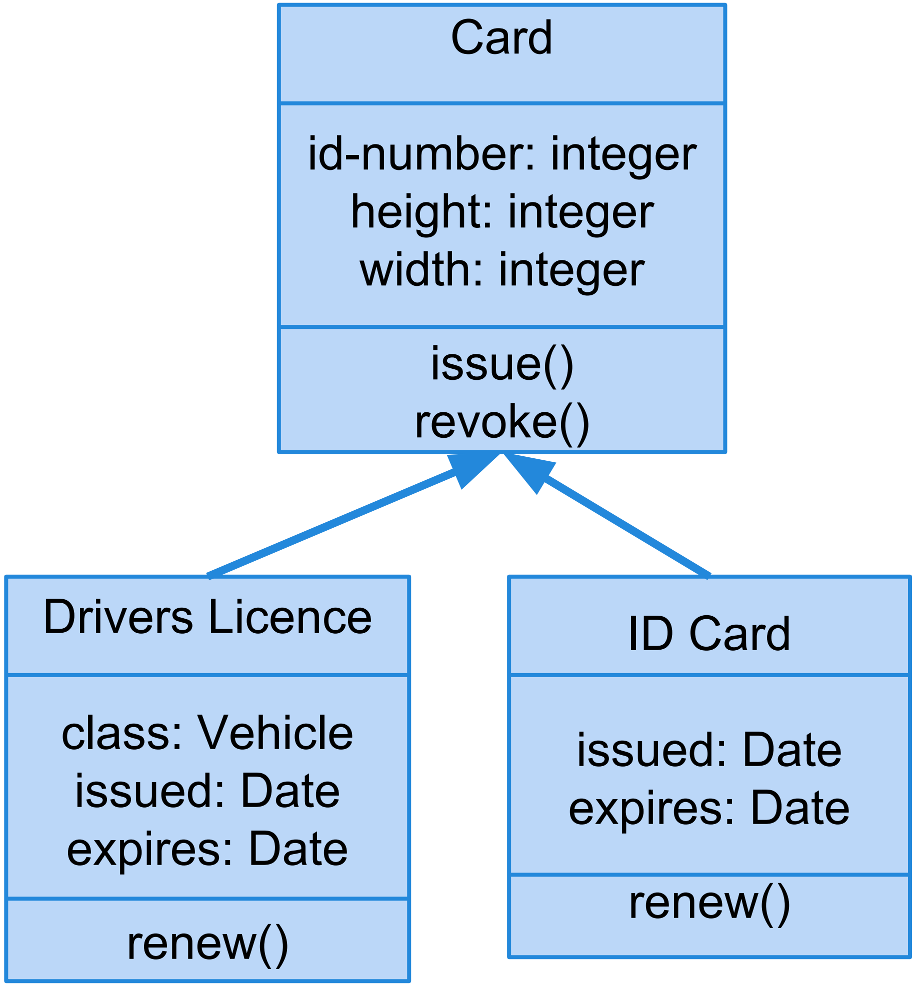
Then use aggregation. If not clear, use association.



Inheritance

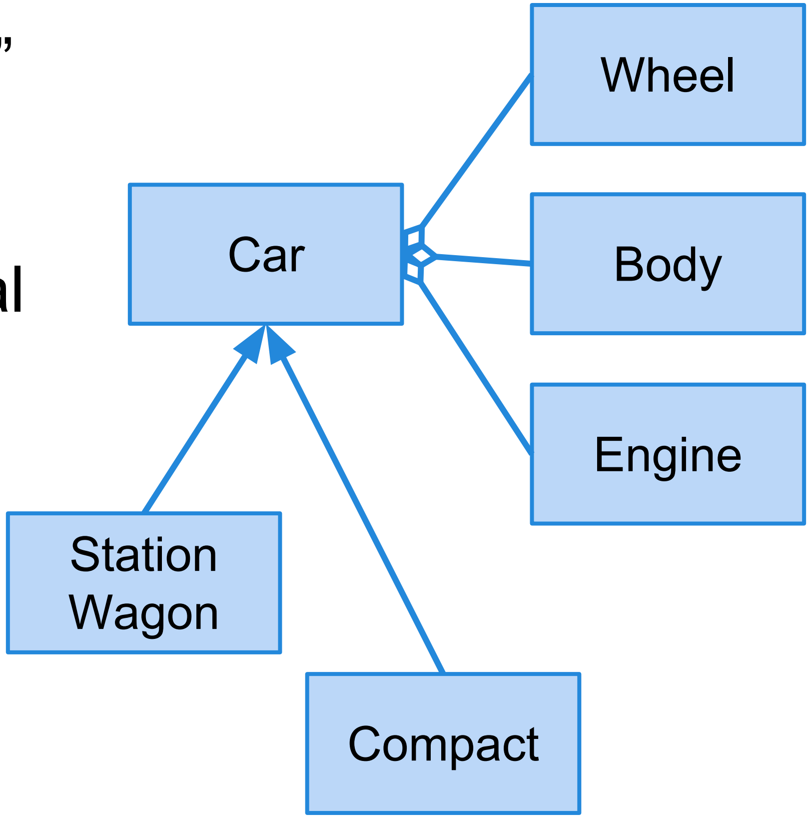
The is-a association.

- Cards have many properties in common.
- Generalize the common properties as a base class.
- Let all card types inherit the common attributes and add their own (Drivers License is-a Card)

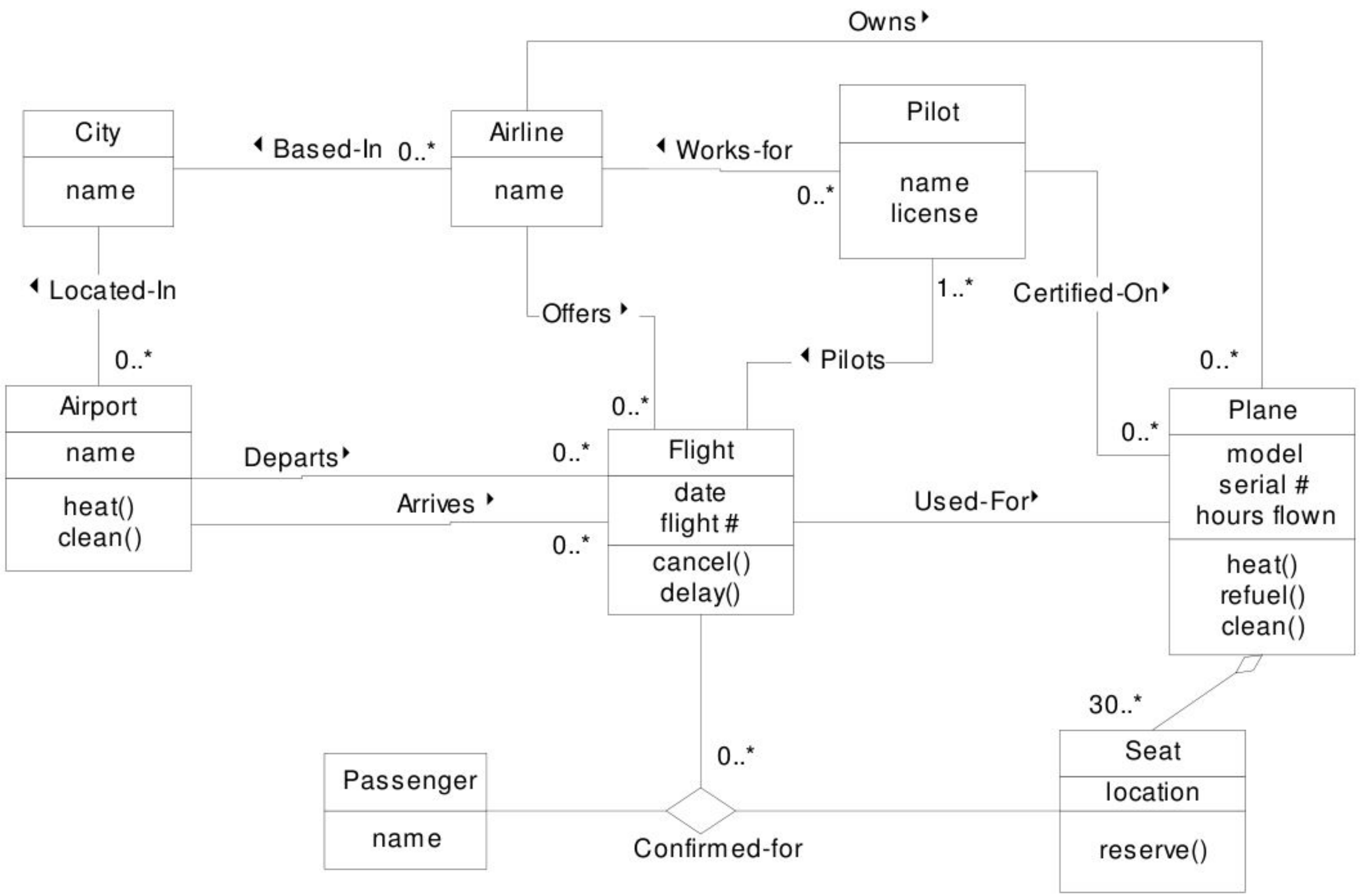


Aggregation Versus Inheritance

- Do not confuse “is-a” (inheritance) with “is-part-of” (aggregation).
- Use inheritance for different special versions of a general concept.
- Use aggregation to indicate components of a whole.



Example



Owns
City
name
Based-In
0..*
Airline
name
Works-for
Pilot
name
license
0..*
Located-In
Offers
1..*
Certified-On
Pilots
0..*
0..*
Airport
name
heat()
clean()
0..*
Plane
model
serial #
hours flown
heat()
refuel()
clean()
0..*
Departs
0..*
Flight
date
flight #
cancel()
delay()
Arrives
Used-For
0..*
30..*
0..*
Seat
location
reserve()
Passenger
name
Confirmed-for

Examples

Draw a class diagram for a book chapter.

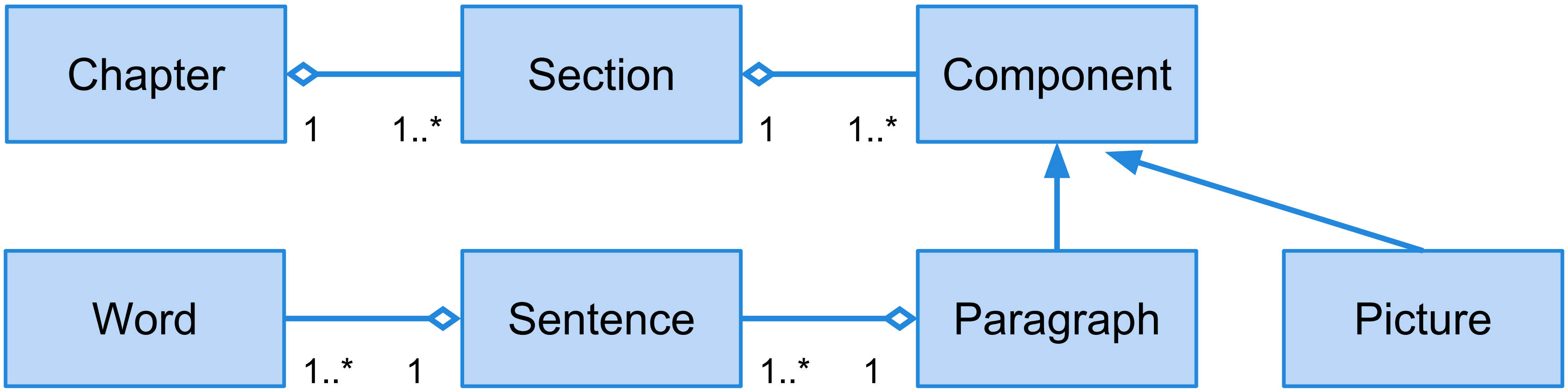
A chapter comprises several sections, each of which comprises several paragraphs and/or figures. A paragraph comprises several sentences, each of which contains several words.

Draw a class diagram (using inheritance) that captures two categories of a company's customers: external customers, which are other companies buying goods from this company, and internal customers, which are the divisions of the company.

Suggested Solution 1

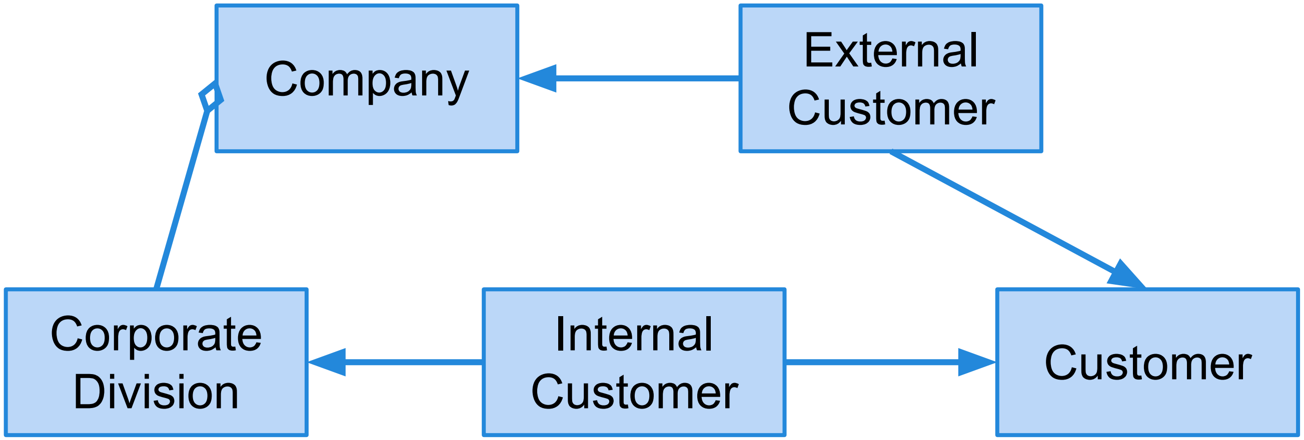
Draw a class diagram for a book chapter.

A chapter comprises several sections, each of which comprises several paragraphs and/or figures. A paragraph comprises several sentences, each of which contains several words.



Suggested Solution 2

Draw a class diagram (using inheritance) that captures two categories of a company's customers: external customers, which are other companies buying goods from this company, and internal customers, which are the divisions of the company.



We Have Learned

- An object is an entity in the problem domain.
- An object is an instantiation of a class (a type of object).
- Classes have attributes and operations.
- Classes are related through associations:
  - Regular association, aggregation, composition, inheritance
- Associations have multiplicity and may have direction.

Next Time

- More on coming up with the classes and associations.
- Reading:
  - Sommerville, chapter 7
  - Fowler UML, chapter 3 and 5
- Assignment 3
  - Draft design - start thinking about this. We will cover a lot of strategies soon.