Class Diagrams

Software Requirements and Design
T-216-GHOH

Skúli Arnlaugsson| 14. október 2019

What are class diagrams ?

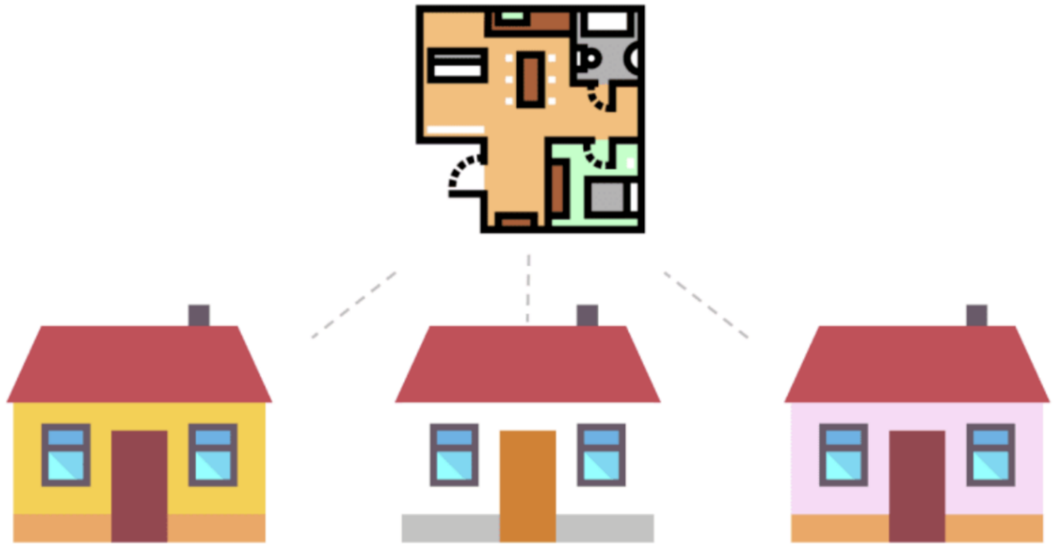
- Class diagrams are the main building block in object-oriented modeling
- They are used to show the **different objects** in a system, their attributes, their operations and the relationships among them

This lecture

- Objects in object-oriented-programming (*ísl. hlutbundindinni forritun*)
- Class diagrams notation (*ísl. skrifháttur klasarita*)
  - Inheritance (*ísl. erfðir*)
  - Association (*ísl. vensl*)
  - Dependency (*ísl. háð tengsl*)
  - Multipliers (*ísl. margfeldisþáttur*)

Class

- A **class** is sort of a template for making a new object
- A class is like a blue print of a house
  - But not quite. A house is still a house even if you add or remove a window, while the blueprint would need to change.
- You can build many houses from the same drawing
  - Each house is one instance



Object Oriented Programming

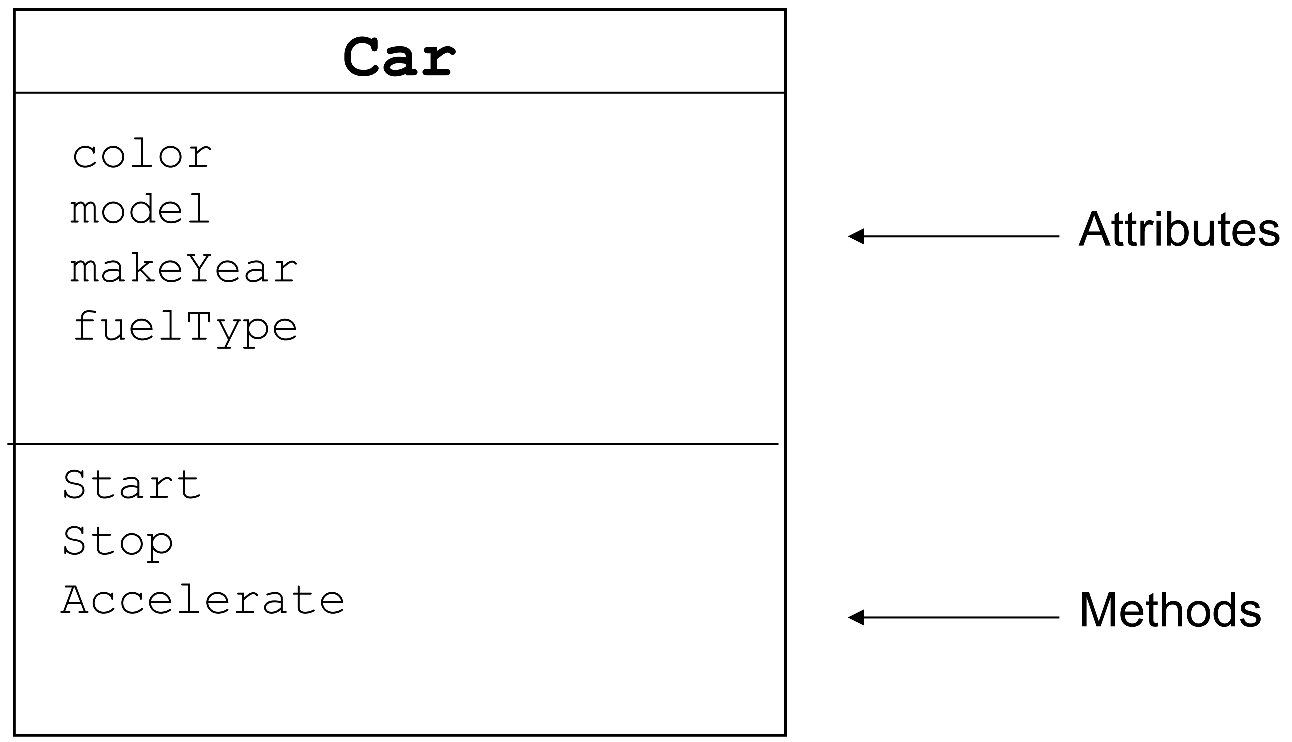
- A program is a set of objects

- Each object has some number of **attributes** that describe the object
  - color, make, name, age

- The object responds to some **methods** that are particular for that object
  - print, move, open

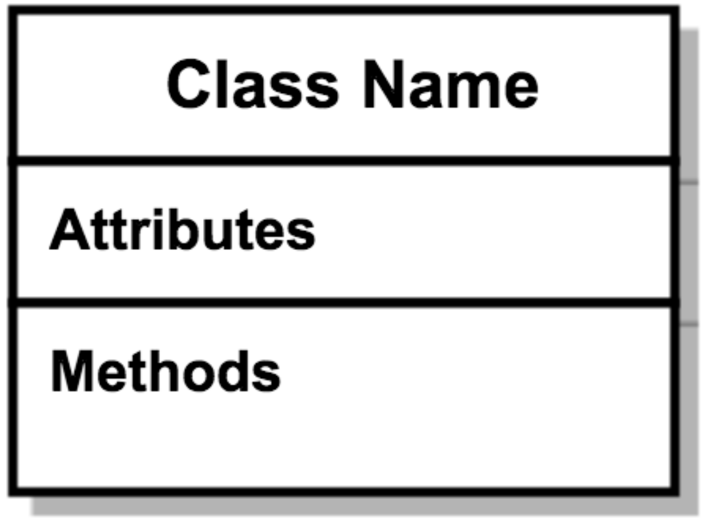
Attributes vs Methods

- Methods describe what the object can do



Classes UML

The UML representation of a class is a rectangle containing three compartments



Classes UML

Attribute notation

```
visability name: type multiplicity = default {property-string}
```

```
+ name: string [1] = "Untitled" {readOnly}
```

`visability:` (+) public, (-) private, (#) protected

`name:` The attribute name - only field that is necessary

`type`: restriction on what kind of object may be placed in this attribute

`multipliciy:` How many objects may fill the attribute(more later)

`default value:` the value for newly created objects if the attribute isn't specified during creation

`{property-string}:` additional properties for the attribute

Classes UML

Methods notation

```
visability name(parameterList): return-type {property-string}
```

```
+ sum(a: int; b: int) : int
```

`visability`: (+) public, (-) private, (#) protected

`name`: The methods name - only field that is necessary

`parameterList` : Parameters passed to the method, if any

```
        parameter1: type1; parameter2: type2; …
```

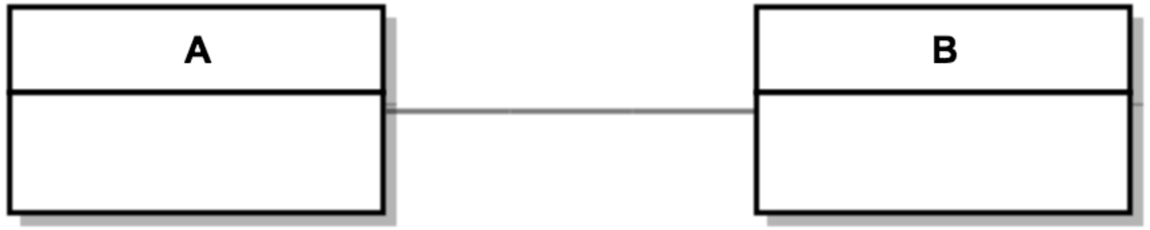
`return-type`: the type of the returned value, if there is one

`{property-string}`: additional properties for the method

Classes - relationships

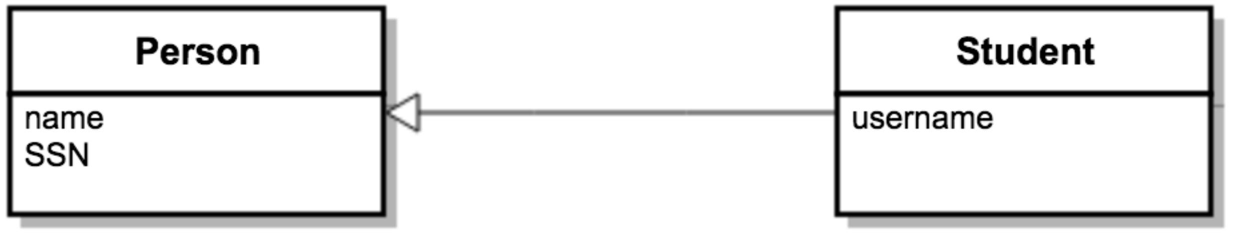
- The relationship between two classes is just as important as the class itself

- In UML, class relationship is defined using a single unbroken line



Class relationships - Inheritance

- Inheritance describes the "is-a" relationship between objects
- **Inheritance**: ability of one class (child class) to *inherit* the identical functionality of another class (super class), and then add new functionality of its own

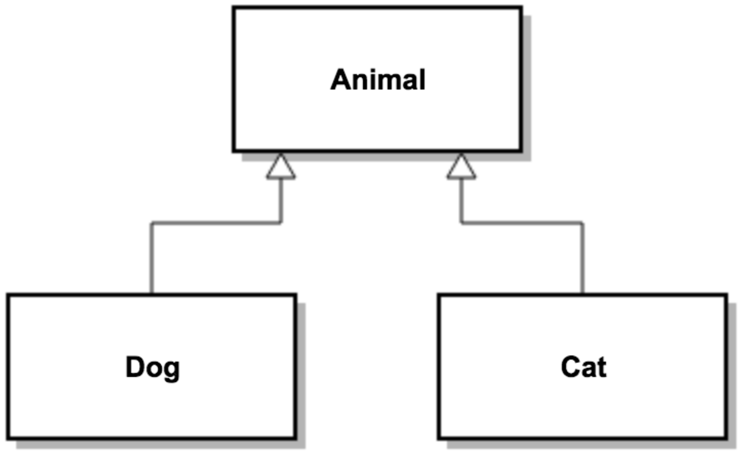


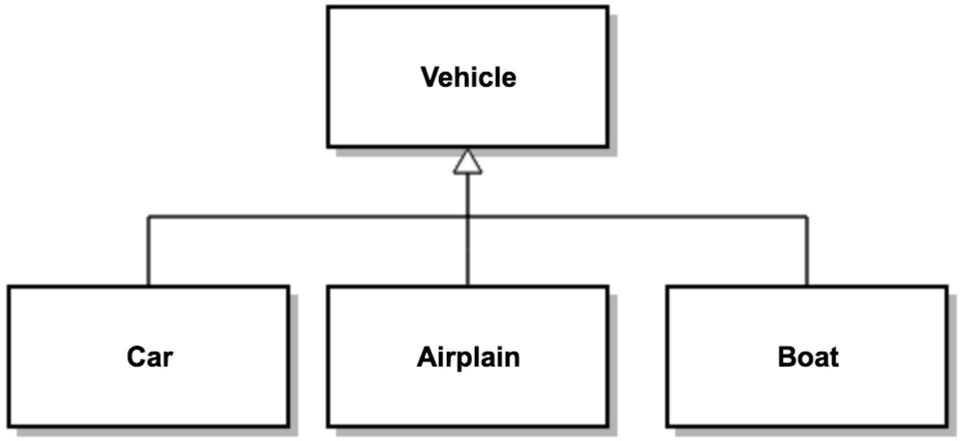
parent class/
super class

child class/
sub class

Inheritance and UML

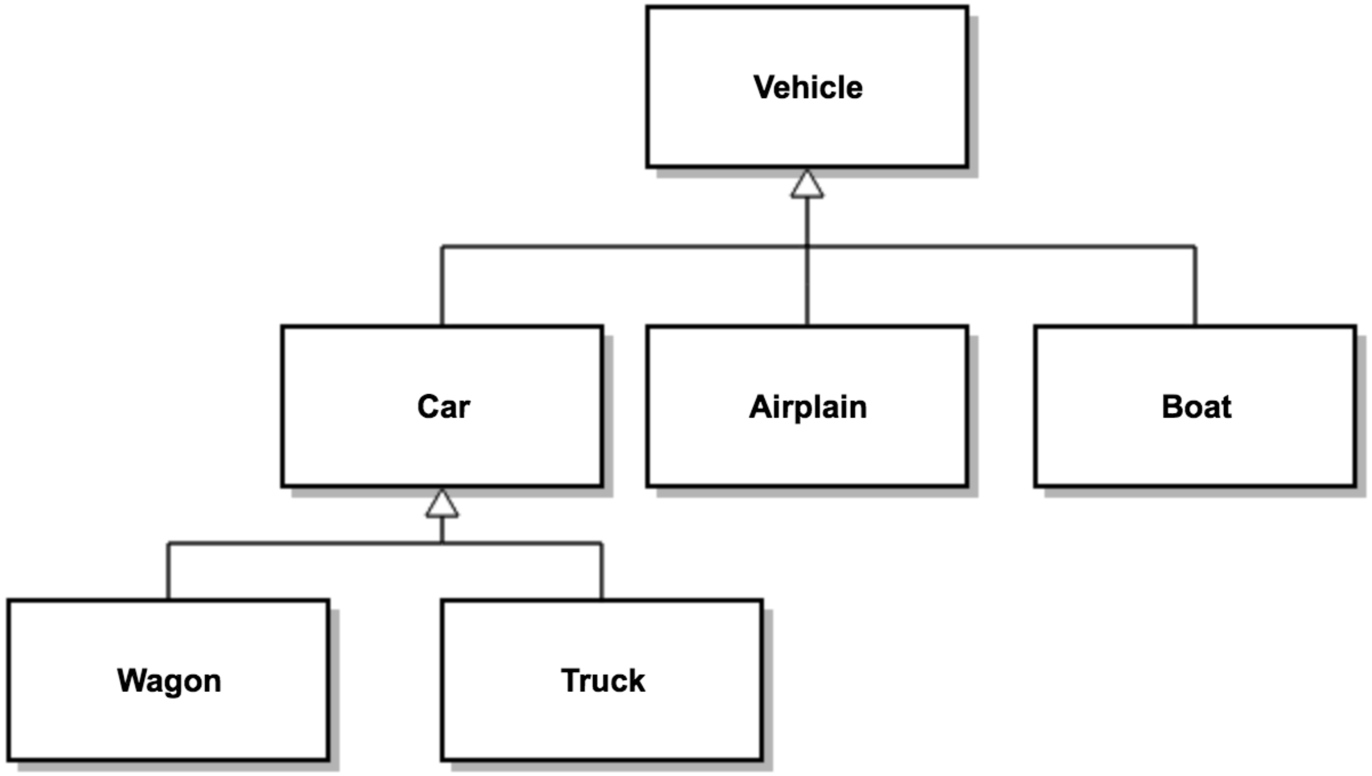
- In UML, inheritance is represented using a hollow arrowhead on one side of the relationship line, pointing to the base class
- Sometimes, the arrowhead is reused





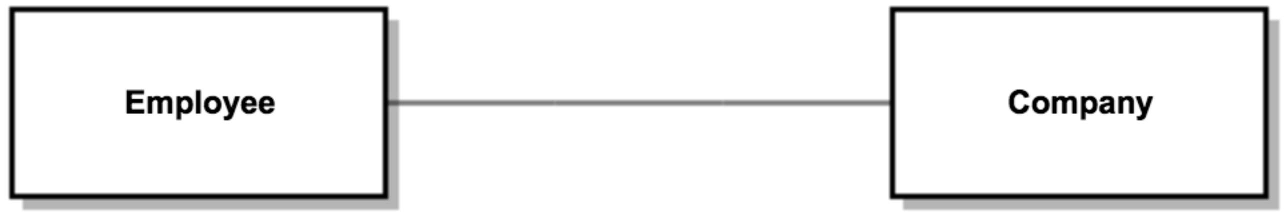
Inheritance chains and trees

- There is nothing that prevents us from using inheritance on more than one le\



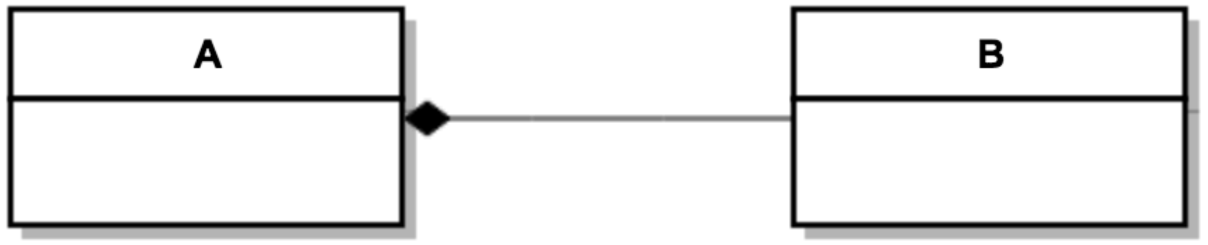
Association

- An association between two classes indicates that objects at one end ”recognize” objects at the other end
- **Example**: An employee works for a company



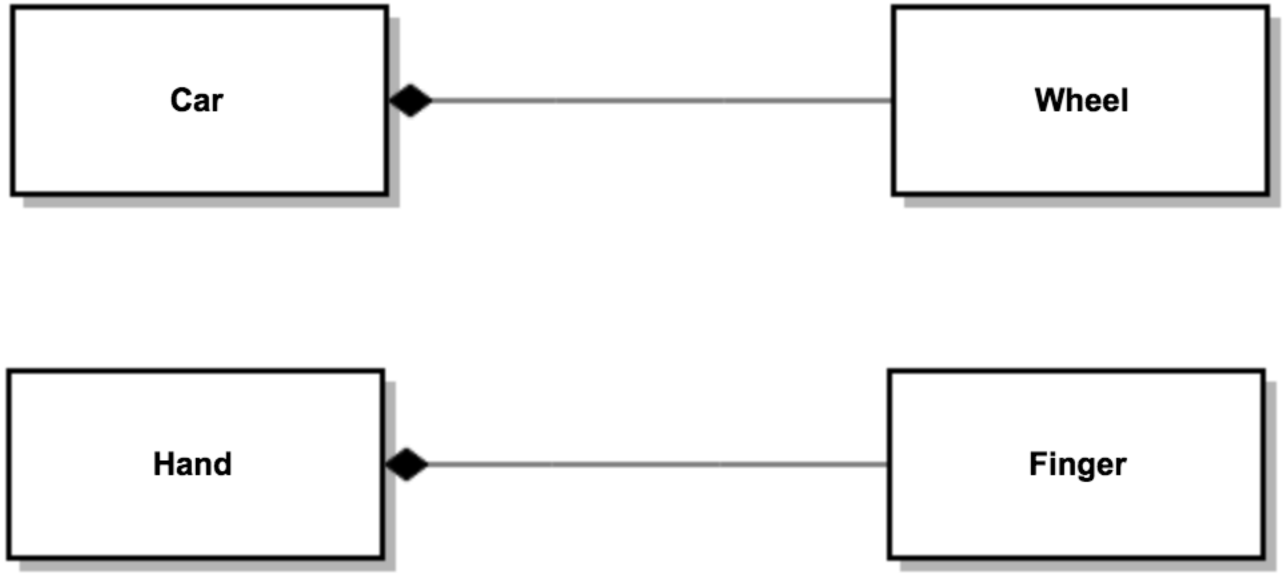
Composition

- A variant of the "has a" association
- More specific than association
- Class A **“has or owns”** one or more instances of another class B



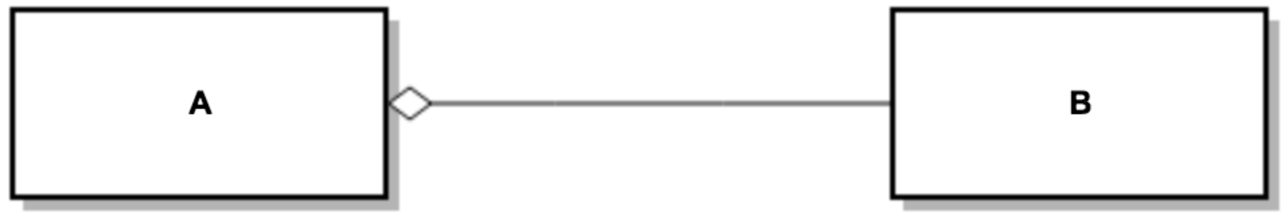
Composition and UML

- Composition is represented with a filled diamond at the end of the relationship line
- Example:

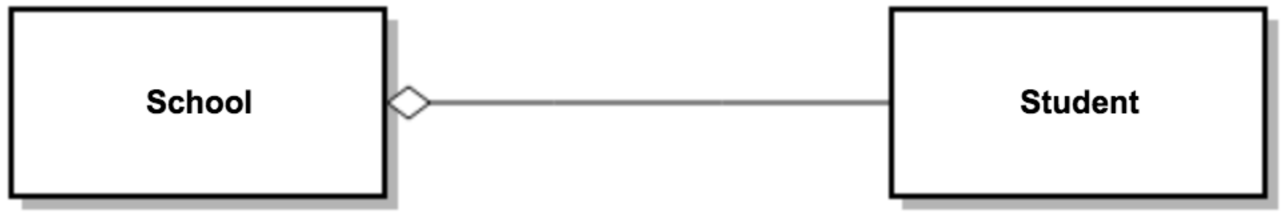


Aggregation

- Aggregation: class A “**has an**” instance of class B, class B may also stand alone

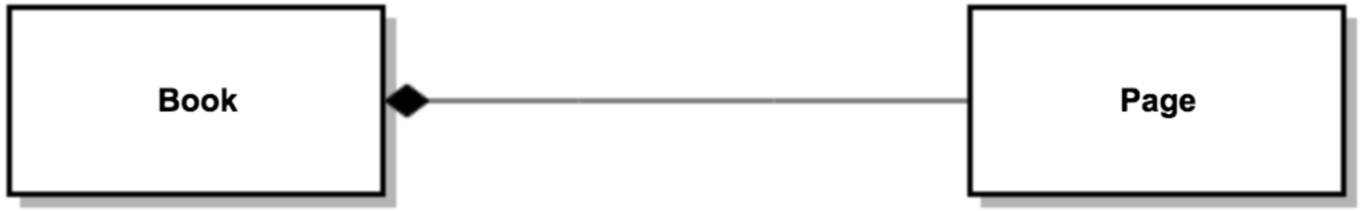


- **Example**: School may own multiple instances of Students, but a Student can also stand alone

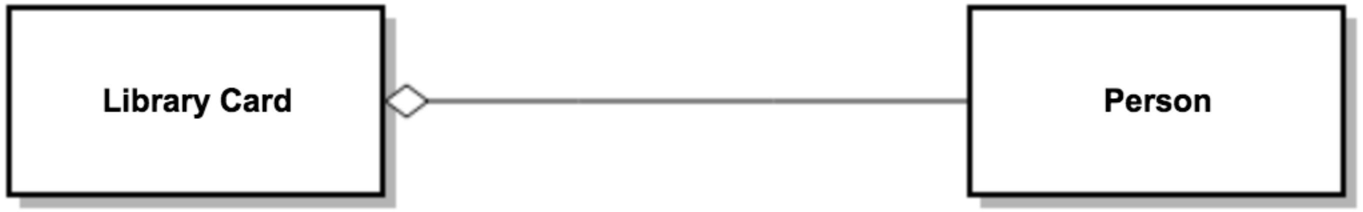


Composition vs Aggregation

- **Composition:** A book “owns” the pages in it. If you destroy the book, you destroy the pages



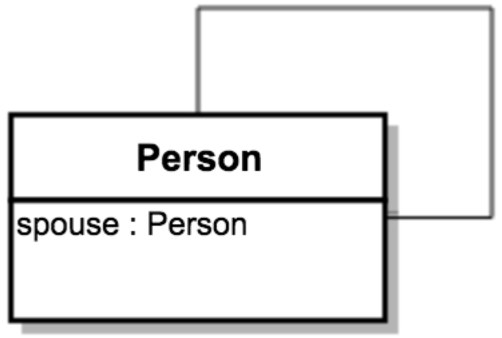
- **Aggregation:** A Library Card “has a” Person. If you destroy the library card, the person still exists



Association - Reflexive

Links can exist between instances of the same class.

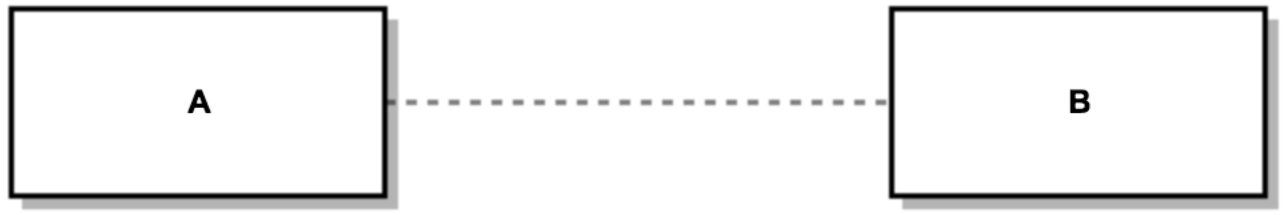
Definition: A reflexive association is an association between instances of the same class.



Dependency

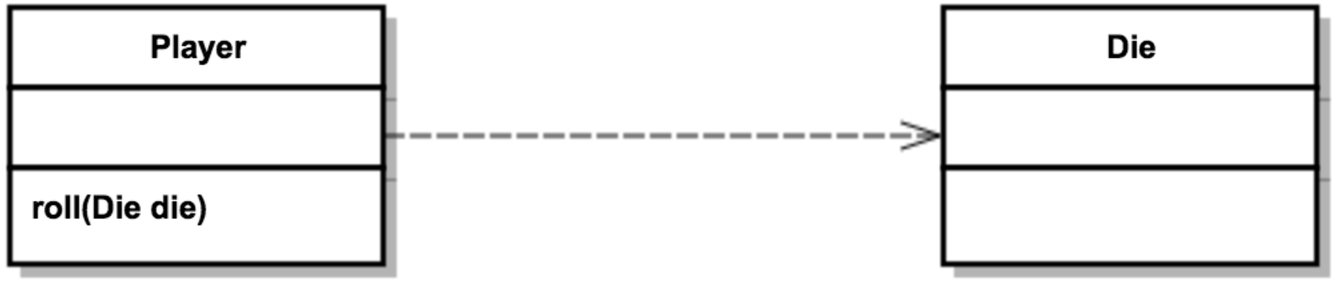
“Uses” relationship

- Dependency is a weaker form of bond that indicates that one class depends on another because it uses it at some point in time.



Dependency

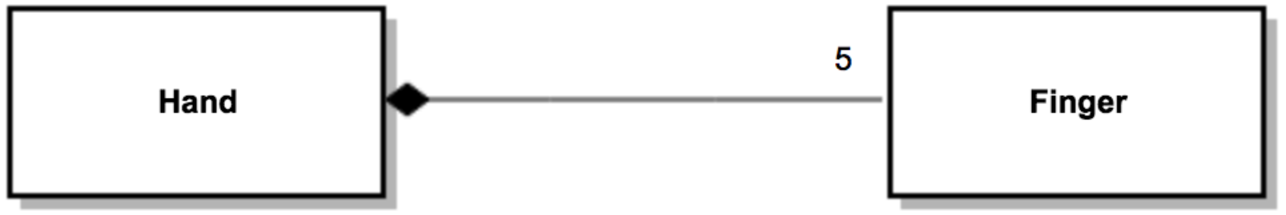
Example: A player rolls a die



Multipliers

- Multiplicity (*ísl. margfeldisþáttur*) defines how many instances of one class can be related to a **single instance** of another class
- Example:

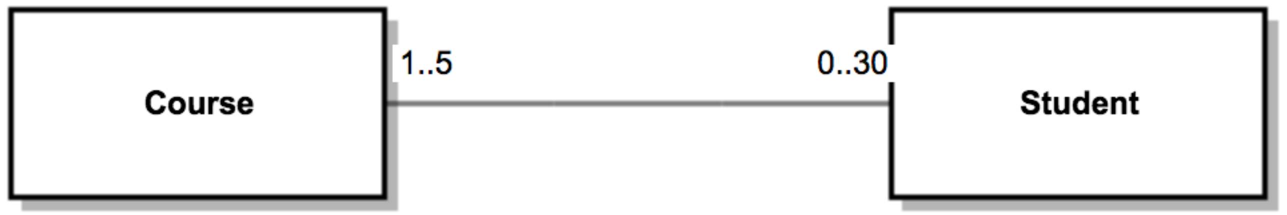
A hand has exactly 5 fingers



Multipliers in both directions

- Relationships are often two-way, therefore we often state the multipliers on both sides
- Example:

A course can have from zero (does that make sense?) and up to but no more than 30 students, while a student must always be enrolled in at least 1 course but no more than 5



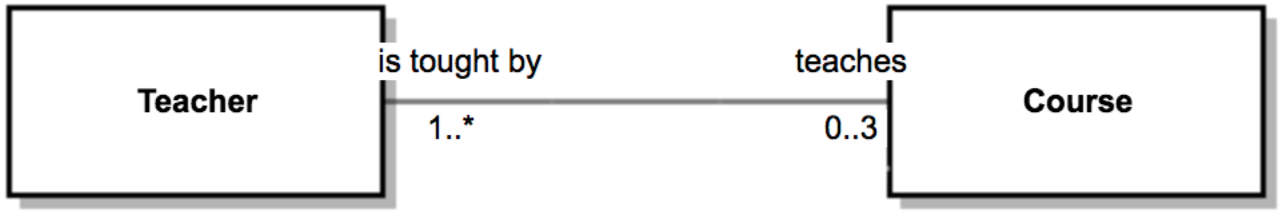
Different multipliers

- Sometimes the multiplier is just a
  `single number`
- Otherwise, they are in the format
  `[lower bound]..[upper bound]`
- If there are no restrictions to the number of class instances, the star (*) is used

*	Any number (including zero and one)
0..*	Same as *
0..1	Either zero or 1
1	Exactly one
1..*	One or more
2..4	2,3 or 4

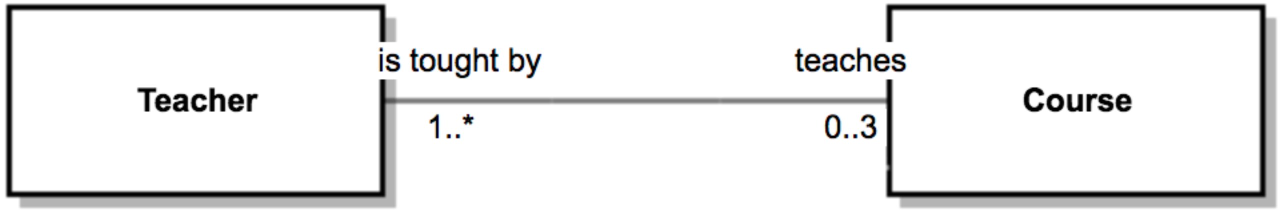
Relationship names

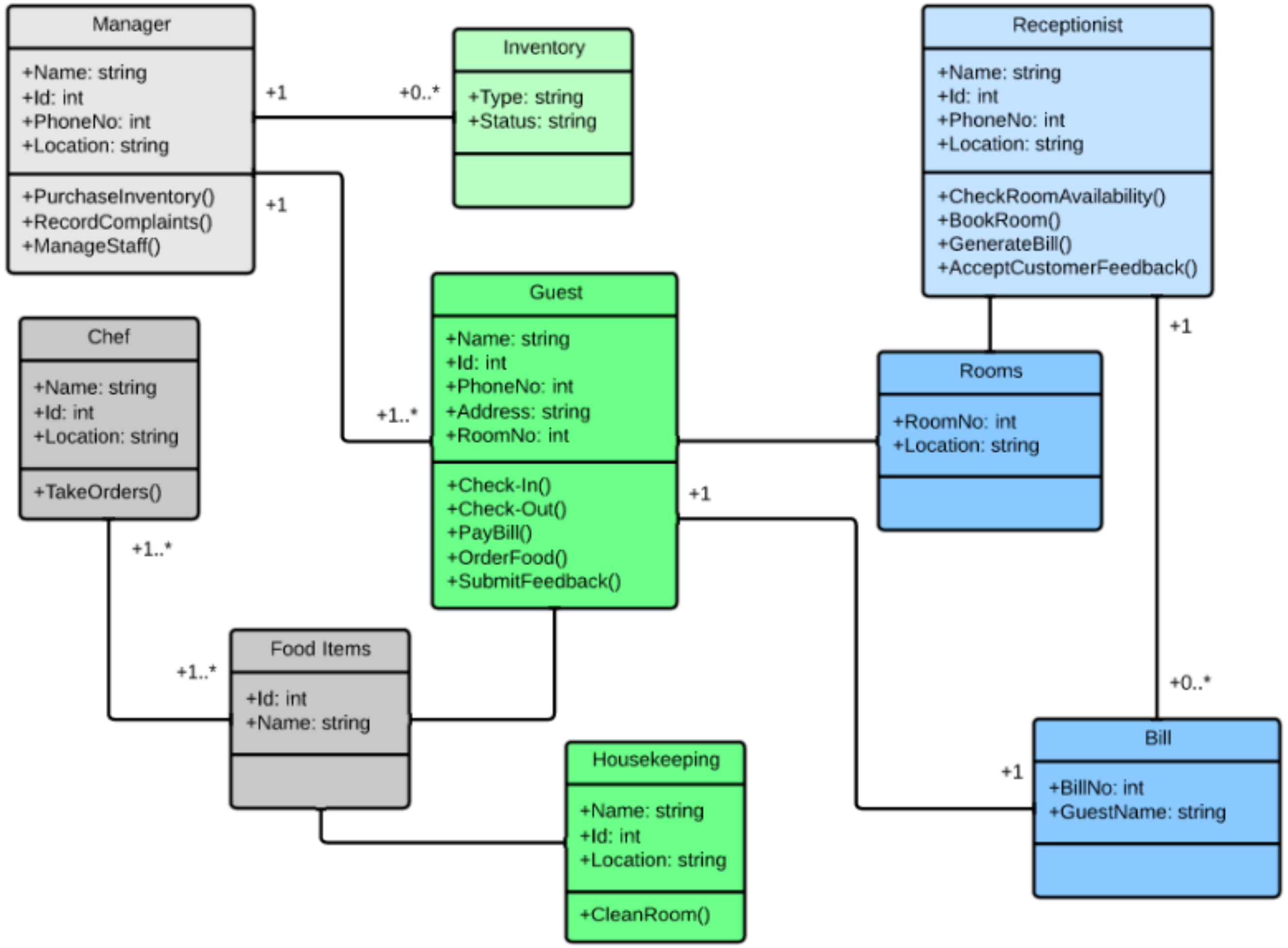
- The name of a relationship - or a sentence which describes it - is sometimes obvious from the context, but is usually helpful to include in a class diagram
- Two classes may have more than one relationship, in that case a name is necessary to identify between them
- Example:



Relationship names

- What can we read from this diagram?
  - a teacher can teach up to 3 courses
  - A teacher doesn't have to teach any courses (they might be doing research for a semester...)
  - a course must have at least one teacher
  - otherwise, there is no upper limit to the number of teachers a course can have




Manager
+Name: string
+Id: int
+PhoneNo: int
+Location: string
+PurchaseInventory()
+RecordComplaints()
+ManageStaff()
+1
+0..*
Inventory
+Type: string
+Status: string
+1
+1..*
Receptionist
+Name: string
+Id: int
+PhoneNo: int
+Location: string
+CheckRoomAvailability()
+BookRoom()
+GenerateBill()
+AcceptCustomerFeedback()
+1
Chef
+Name: string
+Id: int
+Location: string
+TakeOrders()
+1..*
Guest
+Name: string
+Id: int
+PhoneNo: int
+Address: string
+RoomNo: int
+Check-In()
+Check-Out()
+PayBill()
+OrderFood()
+SubmitFeedback()
+1
Rooms
+RoomNo: int
+Location: string
+1..*
Food Items
+Id: int
+Name: string
+0..*
Bill
+BillNo: int
+GuestName: string
+1
Housekeeping
+Name: string
+Id: int
+Location: string
+CleanRoom()