UML Class Diagrams Tutorial, Step by Step

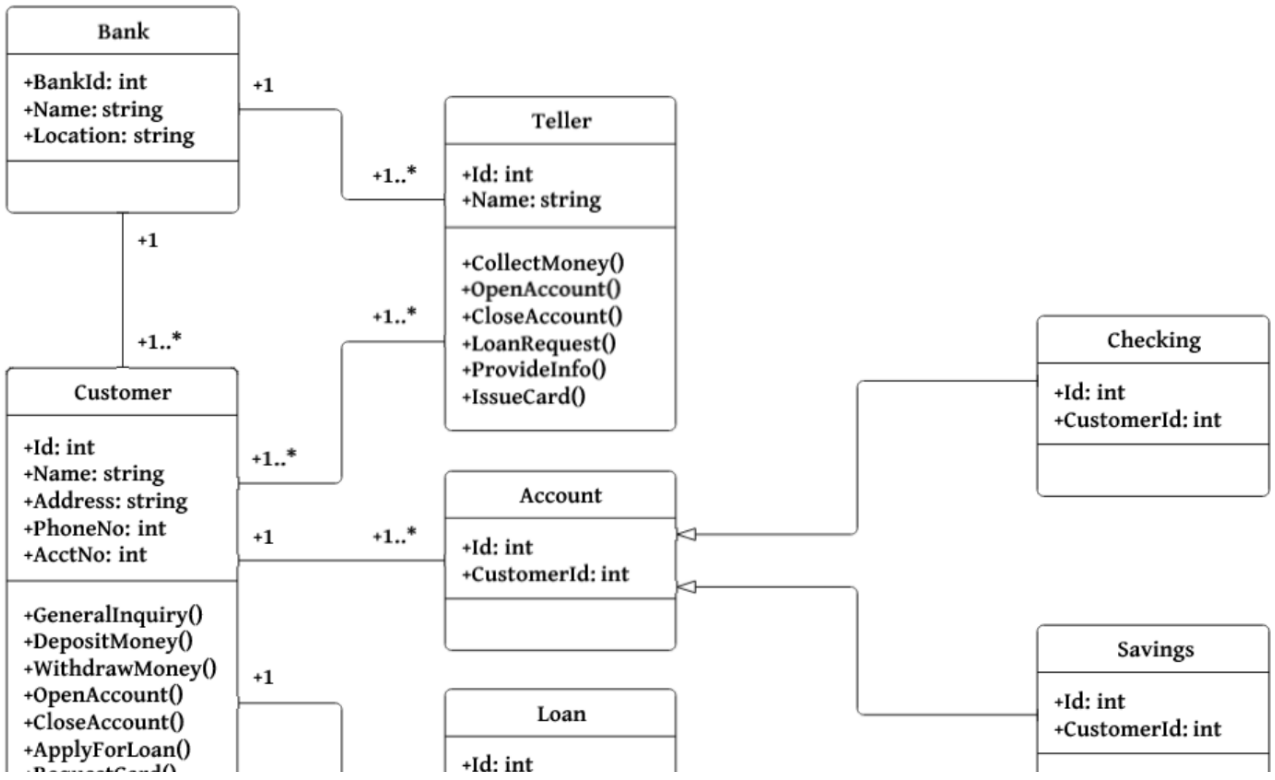
Salma Follow
Sep 1, 2017 · 5 min read

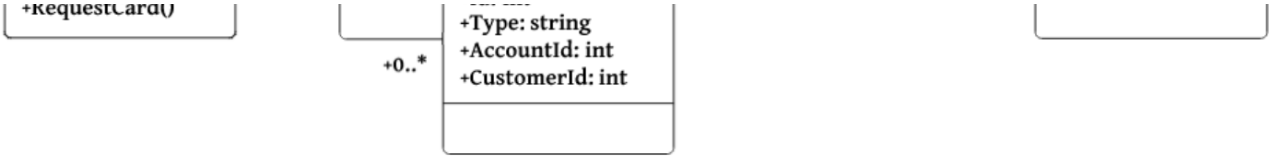
This is a short tutorial on UML Class Diagrams. We'll discuss what they are, why they're needed, some technical stuff, and then we'll dive into an example.

What is a Class Diagram?

Suppose you have to design a system. Before implementating a bunch of classes, you'll want to have a conceptual understanding of the system — that is, what classes do I need? What functionality and information will these classes have? How do they interact with one another? Who can see these classes? And so on.

That's where class diagrams come in. Class diagrams are a neat way of visualizing the classes in your system *before* you actually start coding them up. They're a static representation of your system structure.





Example of a Class Diagram for a Banking System

This is a fairly simple diagram. However, as your system scales and grows, it becomes increasingly difficult to keep track of all these relationships. Having a precise, organized, and straight-forward diagram to do that for you is integral to the success of your system.

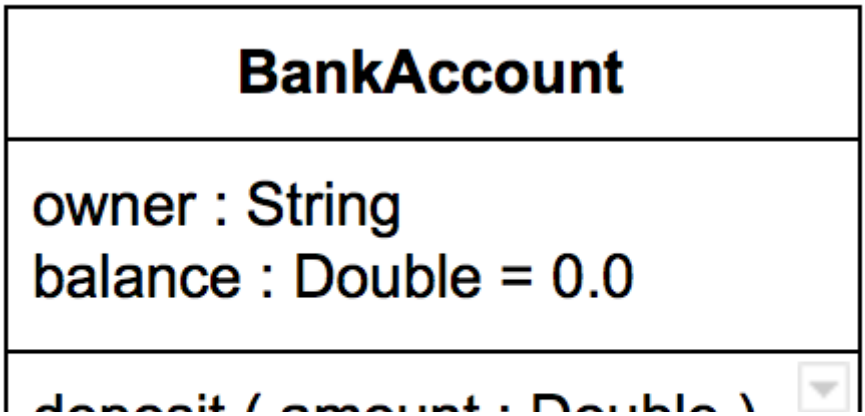
Why do we need class diagrams?

1. Planning and modeling ahead of time make programming much easier.
2. Besides that, making changes to class diagrams is easy, whereas coding differnent functionality after the fact is kind of annoying.
3. When someone wants to build a house, they don't just grab a hammer and get to work. They need to have a blueprint — a design plan — so they can ANALYZE & modify their system.
4. You don't need much technical/language-specific knowledge to understand it.

Some Technical Stuff

Class Representation in UML

A class is represented as a box with 3 compartments. The uppermost one contains the class name. The middle one contains the class attributes and the last one contains the class methods. Like this:



deposit (amount : Double)
withdraw (amount : Double)

They adhere to the following convention:

attribute name : type

method name (parameter: type)

- if you'd like to set a default value to an attribute do as above balance : Dollars = 0
- if a method doesn't take any parameters then leave the parentheses empty. Ex: checkBalance()

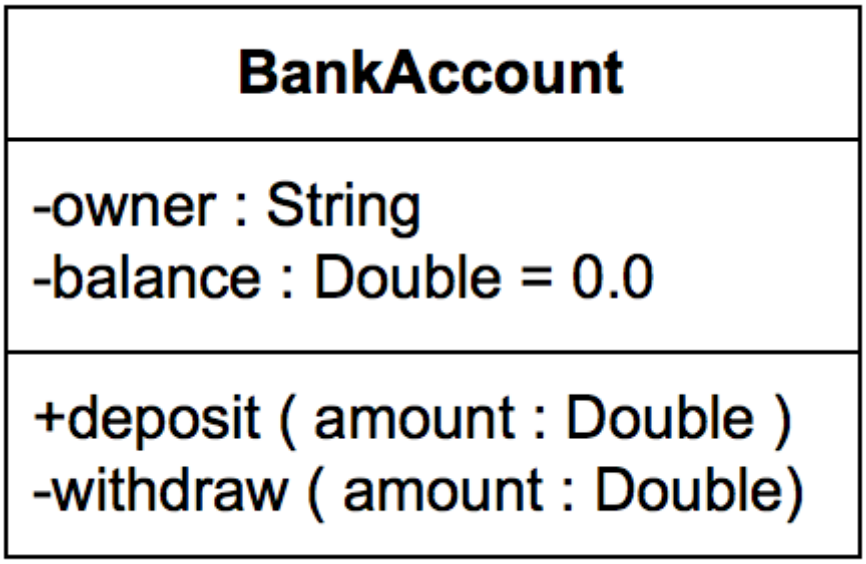
Visibility of Class Members

Class members (attributes and methods) have a specific visibility assigned to them. See table below for how to represent them in UML.

public	+	anywhere in the program and may be called by any object within the system
private	-	the class that defines it
protected	#	(a) the class that defines it or (b) a subclass of that class
package	~	instances of other classes within the same package

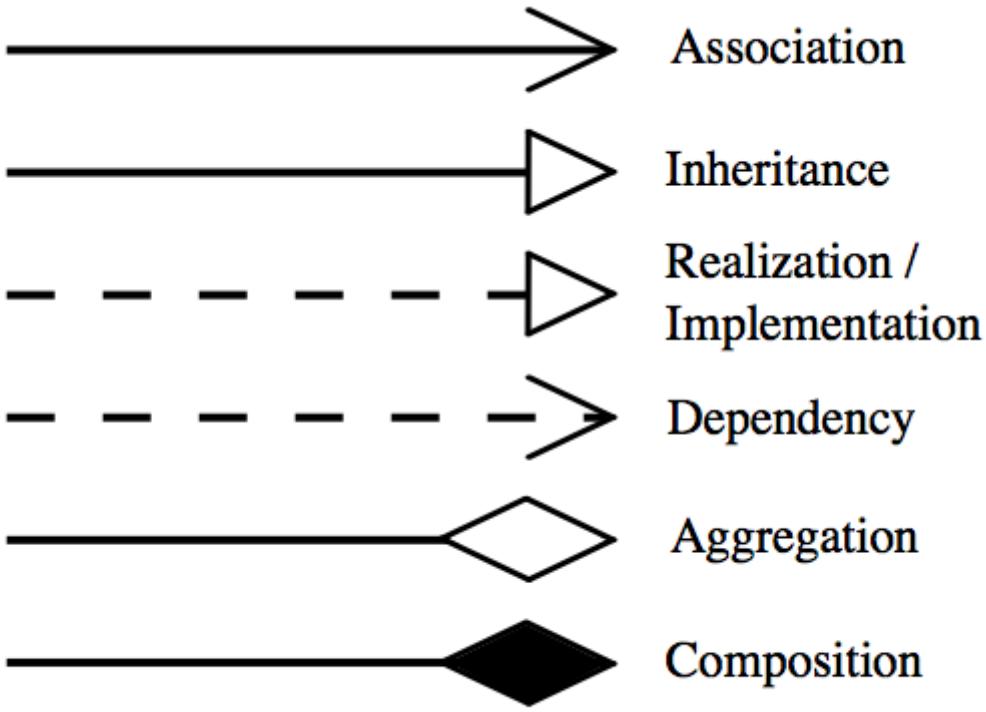
visibility and how to denote it

Let’s specifiy the visibility of the members of the BankAccount class above.



We made the `owner` and balance private as well as the withdraw method. But we kept the deposit method public. (Anyone can put money in, but not everyone can take money out. Just as we like it.)

Relationships



Summary of types of relationships and their notation

Association

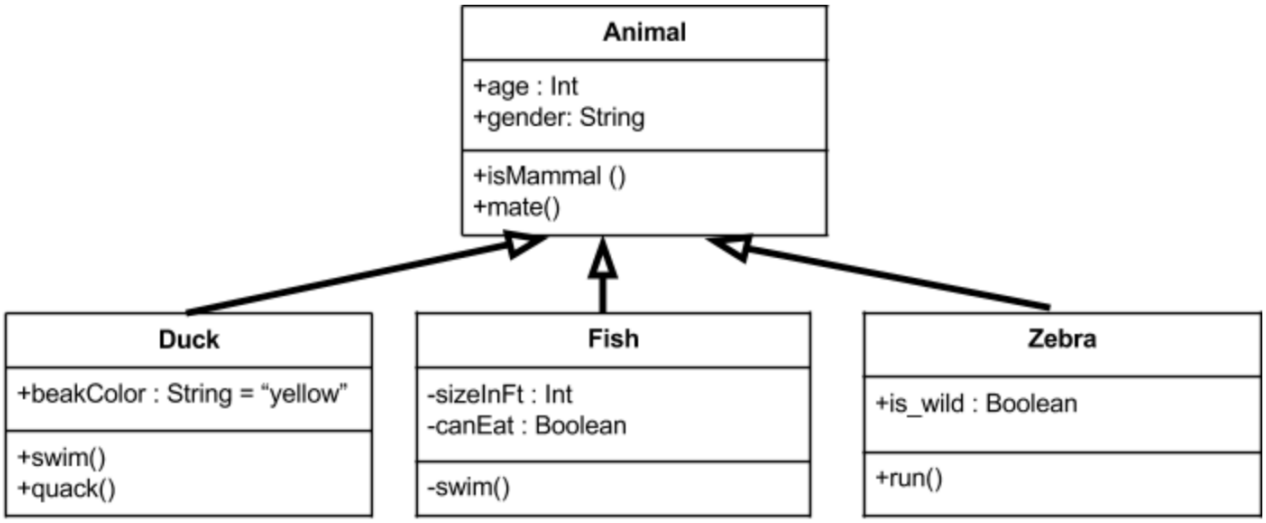
a relationship between two separate classes. It joins two entirely separate entities. There are four different types of association: bi-directional, uni-directional, aggregation (includes composition aggregation) and reflexive. Bi-directional and uni-directional associations are the most common ones.

This can be specified using multiplicity (one to one, one to many, many to many, etc.).

A typical implementation in Java is through the use of an instance field. The relationship can be bi-directional with each class holding a reference to the other.

Inheritance

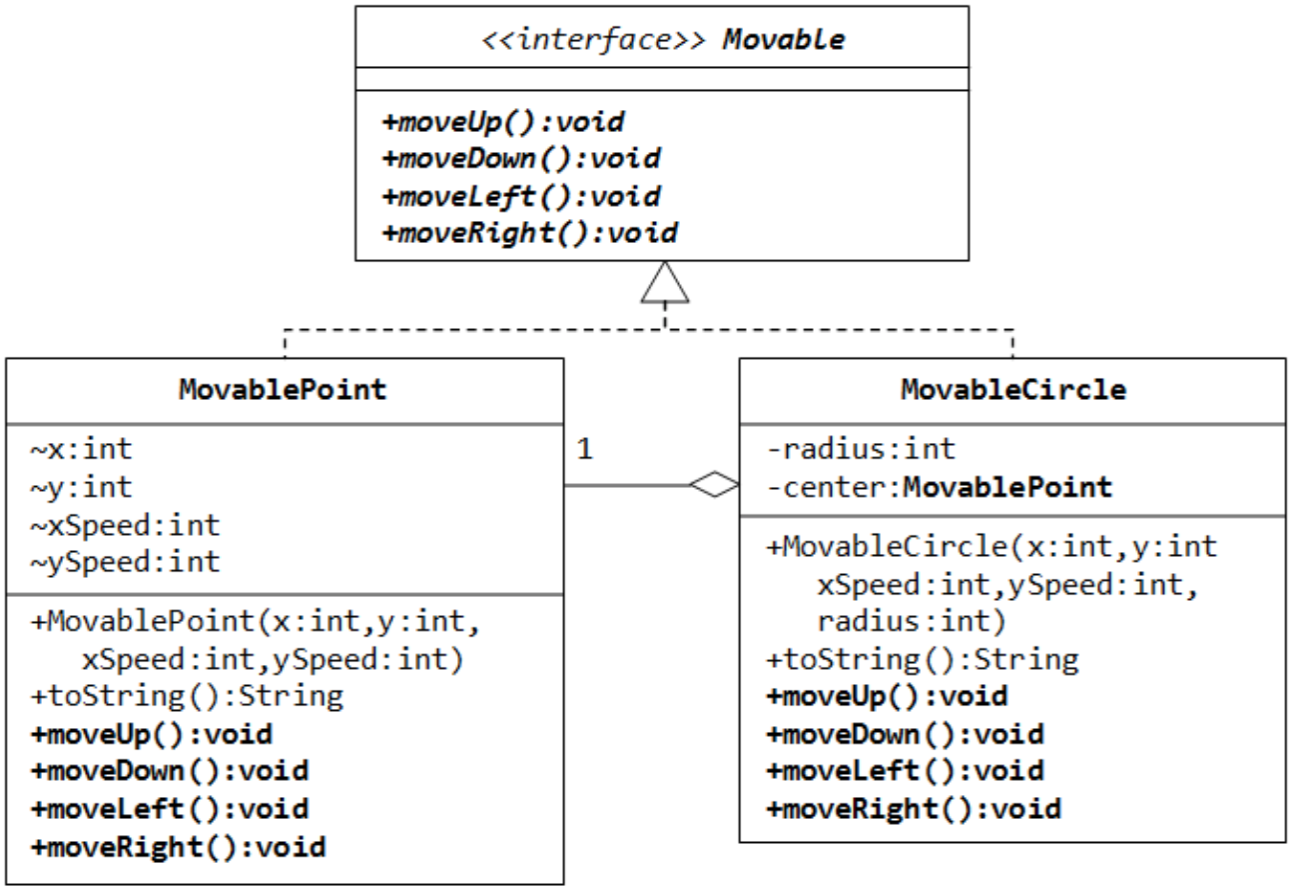
indicates that child (subclass) is considered to be a specialized form of the parent (super class). For example consider the following:



Above we have an animal parent class with all public member fields. You can see the arrows originating from the duck, fish, and zebra child classes which indicate they inherit all the members from the animal class. Not only that, but they also implement their own unique member fields. You can see that the duck class has a swim() method as well as a quack() method.

Realization/Implementation

a relationship between two model elements, in which one model element implements/executes the behavior that the other model element specifies.



example of implements

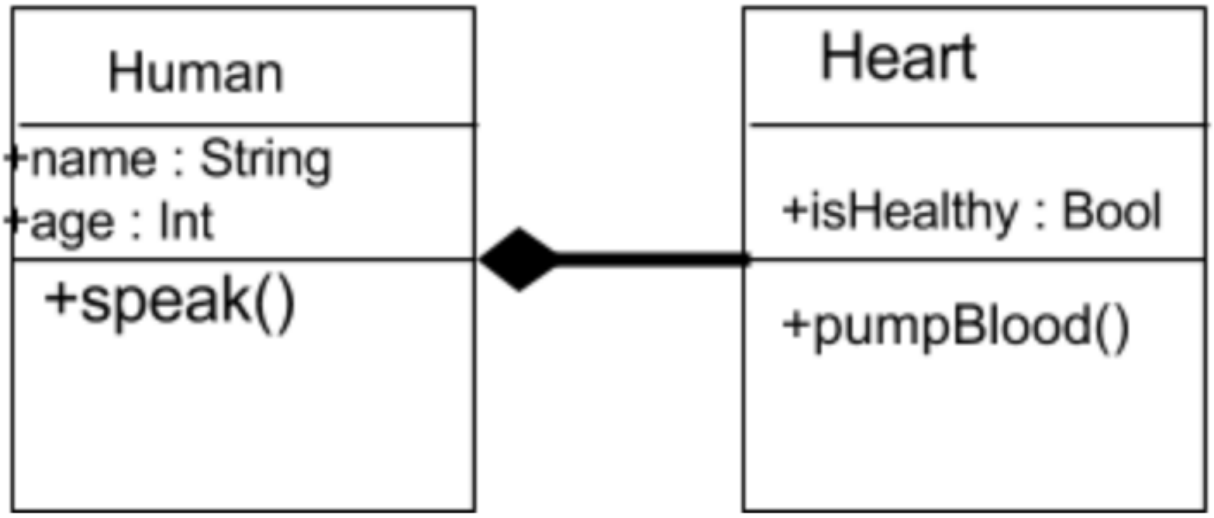
Dependency

Aggregation

a special form of association which is a unidirectional (a.k.a one way) relationship between classes. The best way to understand this relationship is to call it a “has a” or “is part of” relationship. For example, consider the two classes: Wallet and Money. A wallet “has” money. But money doesn’t neccessarily need to have a wallet so it’s a one directional relationship.

Composition

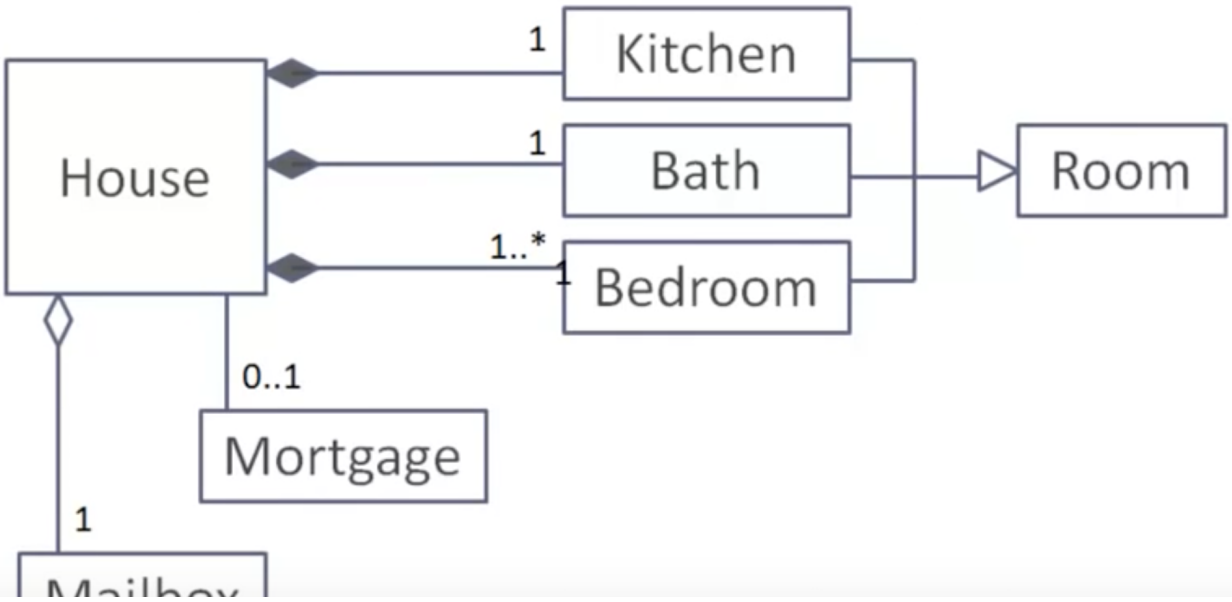
a restricted form of Aggregation in which two entities (or you can say classes) are highly dependent on each other.



A human needs a heart to live and a heart needs a human body to function on. In other words when the classes (entities) are dependent on each other and their life span are same (if one dies then another one too) then its a composition.

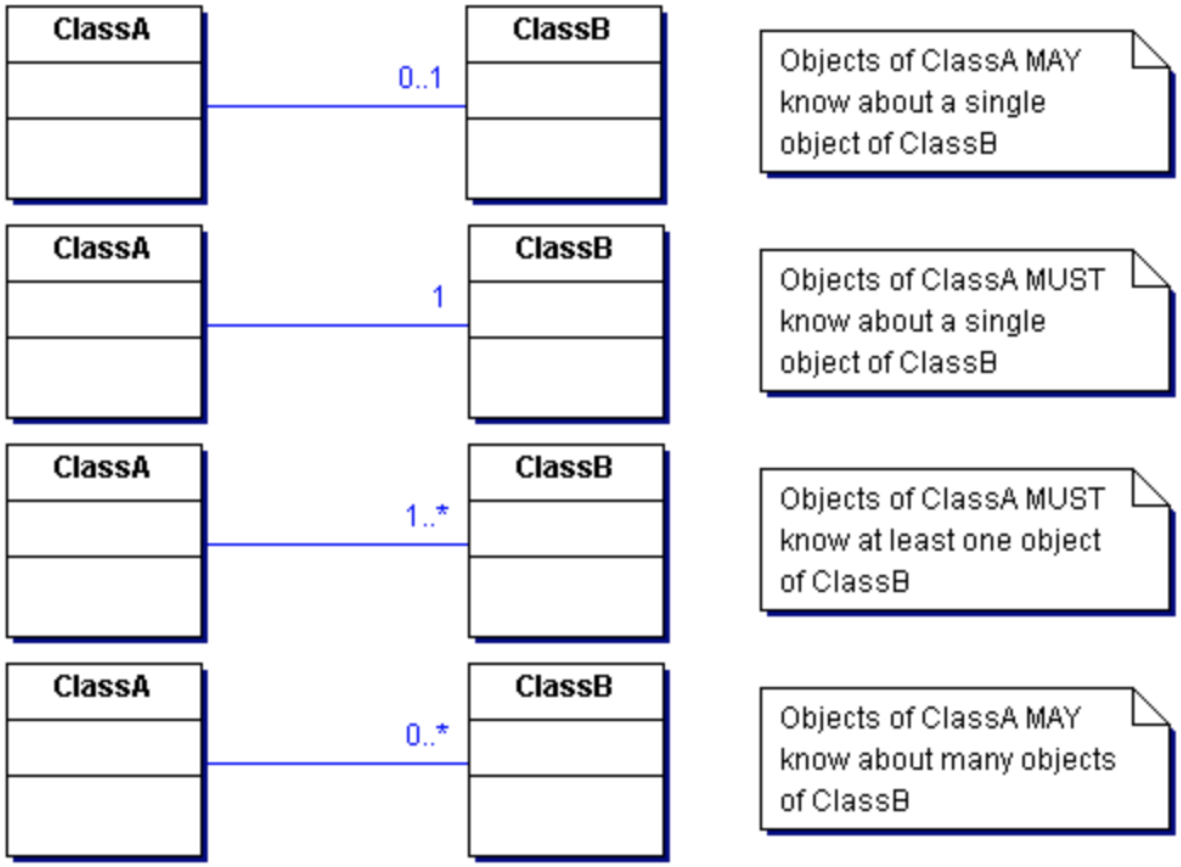
Multiplicity

after specificyfing the type of association relationship by connecting the classes, you can also declare the cardinality between the associated entities. For example:



Mailbox

The above UML diagram shows that a house has exactly one kitchen, exactly one bath, atleast one bedroom (can have many), exactly one mailbox, and at most one mortgage (zero or one).



Now, let's put everything together and do an example! Click here.

Programming