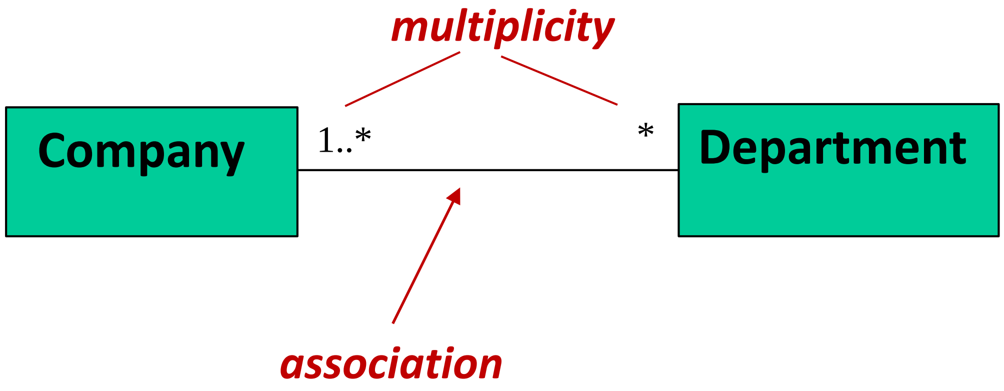
UML - Relationships

Part 2

Multiplicities

- An **association** represents a **structural relationship** among objects.
- In modelling situations, it is important for you to state how many objects may be connected across an instance of an association.
- This **how many** is called multiplicity of an association's role and is written as an expression that evaluates to a range of values or an explicit value.



Multiplicities

- Multiplicities specifies **how many** instances of one class may be associated with a single instance of another class.

- When you apply a multiplicities and association, you are indicating the number of links allowed between one instance of a class and the instances of another class.

Examples

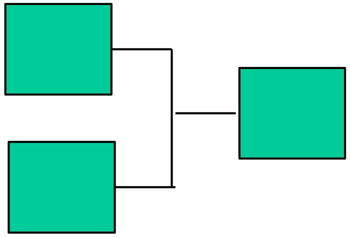
- Vehicle has a Tax Disk, and a Tax Disk is for a Vehicle.
- Is it always true that a Lottery Prize is won by a prize-winner?
- A student must have exactly one Personal-Tutor, but a lecturer may be Personal-Tutor to none, one or more students.

Examples

Associations between objects are called links – these can be useful in establishing multiplicity:

Could Capt. R. Branson be the pilot for more than one aircraft?

Can a student take more than one module for their course?



Multiplicity Notation

Conventions for multiplicity notation:

An exact number – e.g. 5

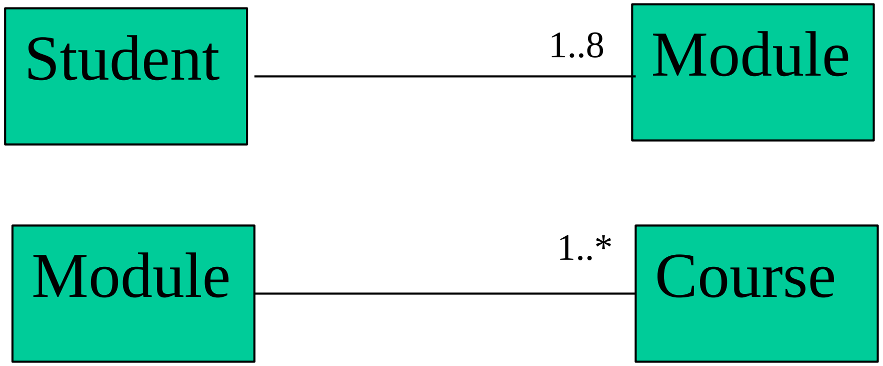
A range of numbers – e.g. 2..10

An unknown number - *

Examples:

A student can take many modules but no more than 8, for their course.

A course has many students enrolled on it.



Self Referencing Association

- A class may have an association with itself as well as other classes – this is known as a self referencing association.

Example:

If a Manager and Team Member are both part of the StaffMember class, one StaffMember would supervise another StaffMember.

Inheritance, Aggregation, Composition

Inheritance, **Aggregation** and **Composition** are UML notation methods which identify similar/special relationships.

Similar elements share common features:

e.g. AdminStaff and LectureStaff will share a number of common attributes and operation, as well as having ‘special’ features of their own.

Sometimes it is pointless having one thing without the other:

e.g. Without a bank account, we cannot make provide bank details for employer to pay us.

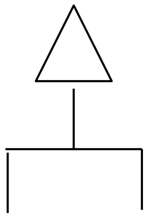
In Systems Analysis and Design, if we can identify and model such relationships the system is easier to develop, maintain and most importantly understand.

Inheritance

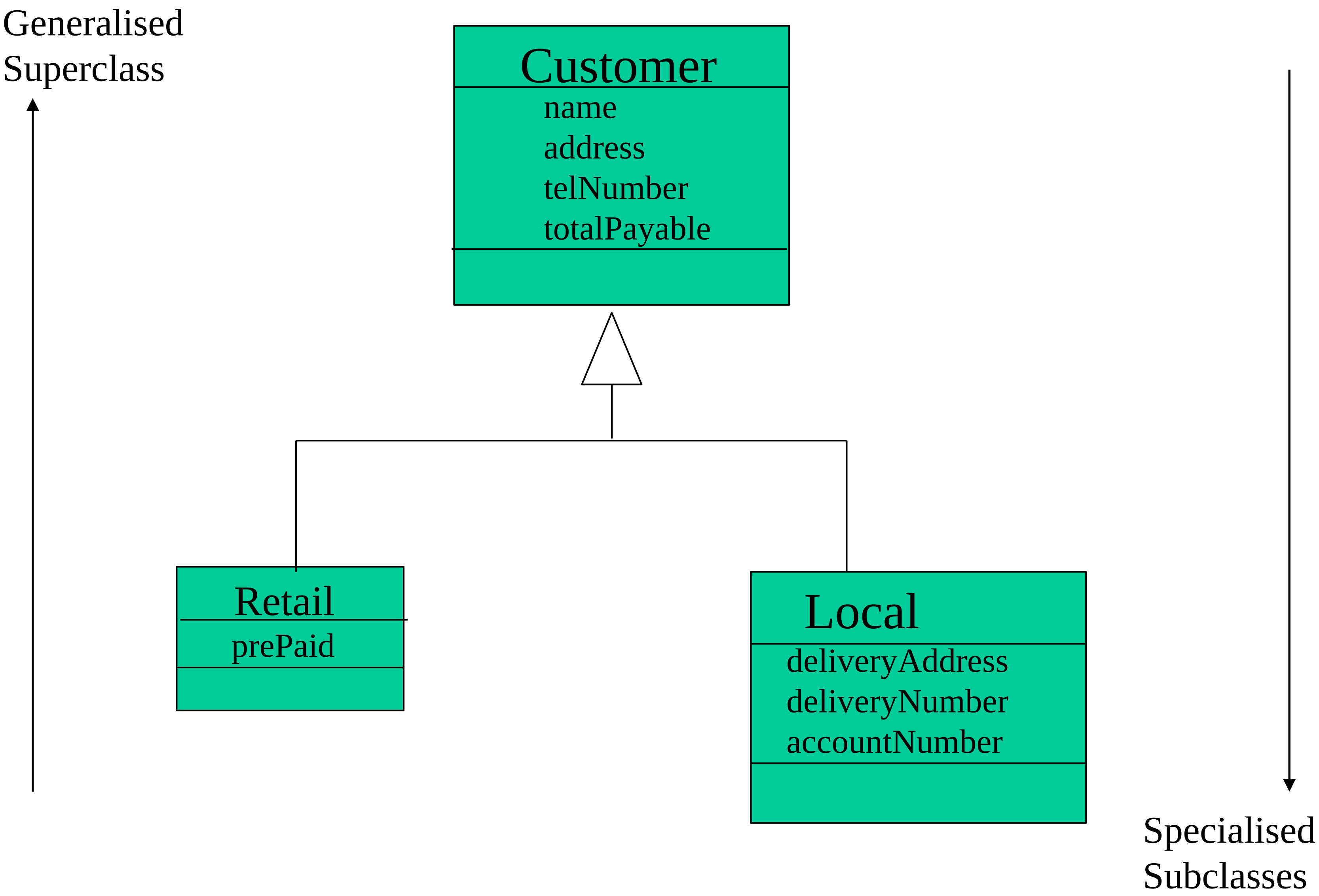
"….a relationship between two classes where one is a refinement of the other; sometimes referred to as the '**is – a**' relationship. A mechanism that allows a class to reuse feature already defined in another class." (Britton & Doake, 2000)

Inheritance

- Different ways of describing the relationship.
- The specialised class inherits from the general class.
- The child class inherits from the parent ancestor class.
- The subclass inherits from the superclass.
- **Inheritance symbol:**



Example



Generalised Superclass

Customer

name

address

telNumber

totalPayable

Retail

prePaid

Local

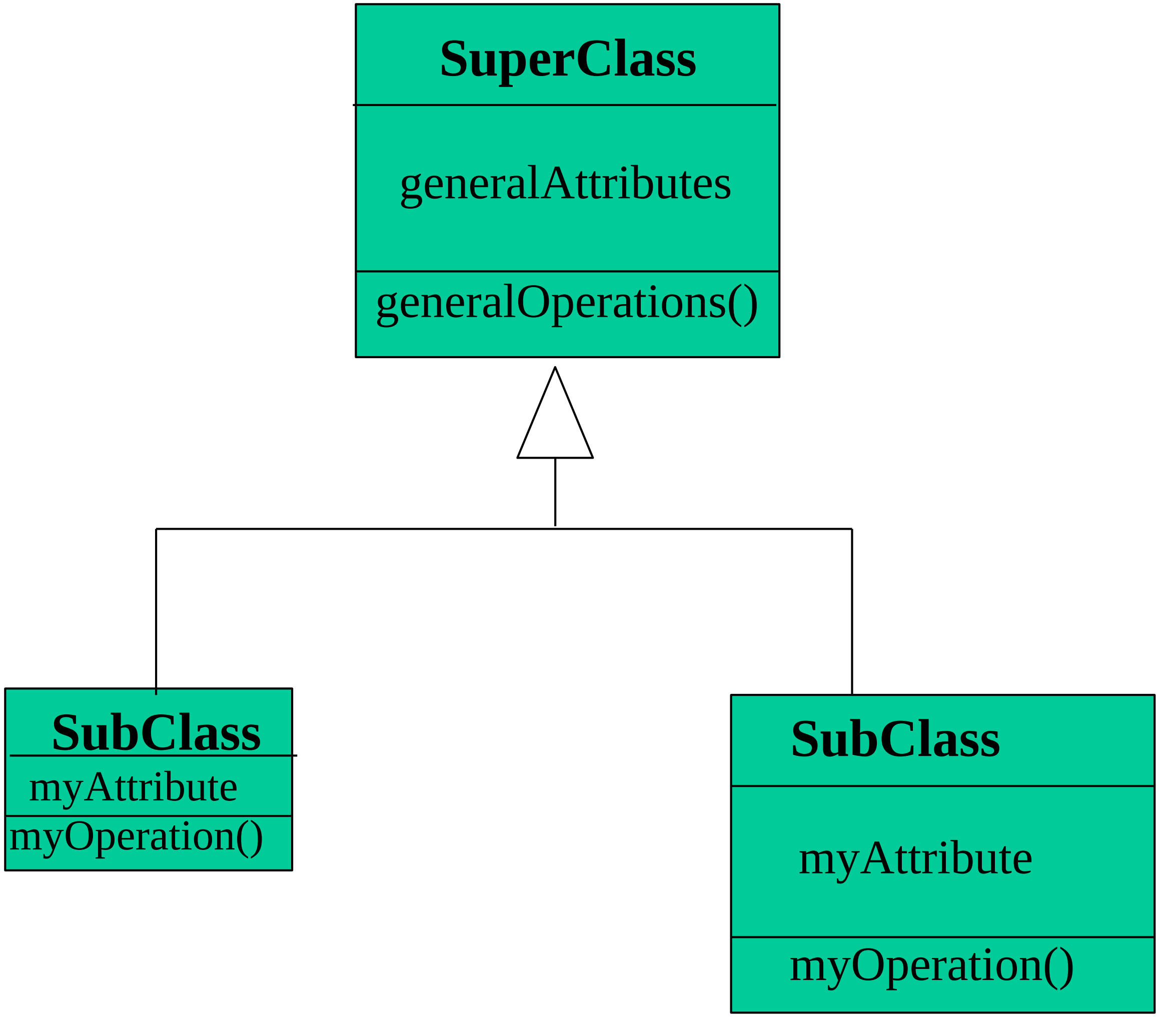
deliveryAddress

deliveryNumber

accountNumber

Specialised Subclasses

Example


SuperClass
generalAttributes
generalOperations()
SubClass
myAttribute
myOperation()
SubClass
myAttribute
myOperation()

Inheritance Rules

Rule #1:

The 100% Rule.

100% of the features of the superclass must apply to the subclass.

Rule #2:

The is-a Rule.

You must be able to say that the subclass is ***a kind of*** the superclass.

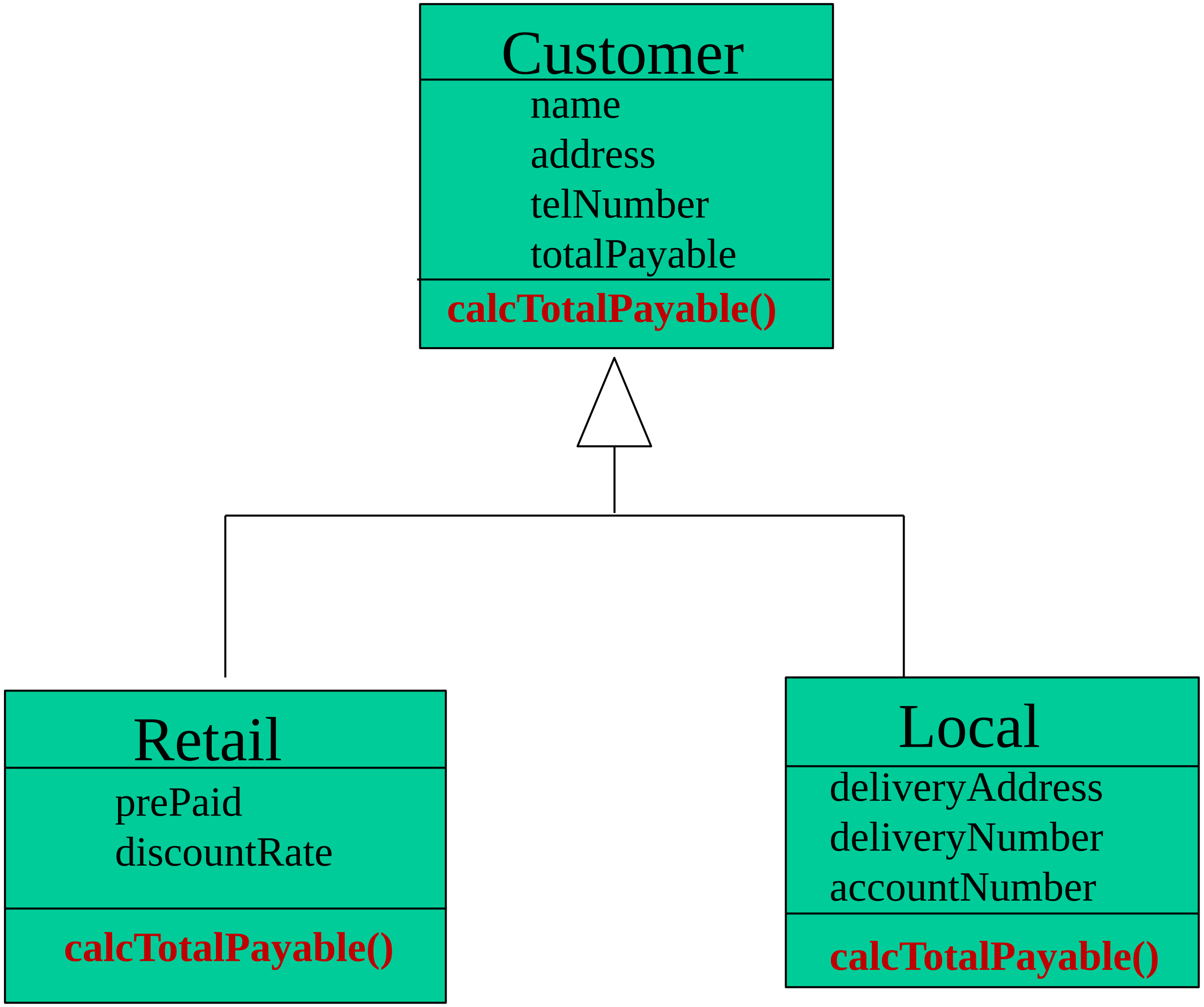
AdminStaff is a (kind of) STaffMember

CreativeStaff is a (kind of) STaffMember

Polymorphism

- Linked with inheritance.
- Allows a single message to be interpreted differently by different objects.
- Which method is executed will depend on which object receives the message.

Example - Polymorphism



Customer

name
address
telNumber
totalPayable

calcTotalPayable()

Retail

prePaid
discountRate

calcTotalPayable()

Local

deliveryAddress
deliveryNumber
accountNumber

calcTotalPayable()

Question

How would you show on a class diagram that relationship between say a CD and all the tracks on a CD?

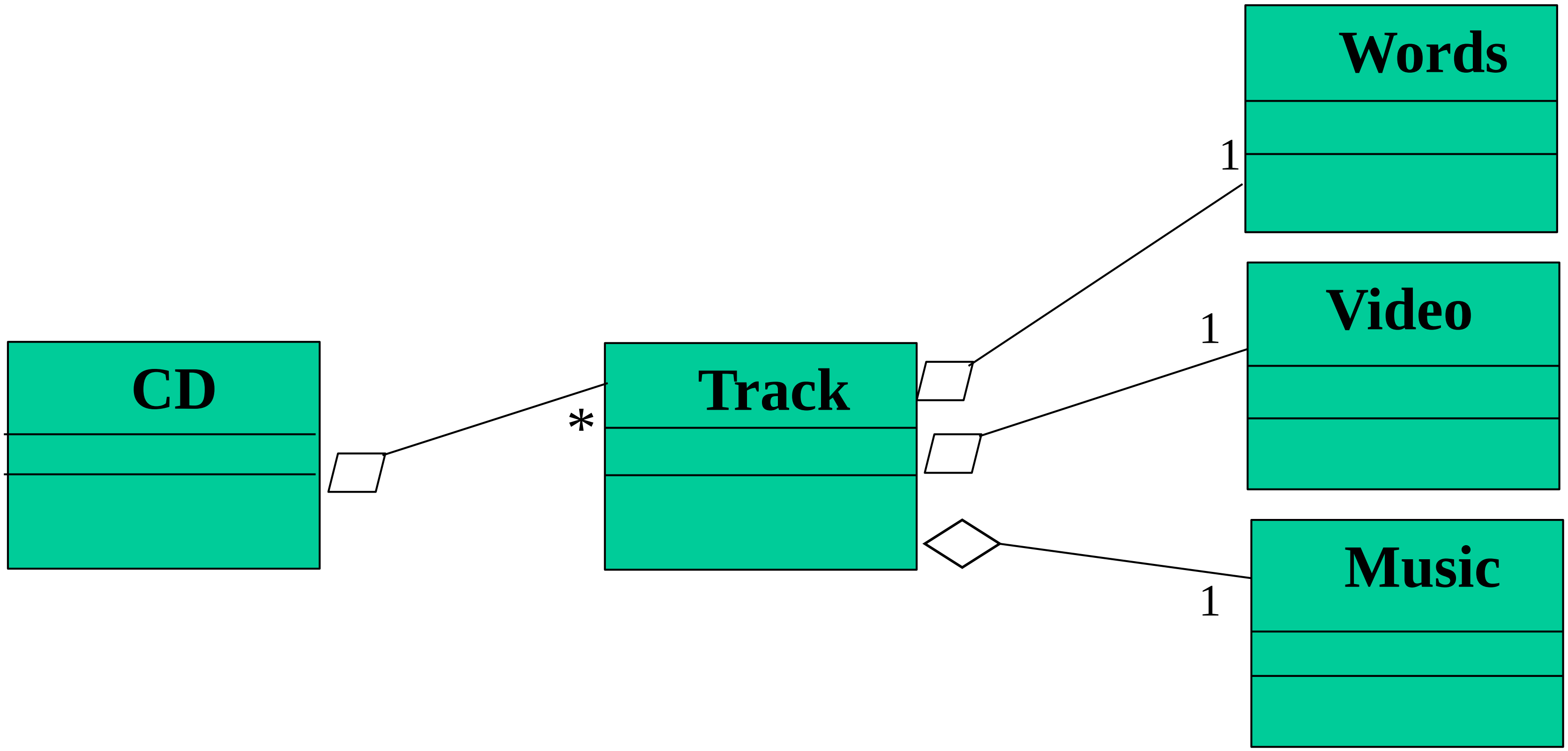
Question

What if some of the tracks appear on several different CDs?

OR

What if you also want to show that a track is made up of words, music and video? How could these elements be re-used elsewhere?

Example



Aggregation

Symbol ◇

When one class is made up of several occurrences of another class.

This is a special kind of relationship that shows how one class is part of another class. (*i.e. Track is part of CD*)

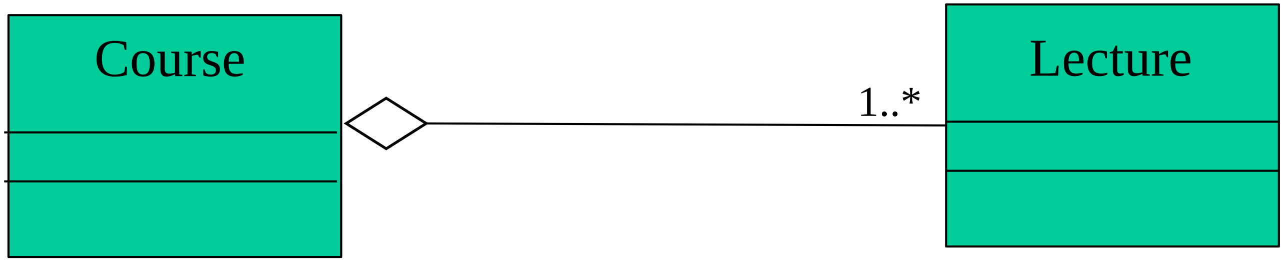
Aggregation

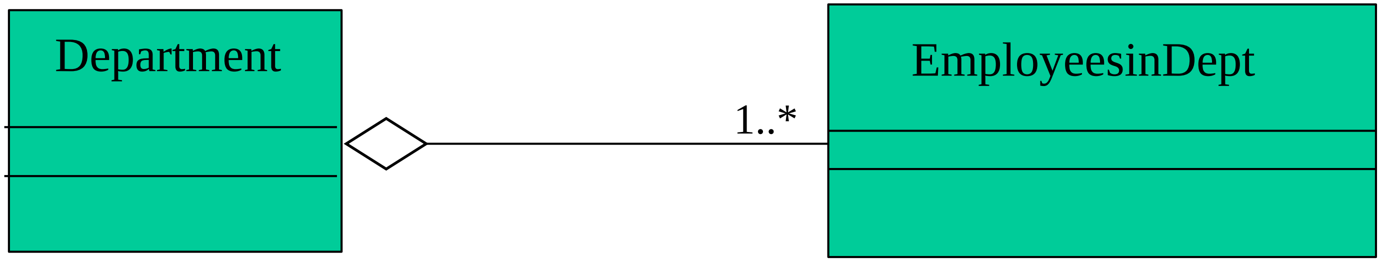
If you delete a particular object of the CD class, the tracks that were linked to it, will be removed but the tracks themselves will still exist and can be used for other CDs.

That might, or might not, be what you want.

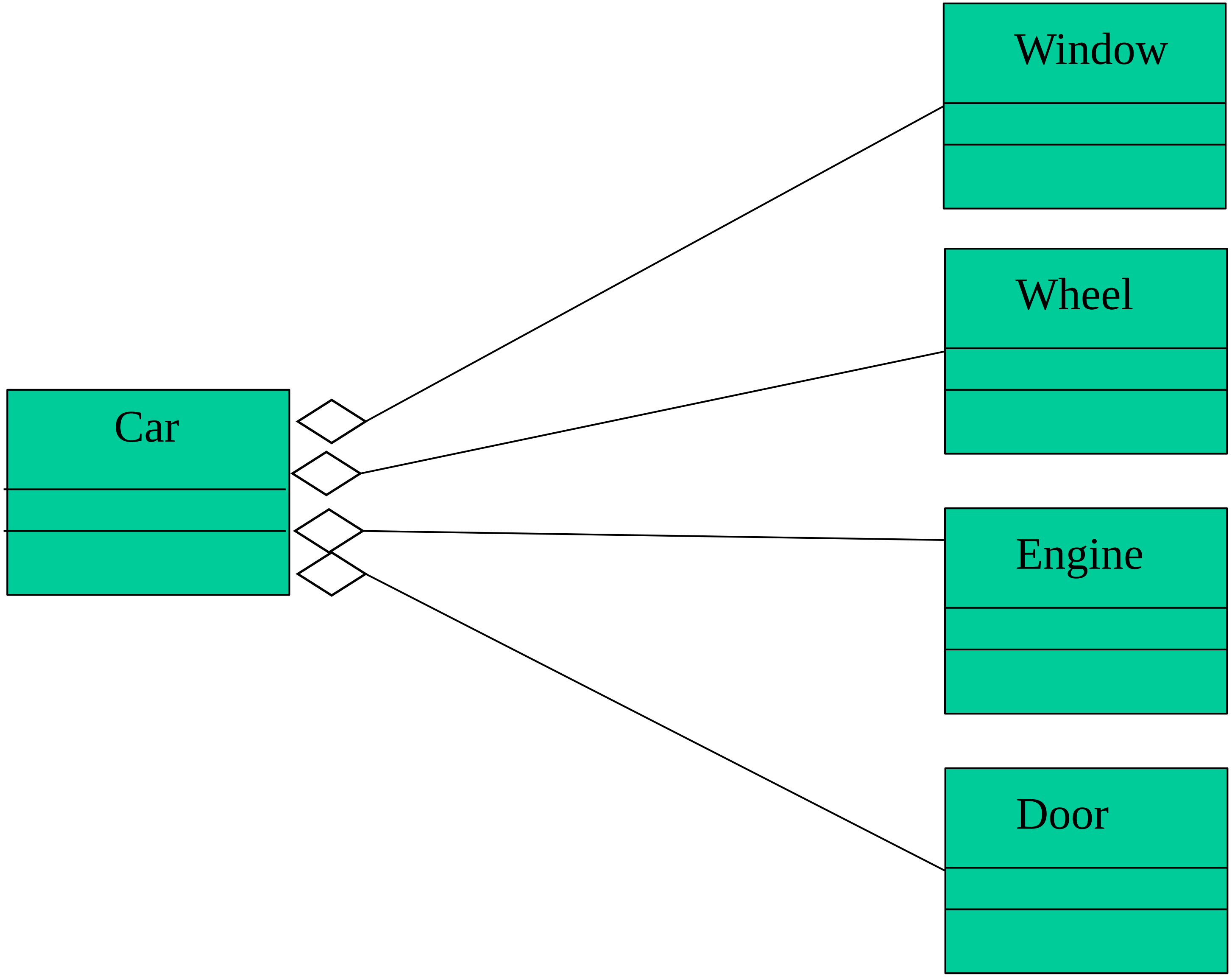
You might need to interview the client to confirm this one way or the other.

Examples





Example



Composition

Symbol ◆

Composition is stronger than aggregation.

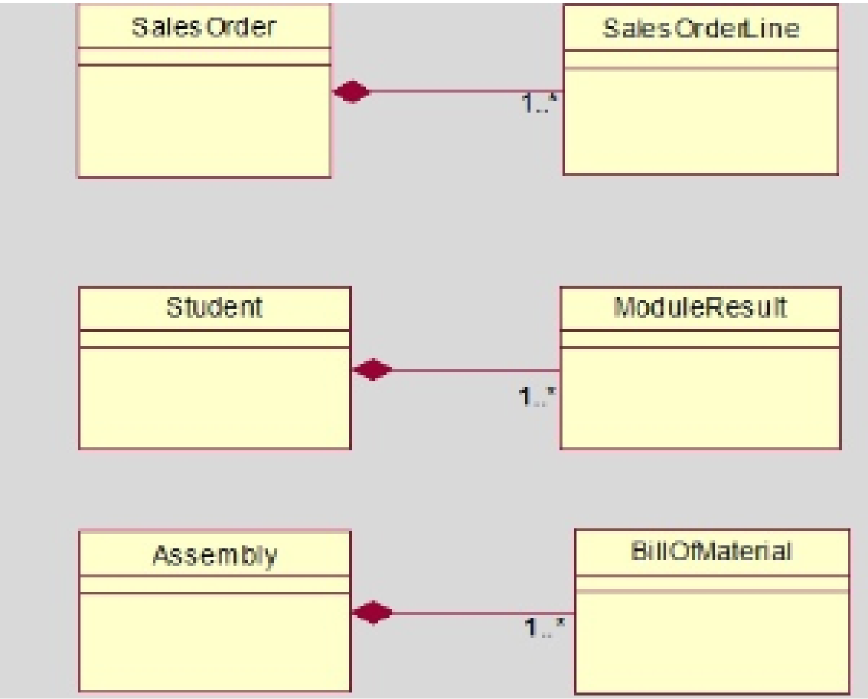
Composition also implies that one class is part of another class.

The solid diamond used in composition is stronger than the clear diamond used to show aggregation.

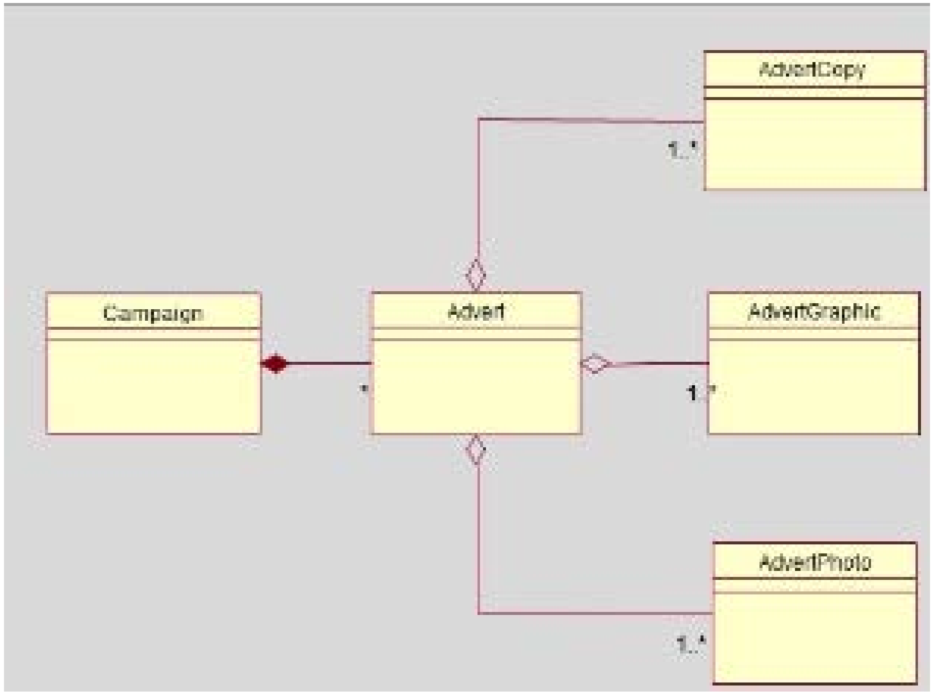
Implications of Composition

If an object of the ‘owning’ class is deleted, the object of the ‘owned’ class is also removed.

Example


SalesOrder
SalesOrderLine
1..*
Student
ModuleResult
1..*
Assembly
BillOfMaterial
1..*

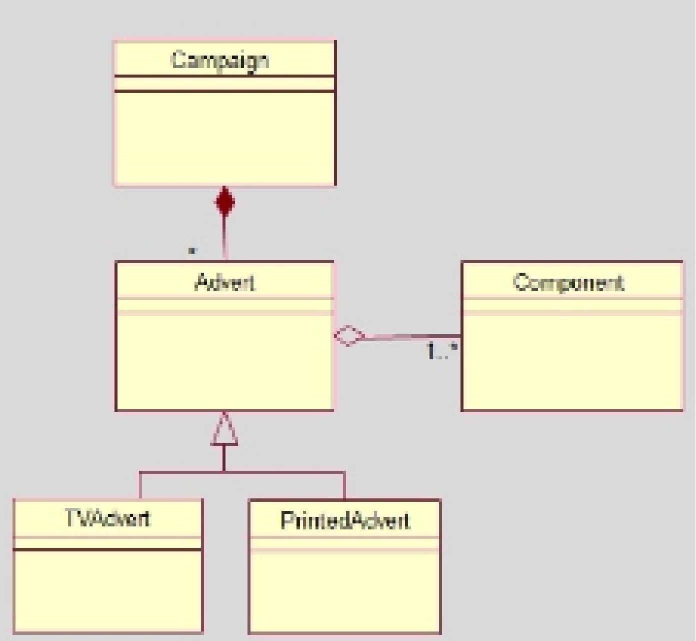
Example



The concern that photographs, graphics and even possible copy might be re-used has led the analyst to show the relationship from Advert to its components as aggregations.

Example

Simplify the model by creating a new class, ‘Component’ which covers all known and unknown parts of any advert.



N.B. Using inheritance, would make it easier to add a new kind of advert

Notes

Some OO experts feel that aggregation and composition are over used. A simple association may be preferable in many cases.

Sometimes an attribute maybe enough to hold data variables, rather than creating a whole new class. (This is also true for inheritance.)

Summary

Inheritance:

All Superclass features will be inherited by the Subclass.

A Subclass is a kind of Superclass (is-a).

Aggregation:

Special relationship; one class is part of another class.

Relationship maybe broken and class removed, but other remains.

Composition:

Special relationship; one class is part of another class.

If one class in relationship is removed, other class is deleted too.

The solid diamond ◆ *in composition is ‘stronger’ than*
the hollow diamond ◇ *used for aggregation.*