Object oriented Thinking, Analysis and Design

Dr. Fatma Meawad

Object Oriented Principles

- **Information Hiding:**
  - Minimize The Accessibility of Classes and Members
- **Encapsulation:**
  - "Encapsulation is a mechanism used to hide the data, internal structure, and implementation details of an object. All interaction with the object is through a public interface of operations." Craig Larman
- **Design by Contract:** Program To An Interface, Not An Implementation
- **The Open-Closed Principle:**
  - Software Entities Should Be Open For Extension, Yet Closed For Modification.
  - When requirements change, you extend the behavior of such modules by adding new code, not by changing old code that already works.

Further Readings if interested:

Effective Java: Josh Bloch

Object-Oriented Analysis and Design with Applications: Grady booch

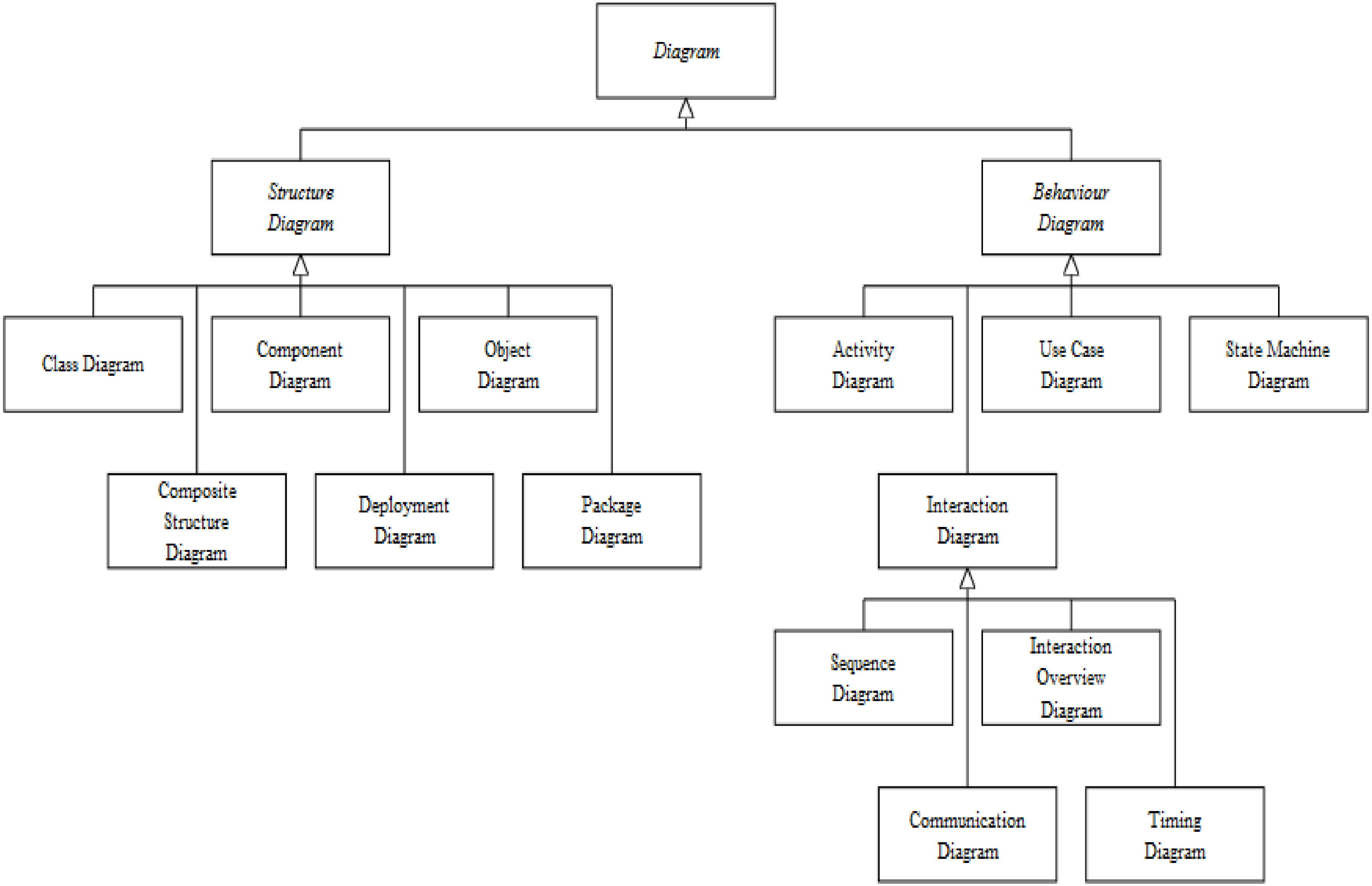
UML

- Unified Modelling Language
- *Three Amigos: Booch, Jackobson and* Rumbaugh
- With UML, we can create different types of diagrams that helps us through out the software development process

Why use UML?

- Abstraction
  - Visualising complex systems
  - Deliver ideas and concepts
- Learning OO
  - It is not easy to make the most of the OO design, the proposed diagrams get you started.
- Generating artefacts for communication

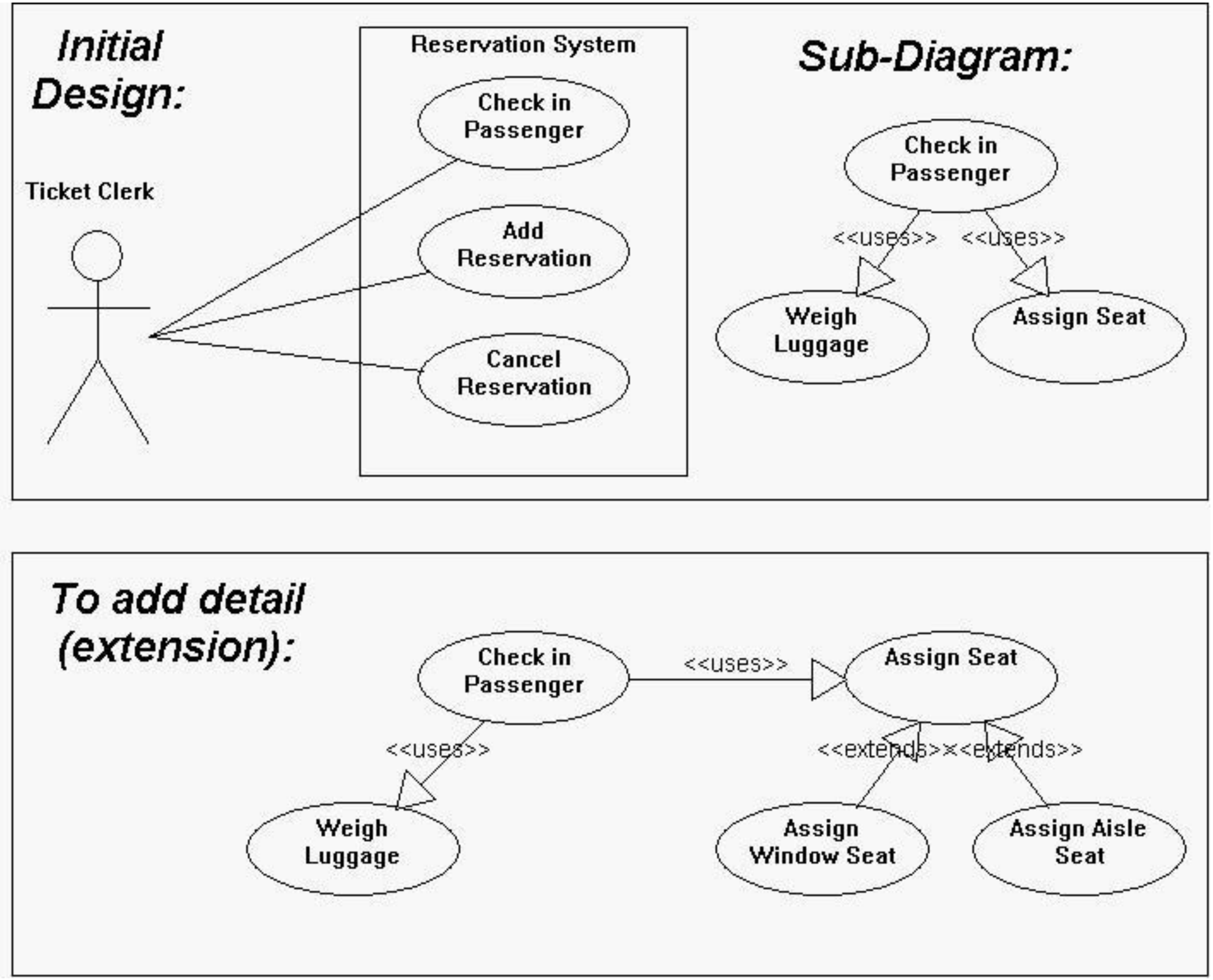
  With (customers, stakeholders, developers, etc)



http://upload.wikimedia.org/wikipedia/en/7/74/Uml_diagram.svg

When to use UML

- There are loads of diagrams, which to use and when
- Each set of diagrams is useful for a certain phase or activity
- Yet, we should only use them to support not to accumulate unnecessary artefacts



http://www.andrew.cmu.edu/course/90-754/airline-extends.jpg

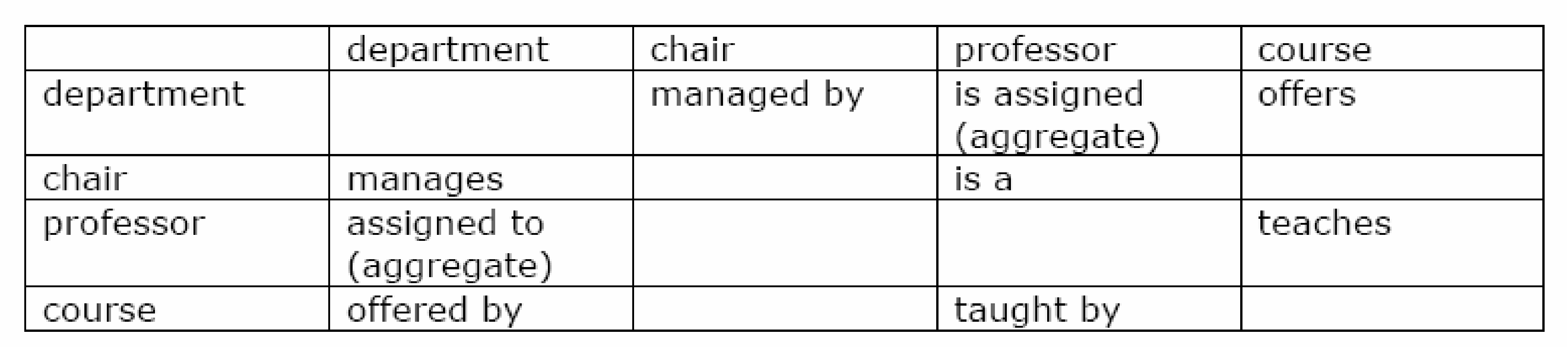
Class Diagrams

The purpose of class diagrams is modelling the types in your domain

1- Identify Nouns in your domain: classes

2- Identify Verbs in your domain: relationships

3- Find Associations: join the nouns with the verbs

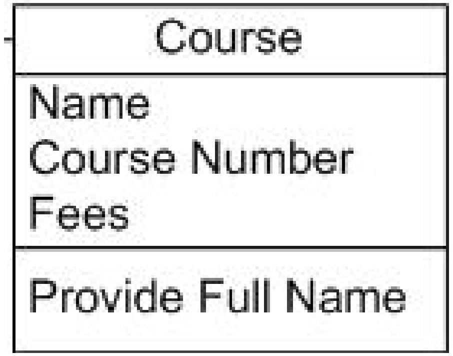


	department	chair	professor	course
department		managed by	is assigned (aggregate)	offers
chair	manages		is a	
professor	assigned to (aggregate)			teaches
course	offered by		taught by	

http://faculty.ksu.edu.sa/zitouni/203%20Haseb%20%20Lecture%20Notes/Class%20and%20ERD.pdf

How to draw Class Diagrams

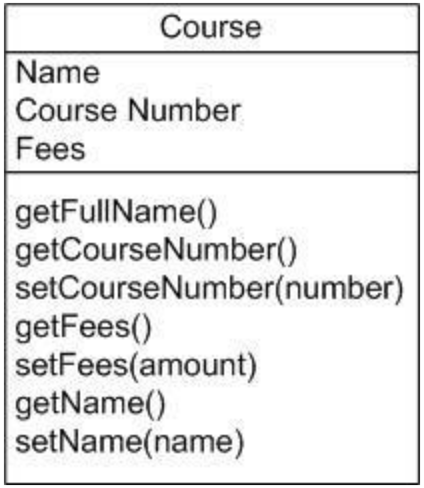
- Objects both know things (they have attributes) and they do things (they have methods)
- Classes are depicted as boxes with three sections, the top one indicates the name of the class, the middle one lists the attributes of the class, and the third one lists the methods



Course

Name
Course Number
Fees

Provide Full Name



Course

Name
Course Number
Fees

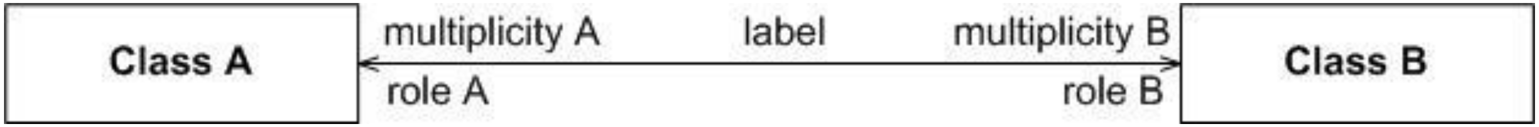
getFullName()
getCourseNumber()
setCourseNumber(number)
getFees()
setFees(amount)
getName()
setName(name)

No need to model getters and setters in your UML

http://www.agilemodeling.com/artifacts/classDiagram.htm

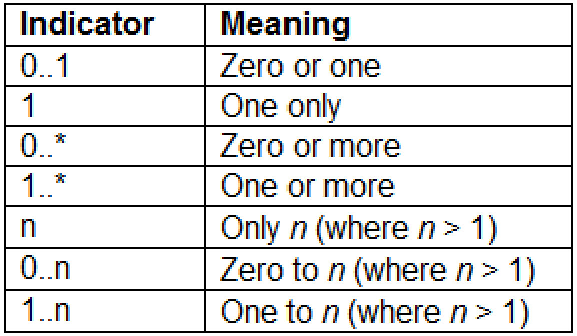
Association

- Objects are often associated with, or related to, other objects.



- Unidirectional
- Bidirectional

Table 1. Multiplicity Indicators.

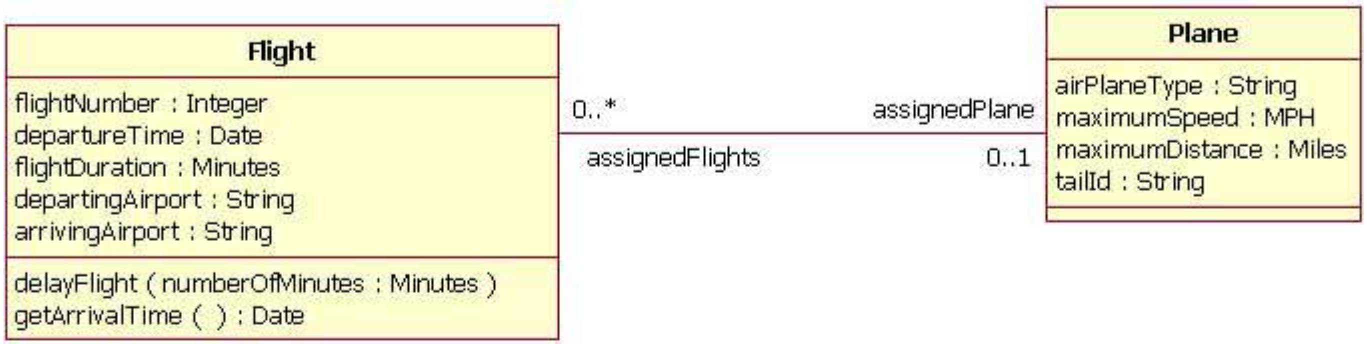


Indicator	Meaning
0..1	Zero or one
1	One only
0..*	Zero or more
1..*	One or more
n	Only *n* (where *n* > 1)
0..n	Zero to *n* (where *n* > 1)
1..n	One to *n* (where *n* > 1)

http://www.agilemodeling.com/artifacts/classDiagram.htm

Association Examples

Bi-Directional: Both classes know about this relationship, drawn by solid line, role names and multiplicity



Flight

flightNumber : Integer
departureTime : Date
flightDuration : Minutes
departingAirport : String
arrivingAirport : String

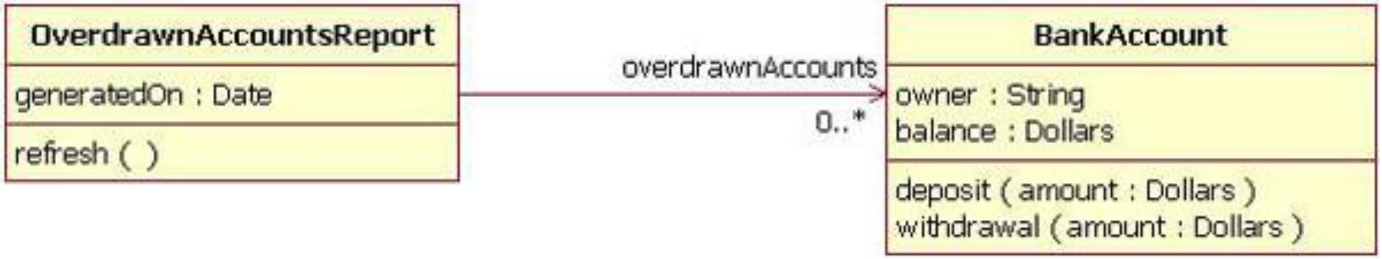
delayFlight (numberOfMinutes : Minutes)
getArrivalTime ( ) : Date

0..*
assignedFlights
assignedPlane
0..1

Plane

airPlaneType : String
maximumSpeed : MPH
maximumDistance : Miles
tailId : String

Uni-directional : Only one Class knows about the other, drawn by a solid line with an open arrowhead, role name and multiplicity



OverdrawnAccountsReport

generatedOn : Date

refresh ( )

overdrawnAccounts
0..*

BankAccount

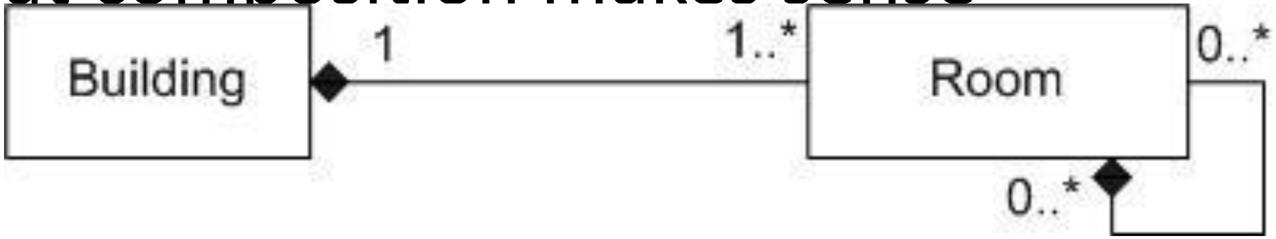
owner : String
balance : Dollars

deposit (amount : Dollars)
withdrawal (amount : Dollars)

http://www.ibm.com/developerworks/rational/library/content/RationalEdge/sep04/bell/

Composition Associations

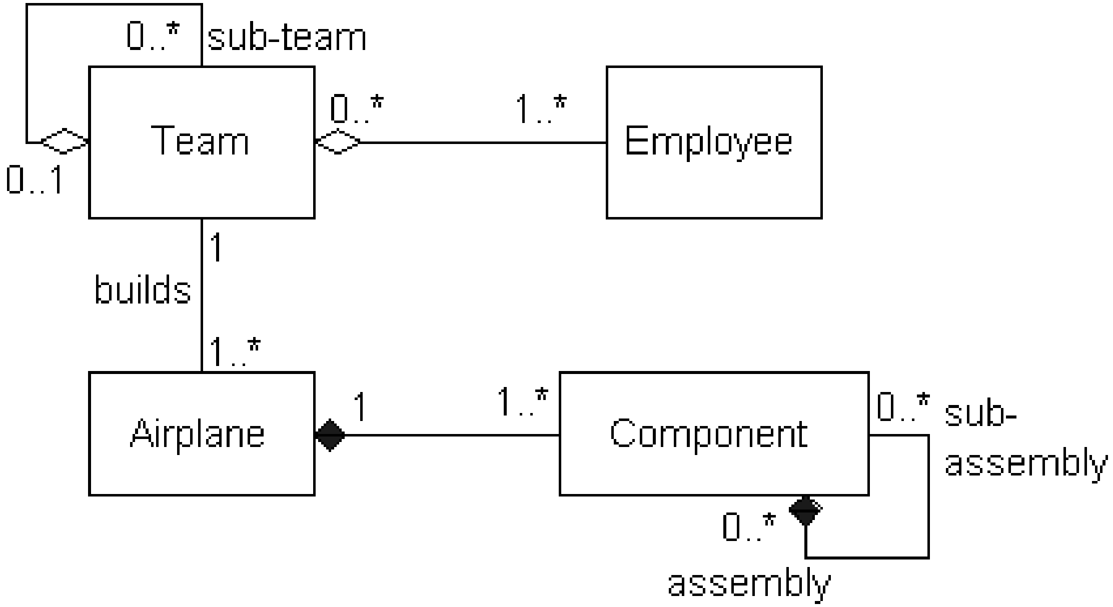
- Sometimes an object is made up of other objects
- if it makes sense to say that something is part of something else then there's a good chance that composition makes sense



http://www.agilemodeling.com/artifacts/classDiagram.htm

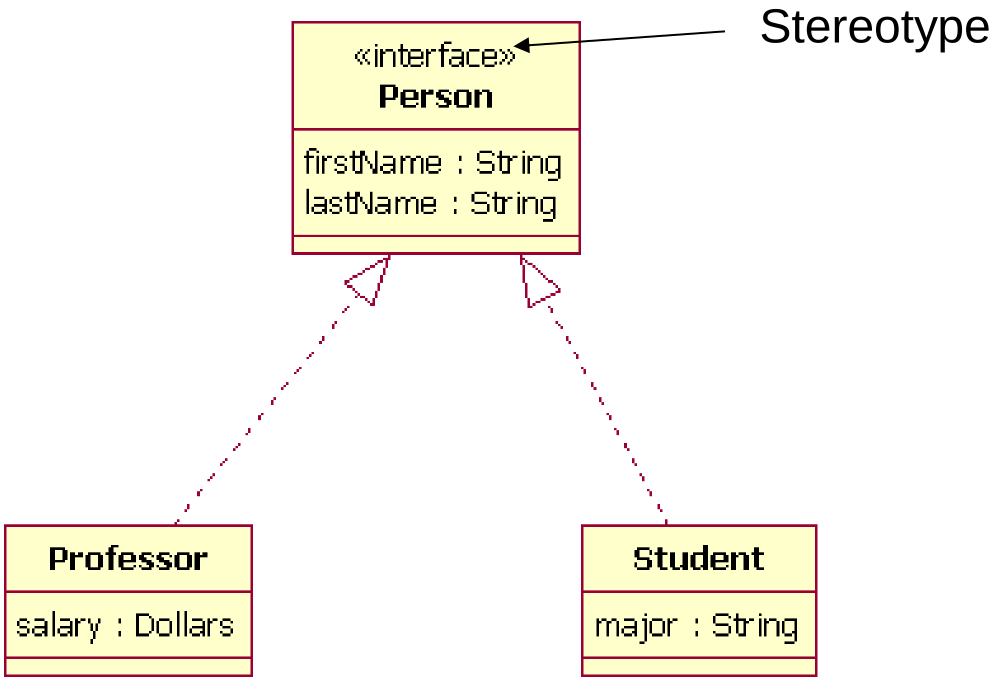
Composition Associations

- Another good indication that composition makes sense is when the lifecycle of the part is managed by the whole



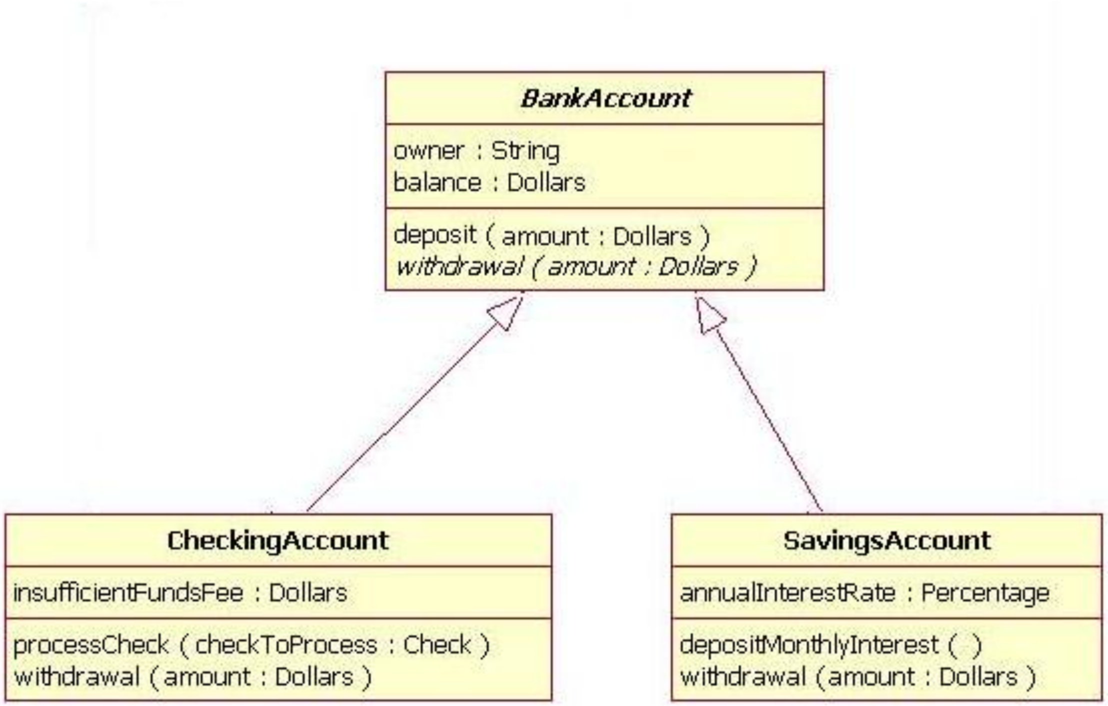
http://www.agilemodeling.com/style/classDiagram.htm#RelationshipGuidelines

Realize interfaces



http://www.ibm.com/developerworks/rational/library/content/RationalEdge/sep04/bell/

Generalization (inheritance)

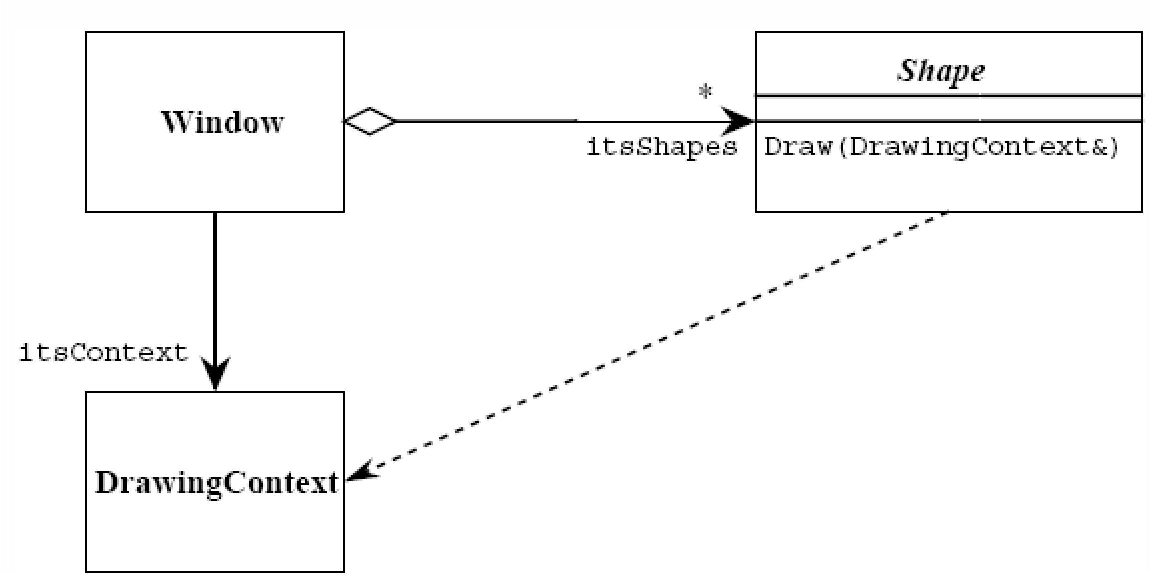


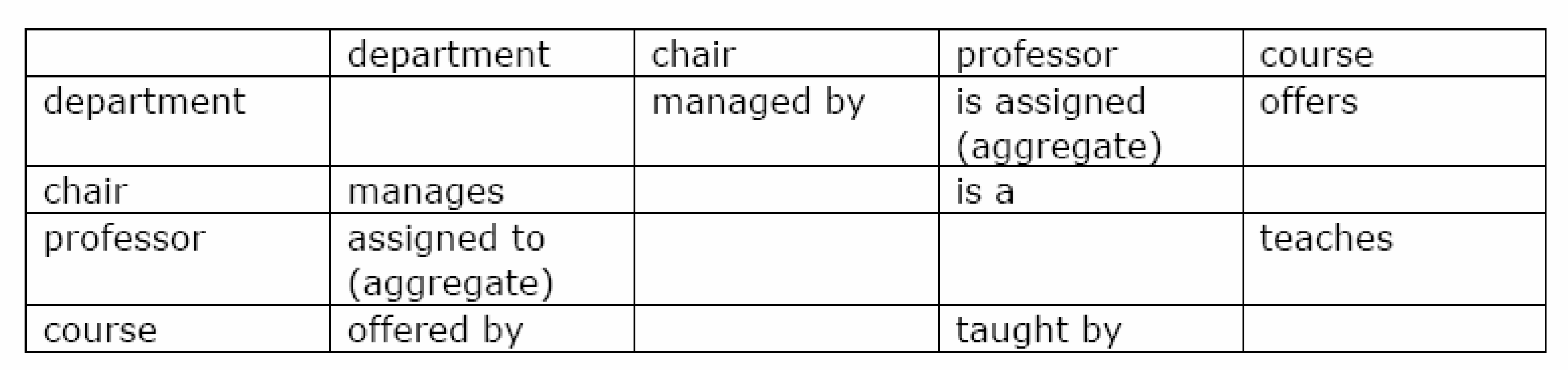
BankAccount
owner : String
balance : Dollars
deposit (amount : Dollars)
withdrawal (amount : Dollars)
CheckingAccount
insufficientFundsFee : Dollars
processCheck (checkToProcess : Check)
withdrawal (amount : Dollars)
SavingsAccount
annualInterestRate : Percentage
depositMonthlyInterest ( )
withdrawal (amount : Dollars)

http://www.ibm.com/developerworks/rational/library/content/RationalEdge/sep04/bell/

Dependency

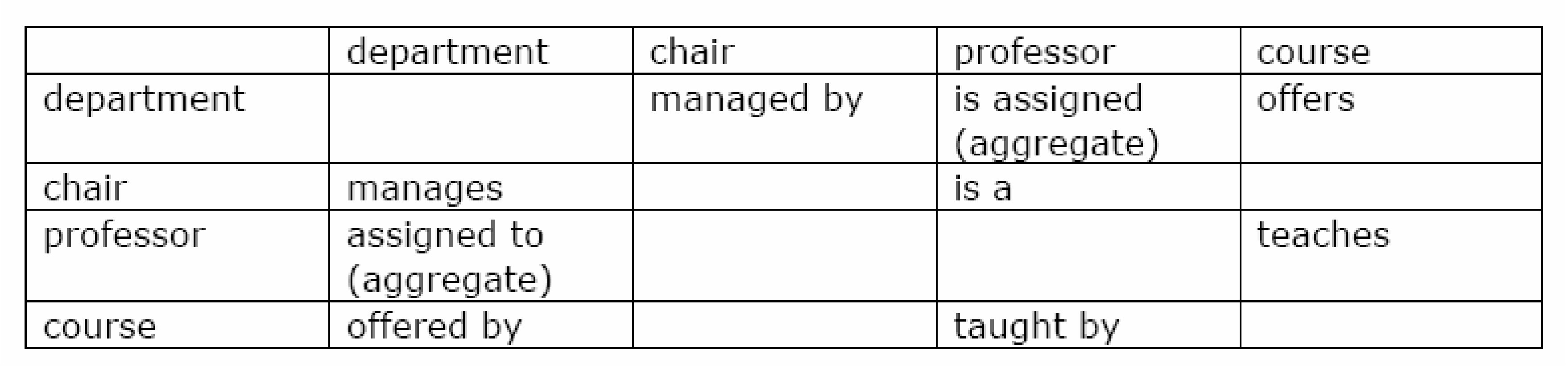
- We use a dashed open arrow when a class is simply using another class. There is no strong relation between them. (ex. A class is simply imported and used in this class)



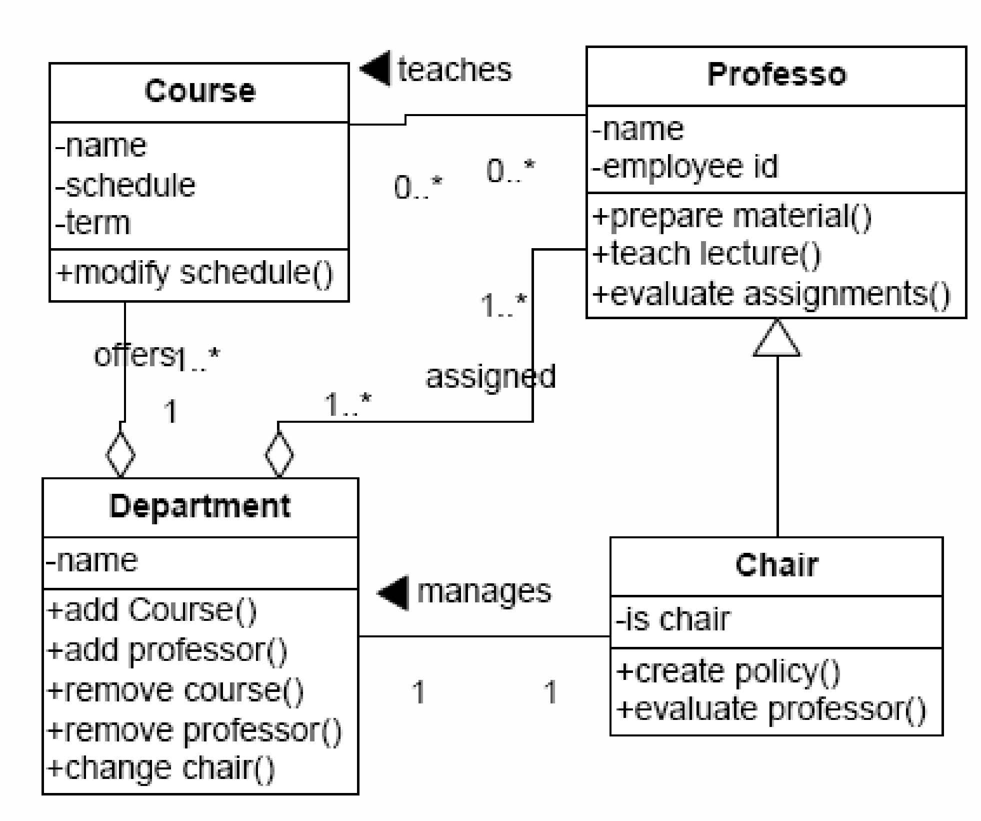


	department	chair	professor	course
department		managed by	is assigned (aggregate)	offers
chair	manages		is a	
professor	assigned to (aggregate)			teaches
course	offered by		taught by	

http://faculty.ksu.edu.sa/zitouni/203%20Haseb%20%20Lecture%20Notes/Class%20and%20ERD.pdf



	department	chair	professor	course
department		managed by	is assigned (aggregate)	offers
chair	manages		is a	
professor	assigned to (aggregate)			teaches
course	offered by		taught by	



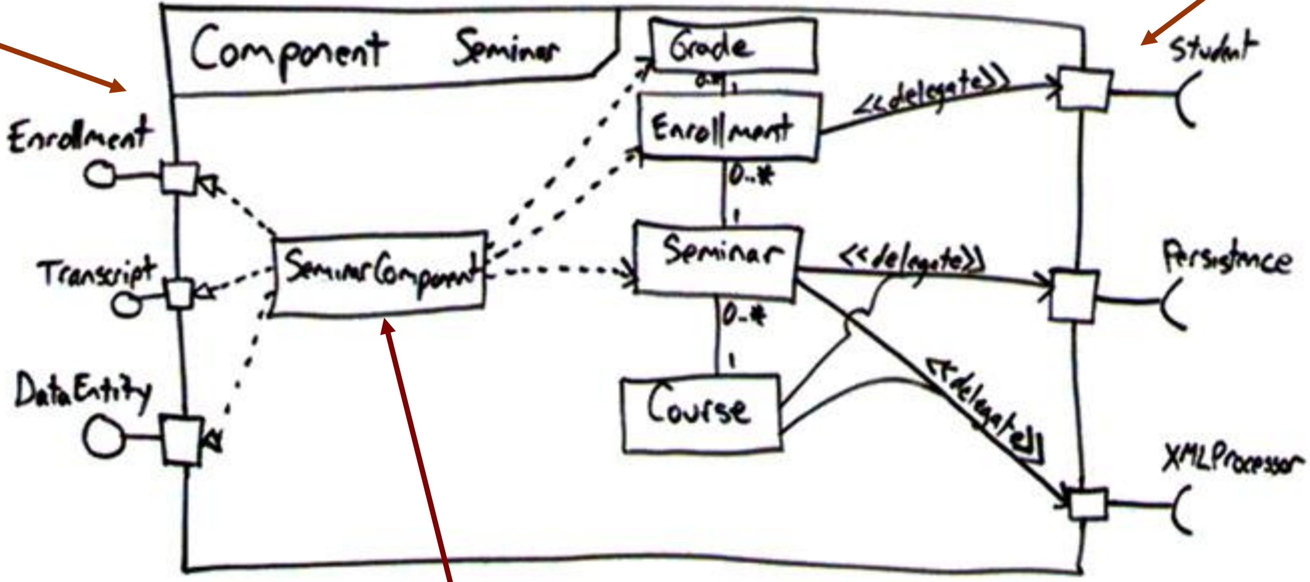
http://faculty.ksu.edu.sa/zitouni/203%20Haseb%20%20Lecture%20Notes/Class%20and%20ERD.pdf

Components' classes

- Hopefully , we will eventually have something like this for each component

An implemented
Interface ports

Required
interface



Class
SeminarComponent

http://www.agilemodeling.com/style/interface.htm