O-O Analysis and UML

What is the difference between analysis and design?

Quick test…

For those of you that don't remember Pacman, here is a brief description:

- Pacman is a game in which the user moves a yellow, chomping pie (Pacman) around a two dimensional maze.
- The maze is the scene of the action and it contains dots (that Pacman can eat), Pacman and the ghosts.
- The ghosts move around the maze trying to catch Pacman. If a ghost catches Pacman, Pacman dies.
- There are a few special, larger dots that render the ghosts harmless for a short period of time (in fact, during this time, Pacman can eat the ghosts).
- Eating all the dots on the board brings Pacman to the next level.

O-O analysis example

Identify the fundamental objects, the methods on those objects & relationships between the objects in Pacman:

Objects: pacman, ghost, bonus cherry, dot, board, game

Methods: move, change direction, eat, die, finish board

Relationships: pacman-dots, pacman-ghost (may change), pacman-cherry, board-game

Development perspectives

These are the terms used in Fowler and Scott, *UML distilled*, Addison Wesley (2000).

Problem (“conceptual” perspective)

- Concepts and relationships which “naturally arise” in the business
- …regardless of how you might actually implement them (which might not actually be o-o at all)

High-level design (“specification” perspective)

- Interfaces between bits of software

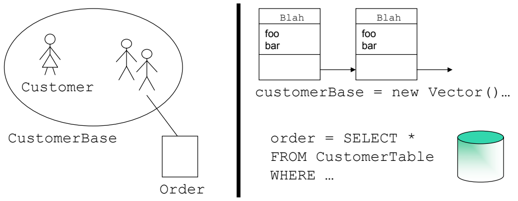
Detailed design (“implementation” perspective)

- The actual bits of code, reflecting the way the system is built

This is also called the solution domain

This falls halfway between problem and solution

Problem *versus* solution domain



```
customerBase = new Vector()…
```

```
order = SELECT *
FROM CustomerTable
WHERE …
```

Problem domain is the only one that counts (sort of)

- You'll probably find lots of possible applications, as well as the one they've asked for

It's important not to confuse problems and solutions

Why not jump straight to design?

Why not just design the system straight from conversations with the users?

Software can be influenced by infinite factors

- Danger of over-committing to a technology too early in the project
- O-O may *not* be the best solution; Java may *not* be the best language
- Solution domain issues don't usually impact the problem domain very much
- There are many solutions to the same problem

Customer only ever really understands the problem

- Conversation must take place at that level

…and probably don't know what they want anyway

- Help them to the "right" system for them

Where analysis meets design

Both are about building *models* of a real-world situation

- Analysis is about *understanding* the world; design is more about *simulating and manipulating* it

Seeking the *simplest adequate* model

- A snooker simulator needs to account for friction and spin
- …but probably not for special-relativity effects

Could do analysis using a programming pseudo-code

- Only really accessible to a software team, where what you want is user involvement
- Easy to get lost in the detail
- Structured graphical notations offer a good compromise between formality (for the software team) and intuitiveness (for the users)

Object-Oriented Analysis and Design

The heart of object-oriented problem solving is the construction of a model.

- The model abstracts the essential details of the underlying problem from its usually complicated real world.

O-O Analysis

- Real world entities represent objects

O-O Design

- Decomposing a system into objects
- Use some analysis technique like CRC cards

Brief Introduction to CRC Card

Class, Responsibilities, Collaborator (CRC)

- Index card that is used to represent the responsibilities of the class and its interactions with other classes
- CRC cards are an informal approach to object oriented modelling
- They are portable... No computers are required so they can be used anywhere
- A group interacts in a session using a set of CRC cards
- Once a reasonable set of classes have be assigned to a group, responsibilities can be added.

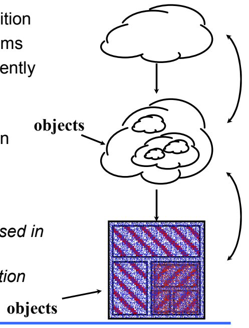
Object-Orientation

Manage the complexity using decomposition

- break the problem into sub-problems
- address those relatively independently

“Objects”

- used in design and implementation
  - *object-oriented design*
  - *object-oriented programming*
- in this course
  - *Unified Modelling Language used in Analysis and Design*
  - *Java/C++ used in implementation*



Where do objects come from?

From the sentence structure in interviews

- Nouns – the classes of object in the problem space
- Verbs – the activities (methods) those objects engage in
- “x of y” – possibly an association, possibly an attribute of an object

“Each customer’s account manager generates a bill for each of them at the end of each month”

From organisational charts and process descriptions

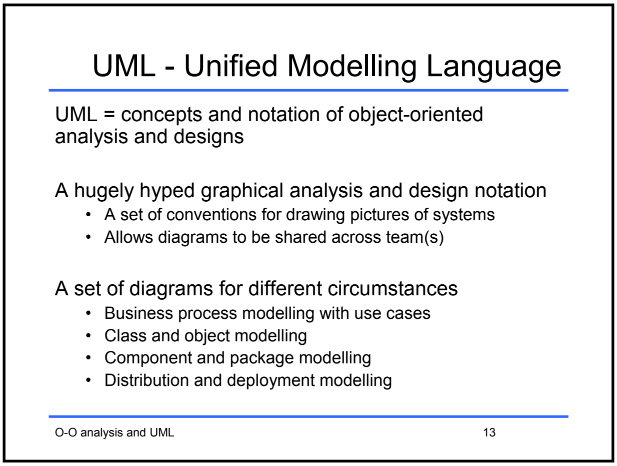
From observation of premises

From UML Diagrams….

UML - Unified Modelling Language

The Object Management Group (OMG) specification states:

"*The Unified Modelling Language (UML) is a graphical language for visualizing, specifying, constructing, and documenting the artefacts of a software-intensive system. The UML offers a standard way to write a system's blueprints, including conceptual things such as business processes and system functions as well as concrete things such as programming language statements, database schemas, and reusable software components.*"



UML - Unified Modelling Language

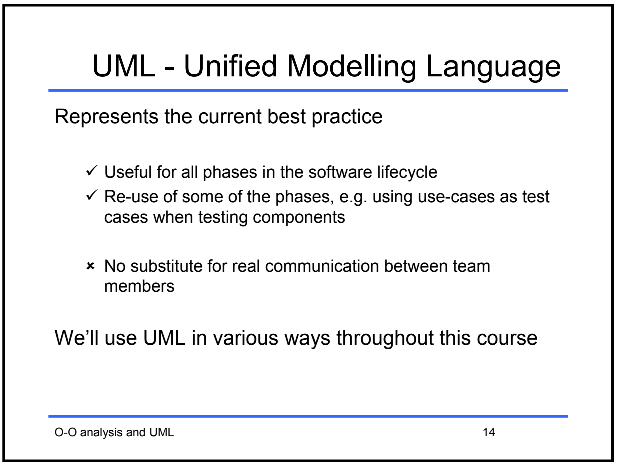
UML = concepts and notation of object-oriented analysis and designs

A hugely hyped graphical analysis and design notation

- A set of conventions for drawing pictures of systems
- Allows diagrams to be shared across team(s)

A set of diagrams for different circumstances

- Business process modelling with use cases
- Class and object modelling
- Component and package modelling
- Distribution and deployment modelling



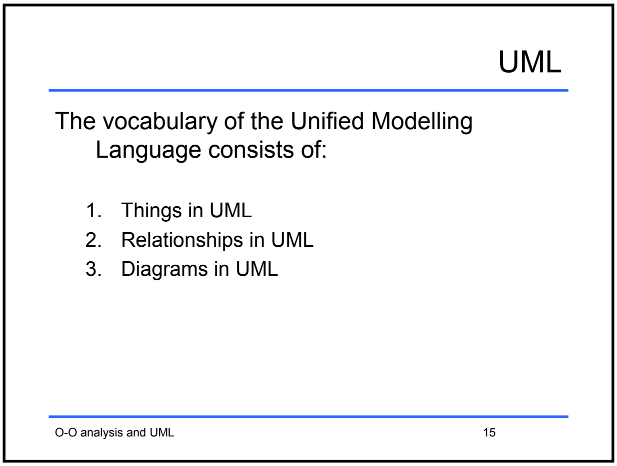
UML - Unified Modelling Language

Represents the current best practice

✓ Useful for all phases in the software lifecycle
✓ Re-use of some of the phases, e.g. using use-cases as test cases when testing components

✗ No substitute for real communication between team members

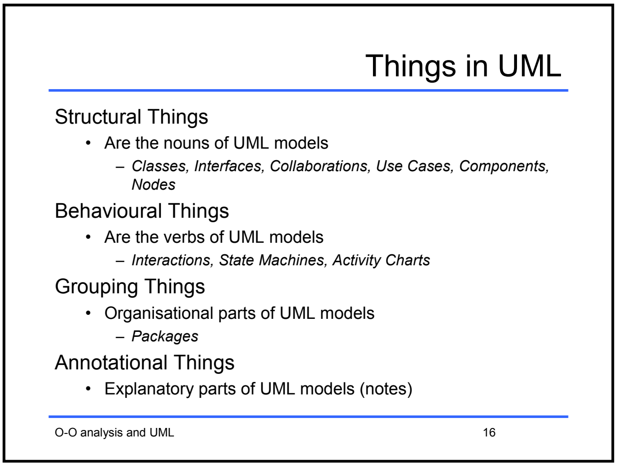
We'll use UML in various ways throughout this course



UML

The vocabulary of the Unified Modelling Language consists of:

1. Things in UML
2. Relationships in UML
3. Diagrams in UML



Things in UML

Structural Things

- Are the nouns of UML models
  - *Classes, Interfaces, Collaborations, Use Cases, Components, Nodes*

Behavioural Things

- Are the verbs of UML models
  - *Interactions, State Machines, Activity Charts*

Grouping Things

- Organisational parts of UML models
  - *Packages*

Annotational Things

- Explanatory parts of UML models (notes)

Relationships in UML

A Relationship is a connection between things:

- Connections between structural things:
  - *Connections between classes*
  - *Connections between use-cases*
- Connections between behavioural things:
  - *Objects in interaction diagrams*
- Connections between grouping Things:
  - *Dependencies between packages*

Diagrams in UML

1. Use Case Diagram
2. Sequence (or Interaction) Diagram
3. Collaboration Diagram
4. Class Diagram
5. Object Diagram
6. Statechart Diagram
7. Activity Diagram
8. Component Diagram
9. Deployment Diagram

Use cases

An interaction between a person and a computer

- Something the user does to achieve a recognisable goal

Actors

- A role someone takes on when using the system (accounts manager, customer, …)
- Not necessarily the same as a person – one person might have many roles and therefore be represented by several actors

Use case

- A named task or goal performed using the system
- One or more actors *benefit* from the use case, *e.g.* they make use of the task or function provided

Use Cases

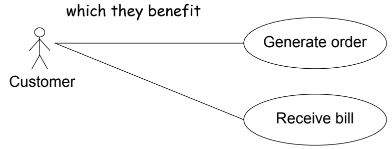
Use case diagrams are helpful in three areas:

- determining features (requirements)
  - *New use cases often generate new requirements as the system is analysed and the design takes shape*
- communicating with clients
  - *Their notational simplicity makes use case diagrams a good way for developers to communicate with clients*
- generating test cases
  - *The collection of scenarios for a use case may suggest a suite of test cases for those scenarios*

Use cases drive the whole development process

Use case set-up

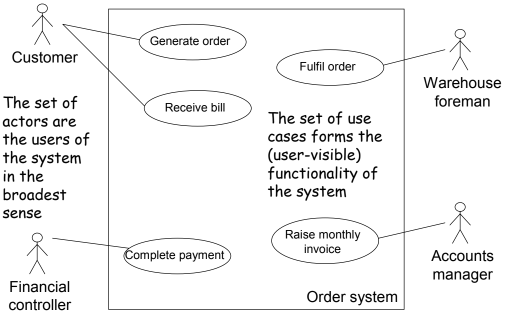
Connect each actor to the use cases from which they benefit



The communications indicate the stakeholding of actors in use cases – think about the WinWin spiral idea of reconciling people's win conditions

Should be stated in the users' terms: "add the receiver's address to the letter" not "generate a right-justified text block"

A use case system



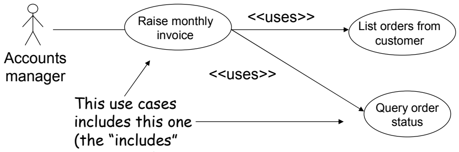
The set of actors are the users of the system in the broadest sense

The set of use cases forms the (user-visible) functionality of the system

Use case relationships

Use cases are related when (for example) one task makes use of the function described in another

- Might indicate something which can be re-used
- Can help with planning – might indicate a critical path in the development process, or something to rapidly prototype



This use cases includes this one (the "includes" stereotype)

Sequence (Interaction) diagrams

Intended to capture interactions

- What happens when, between whom, and in what order

Typically, a sequence diagram captures the behaviour of a *single* use case

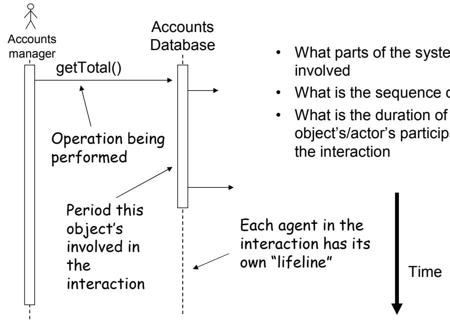
- Sequence diagrams show a number of objects and the messages that are passed between these objects within the use case
- Sequence diagrams are good at identifying the "objects" in a system and the messages (methods) between them

Surgeon’s warning

It’s here that the danger of confusing problem and solution domains really starts to bite

- It’s all too easy to get caught up in how the *application* will deal with a situation
- …when what you should be looking at is how the *people* deal with the situation
- …then you can refine towards the computer solution in design

The basic format



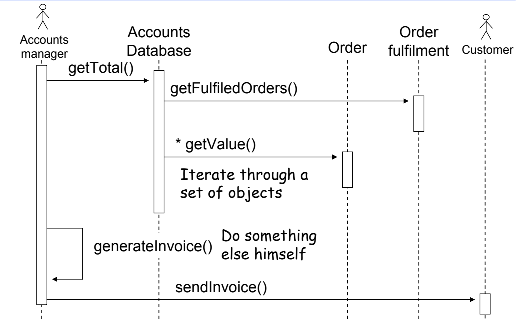
Operation being performed

Period this object’s involved in the interaction

Each agent in the interaction has its own “lifeline”

- What parts of the system are involved
- What is the sequence of actions
- What is the duration of each object’s/actor’s participation in the interaction

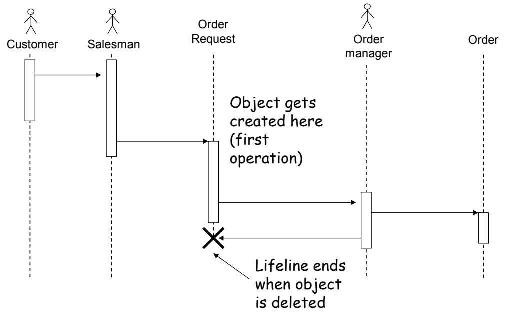
Raising invoices



Iterate through a set of objects

Do something else himself

Object lifespans



Object gets created here (first operation)

Lifeline ends when object is deleted

More detailed processes

Sequence diagrams give the high-level view of interactions between agents

Often need some way to write down the detail of a process

Could use pseudo-code

- More precise, less customer-accessible, too much detail

UML provides *activity diagrams* for this

- The sequence of actions and their consequences
- Remember flowcharts?…

Activity Diagrams

Write down the “atomic” activities in a process, and combine them into a process

- Sequential actions
- Any available parallelism, any synchronisation points

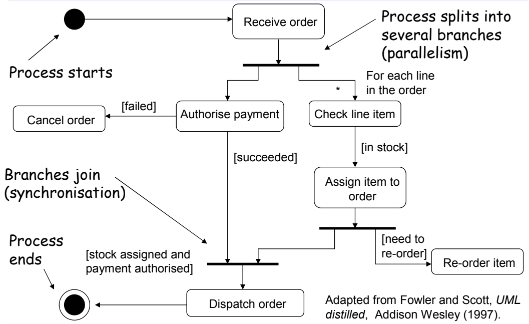
Remember, we’re still in the problem domain

- Don’t have to be fully specified or directly implementable

What they *do*, on the way to what they *should* do

- Changing the customer’s processes is process re-engineering
- The computer system may (should!) simplify many business processes

Order processing



Receive order

Authorise payment

Check line item

Cancel order

Assign item to order

Re-order item

Dispatch order

[failed]

[succeeded]

For each line in the order

*

[in stock]

[need to re-order]

[stock assigned and payment authorised]

Process starts

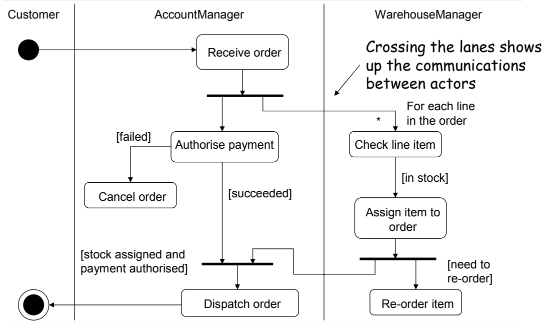
Process splits into several branches (parallelism)

Branches join (synchronisation)

Process ends

Adapted from Fowler and Scott, *UML distilled*, Addison Wesley (1997).

Swim lanes



Customer

AccountManager

WarehouseManager

Receive order

Authorise payment

Check line item

Cancel order

Assign item to order

Dispatch order

Re-order item

[failed]

[succeeded]

For each line in the order

*

[in stock]

[need to re-order]

[stock assigned and payment authorised]

Crossing the lanes shows up the communications between actors

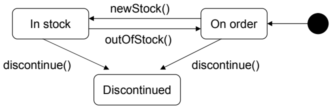
State Diagrams

An object often has some kind of state or mode that changes as a result of interactions

- Stock is acquired, depleted and discontinued

Show the legal state transitions as arrows

- Record all the important states
- Show how these states can evolve



Quick test…

Consider the features of a web-based reservation system for hotels

- List available hotels and rooms
- Book rooms in a particular hotel
- Provide booking confirmations

Describe one of the features of such a system as a use case

Model this use case as a sequence diagram

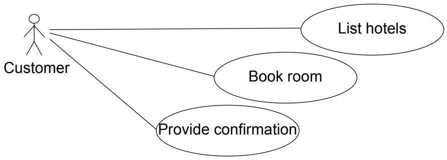
- What are the objects and the interactions between them

Present your solution…

Analysing a hotel reservation system

Use cases

- Based on observation, requirements, experience, ..
  - *List available hotels and rooms*
  - *Book rooms in a particular hotel*
  - *Provide booking confirmations*



Analysing a hotel reservation system

Sequence diagram

- One per use case

Identify objects (include actors)

- List hotels
  - *Customer, hotels database, hotel manager*
- Book room
  - *Customer, hotel manager, reservation system*
- Provide confirmation
  - *Customer, reservation system*

Identify operation / messages

- List hotels: getAvailableHotelList(), getRoomNumber()

Draw diagrams

- Put interactions in sequence, identify lifelines, include labels

Links across the problem model

There are obviously relationships between the different parts of a model

- States within an activity diagram should be defined by a class diagram
- Interactions probably imply associations and operations
- Can be tricky to track by hand as models grow

Computer-Aided Software Engineering (CASE) tools

- ✓ A structured editor to draw the diagrams
- ✓ Easier to add features to diagrams as they are discovered
- ✗ Can be big and complex to use
- ✗ Tools are no substitute for understanding the underlying processes of software engineering

UML Class Diagrams

A description of a set of common *problem domain* objects

A class diagram gives an overview of a system by showing its classes and the relationships among them

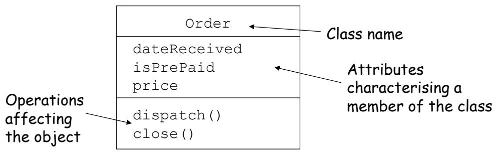
- Class diagrams are static -- they display what interacts but not what happens when they do interact

Arguably the most important model in UML

Classes

You can see the relationships with Java classes

- …and that's a two-edged sword – can be easy to over-commit to a *solution* when you're trying to think about a *problem*
- …but it also gives you a first-cut design later

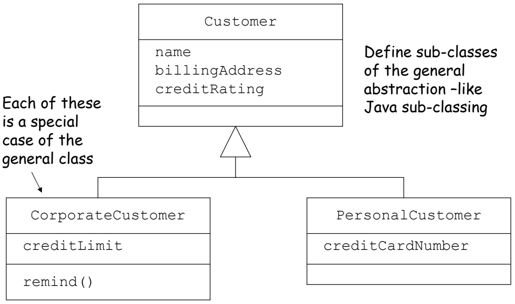

Order
dateReceived
isPrePaid
price
dispatch()
close()
Class name
Attributes characterising a member of the class
Operations affecting the object

Relationships in Class Diagrams

Relationships in UML class diagrams include:

- Generalization Relationship
- Realisation Relationship
- Association Relationship
- Aggregation Relationship
- Dependency Relationship

Generalisation Relationship



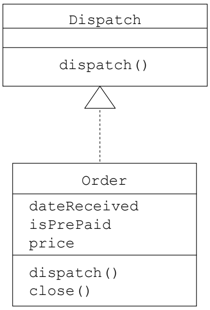
Each of these is a special case of the general class

Define sub-classes of the general abstraction -like Java sub-classing

Realisation Relationship

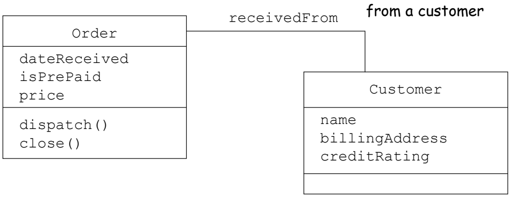
Realisation indicates that one class implements the behaviour specified by another

- One implementation class *realises* another
- The realising class must conform to the interface; but need not use inheritance
- The keyword `implements` in Java is an example implementation of the realisation relationship



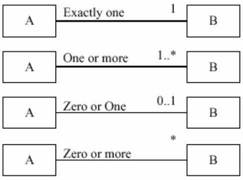
Association Relationship

Express the relationships between the problem domain classes



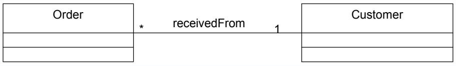
An order is received from a customer

Class associations - multiplicities

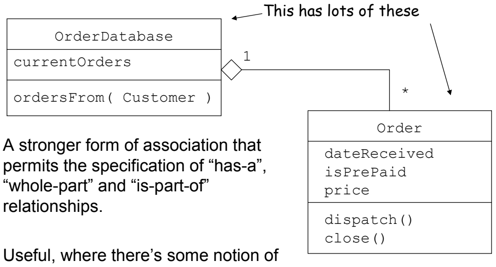


In the diagram below:

- Each customer has * (many) orders
- Each order has 1 customer



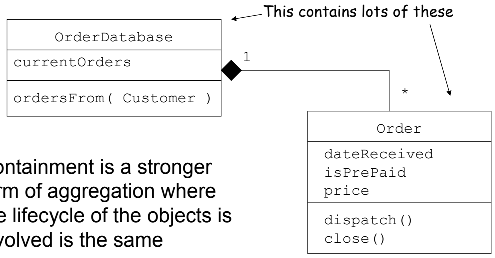
Aggregation Relationship



A stronger form of association that permits the specification of “has-a”, “whole-part” and “is-part-of” relationships.

Useful, where there’s some notion of objects of one class being “part-of” those of another.

Containment Relationship

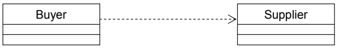


Containment is a stronger form of aggregation where the lifecycle of the objects is involved is the same

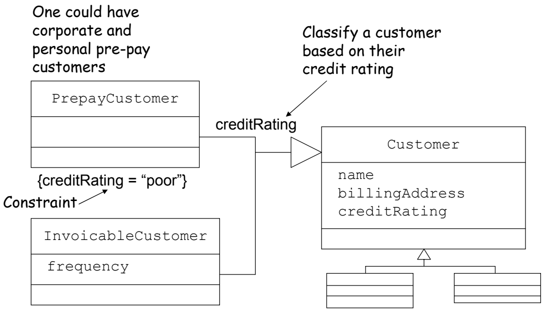
Dependency Relationship

Dependency is a using relationship that states that a change in the specification of a thing may affect another thing that uses it

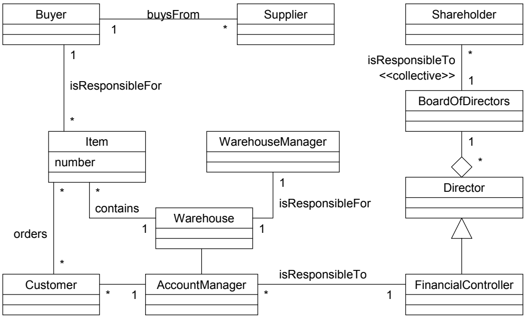
- E.g. the Buyer depends on the Supplier



Constraints: orthogonal classification



Classes – organisational structure



UML Models Summary

Major Area	View	Diagrams	Main Concepts
Structural	Static View	Class Diagram	class, association, generalization, dependency, realization, interface
	Use case view	Use case diagram	Use case, actor, association, extended, include, use case, generalization
Dynamic	State machine view	Statechart diagram	State, event, transition, action
	Activity view	Activity diagram	State, activity, completion transition, fork, join
	Interaction view	Sequence diagram	Interaction, object, message, activation

Bibliography

UML Distilled, Second Edition, Martin Fowler, Addison-Wesley, 2000.

Unified Modelling Language User Guide by Booch, Rumbaugh, Jacobson.

XP Tutorial

The Rules and Practices of Extreme Programming

http://www.extremeprogramming.org/rules.html