#### LECTURE 18: UML

Software Engineering Mike Wooldridge Lecture 18

Software Engineering

#### 1 What is UML?

- In the mid 1980s, a number of techniques began to emerge for *object-oriented-analysis and design*.
- Examples:
  - Coad & Yourdon;
  - Booch;
  - Rumbaugh (the OMT technique);
  - Coleman (FUSION).
- All used similar techniques, but differed on details of notation, etc.
- Mid 1990s: a move towards standardisation, driven by the *Object Management Group* (*OMG*):

The Unified Modelling Language (UML).

• UML is essentially a *notation*, and *not* a technique.

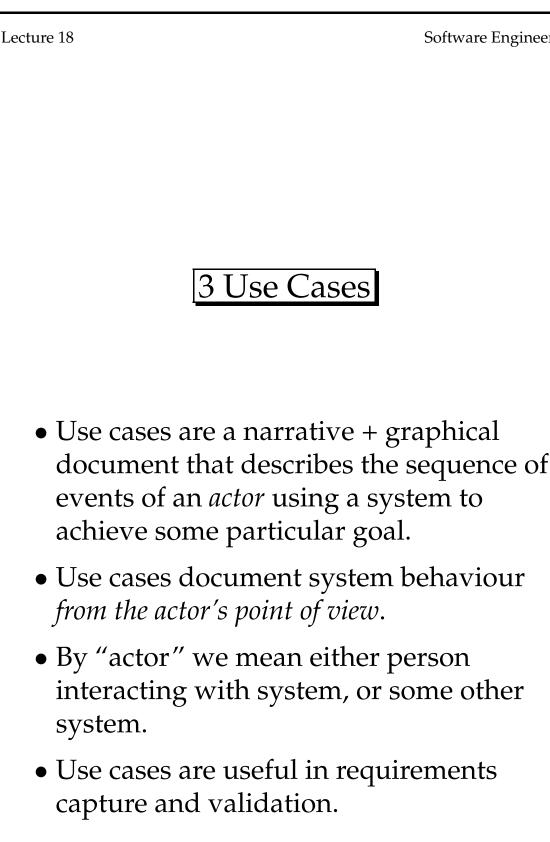
Notation can be used in many different ways: we show one.

## 2 UML Models

- UML provides a rich graphical notation for developing a series of *system models*.
- These models become increasingly less abstract, and more detailed.
- The models we discuss are *Analysis*:
  - use cases;
  - conceptual model;

Design:

- class model;
- interaction and collaboration model;



### 3.1 Schema for Use Cases

- Use case: [name of use case]
- Actors:

[list of actors that can participate, naming the initiator, and indicating key player]

• Purpose:

[one-line summary of *purpose* of the activity]

• Overview:

[short summary of the use case]

• *Type*:

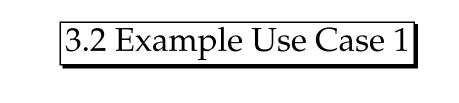
[primary or secondary, essential or optional]

• Cross references:

[references to requirements document]

• Course of events:

[narrative summary of use case]



- Use case: Buy items with cash.
- Actors: Customer (initiator), cashier
- Purpose:
   Capture a sale and its cash payment
- Overview:

Customer arrives at checkout with items to purchase in cash. Cashier records the items and takes cash payment. On completion, customer leaves with items.

• Type:

Essential, primary.

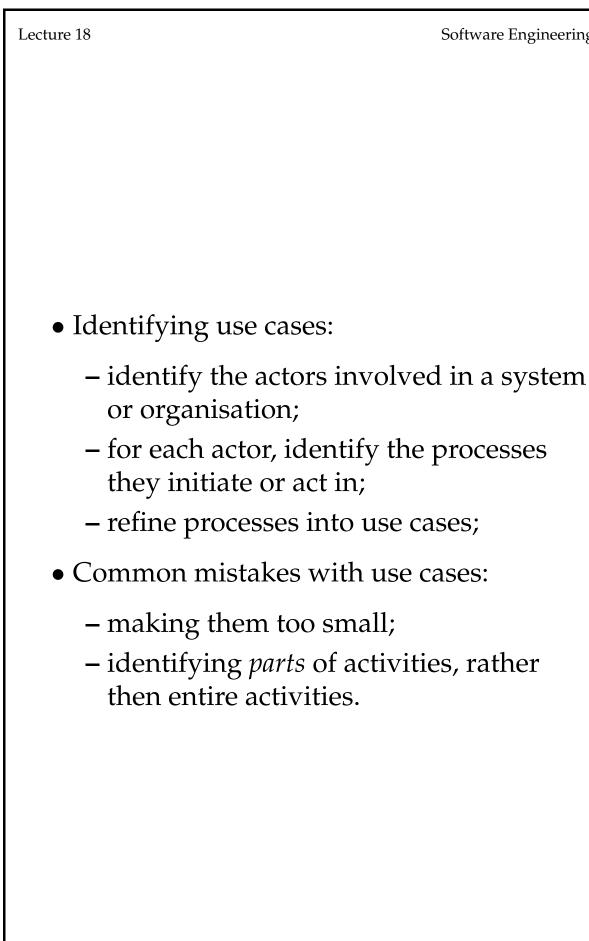
• Cross references:

Buy items with credit card, buy items with cheque.

- Course of events:
- 1. Customer arrives at checkout with items.
- 2. Cashier records identifier of each item.
- 3. As each item is recorded, system responds with running total of cost of items.
- 4. Cashier indicates to system that there are no more items.
- 5. System responds with overall total cost of items.
- 6. Cashier takes cash payment from customer for total cost, and indicates this to system.
- 7. System prints receipt for amount tendered.

Lecture 18	Software Engineering
<b>X</b> A7 1'	. 1 1
<ul> <li>We use use case diagram participants in use case</li> </ul>	
Mike Wooldridge	7

# • We capture a number of use cases in a single use case diagram:



Lecture 18

### 4 Conceptual Models

• The next stage in UML development involves identifying the key *concepts* in the system, and documenting the relationships between these concepts.

The resulting *conceptual* model will evolve into the *object* model.

• A conceptual model documents:

- the concepts in a system;
- relationships between concepts;
- attributes of concepts.
- Conceptual models are *not* design models.
   So avoid such concepts as:

– "database";

- "GUI" or "window";
- Key distinguishing feature of OO development:

Understanding system in terms of concepts rather than functions.

• What are candidates for concepts? – physical or tangible things e.g., receipt, plane - specifications, designs, or descriptions of things; e.g., product specification - places e.g., airport, point of sale terminal - transactions e.g., deposit, withdrawal - roles of people e.g., casher, customer - containers of things e.g., plane, store room - things in a container e.g., passenger

- other systems *e.g., www site* 

abstract noun concepts
 e.g., hunger

- organisations *e.g., sales department* 

– events

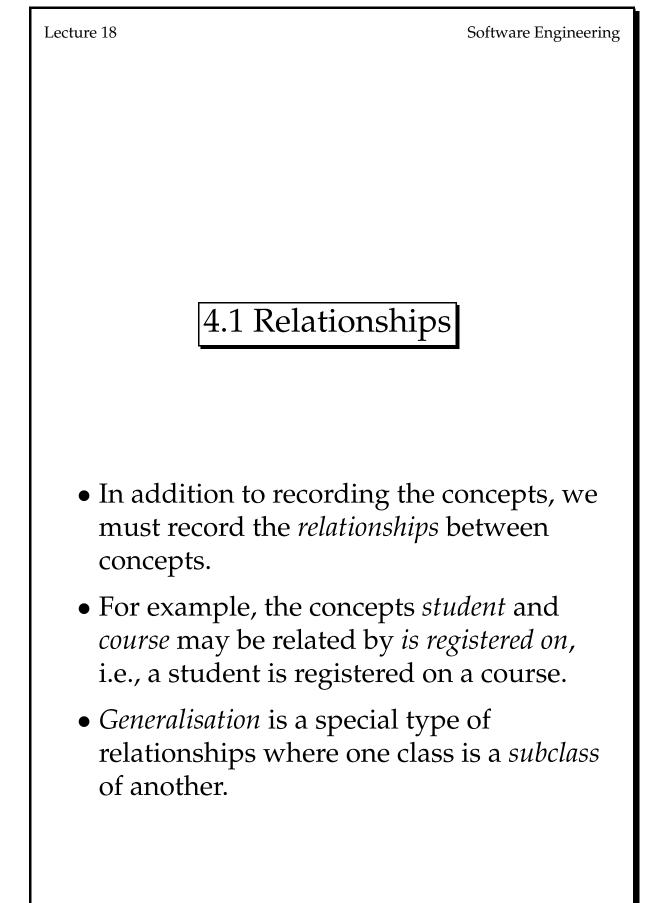
e.g., robbery, death

processes (sometimes)
 e.g., deposit

- rules and policies *e.g., refund policy* 

– catalogues
 *e.g., parts catalogues*

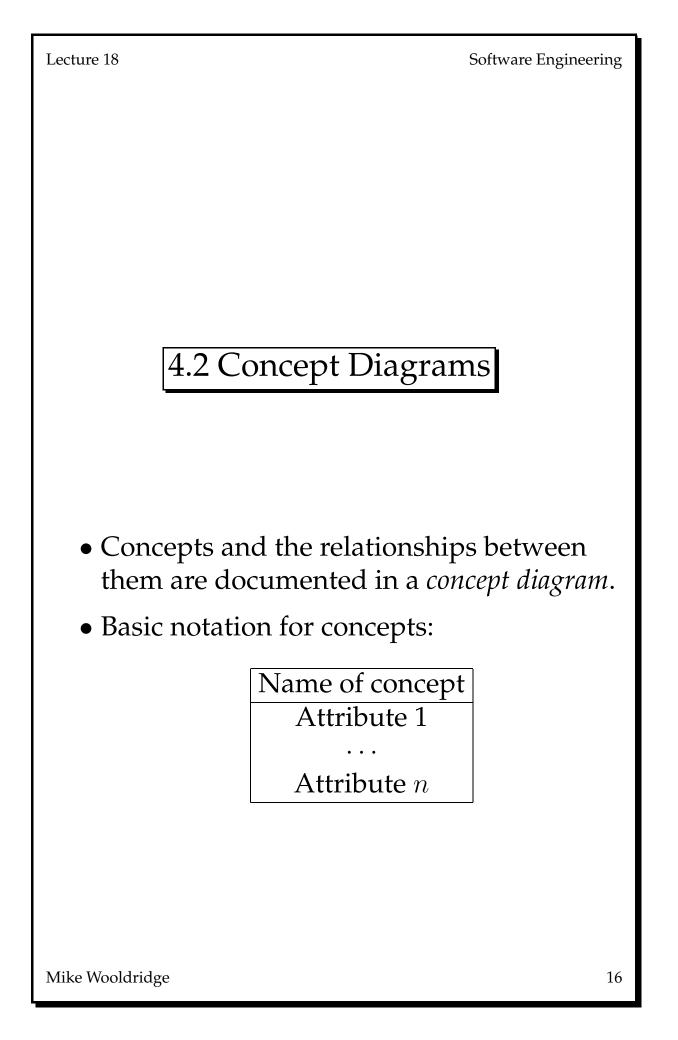
– contracts & legal documents



Common types of relationship:

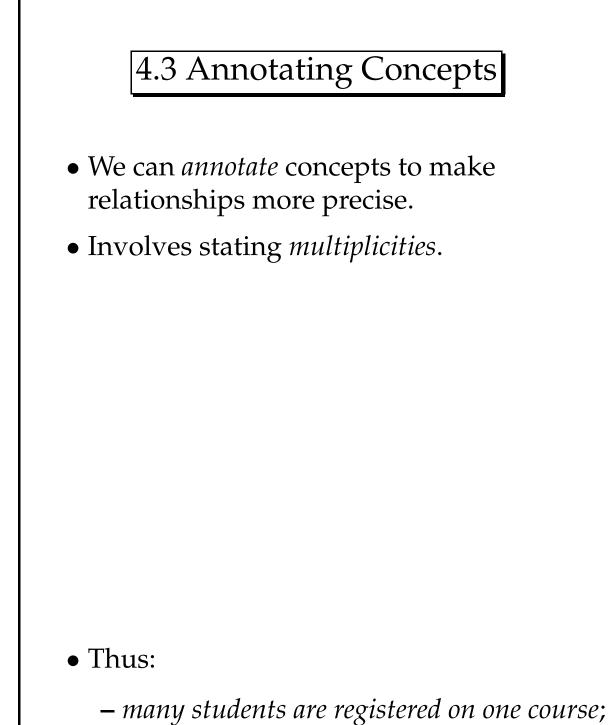
- is a physical part of; *e.g., wing-plane*
- is a logical part of *e.g., module-course*
- is physically contained in *e.g., passenger-plane*
- is logically contained in *e.g., flight-flight schedule*
- is a description for *e.g., flight description-flight*
- is reported/recorded in *e.g., reservation-flight manifest*

- is a member of *e.g., pilot-airline*
- is an organisational subunit of *e.g., department-store*
- uses or manages *e.g., pilot-plane*
- communicates with *e.g., customer-cashier*
- is related to a transaction *e.g., passenger-ticket*
- is owned by *e.g., plane-airline*



• Example:

- *Concepts*: student, course, module
- *Relationships*: registered-on, has
- Thus:
  - *students are registered on a course;*
  - a course has modules.



- one course has many modules.

Lecture 18	Software Engineer	ing
• Possible annot	ations:	
1	one	
n	exactly <i>n</i>	
mn	between $m$ and $n$	
*	zero or more	
m, n	either $m$ or $n$	
	tifying concepts is much nt than relationships or	
Mike Wooldridge		19

# 4.4 Concept Attributes

- Attributes should correspond to *simple* data types.
- Common attributes:
  - address;
  - colour;
  - phone number;
  - serial number;
  - universal product code/barcode;
  - postal or ZIP code;
- Attributes should *not* be *composite*!
- Bad attributes:
  - URL
    - is composed of three things
  - flight destination
    - is an airport a separate concept altogether;

	Software Engineering
<ul> <li><u>4.5 Generalisati</u></li> <li>UML includes a special sy</li> </ul>	
generalisations.	
• Thus:	
<ul> <li><i>rectangle</i> is a generalisa</li> <li><i>shape</i> is a generalisation</li> </ul>	-
Mike Wooldridge	21

- The generalisation relationship is actually the *subclass* relationship.
- Thus rectangle is a subclass of shape, and square is a subclass of rectangle.
- A subclass *inherits* all the attributes of its superclass.
- The generalisation relation is *transitive*: thus square is also a subclass of shape.
- Generalisation can prevent proliferation of similar classes ... but use it sparingly!

# 4.6 Finding Concepts

- We find concepts by looking for nouns: The <u>library</u> contains <u>books</u> and journals. It may have several copies of a <u>book</u>. Books are written by <u>authors</u>. Some <u>books</u> are for <u>short term loan</u> only. All other <u>books</u> may be borrowed by any <u>library</u> member for up to three <u>weeks</u>. Members of the <u>library</u> can normally borrow up to 6 <u>items</u> at a time, but <u>members of staff</u> may borrow up to 12 <u>items</u> at a time.
- Avoid concepts that are:
  - trivial (attributes);
  - redundant (duplicated concepts);
  - vague;
  - part of the meta-language;
  - outside the scope of the system;

Lecture 1	8
-----------	---

• The library conceptual model:

• Missing concepts?



- The conceptual model of a system will evolve into an *object model*, which captures the *static* aspects of a system.
- To capture the dynamics of a system, we need a *collaboration diagram*.
- A collaboration is:
  - a collection of *linked* objects;
  - which work together to achieve a task;
  - by invoking methods on each other.
- A collaboration captures stereotypical *control flows* (method invocations) between collaborating objects.
- NB: At least one collaboration diagram for each use case.

# 5.1 Example Collaboration Diagram

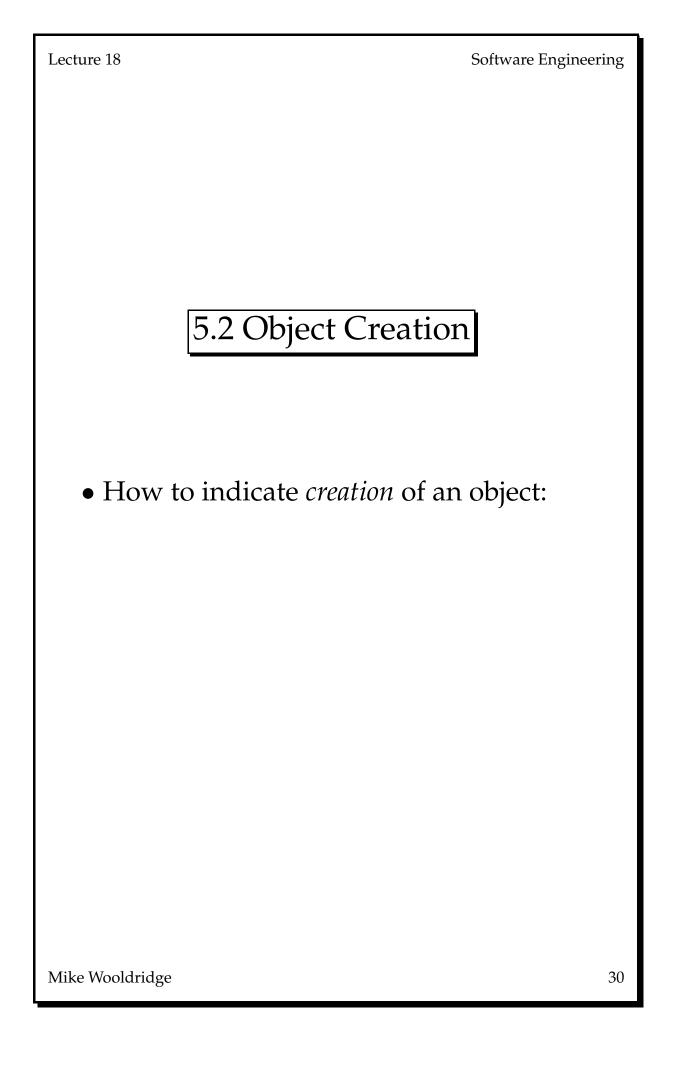
#### Intuition:

- Person (borrower) begins by asking to borrow a copy of a book.
   This corresponds to a *method* borrow(theCopy) being invoked on the corresponding libraryMember object.
- 2. The library member object then asks *itself* whether it is allowed to borrow a book.
- 3. The LibraryMember object then invokes the borrow() method on the Copy object corresponding to the copy of the book that the user requests.

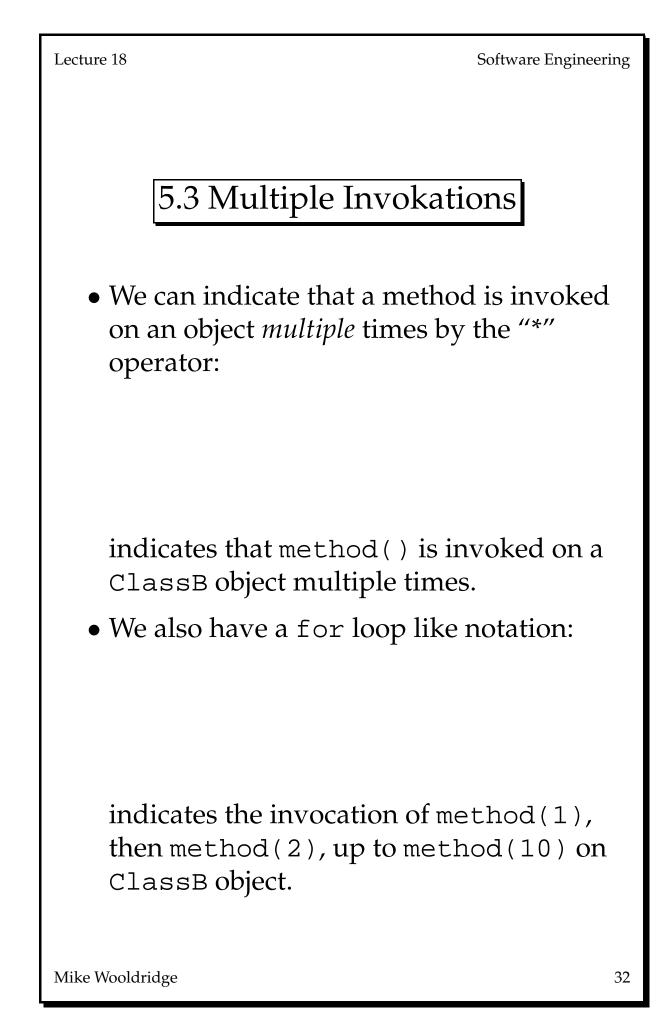
#### Note that:

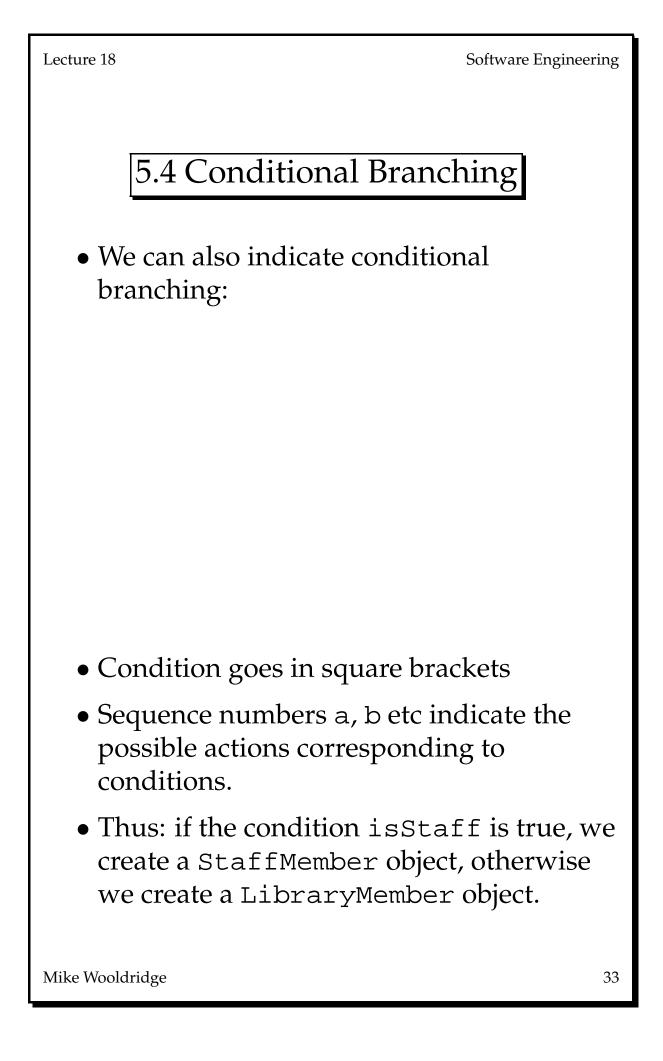
- Boxes correspond to objects.
- Names inside boxes are the class of the object.
- Arrows correspond to method invocations.
- The numbers that label method invocations indicate the order of events.
- If an object is invoked by invocation number x then its method invocations will be numbered x.1, x.2, and so on.

Lecture 18	Software Engineering
<ul> <li>An object cannot "spontaneor a method: it can only invoke <i>response</i> to methods being in</li> </ul>	e methods <i>in</i>
<ul> <li>In order for an object o1 to in method on another object o2 <i>a handle on it</i>:</li> </ul>	
<ul> <li>– 01 may have 02 as an ins</li> <li>– 01 may have been passed argument.</li> </ul>	_
There are clear implications <i>implementation</i> of objects.	for the
Mike Wooldridge	29



```
Lecture 18
                                         Software Engineering
   • Equivalent of following Java code:
     class Library extends Object {
       . . .
       void newLibraryMember(details : Details) {
         LibraryMember newLibraryMember =
           new LibraryMember(details);
         addNewMember(newLibraryMember);
       }
        . . .
     }
Mike Wooldridge
                                                      31
```





# 6 Sequence Diagrams (Swimlanes)

• An alternative to collaboration diagrams is sequence diagrams:

• The *y* axis is time;

Lecture 18

- Dotted line indicates the lifeline of an object
- Horizontal arrows indicate method invocation;
- Vertical blocks indicate periods of object activity.



- Statecharts, or state diagrams, illustrate the behaviour of an object its *lifecycle*.
- UML uses a finite state machine-like model of statecharts:
  - directed graph;
  - nodes correspond to states of the system;
  - arcs correspond to events that cause state changes;
  - start state:
  - end state:

Mike Wooldridge

Lecture 18

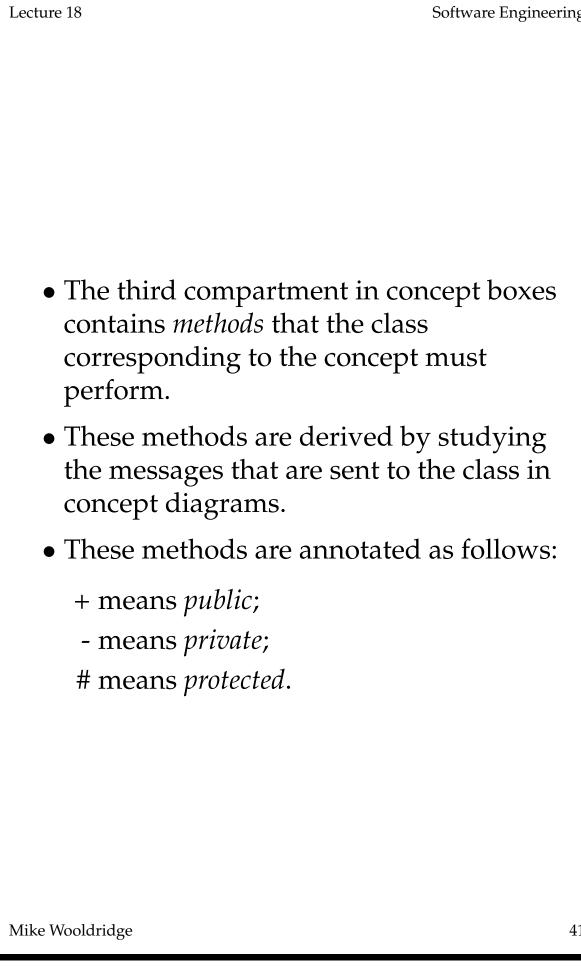
#### Lecture 18

• We can associate *guards* with transitions; these are conditions that must be true if the state transition is to occur.

### 8 Class Diagrams

- The final UML model we look at is the *class diagram* or *object model*.
- This model is an *elaboration* of the *conceptual model*.
- How to construct a class diagram:
  - using concept model and collaboration diagrams, identify the classes in your system;
  - 2. using collaboration diagrams, name the methods;
    (if you invoke m on an object o of class C, then C must provide method m)
  - 3. determine visibility of methods;
  - 4. determine *navigability*.

• An example class diagram.



# 8.1 Generating Code

- The transformation of class diagrams is now straightforward:
  - From class diagrams...
    - attributes become instance variables;
    - methods become methods!
- One final step involves determining *navigability*.
- If there is a relationship between C<sub>1</sub> and C<sub>2</sub>, then C<sub>1</sub> may need an instance variable corresponding to record this.
- Example: Book class has relationship to Author.

So may want instance variable Author in Book class.