From requirements to designs (EE5551)

Prof. Peter R Hobson

Modelling a Library

- This is a case study based on that described in Stevens P, Pooley R “Using UML”
- Used because it is simple to understand but detailed enough not to be trivial.
- I will try to bring out some of the system modelling aspects as well as the UML.
- I won’t discuss implementation details, *that is we will concentrate on analysis and design*

The problem

- “You have been contracted to develop a computer system for a university library. The library currently uses a 1960’s program, written in an obsolete language, for some simple bookkeeping tasks, and a card index for user browsing”

Clarify the requirements

- Detailed analysis of user requirements
- A complex task
  - Different users have different priorities
  - Users are often not clear about what they want
  - Problem of imagination (i.e. model compared to reality)
  - Managers, talking to developers, may not be users

Simplified system

- Books & journals
  – May be multiple copies of a book
  – Only library staff members may borrow journals
- Borrowing
  – System must keep track of borrowing and returning
- Browsing
  – Users must be allowed to search for a book by topic, author etc.
  – All users of the library can browse

Use Cases - definitions

- A *reminder* of some definitions
  - Actor: something (person, machine etc.) with behaviour. An actor plays a role. An actor has a goal.
  - Scenario: a *specific* sequence of actions and interactions between actors and the system
  - Use case: a collection of related success and failure scenarios that describe how actors use the system to achieve a goal.

Use Cases

- Use cases are *functional requirements,* they indicate what the system will do.
- Use cases are *primarily* text documents *not* diagrams.
- Use cases are *black-box*. The internals of the system are not described. The system is assumed to have *responsibilities*.
- Use cases help us to describe *what* while ignoring *how*.

Identifying classes

- This is one of the core skills of OO development.
- Crucial to building extensible and resuabale systems.
- A number of approaches – there is no single correct way to do this.
- We will illustrate the *noun identification technique*.

Precise statement of requirements

- Books and Periodicals

  The library contains books and journals. It may have several copies of a given book. Some of the books are for short term loan only. All other books may be borrowed by any library member for three weeks. Members of the library can normally borrow up to six items at a time, but members of staff may borrow up to 12 items at one time. Only members of staff my borrow journals.

- Borrowing
  - The system must keep track of when books and journals are borrowed and returned, enforcing the rules described above.

Precise statement of requirements

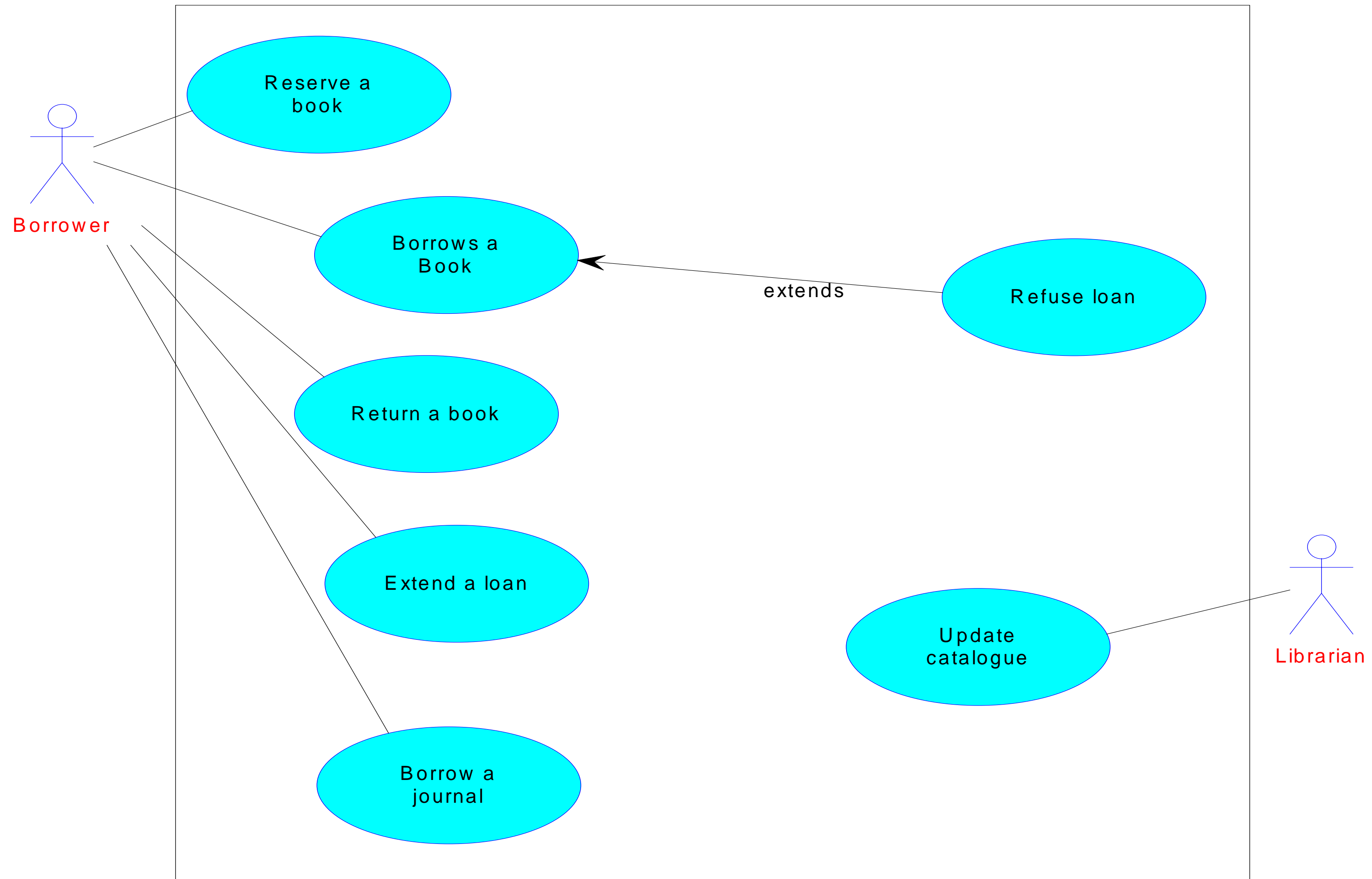
- Books and Periodicals

  The library contains books and journals. It may have several copies of a given book. Some of the books are for short term loan only. All other books may be borrowed by any library member for three weeks. Members of the library can normally borrow up to six items at a time, but members of staff may borrow up to 12 items at one time. Only members of staff my borrow journals.

- Borrowing
  - The system must keep track of when books and journals are borrowed and returned, enforcing the rules described above.

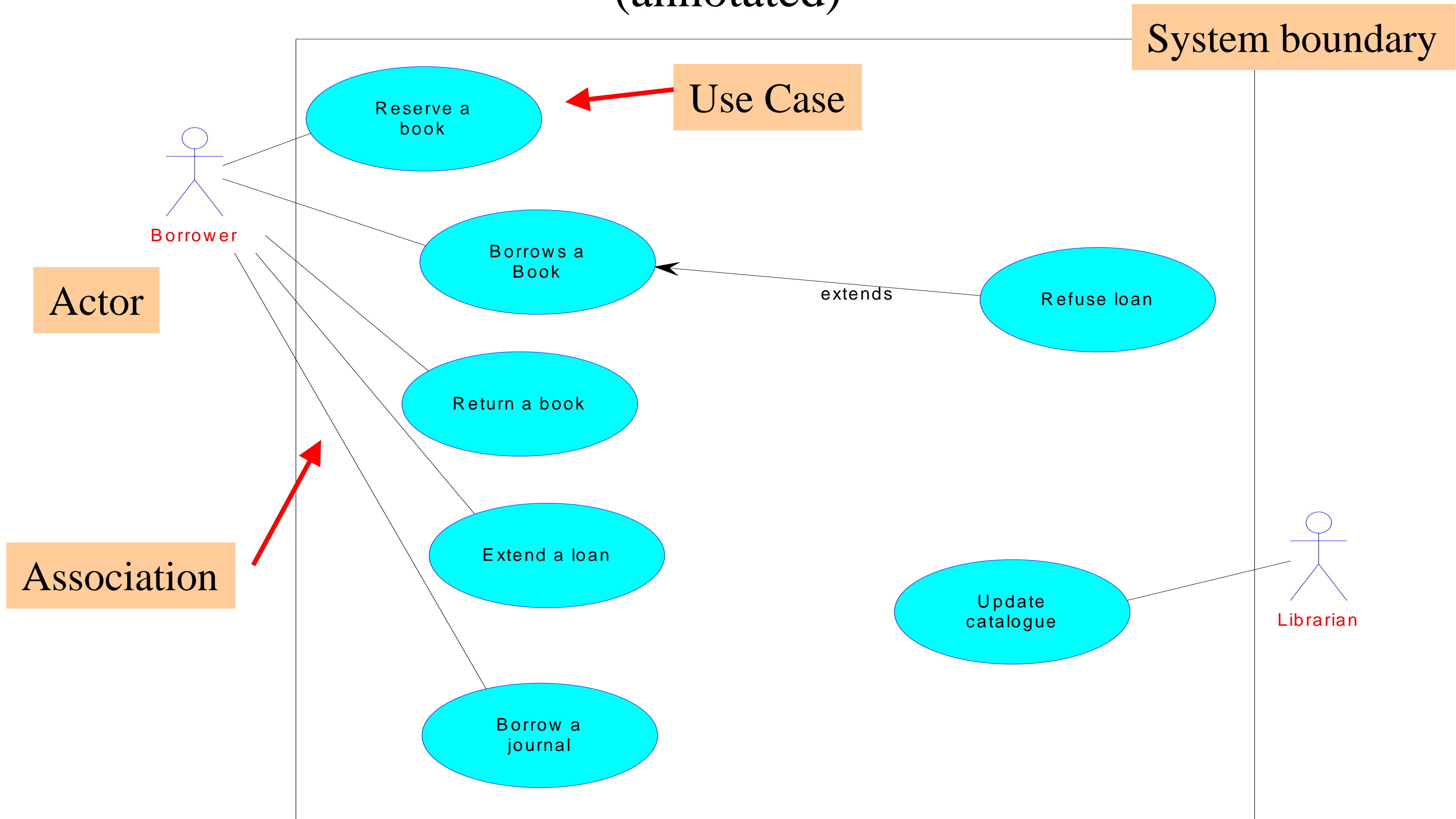
Use cases for Library

(partial list)



Use cases for Library

(annotated)

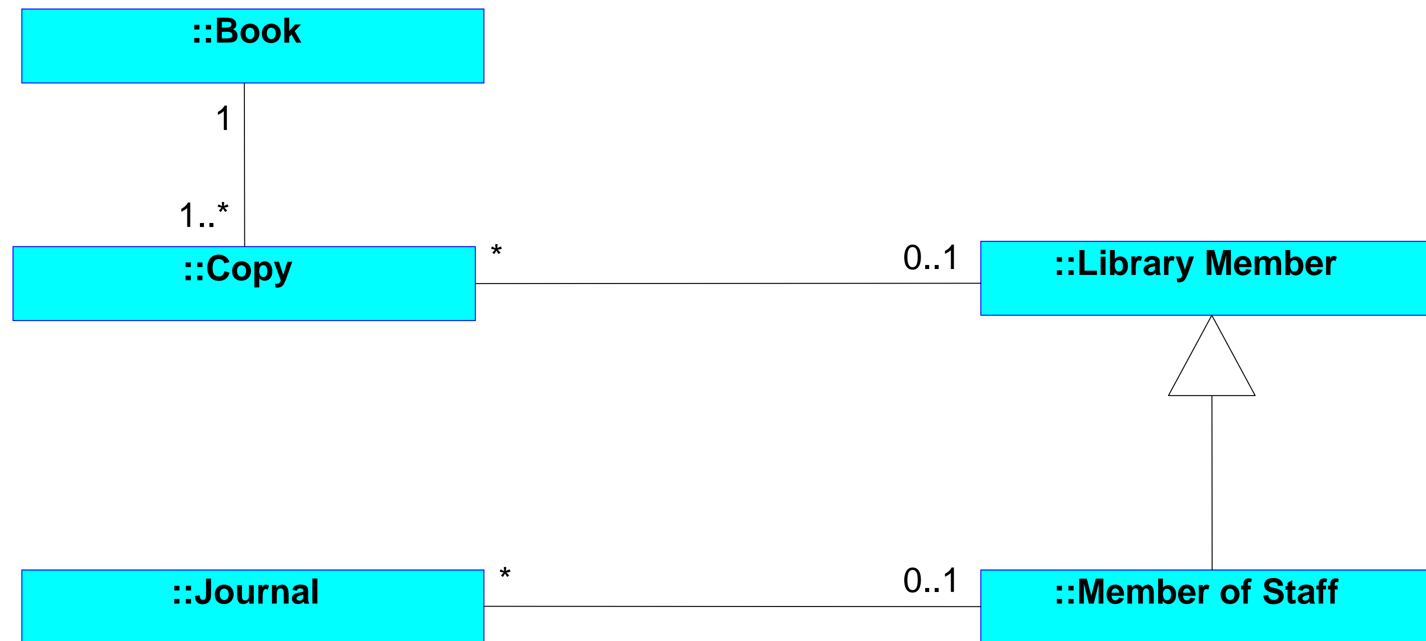


This UML diagram is useful as a CONTENTS page only!

Getting to candidate classes

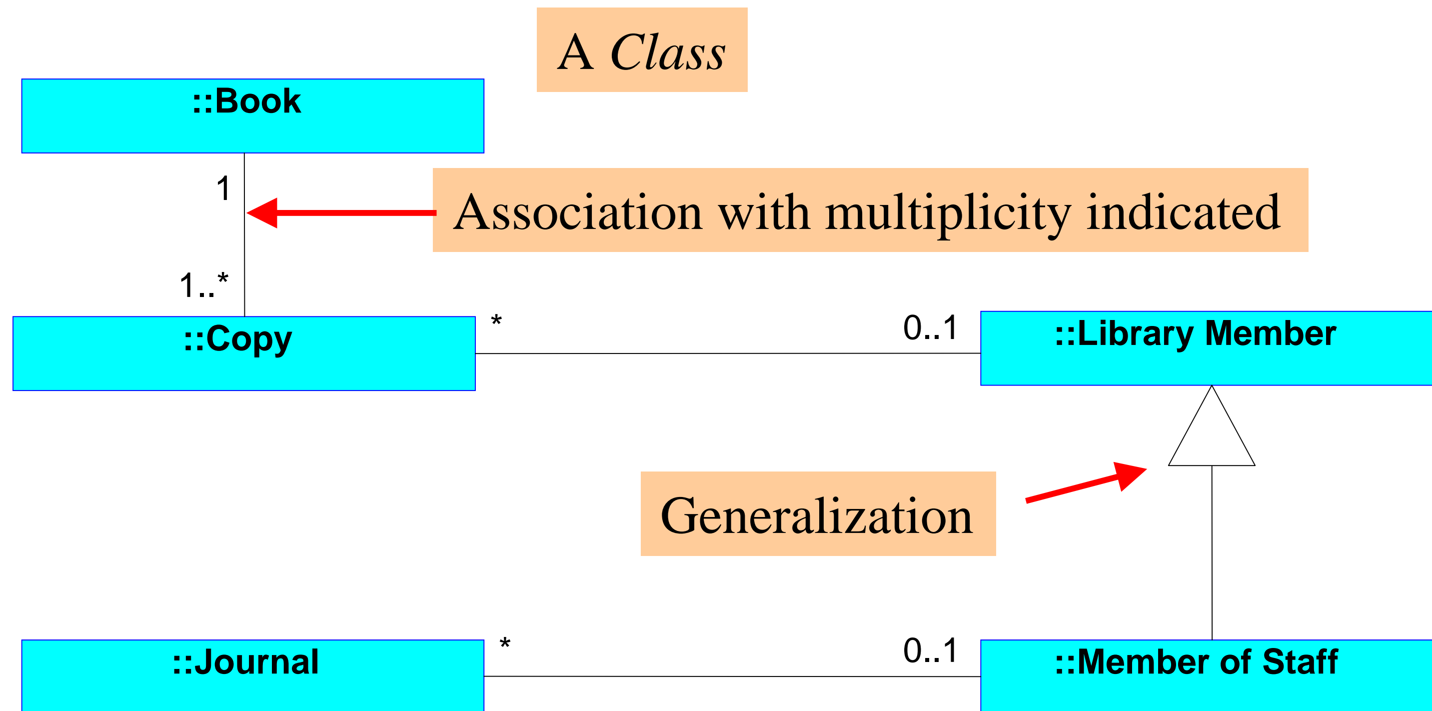
- Discard the following
  - Library — outside scope
  - Loans — events
  - Item — vague = book or journal
  - Week — measure of time
  - Time — outside scope
  - System and rule — meta-language of requirements
- Keep
  - Book, journal, copy (of book), library-member, member-of-staff

Library Class Model



Library Class Model

(annotated)

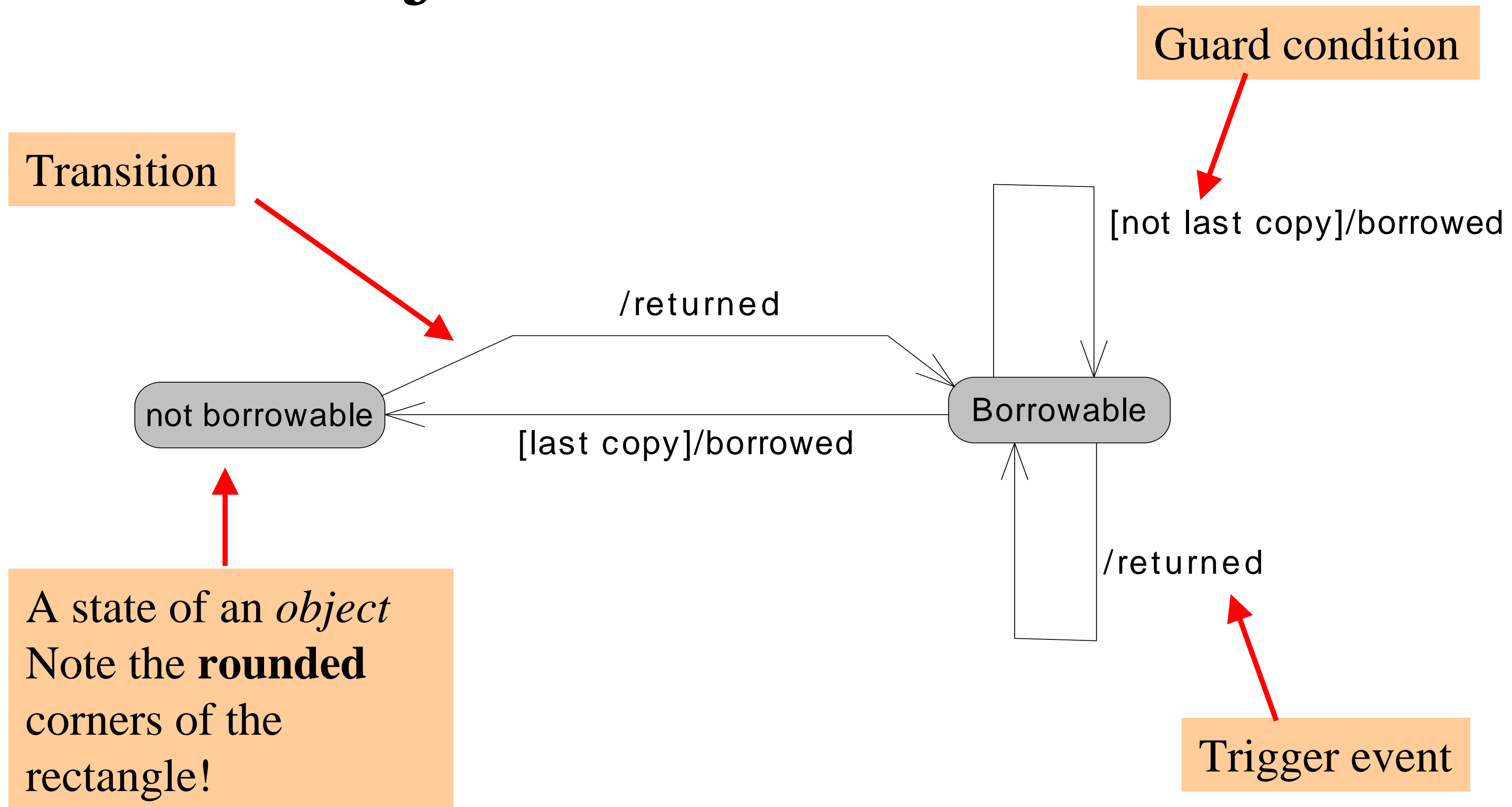


Multiplicity: * = any, 1 = 1 *and only* 1, 0..1 = 0 or 1 1..* = *at least* 1 etc.

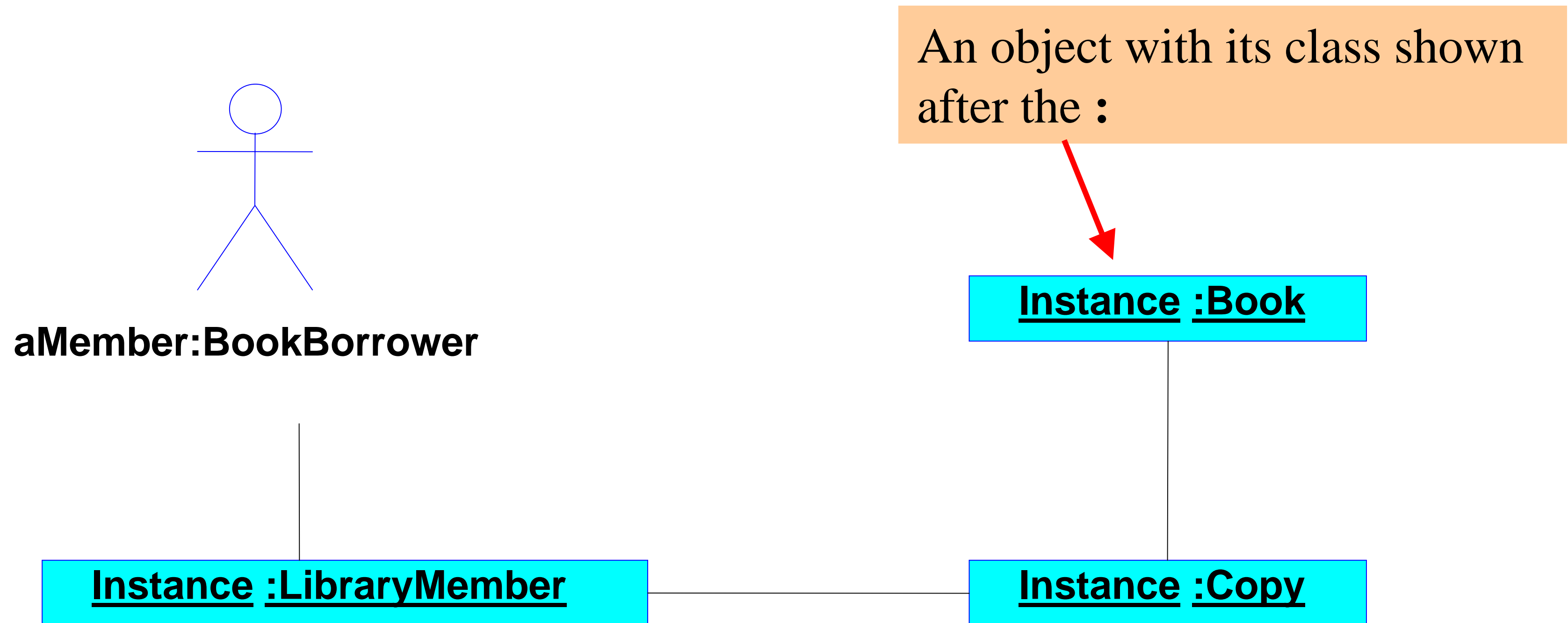
State Machine View

- Describes the *dynamic* behaviour of *objects*.
- Each object is considered to be isolated and communicates with the rest of the system by detecting *events*.
- An *event* is localized in space and time, it has no duration. Events may have parameters.

State transition diagram for an object of class Book

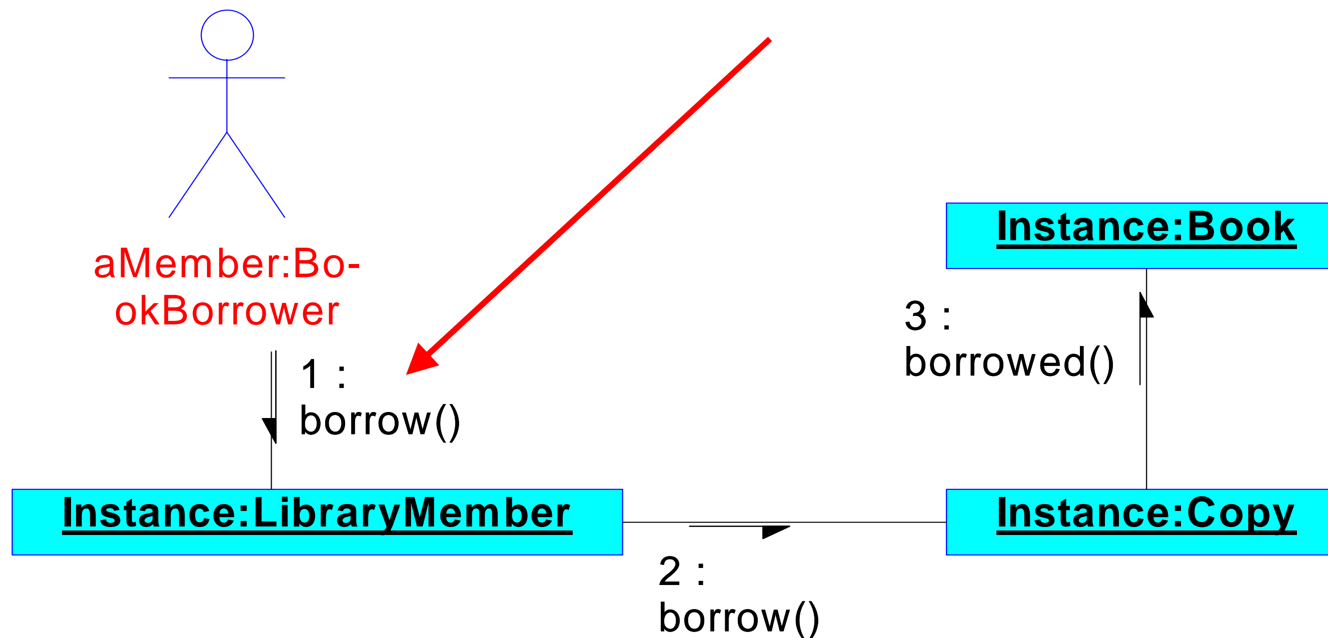


Simple Collaboration



Collaboration with interaction

Message sequence number with operation being described



The *elevator (or lift)* problem

- This is a classic problem in software engineering (D. Knuth, 1968)
- Used because it is simple to understand but far from trivial.
- A classic problem for a finite state machine
- I won't discuss *implementation details*

The problem

Build a system that enables *n* lifts between floors, subject to these constraints:

1. Each lift has *m* buttons, one for each floor. These illuminate when pressed, the light goes out when the correct floor is reached.
2. Each floor (except the basement and the top) has two buttons to summon an lift. One is $\Uparrow$ the other is $\Downarrow$. Each of these buttons illuminate when pressed and go out when an lift arrives.
3. When there are no pending requests each lift remains at its current floor with its doors closed.

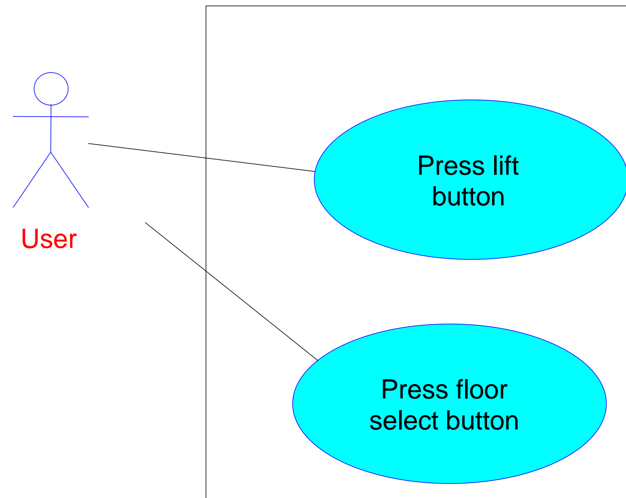
Use Cases

A use case describes the *externally visible* funtionality in a generic form.

A scenario is a specific instantiation of a use case (compare object and class)

For the lift problem the use case is very simple and there are a vast number of distinct scenarios. One should study sufficient scenarios in the OO analysis phase to gain a detailed insight in the behaviour of the system being modelled.

Use case for the lift



Class model

- In this example I will show three iterations during the development of the class model.
- A characteristic of OAD is that most of the steps are *difficult* to carry out. Don't be surprised that extracting classes and attributes is not easy to get right first time.
- One can get a good idea of candidate classes from scenarios, but note that this may generate too many – it is usually a good idea to minimise the initial number of classes and then add to them rather than removing a candidate class that should not have been included.

Class model – noun extraction

1. Specify problem concisely.
   - “Buttons in lifts and at each floor control the motion of *n* lifts between *m* floors”
2. Develop an informal strategy
   - Using 1) add the constraints and express if possible in a single paragraph
3. Formalize the strategy
   - Identify the nouns and use these as candidate classes

Classes – iteration 1)

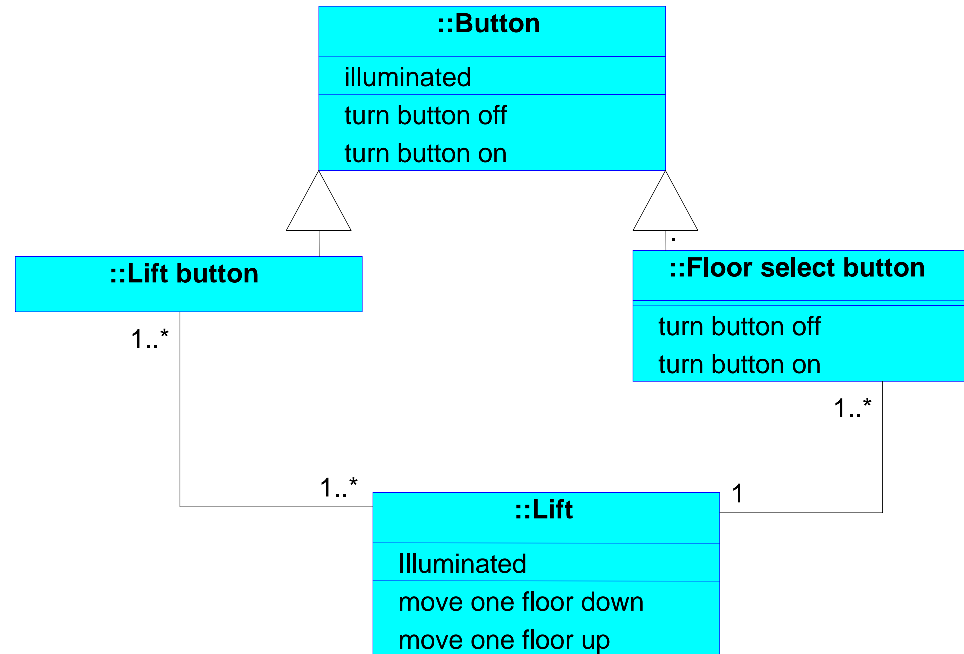
Buttons in lifts and on the floors control the movement of *n* lifts in a building with *m* floors. Buttons illuminate when pressed to request a lift to stop at a specific floor. The illumination is cancelled when the request has been satisfied. When an lift has no requests it remains at its current floor with its doors closed.

Nouns shown in red

Classes – iteration 1)

- Nouns
  - Button, lift, floor, movement, building, illumination, request, door
- Outside problem boundary are
  - Floor, building, door
- There are three *abstract nouns* (may be identified as attributes of classes)
  - Movement, illumination, request
- Our candidate classes are therefore
  - **Lift** and **Button**

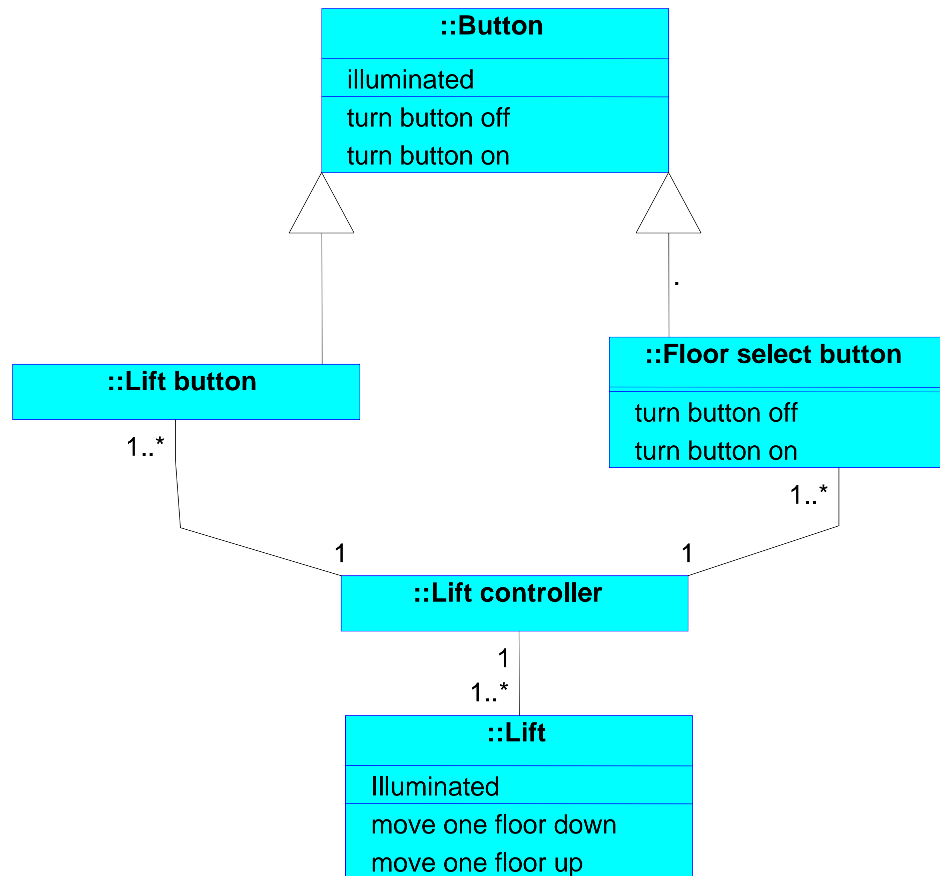
First iteration of class diagram



Second iteration

- Two types of button therefore two subclasses defined: **Lift Button** and **Floor select Button**
- But in a real lift buttons do not communicate directly, but via some type of controller. A controller was *not* mentioned in the specification hence no class was identified.
- The lesson here is that the noun identification strategy is a start but is not in itself complete. We now get the second iteration of the class diagram.

Second iteration of class diagram



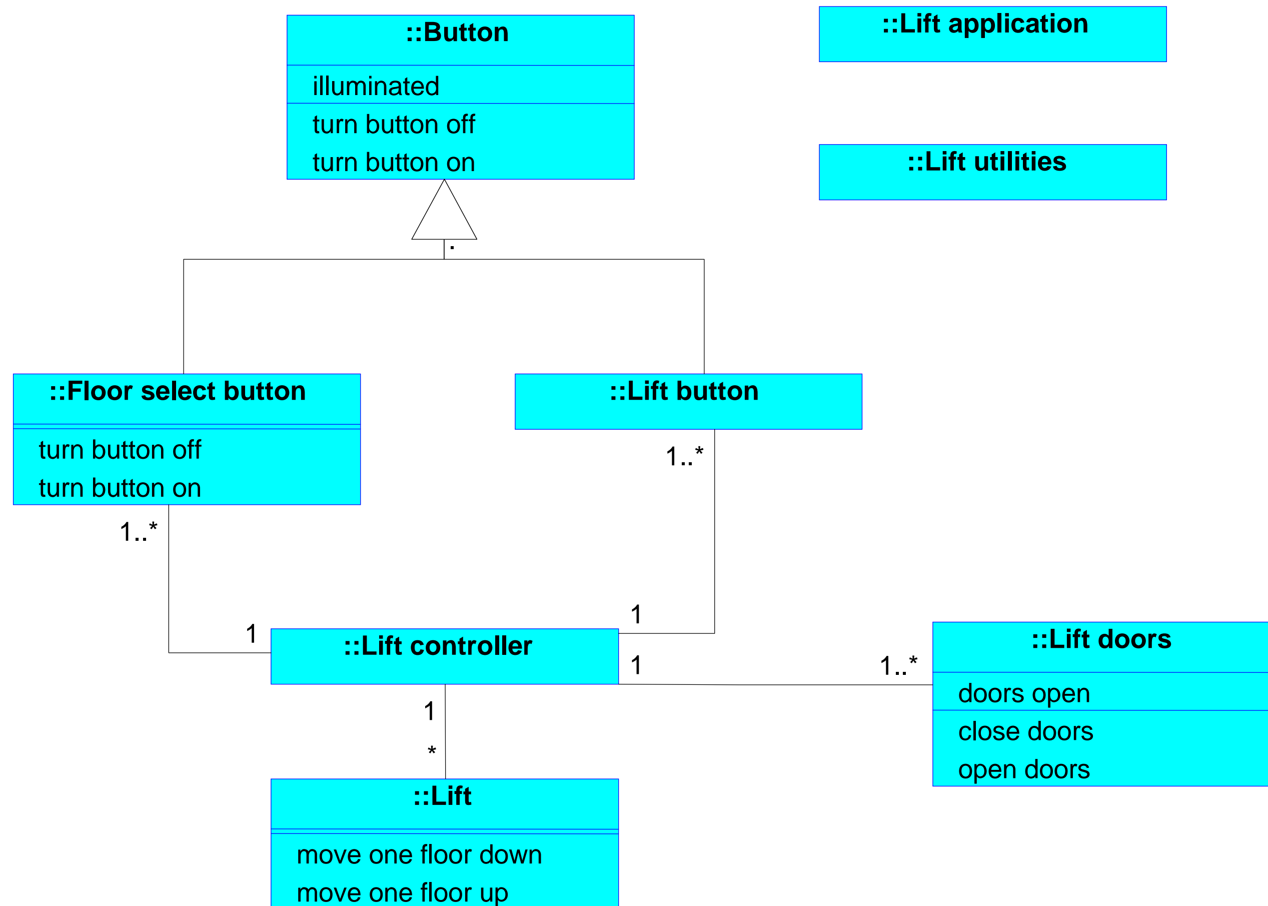
Third iteration

- What are the responsibilities of the class **Lift Controller**?
    1. Turn on lift button
    2. Turn off lift button
    3. Turn on floor button
    4. Turn off floor button
    5. Open doors
    6. Close doors
    7. Move up one floor
    8. Move down one floor

Third iteration

- Do you see the problems here? The responsibility for carrying out 1. & 2. lies with **Lift Button** and not with **Lift Controller.** The correct response is to send appropriate messages.
- Also a class has been overlooked. Consider 5. & 6. Since the doors of the lift possess a state which will be changed during execution of the implementation, then probably it should be modelled as a class: **Lift Doors**
- From these considerations we arrive at the third iteration of the class diagram. Two additional **Lift** classes have also been added, one to act as the overall class and one providing a set of untility functions.

Third iteration of the class diagram



How do I find out more?

- **P Stevens and R Pooley, "Using UML: software engineering with objects and components"**
  - http://www.dcs.ed.ac.uk/home/pxs/Book/
- The elevator problem is nicely discussed in
  - **S R Schach, "Classical and object-oriented software engineering"** 4th edition, McGraw-Hill, 1999
- For more on the elevator problem see
  - http://www.bit-net.org/java/elevator.pdf