UNIVERSITY OF LEICESTER

CLASS MODELLING

CO1106 Requirements Engineering and **Professional Practice**

Dr Matthias Heintz , Prof. Shigang Yue
(mmh21@leicester.ac.uk) (sy237@Leicester.ac.uk)

Please join TopHat activities:
https://app.tophat.com/login
Join: **147651**

Schedule

Week	Start Date	Monday / Wednesday - Lecture	Thursday / Friday - Surgery	Assessment
26	15/01/2024	Introduction & Why Requirements?	Icebreaker activity for groups & work on Project Description	
27	22/01/2024	Requirements gathering (Quan. & Qual. User Studies)	Work on requirements gathering for Assessment 1	
28	29/01/2024	Functional Requirements	Work on building list of funct. requirements for Assessment 1	
29	05/02/2024	Non-Functional Requirements	Work on building list of non-funct. requirements for Assessment 1	
30	12/02/2024	Overview of UML; Use Case diagrams and descriptions	Work on Use Case diagram and Use Case description for Assessment 1	
31	19/02/2024	Basics of git version control	Checkout and setup group git repository and set up Weekly Log .md file	Assessment 1 (50%)
32	26/02/2024	More advanced git topics	Work on reworked list of functional requirements	
33	04/03/2024	Class Diagrams	Work on Class diagram	
34	11/03/2024	Class Modelling	Rework Class diagram	
35	18/03/2024	Sketching and Lo-fi prototyping	Work on wireframes/lo-fi prototypes	
36	25/03/2024	Software Laws & Professionalism	**none**	Effective use of Git (10%)
37-40	01/04/2024	break	break	
41	29/04/2024	none	none	Blackboard Test (40%)

Matthias
Shigang

Session objectives

- At the end of the lecture you will be able to:
  - *Apply techniques for producing a class model based on system requirements/use case models*
  - *Recall some examples of how the techniques are used to produce class models*

Group coursework in CO1106

- Main group project
  - ***Part 1*** *(50% - due 23rd February)*
    - Project description (10%)
    - Quantitative and qualitative studies (10%)
    - Written requirements (20%)
    - Use Case UML Diagram and Use Case Description (10%)
  - *Part 2 (10% - due 27th March)*
    - **Effective usage of git version control**
      - ***Upload your initial class diagrams***
    - **Individual Blackboard test (40%)**

Group Coursework Part 2 – Effective use of git version control

- Your group will utilise a git repository in order to manage/submit any files produced as part of the second part of the group project for CO1106 (details of how to access the repository will be provided to you in the tutorial of Week 6).
- Groups should **make frequent usage** of their group repository - any shared files that you work on (for example, the .md files containing functional requirements and use case descriptions) should be added to the repository as soon as they are made, with regular changes being committed by group members until that particular file is finished. Each group will be responsible for coordinating their git usage.

Group Coursework Part 2 – Instructions

- A maximum of 3 marks are available, depending on **how effectively your group used git**. We will decide the number of marks you receive by inspecting the contents of your repository as well as the commit history of your repository:
  - For 1 mark, at least one member of your group needs to make a commit each week; no advanced features (i.e., branching/merging) have been used, and commit descriptions (included when you made the commit) may be non-descriptive and not give a good idea of the changes included in a particular commit.
  - For 2 marks, multiple commits should be made each week, and an initial (possibly incomplete) version of the artefact worked on during each tutorial session needs to be submitted in the week it was worked on. Commit messages must be descriptive (but succinct) and give a good idea of the changes that have been committed.
  - For 3 marks, you must satisfy all points from the previous two bullets, and it should be apparent from your commit log that all members of the group have

Group Coursework Part 2 – Weekly log

- Each group must also produce **a 'Weekly Log' (.md format)** that is **updated** with the following contents in **each** of the **week**s 6-10 (5 weeks in total):
  - A 'beginning of week' entry containing a summary of the work that the group plans to complete during that week, along with a breakdown of which members will complete which tasks. The beginning of week entry for each week should be produced by the group; for example, during your first groupwork meeting of that week.
  - An 'end of week' entry which lists the tasks that each member of the group has completed; any outstanding work; and any other additional information that your group feel is relevant to add .

Group Coursework Part 2 – Weekly log (continued)

- Each group will be responsible for **designing the Markdown structure** of their Weekly Log, ensuring that it is easy to read and maintained properly. The entries in the Weekly Log will be checked on a weekly basis (the entry for Week X will be checked by us during Week X+1). For the entries of Week X, there is a maximum of 0.75 marks available (up to a total of 3 marks for weekly updates):
  - 0.25 marks depending on whether both the 'beginning of week' and 'end of week' entries have actually been added to the log (if either one is missing, you receive 0 marks for that week)
  - 0.5 marks will be awarded depending on the quality of the entry (is it descriptive enough? does it contain all the information listed above?)
  - An additional mark out of 4 will be awarded based on the readability/quality of the Weekly Log document.

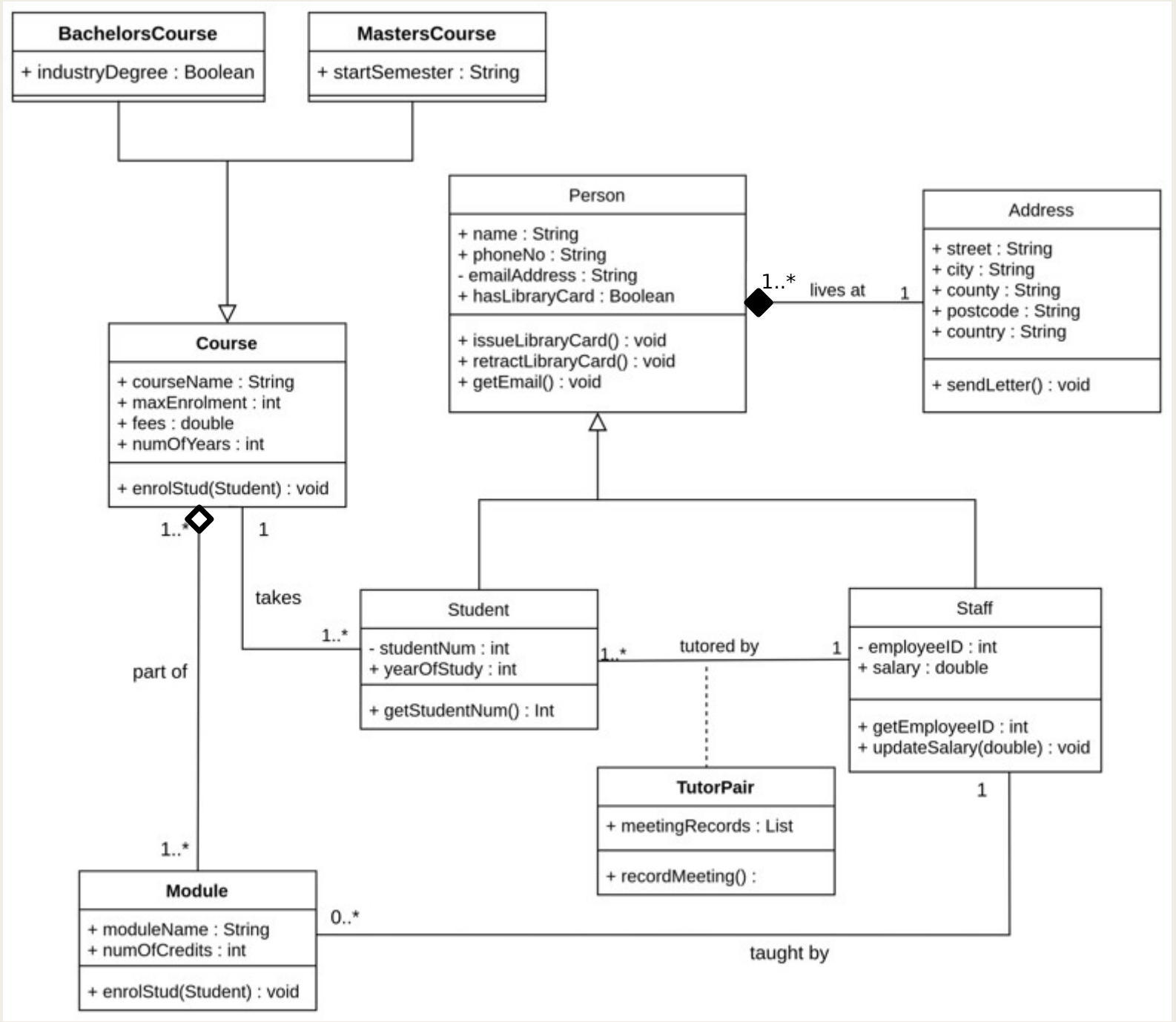
Marking rubrics

Markdown Usage, Marking Rubric					
Fail	Poor	Requires Improvements	Satisfactory	Good	Excellent
0	0.5	1	2	3	4
No serious attempt is made.	Required information is contained in the document but no Markdown syntax has been utilised in order to improve readability.	A limited amount of basic Markdown syntax has been used to produce a document that is readable but not aesthetically pleasing or easy to navigate. There may be some syntactical errors in the Markdown usage that cause the document to look untidy.	Basic Markdown syntax has been used to reasonable effect. The produced document is navigable and its contents is clearly displayed. Most audiences should have little to no problem with understanding the document, but it is not particularly aesthetically pleasing.	Same as Satisfactory, but with a number of more advanced Markdown features used in order to improve the readability of the document. Any member of the intended audience would be able to navigate the document easily.	Same as Good, but a clear effort to make the document aesthetically pleasing as well as easily navigable has been made.

Re-visit

Class Diagrams: University management system

- Composition association
- Aggregation association
- Inherit
- Multiplicities



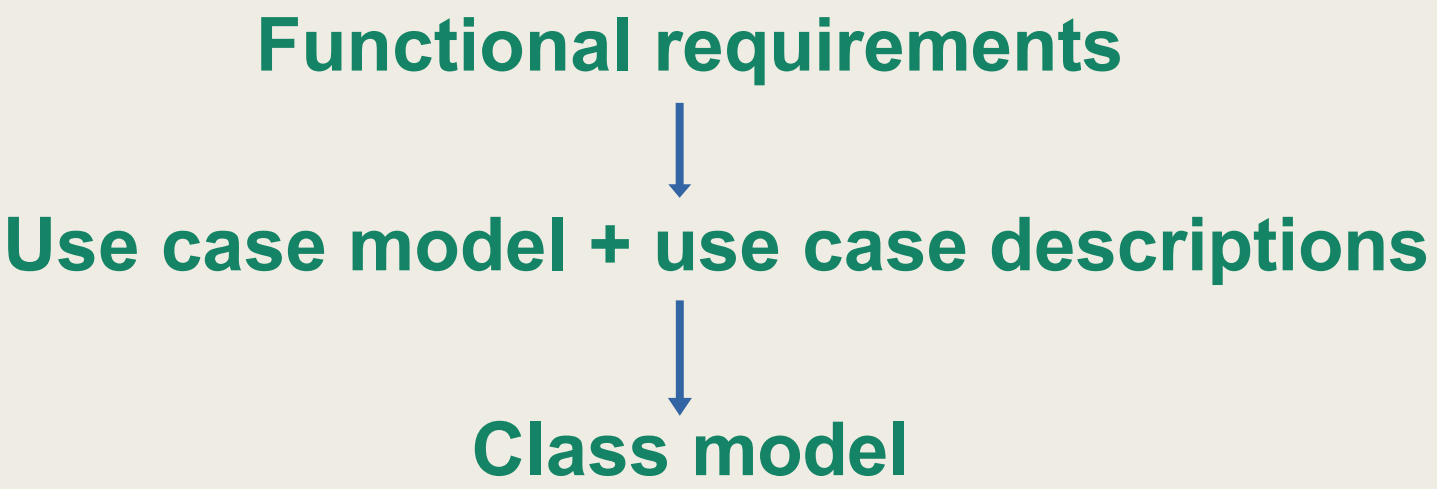
CLASS MODELLING PROCESS

The class modelling process - from requirments to class diagrams

- Last week we learnt about the notational aspects of Class Diagrams and how / when they should be used
- Knowing how to generate a formal class diagram from a set of requirements is a **different process all together**
- It involves:
  - *Identifying **necessary classes and their attributes***
  - *Establishing **which classes need to communicate and how***
  - *Ensuring that the class model **satisfies your requirements***

Design ‘flow’

- The general flow when attempting to generate a class model for a system:

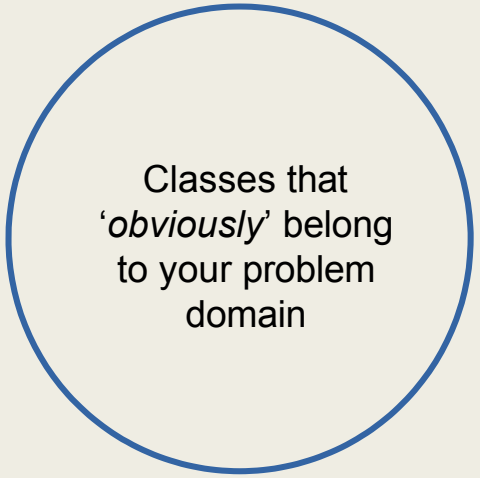
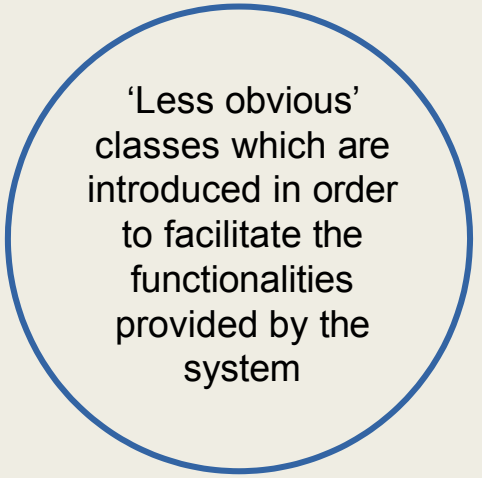


- **The aim:** Design a class model/diagram that ***enables all use cases in your diagram to be carried out***

IDENTIFYING CLASSES

Identifying classes

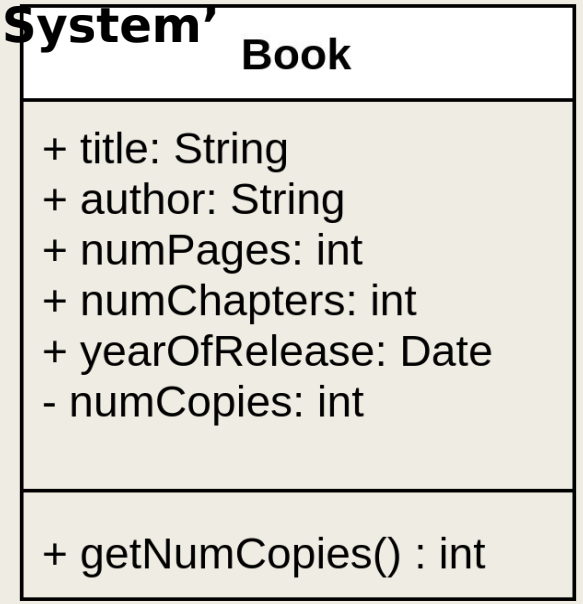
- Identifying your classes is arguably the most important part of the process and '***lays the foundations***' for your class model
- Classes can be roughly grouped into two categories:

Domain classes	***Solution classes***
	
Usually straightforward to identify from	Harder to identify; often discovered during the modelling process

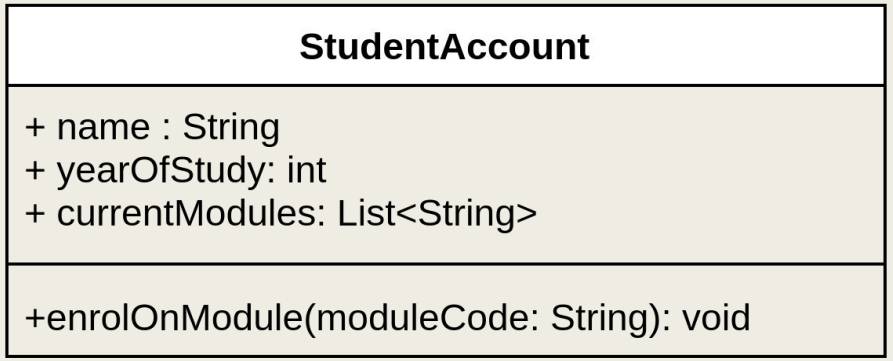
Domain classes

- Domain classes are usually quite easy to identify; they often correspond to physical items/entities that your system will record information about:
- **Examples:**

A book class in a 'Library Management System'



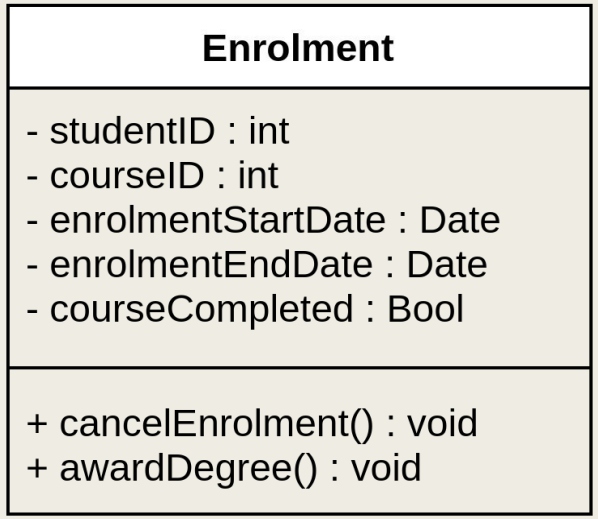
A student account in an online learning platform



Project Description/ Functional Req.'s/UCD
are all a good source of information
when trying to establish your domain classes

Solution classes

- Solution classes more often come about as a means to achieving a piece of functionality your system needs to provide
- **Example:** An ‘Enrolment’ class in a University management system



An enrolment of a student on a course is not a physical entity, but it represents
something that the system facilitates and stores information about

- The need for solution classes usually reveals itself as you add more domain classes to your model, as you begin to understand the use cases / requirements more deeply

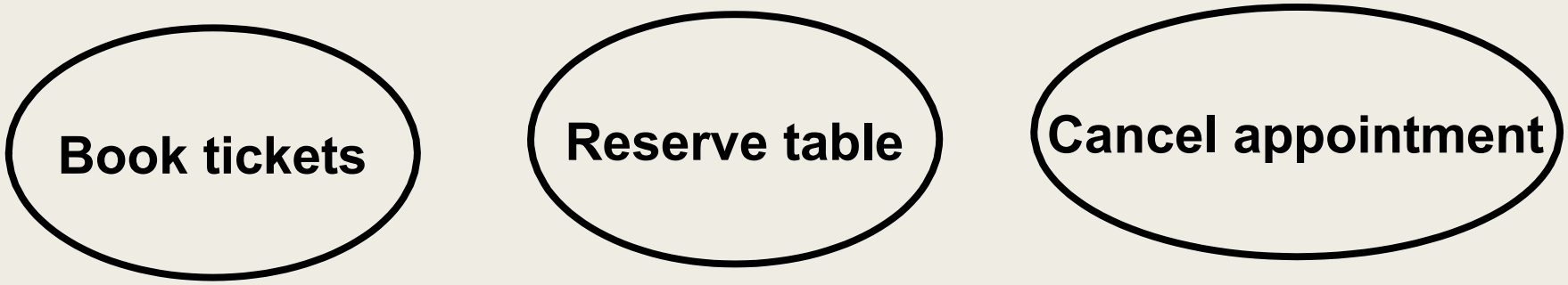
Notes & tips

- In practice, **we don’t distinguish between Domain and Solution classes** within a class diagram, but you should be aware of the differences
- **Look for your Domain classes first** by analyzing the functional requirements / Use Cases of your system
- A need for Solution classes will then become apparent as you **find out more about the information / operations** that your system needs to store / perform

USING USE CASES TO IDENTIFY CLASSES

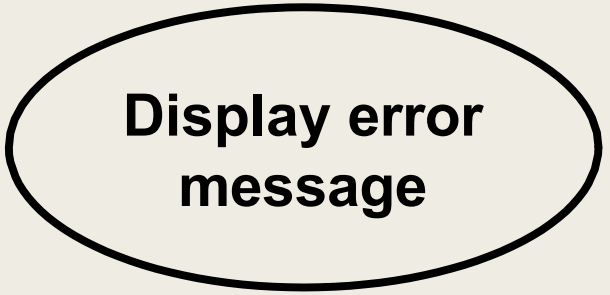
Reminders about use case diagrams

- Each use case should represent a **singular use of the system;** it should describe a task from a **stakeholders perspective**, i.e.:



Reminders about use case diagrams

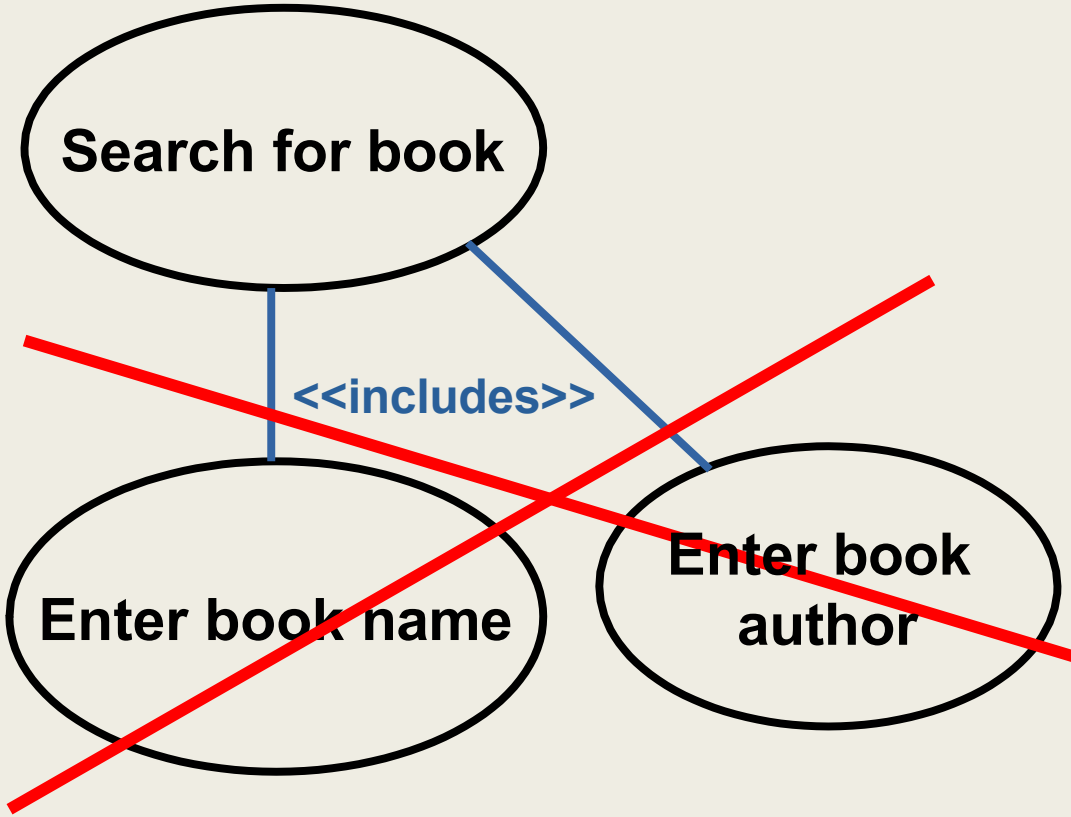
- Use Cases should not be written from the perspective of the system, for example



- is something the system does, rather than a task that a stakeholder can complete using the system

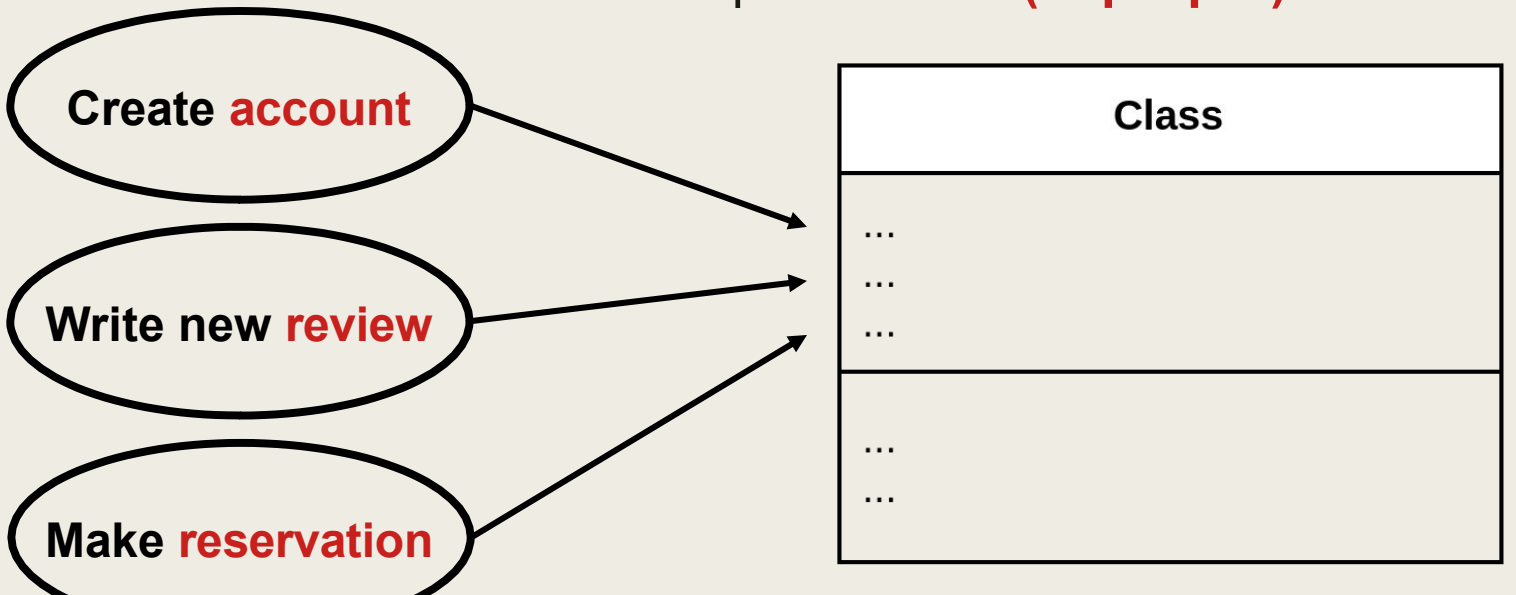
Reminders about use case diagrams

- Each use case should be **stand-alone**, you should not attempt to break down one usage of the system into multiple tasks
- E.g.: '***Search for book***' **should not be** broken down into '***Enter book name***', '***Enter book author***' etc., since these are not stand-alone uses of the system
- These use cases **should be** part of the ***'Search for book'* Use Case Description,** not indiviual use cases in the diagram



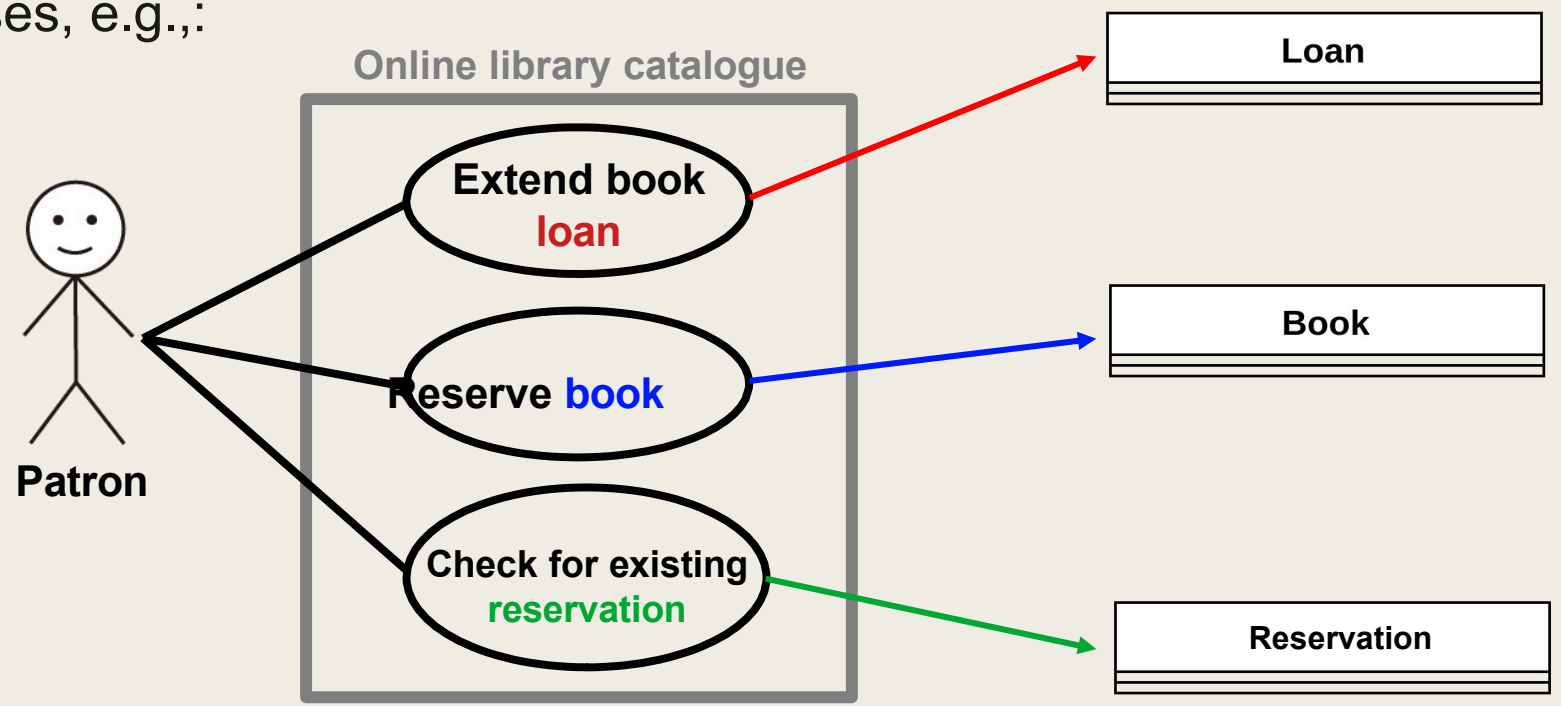
Using a Use Cases to identify classes

- Since a class model should enable all of the use cases of a system to be carried out, use case analysis is a good place to start
- ***Remember*** – use cases reflect **what** a system should provide for its stakeholder and not **how**…
- **Idea**: Focus in on each use case and pick out the **(improper) nouns**:



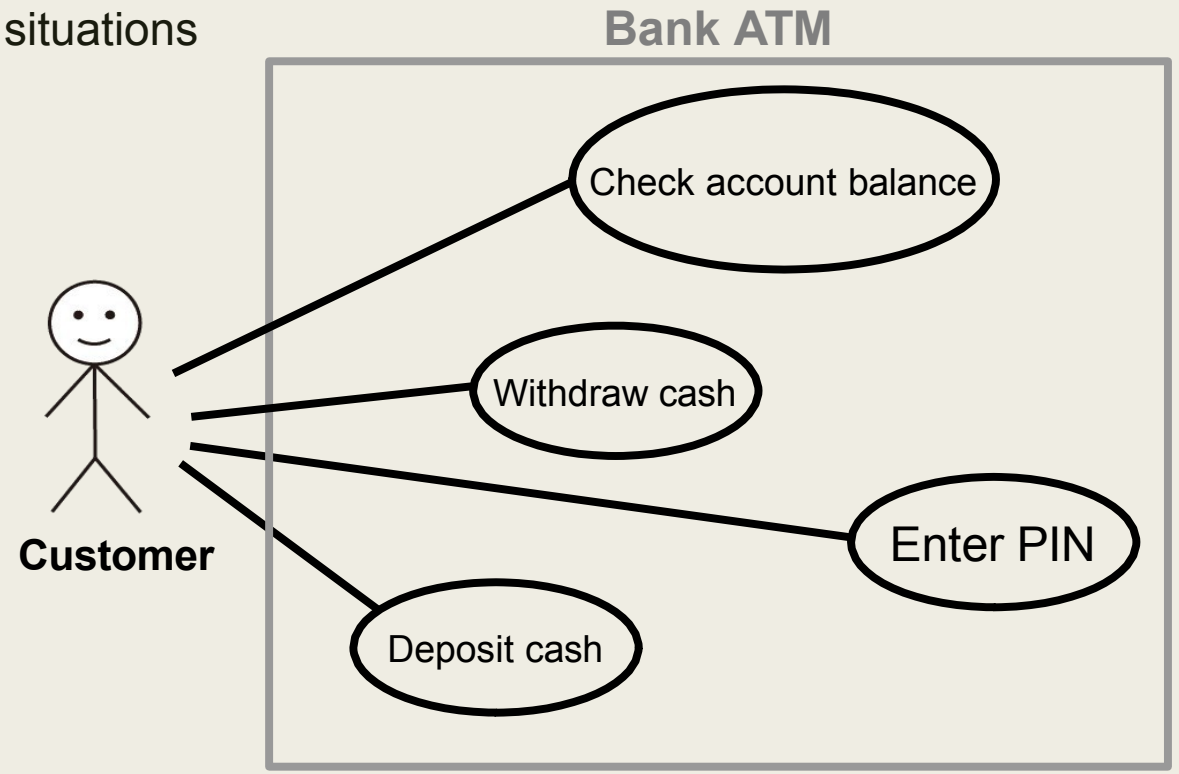
Abbot’s heuristic (Textual analysis)

- Identifying nouns in order to obtain classes is a popular class-identification technique introduced by Abbot in the 1980s
- The technique can be applied to a Use Case diagram to establish a preliminary set of classes, e.g.,:



Abbot's heuristic doesn't always work...

- Be warned though – there are situations in which Abbot's heuristic won't necessarily work with a Use Case Diagram...
  Here's one:

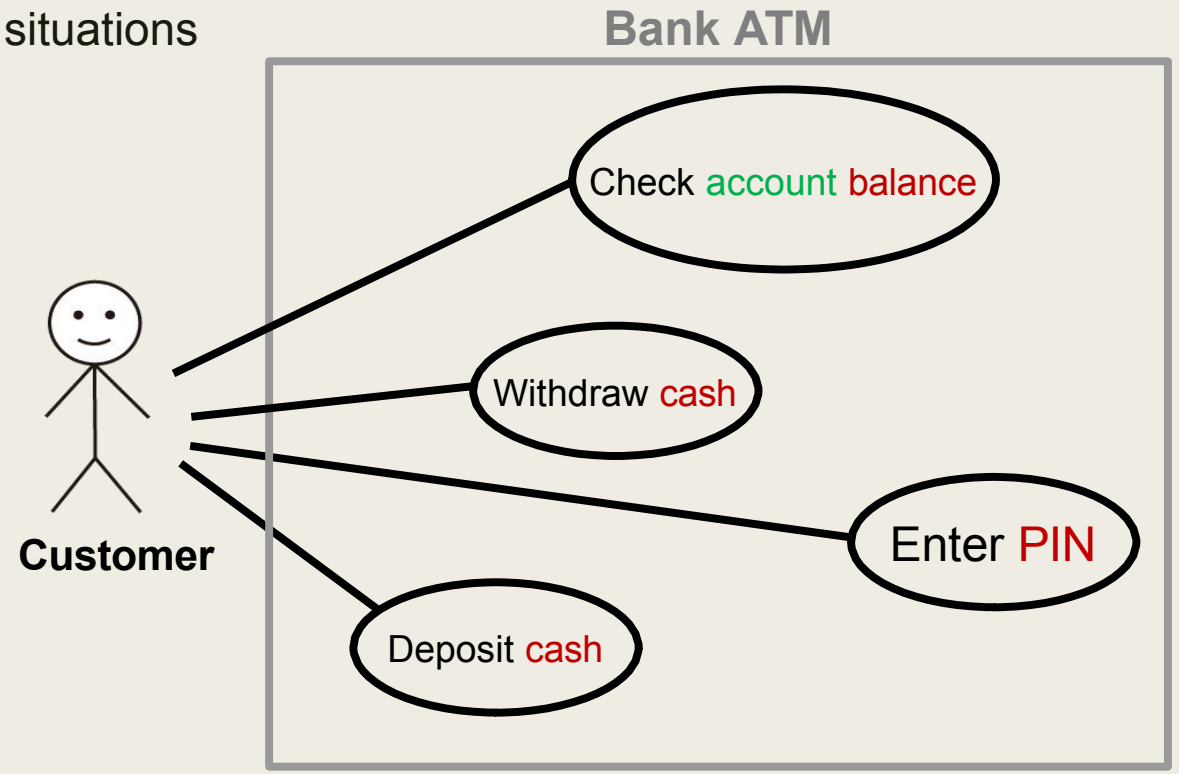


- **Question**: Can you see the problem with the '***noun → class***' approach here?

Join: https://app.tophat.com/login 147651

Abbot's heuristic doesn't always work...

- Be warned though – there are situations in which Abbot's heuristic won't necessarily work with a Use Case Diagram... ***Here's one***:



- **Question**: Can you see the problem with the '***noun → class***' approach here?

USING USE CASES DESCRIPTIONS TO IDENTIFY CLASSES

Identifying model aspects from UC descriptions

- Checking the UC diagram for nouns can only get us so far, and as we have seen it has its drawbacks; so we consider **use case descriptions**
- We analyse the behavior during the course of a use case, and apply Abbot's techniques even further, using the following '***translation rules***':

Word/phrase type	*Model component*	*Example*
Improper noun	Class	'Book'
Doing verb	Operation/method	'creates', 'submits', 'selects'
Being verb	Inheritance	'is a kind/type of'
Having verb	Aggregation / composition	'has', 'consists of'/ 'is part of'

- **Note:** Attributes should be straightforward to find, by examining the text surrounding a potential class for any mentioned details / characteristics...

Example / Exercise: Coach booking system

Use case: Purchase tickets

Typical flow of events:

- The user logs into their account and navigates to the search box, where they specify one desired destination, one start date and one end date.
- The user presses the search button to perform a search and the system returns a results list consisting of 0 or more available coach trips travelling to the specified destination taking place within the time frame of the specified dates
- The user selects a trip and view it's details (trip ID, destination, start time, duration).
- The user clicks a ‘Purchase tickets’ button, selects one of the payment cards associated with their account and pays for the tickets.

Key: **Class** **Association/Multiplicities**

Attributes

Example / Exercise: Coach booking system

Use case: Purchase tickets

Typical flow of events:

- The user logs into their account and navigates to the search box, where they specify one desired destination, one start date and one end date.
- The user presses the search button to perform a search and the system returns a results list consisting of 0 or more available coach trips travelling to the specified destination taking place within the time frame of the specified dates
- The user selects a trip and view it's details (trip ID, destination, start time, duration).
- The user clicks a ‘Purchase tickets’ button, selects one of the payment cards associated with their account and pays for the tickets.

Key: **Class** Association/Multiplicities Attributes

Example / Exercise: Coach booking system

Use case: Purchase tickets

Typical flow of events:

- The user logs into their account and navigates to the search box, where they specify one desired destination, one start date and one end date.
- The user presses the search button to perform a search and the system returns a results list consisting of 0 or more available coach trips travelling to the specified destination taking place within the time frame of the specified dates
- The user selects a trip and view it's details (trip ID, destination, start time, duration).
- The user clicks a ‘Purchase tickets’ button, selects one of the payment cards associated with their account and pays for the tickets.

Key: **Class** Association/Multiplicities Attributes

Example / Exercise: Coach booking system

Use case: Purchase tickets

Typical flow of events:

- The user logs into their account and navigates to the search box, where they specify one desired destination, one start date and one end date.
- The user presses the search button to perform a search and the system returns a results list consisting of 0 or more available coach trips travelling to the specified destination taking place within the time frame of the specified dates
- The user selects a trip and view it's details (trip ID, destination, start time, duration).
- The user clicks a ‘Purchase tickets’ button, selects one of the payment cards associated with their account and pays for the tickets.

Key: Class Association/Multiplicities

Attributes

Example / Exercise: Coach booking system

Use case: Purchase tickets

Typical flow of events:

- The user logs into their account and navigates to the search box, where they specify one desired destination, one start date and one end date.
- The user presses the search button to perform a search and the system returns a results list consisting of 0 or more available coach trips travelling to the specified destination taking place within the time frame of the specified dates
- The user selects a trip and view it's details (trip ID, destination, start time, duration).
- The user clicks a ‘Purchase tickets’ button, selects one of the payment cards associated with their account and pays for the tickets.

Key: Class Association/Multiplicities

Attributes

Example / Exercise: Coach booking system

Use case: Purchase tickets

Typical flow of events:

- The user logs into their account and navigates to the search box, where they specify one desired destination, one start date and one end date.
- The user presses the search button to perform a search and the system returns a results list consisting of 0 or more available coach trips travelling to the specified destination taking place within the time frame of the specified dates
- The user selects a trip and view it's details (trip ID, destination, start time, duration).
- The user clicks a ‘Purchase tickets’ button, selects one of the payment cards associated with their account and pays for the tickets.

Key: Class **Association/Multiplicities**

Attributes

Example / Exercise: Coach booking system

Use case: Purchase tickets

Typical flow of events:

- The user logs into their account and navigates to the search box, where they specify one desired destination, one start date and one end date.
- The user presses the search button to perform a search and the system returns a results list consisting of 0 or more available coach trips travelling to the specified destination taking place within the time frame of the specified dates
- The user selects a trip and view it's details (trip ID, destination, start time, duration).
- The user clicks a ‘Purchase tickets’ button, selects one of the payment cards associated with their account and pays for the tickets.

Key: Class **Association/Multiplicities**

Attributes

Example / Exercise: Coach booking system

Use case: Purchase tickets

Typical flow of events:

- The user logs into their account and navigates to the search box, where they specify one desired destination, one start date and one end date.
- The user presses the search button to perform a search and the system returns a results list consisting of 0 or more available coach trips travelling to the specified destination taking place within the time frame of the specified dates
- The user selects a trip and view it's details (trip ID, destination, start time, duration).
- The user clicks a ‘Purchase tickets’ button, selects one of the payment cards associated with their account and pays for the tickets.

Key:

Class

Association/Multiplicities

Attributes

Example / Exercise: Coach booking system

Use case: Purchase tickets

Typical flow of events:

- The user logs into their account and navigates to the search box, where they specify one desired destination, one start date and one end date.
- The user presses the search button to perform a search and the system returns a results list consisting of 0 or more available coach trips travelling to the specified destination taking place within the time frame of the specified dates
- The user selects a trip and view it's details (trip ID, destination, start time, duration).
- The user clicks a 'Purchase tickets' button, selects one of the payment cards associated with their account and pays for the tickets.

Key:

Class

Association/Multiplicities

Attributes

Example / Exercise: Coach booking system

Use case: Purchase tickets

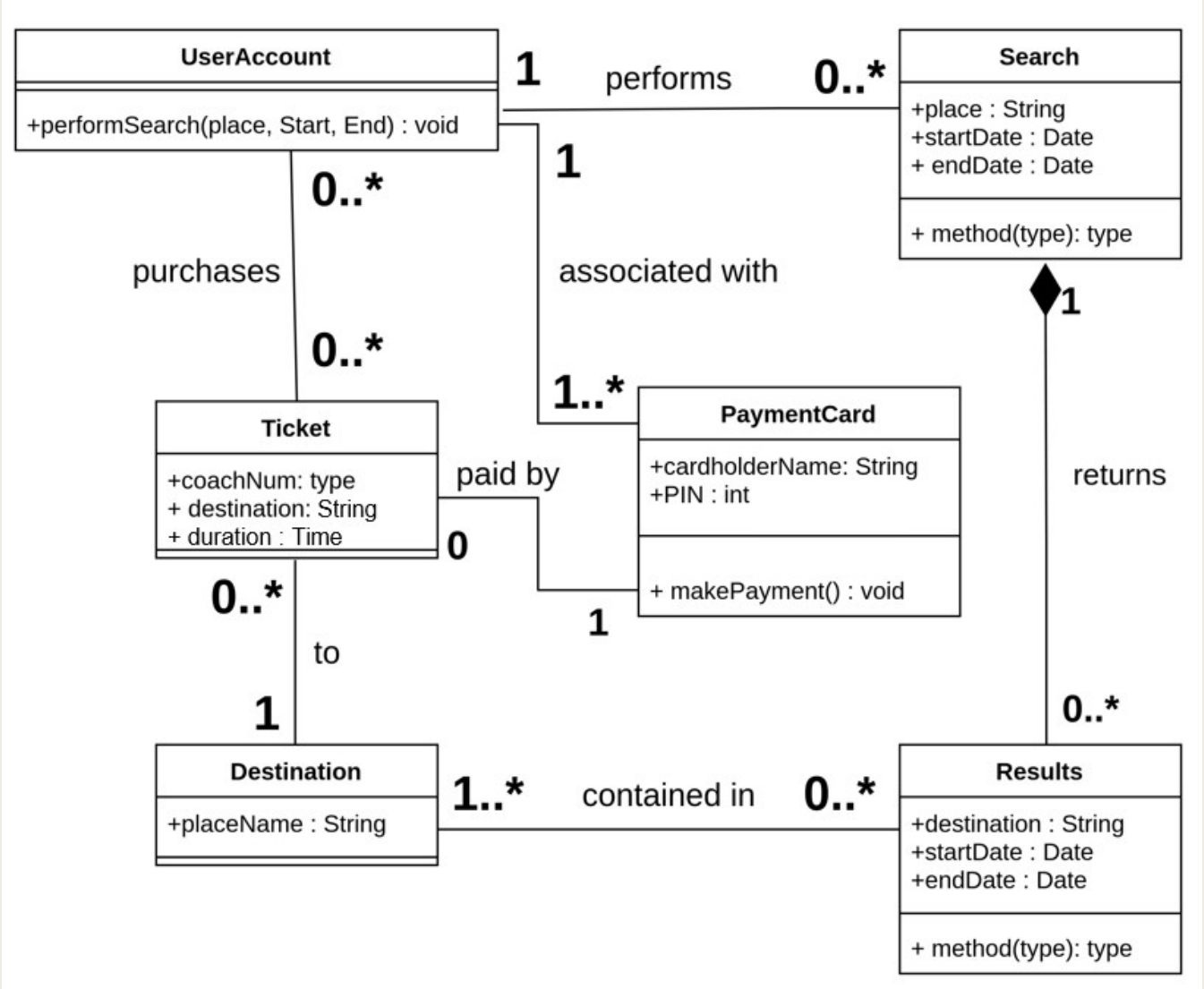
Typical flow of events:

- The user logs into their account and navigates to the search box, where they specify one desired destination, one start date and one end date.
- The user presses the search button to perform a search and the system returns a results list consisting of 0 or more available coach trips travelling to the specified destination taking place within the time frame of the specified dates
- The user selects a trip and view it's details (trip ID, destination, start time, duration).
- The user clicks a ‘Purchase tickets’ button, selects one of the payment cards associated with their account and pays for the tickets.

Key: **Class** **Association/Multiplicities**

Attributes

One possible solution



CLASS MODEL DESIGN APPROACH

One approach to creating a complete class diagram is the following:

- Consider each of the use cases in your Use Case Diagram as a group and write down a rough description of the typical flow of events
- Apply Abbots heuristic to the descriptions you obtain and try to establish a set of candidate classes and related attributes
- ‘Shortlist’ the sets of candidates; think about which are potential candidates / non-candidates for your model.
- Amongst shortlisted candidates, try to establish any associations that are suggested by your use case descriptions and add them to your model

Designing class models

- There is **no systematic way** of producing a class model based on functional requirements/use case diagrams
- You need to decide how to model classes / relationships / attributes / functions as clearly and unambiguously as possible
- Your **decisions need to be justified**
- Remember that **two people could produce 2 completely different models** based on the same description – as a group you need to come to an agreement on your model

What’s next?

- In this week’s tutorial you will:
  - *Refine your UML Class Diagram for your project*
  - *Please don’t forget to update your WeeklyLog!*

- Lecture next week:
  Sketching and Low-fidelity prototyping

Task for the Surgery this week

- Refine your UML Class Diagram using the techniques and approaches discussed in today's lecture
- **The diagram should contain:**
  - *A set of appropriately named classes with sensible attributes and methods*
  - *Appropriate associations between classes*
- **It should also:**
  - *Satisfy each of the functional requirements of your system*
  - *Enable each of the use cases in your Use Case diagram to be carried out by your system's end users*

Questions?

- Dr. Matthias Heintz or Prof. Shigang Yue
  - *mmh21@leicester.ac.uk or sy237@leicester.ac.uk*
  - *Microsoft Teams*
  - *Office 613 or 608 in Ken Edwards Building*

UNIVERSITY OF LEICESTER