G64OOS (Spring 2014)

Lecture 03 full

Object Oriented Design

Peer-Olaf Siebers

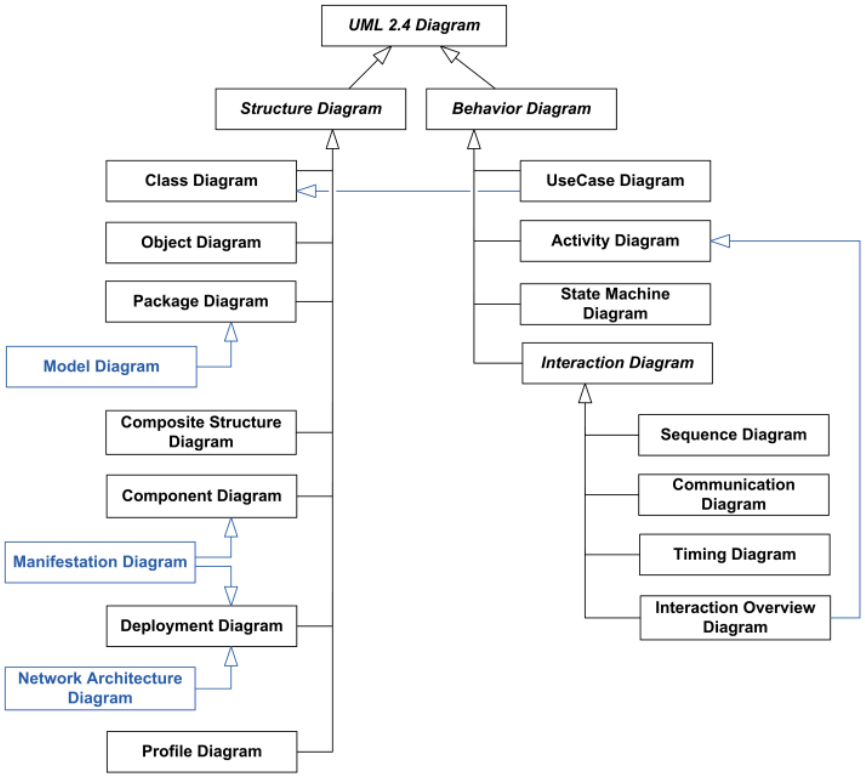
The University of Nottingham
UNITED KINGDOM · CHINA · MALAYSIA

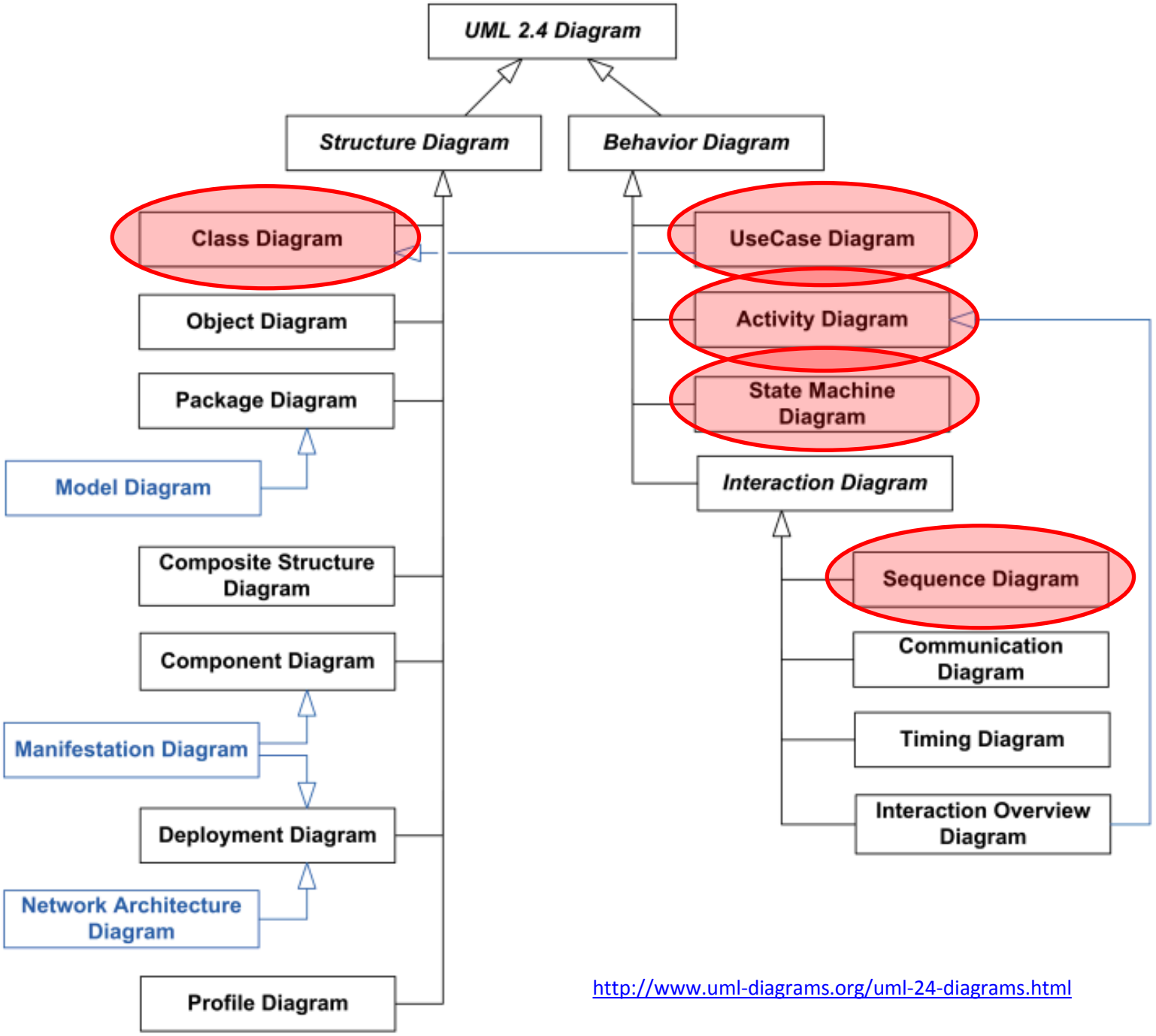
Motivation for Lecture 03

- Today you will learn:
  - Everything you always wanted to know about UML
  - How to describe the requirements of a system using Use Case Diagrams
  - How to model the static structure of a system using Class Diagrams
  - How to model the way objects interact using Sequence Diagrams
  - How to model state co-ordination using State Machine Diagrams
  - How to model activity co-ordination using Activity Diagrams

UML

- UML (Unified Modelling Language) is a family of graphical notations that help in describing, designing and organising object oriented software systems
- Latest version: 2.4.1



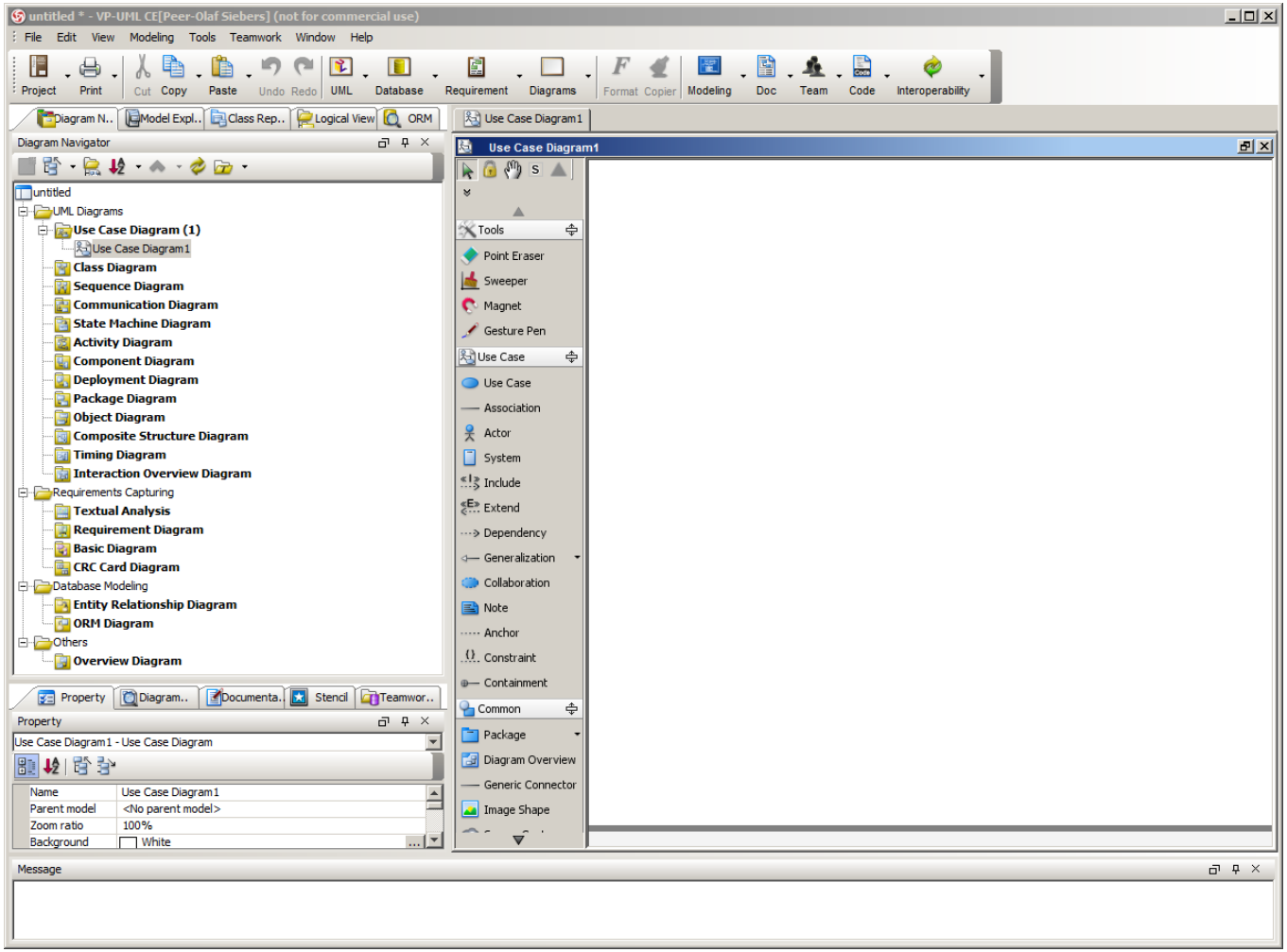


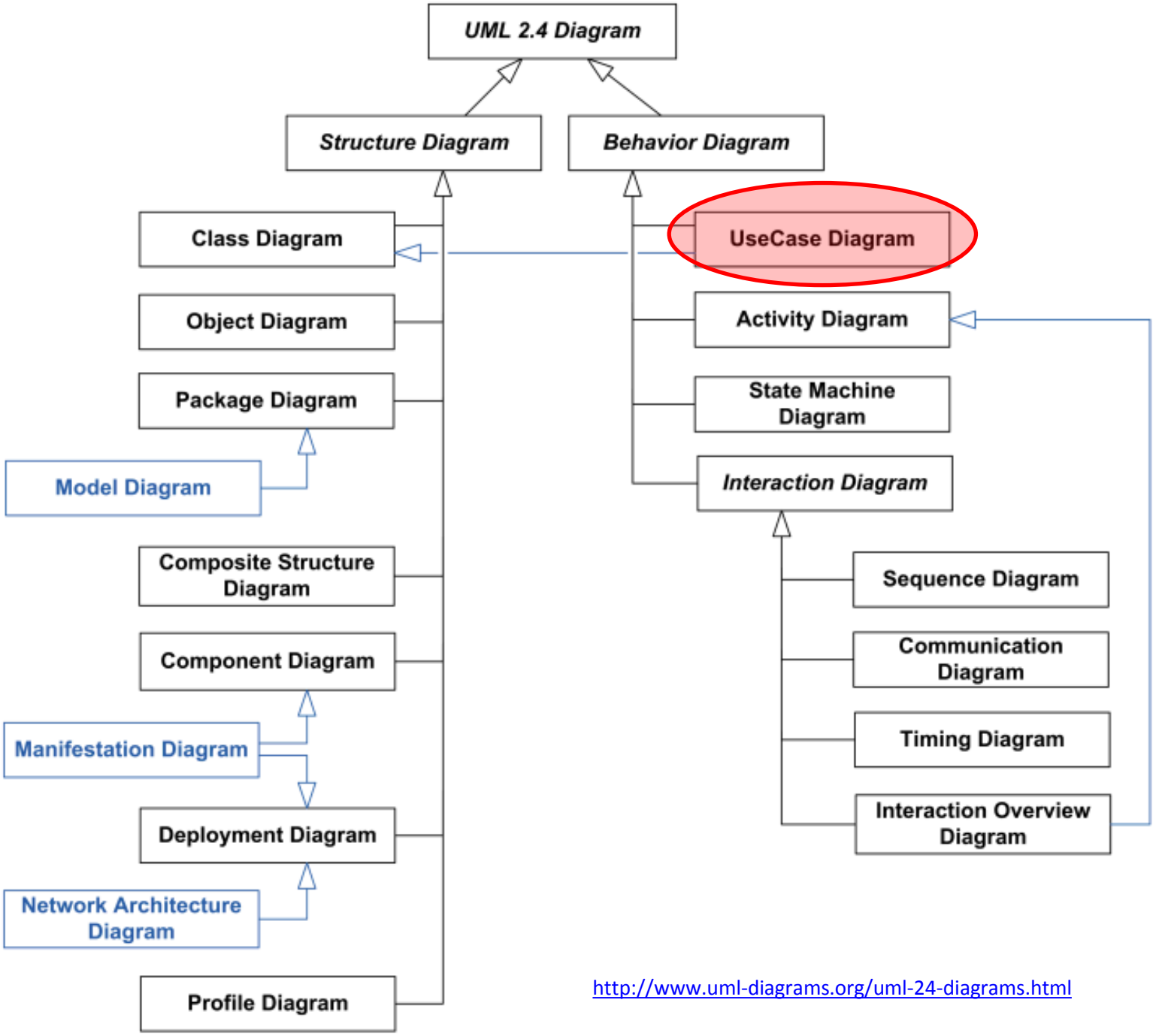
http://www.uml-diagrams.org/uml-24-diagrams.html

UML

- Advantages of using UML:
  - Enhances communication and ensures the right communication
  - Captures the logical software architecture independent of the implementation language
  - Helps to manage the complexity
  - Enables reuse of design

UML Software



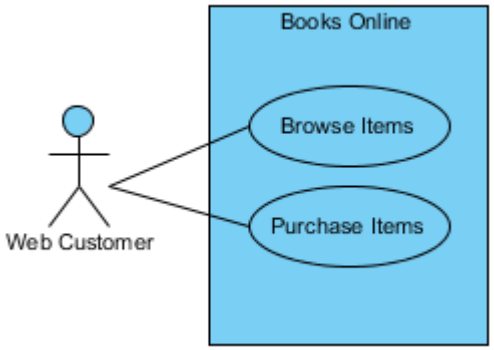


http://www.uml-diagrams.org/uml-24-diagrams.html

Use Case Diagrams

- Use case diagrams
  - Behaviour diagrams used to describe a set of actions (use cases) that some system or systems (subject) should or can perform in collaboration with one or more external users of the system (actors)
  - They do not make any attempt to represent the order or number of times that the systems actions and sub-actions should be executed

- Use case diagram components:
  - Actors
  - Use cases
  - System boundary
  - Relationships



Use Case Diagrams

- Actors
  - Entities that interface with the system
  - Can be people or other systems
  - Think of actors by considering the roles they play

- Use cases
  - Represent what the actor wants your system to do for them
  - Must be a complete flow of activity (from the actors point of view) that provides observable and valuable result to the actor(s)

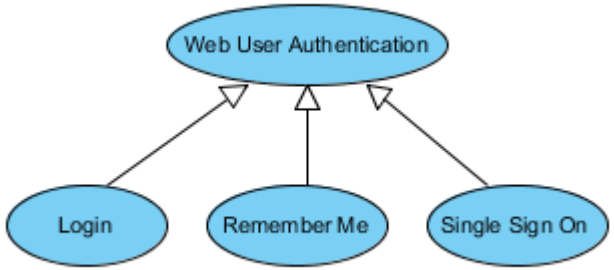
Use Case Diagrams

- Relationship between use case and actor:
  - Associations indicate which actors initiate which use cases
- Relationship between two use cases:
  - Using <<include>> or <<extend>> or generalisation

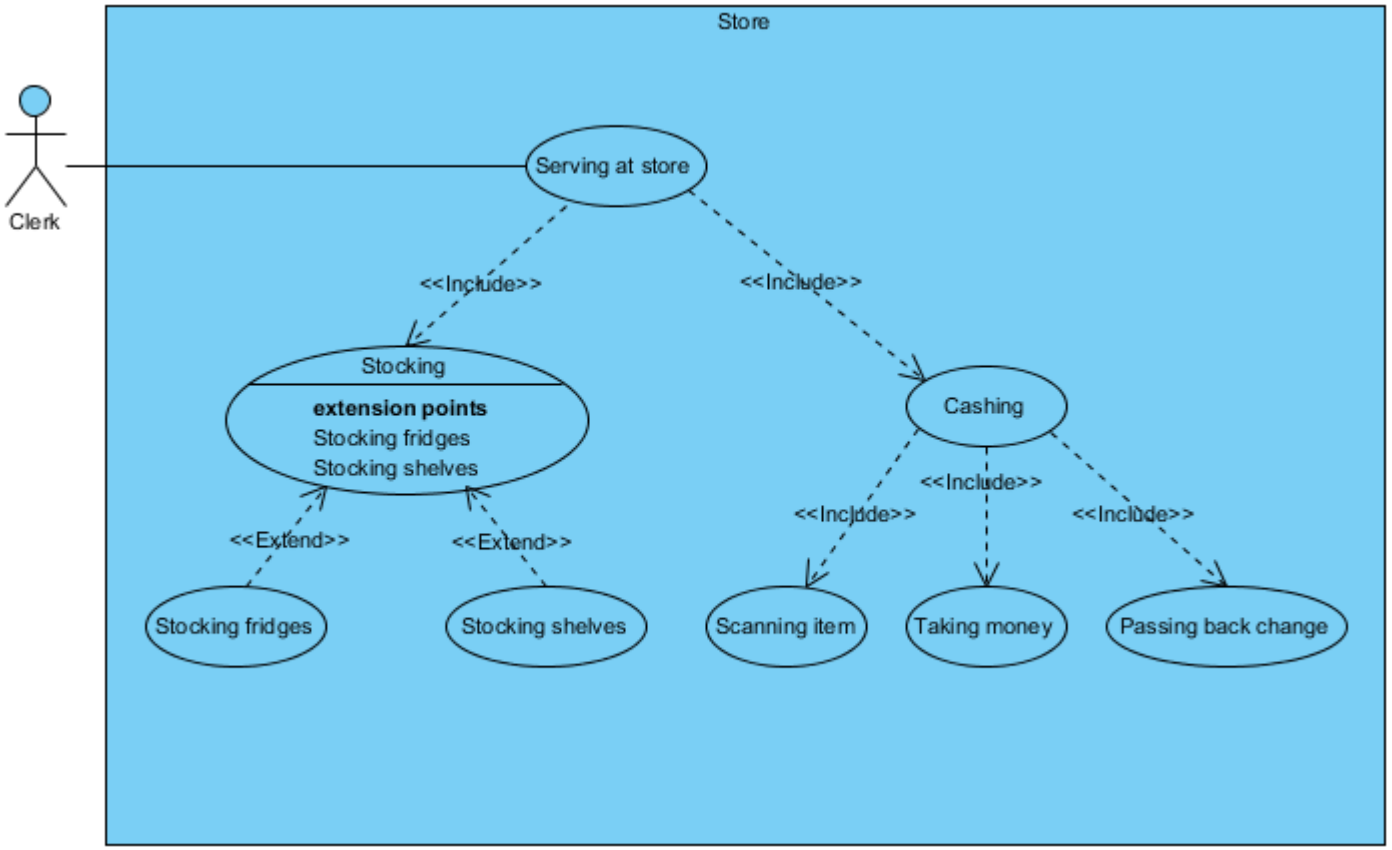
- <<include>> (specifying common functionality):
  - Multiple use cases share a piece of same functionality which is placed in a separate use case rather than documented in every use case that needs it
- <<extend>> (simplifying use case flows):
  - Activities that are performed as part of the use case but are not mandatory for that use case to run successfully

Use Case Diagrams

- Generalisation
  - Showing a task and specialised versions of it
  - Similar to <<extend>>
    - Use <<extend>> for behaviour that should sometimes be added depending on runtime conditions
    - Use "generalisation" if you want a label for a specialised version of a whole task



Use Case Diagrams



Case Study 1

- Development of a computer system for the university library
  - We use an iterative process
  - After discussing priorities with the university we decided that the first iteration of the system should provide the following use cases:
    - Borrow copy of book
    - Return copy of book
    - Borrow journal
    - Return journal

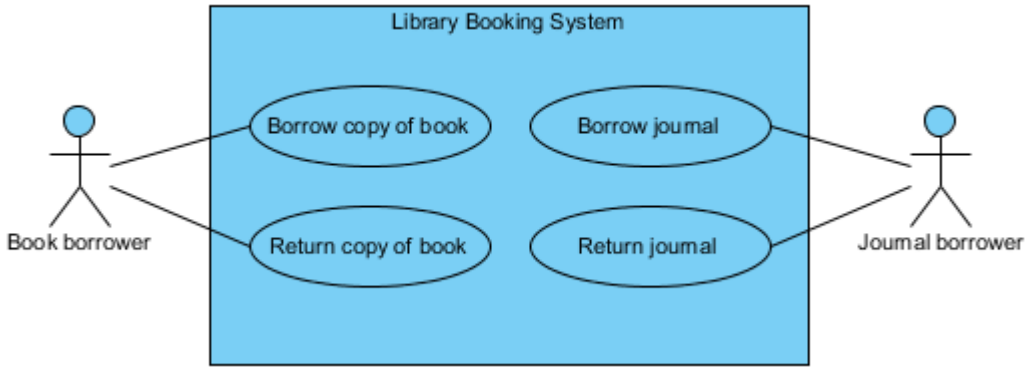
Case Study 1

- Books and journals
  - The library contains books and journals; it may have several copies of a given book; some are for short term loan only; the others can be borrowed by any library member for three weeks
  - Normal members can borrow up to 6 books at the same time, staff members up to 12
  - Only staff members can borrow journals

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

Case Study 1: Use Case Diagram

- Reminder
  - The library contains books and journals; it may have several copies of a given book; only staff members can borrow journals

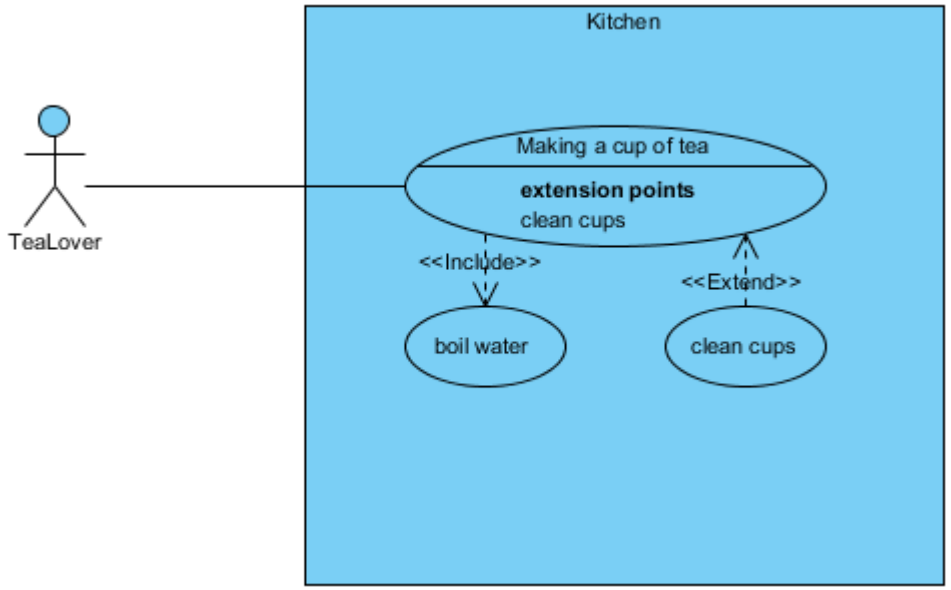


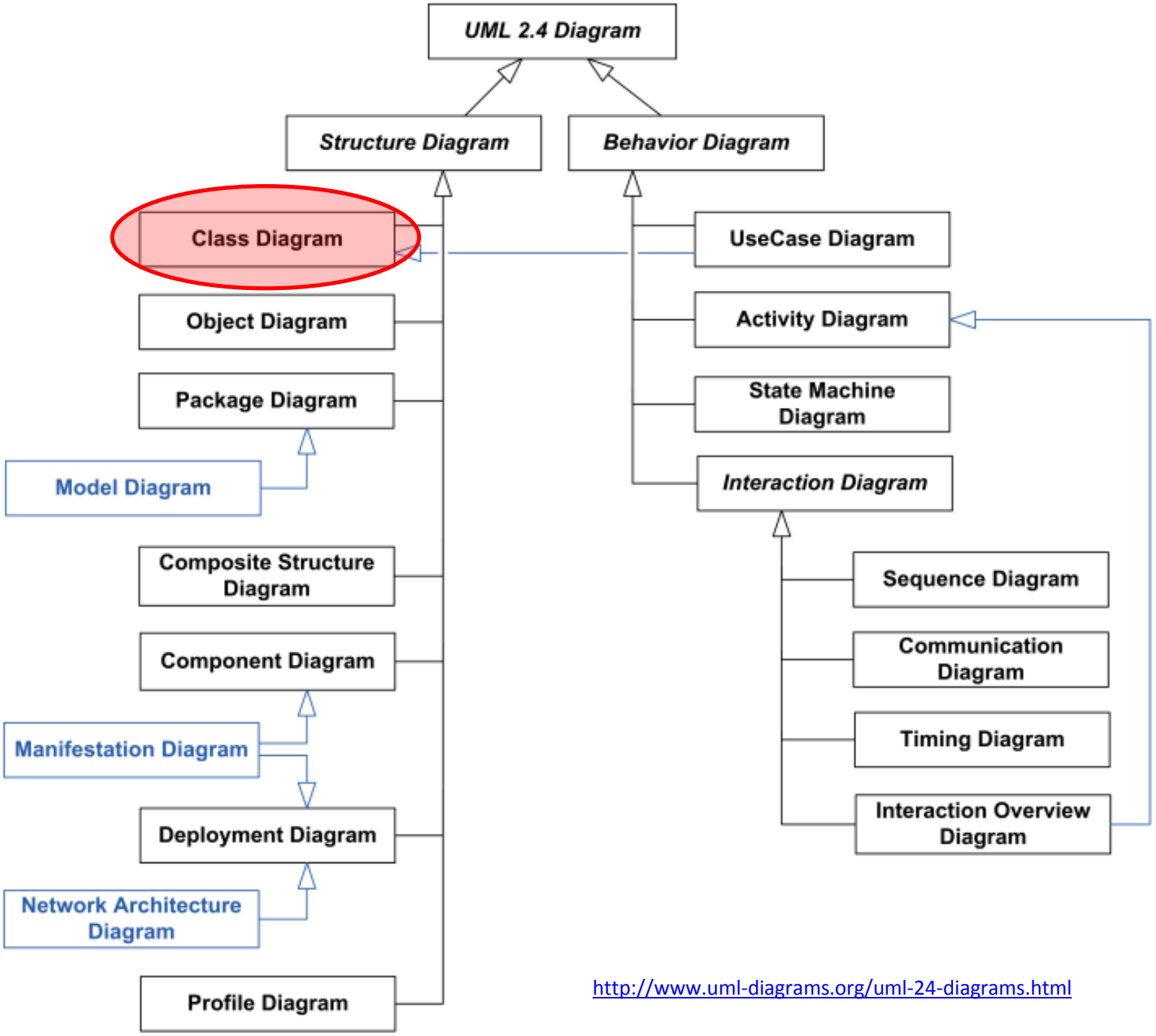
Activity 1: Use Case Diagram

- How to make a decent cup of tea (milk, no sugar) …

Activity 1: Use Case Diagram

- Making a cup of tea is already the use case (a use case is anything that has a value in its own right to the actor)





http://www.uml-diagrams.org/uml-24-diagrams.html

Class Diagrams

- Class diagrams shows the existence of classes and their structures and relationships in the logical view of a system

- Class diagram components:
  - Classes (their structure and behaviour)
  - Class relationships
    - Association
    - Aggregation
    - Composition
    - Generalisation
    - Dependency
  - Multiplicity and navigation indicators

Class Diagrams

- What makes a class model good?
  - Build a system quickly and cheaply to the satisfaction of the client
  - Build a system that is easy to maintain and adapt

- Identifying classes
  - A class describes a set of objects with an equivalent role
  - Identify candidate classes by picking all nouns and noun phrases out of a requirement specification of a system
  - Discard candidates which appear to be inappropriate (redundant, vague, an event or operation, meta-language, outside the scope of the system, an attribute)

Class Diagrams

- What kind of things are classes?
  - Tangible (real world things)
  - Roles
  - Events
  - Interactions

- First two are much more common sources for classes – the other two might help to find and name associations between them

Class Diagrams (CDs)

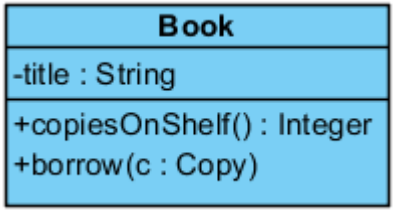
- Associations between classes
  - Correspond to verbs
  - Real world association that can be described by a short sentence (reader borrows a book)
  - Classes are associated if some object of class A has to know about some object of class B or vice versa

- Multiplicity
  - Number of links between each instance of the source class and instances of the target class
    - 1 = exactly 1; * = unlimited number (zero or more); 0..* = zero or more; 1..* = one or more; 0..1 = zero or 1; 3..7 = specified range (from 3 to 7)

Class Diagrams

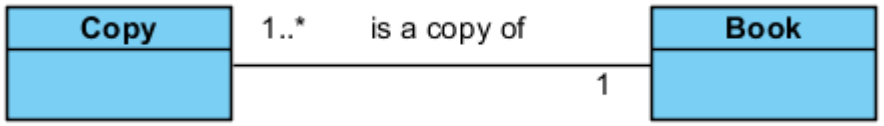
- Class representation
  - In UML classes are depicted as rectangles with three compartments
    - Class name
    - Attributes: Describe the data contained in an object of the class
    - Operations: Define the ways in which objects interact
  - Additional symbols

+	public
#	protected
-	private
/	derived
$	static



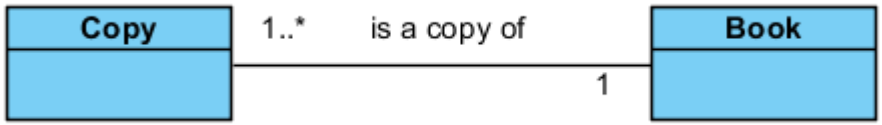
Class Diagrams

- Relationships: Associations
  - These are the most general types of relationships
  - It shows bi-directional connection between two classes
  - It is a weak coupling as associated classes remain somewhat independent of each other



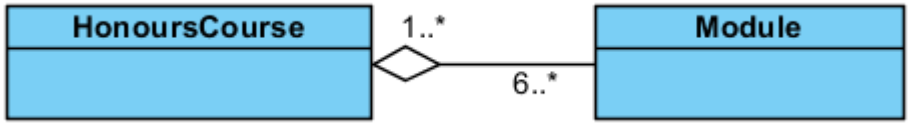
Class Diagrams

- Multiplicity:
  - The 1 at the book end of the association "is a copy of" shows that every copy (object of class Copy) is associated only with one book (object of class Book)
  - But every book is associated with one or more copies. So the multiplicity on the copy end is 1..*



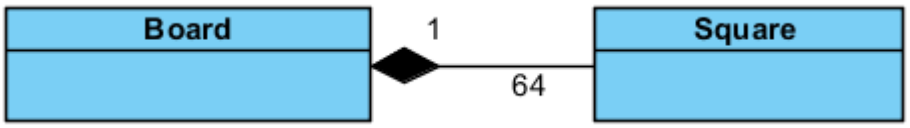
Class Diagrams

- Relationship: Aggregation ("is part of" relationship)
  - This is a special type of association
  - An association with an unfilled diamond at the end denotes the aggregate (the whole)
  - It is used when one object logically or physically contains another; the container is called "aggregate"
  - The components of aggregate can be shared with others



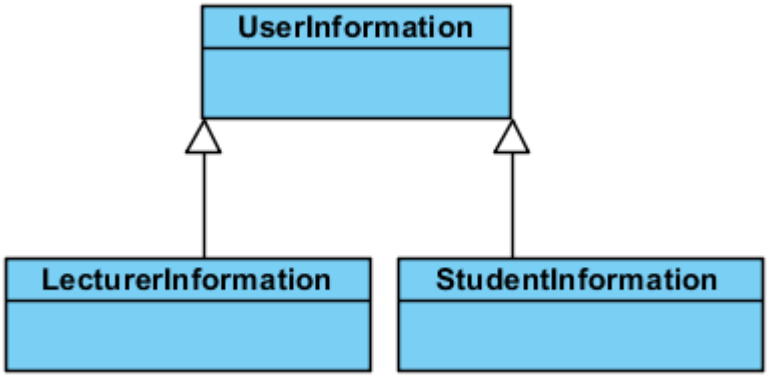
Class Diagrams

- Relationship: Composition
  - This is a strong form of aggregation
  - An association with a filled diamond at the end denotes the composition (physical containment)
  - The multiplicity at the composition end is always 1 as the parts have no meaning outside the whole
  - If the whole is copied or deleted its parts are copied or deleted together with it; the owner is explicitly responsible for creation and deletion of the parts



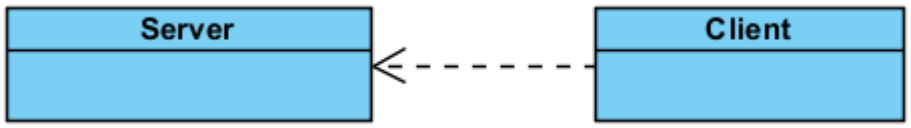
Class Diagrams

- Relationship: Generalisation ("is a" relationship)
  - Implemented by inheritance



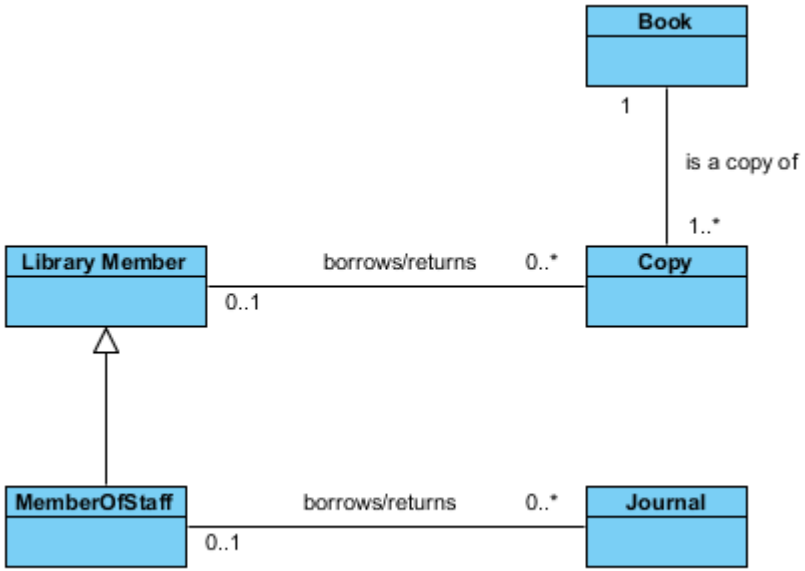
Class Diagrams

- Dependency
  - Dependency is semantic connection between dependent and independent model elements
  - A dependency exists between two elements if changes to the definition of one element (the supplier or target) may cause changes to the other (the client or source)
  - This association is uni-directional



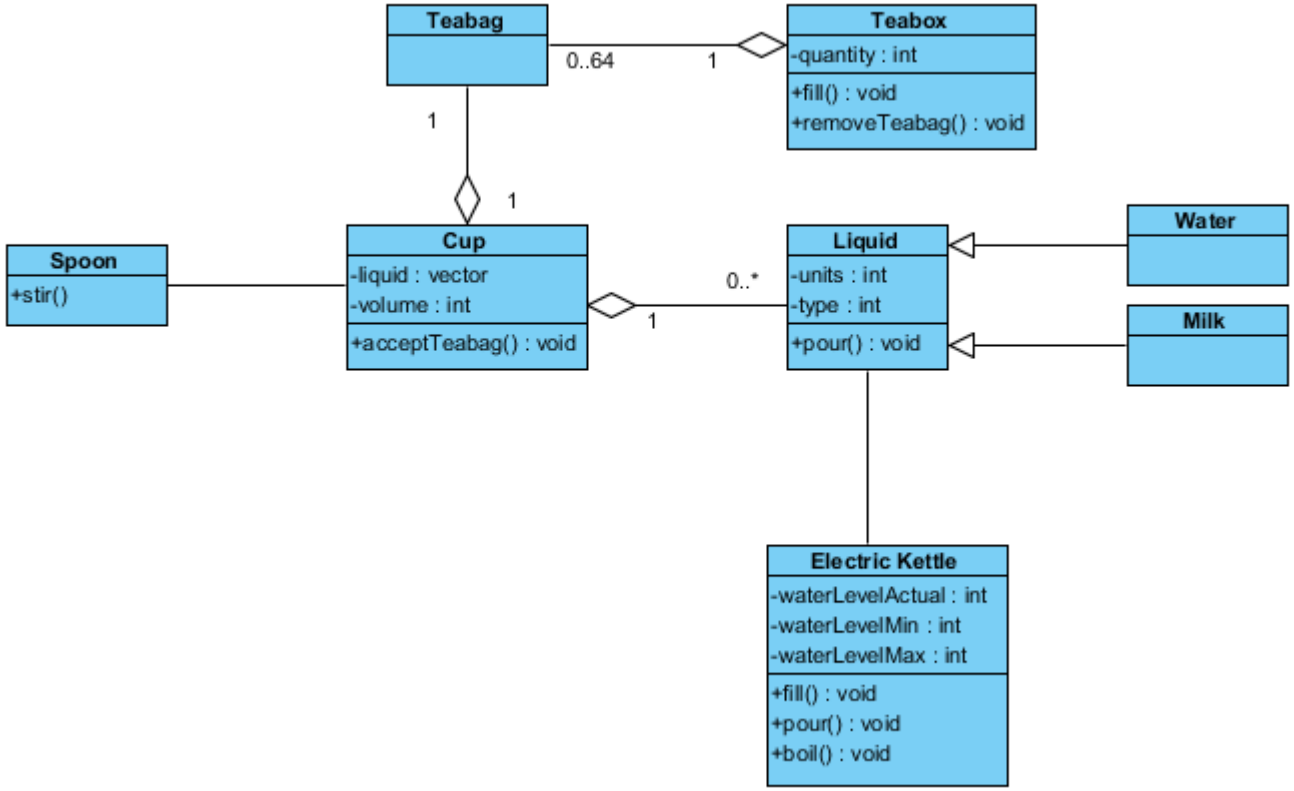
Case Study 1: Class Diagram

- Reminder
  - The library contains books and journals; it may have several copies of a given book; only staff members can borrow journals



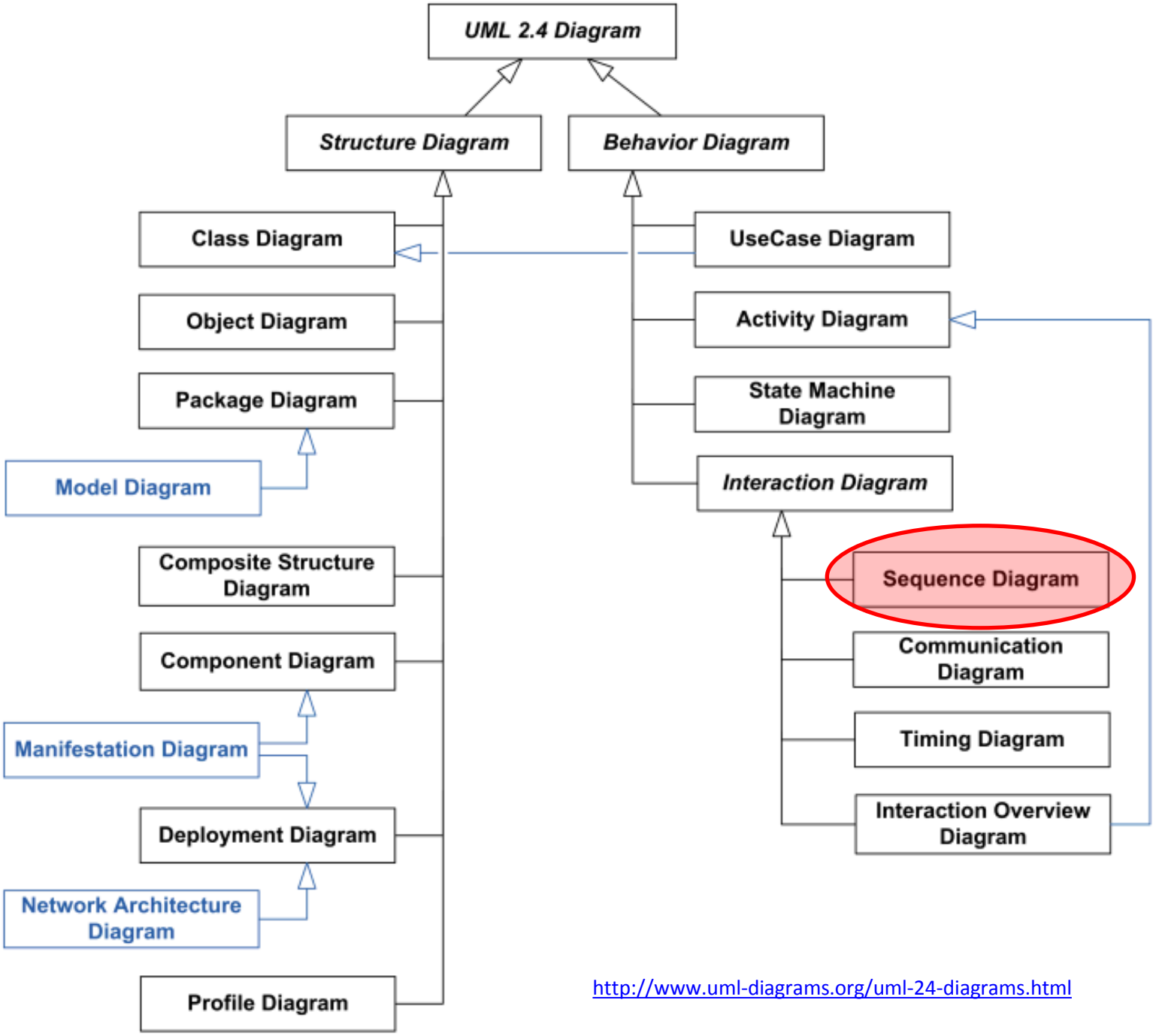
Activity 1: Class Diagram

- How to make a decent cup of tea (milk, no sugar) …


Teabag
0..64
1
Teabox
-quantity : int
+fill() : void
+removeTeabag() : void
1
1
Spoon
+stir()
Cup
-liquid : vector
-volume : int
+acceptTeabag() : void
0..*
1
Liquid
-units : int
-type : int
+pour() : void
Water
Milk
Electric Kettle
-waterLevelActual : int
-waterLevelMin : int
-waterLevelMax : int
+fill() : void
+pour() : void
+boil() : void

Break

- See you back in 10 minutes

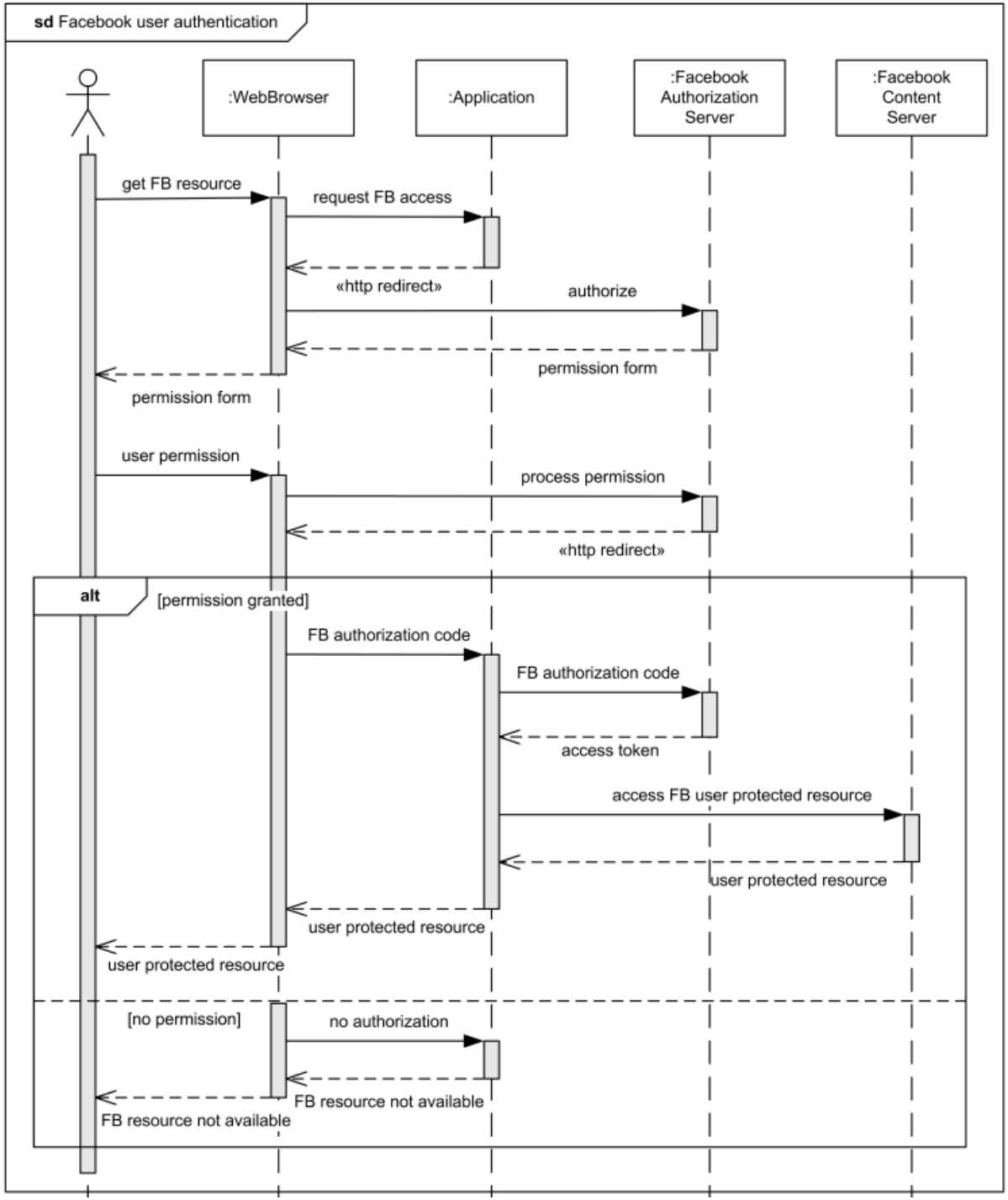


http://www.uml-diagrams.org/uml-24-diagrams.html

Sequence Diagrams

- Sequence diagrams are a temporal representation of objects and their interactions; they shows the **objects** and **actors** taking part in a collaboration at the top of dashed lines

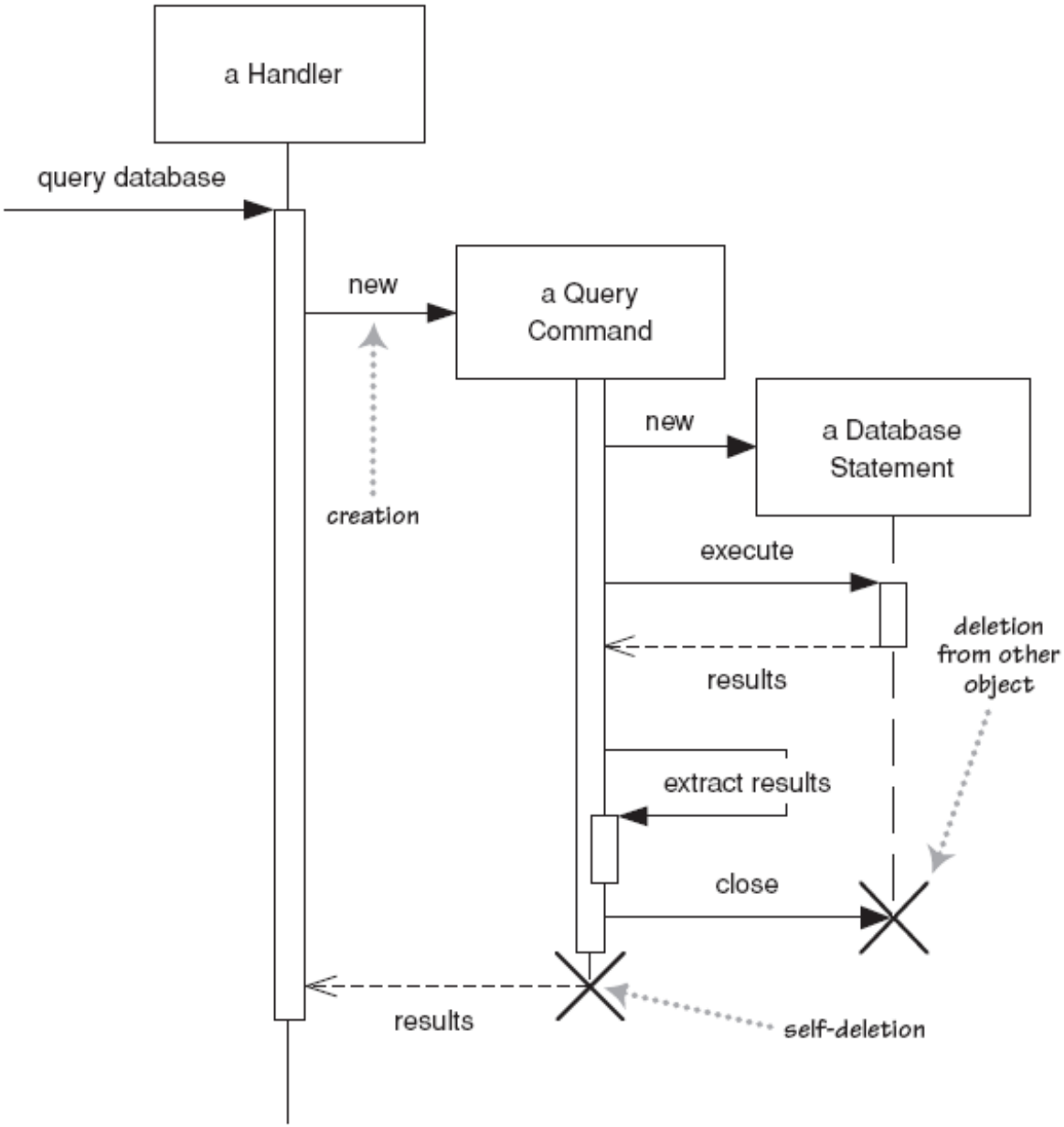
- Sequence diagrams components
  - Participants are objects or actors that act in the sequence diagram
  - Lines represent time as seen by the object (lifeline)
  - Arrows from lifeline of sender to lifeline of receiver are messages (denoting events or the invocation of operations)
  - A narrow rectangle covering an object's life line shows a live activation of the object


sd Facebook user authentication
:WebBrowser
:Application
:Facebook Authorization Server
:Facebook Content Server
get FB resource
request FB access
«http redirect»
authorize
permission form
permission form
user permission
process permission
«http redirect»
alt
[permission granted]
FB authorization code
FB authorization code
access token
access FB user protected resource
user protected resource
user protected resource
user protected resource
[no permission]
no authorization
FB resource not available
FB resource not available

Sequence Diagrams

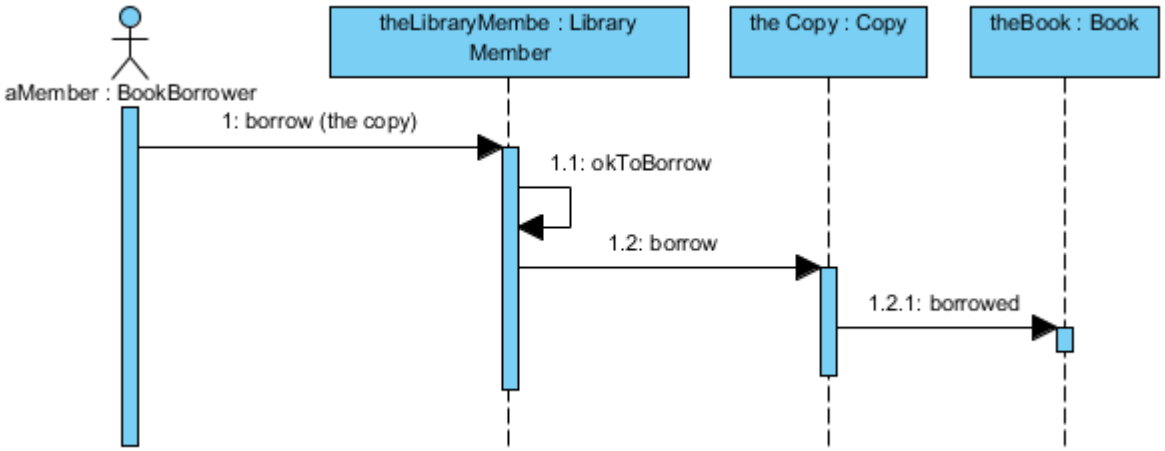
- The axes in a sequence diagram
  - Horizontal: which object/participant is acting
  - Vertical: time (down -> forward in time)

- Creation: arrow with 'new' written above it
  - Notice that an object created after the start of the scenario appears lower than the others
- Deletion: an X at bottom of object's lifeline
  - In some OOP languages this is handled automatically


a Handler
query database
new
a Query Command
creation
new
a Database Statement
execute
deletion from other object
results
extract results
close
results
self-deletion

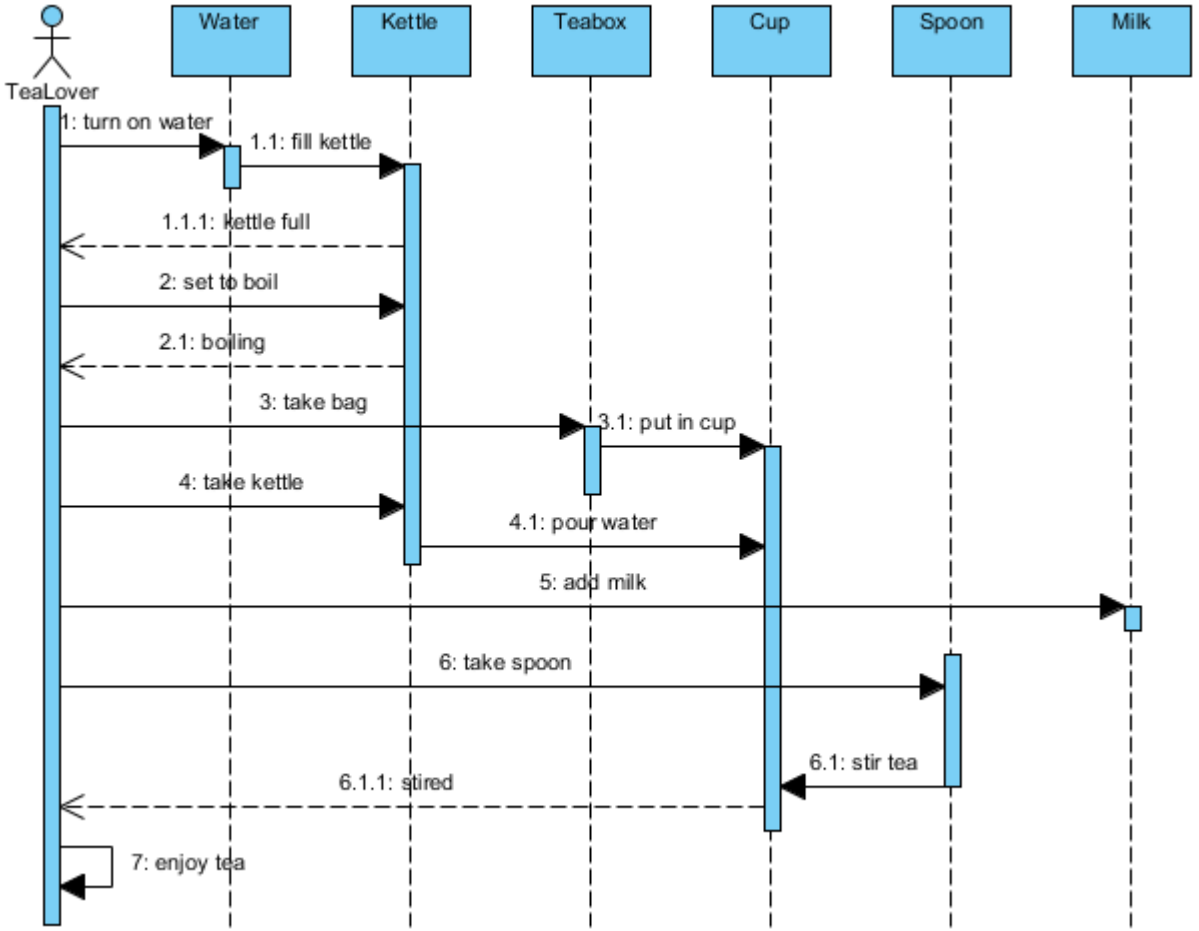
Case Study 1: Sequence Diagram

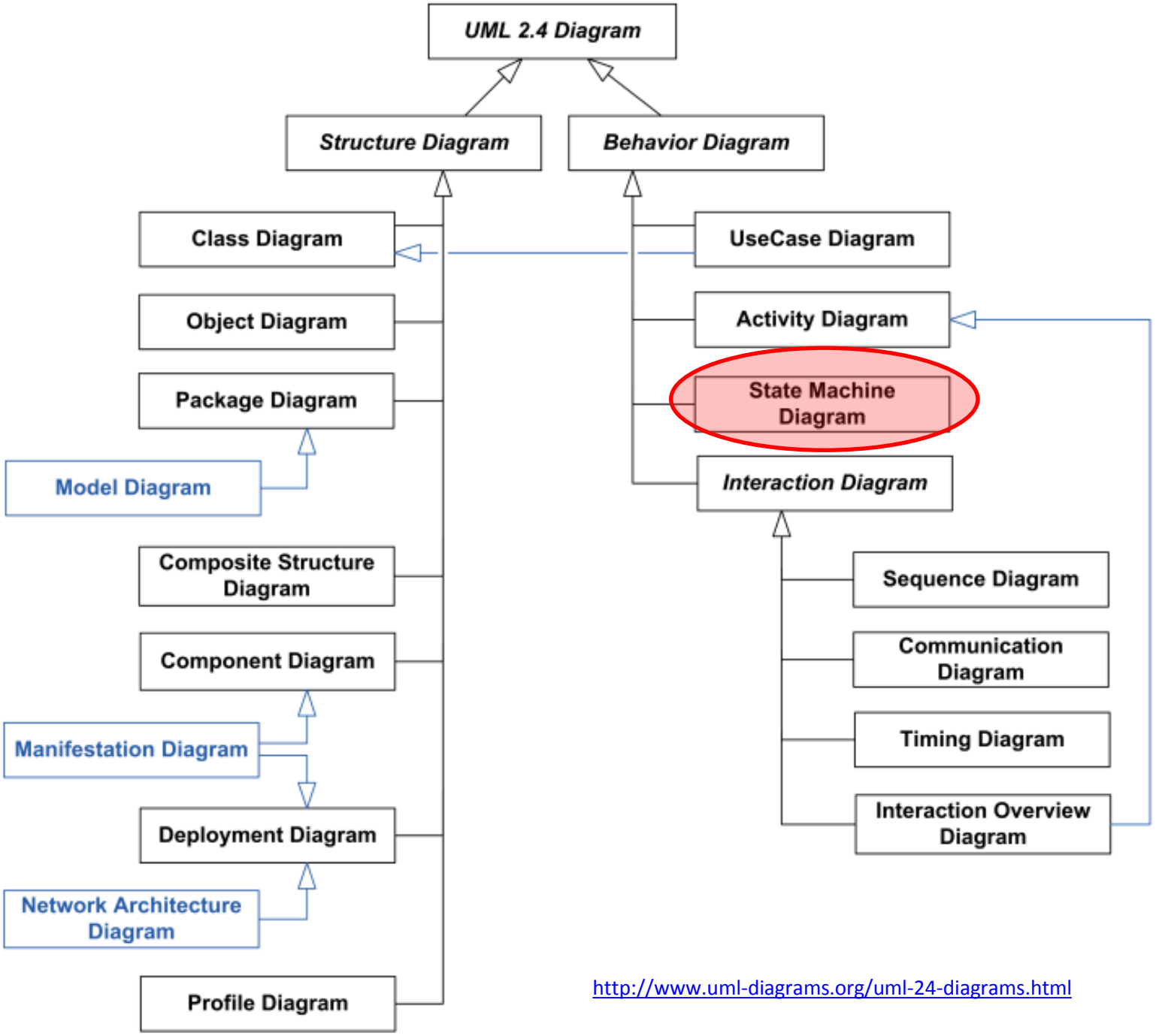
- Reminder
  - The library contains books and journals; it may have several copies of a given book; only staff members can borrow journals



Activity 1: Sequence Diagram

- How to make a decent cup of tea (milk, no sugar) …


TeaLover
Water
Kettle
Teabox
Cup
Spoon
Milk
1: turn on water
1.1: fill kettle
1.1.1: kettle full
2: set to boil
2.1: boiling
3: take bag
3.1: put in cup
4: take kettle
4.1: pour water
5: add milk
6: take spoon
6.1: stir tea
6.1.1: stired
7: enjoy tea



http://www.uml-diagrams.org/uml-24-diagrams.html

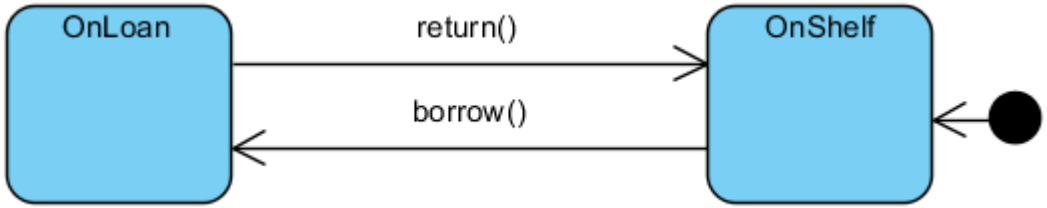
State Machine Diagrams

- State machine diagrams
  - In order to implement a class we need to understand what the dependencies are between the state of an object and its reaction to messages or other events
  - State machine diagrams show the states of a single object, the events or the messages that cause a transition from one state to another and the action that result from a state change.
  - You do not have to create a state machine diagram for every class!

State Machine Diagrams

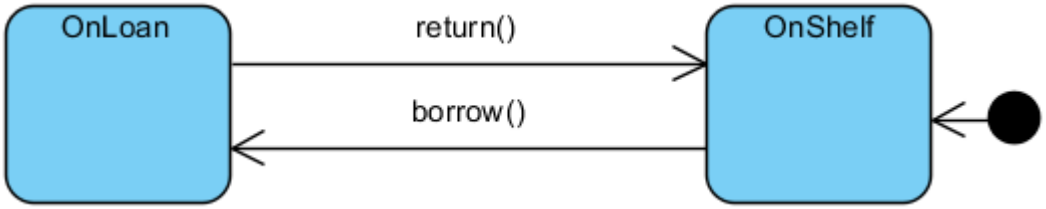
- State: A condition during the life of an object when it satisfies some condition, performs some action, or waits for an event

- There are two special states
  - Start state: Each state diagram must have one and only one start state
  - Stop State: An object can have multiple stop states



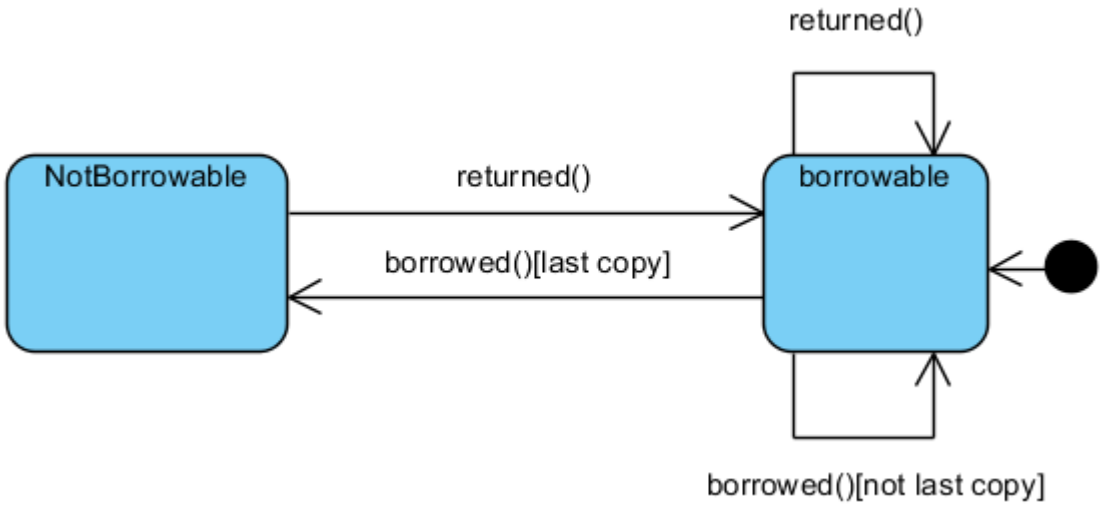
Case Study 1: State Machine Diagram

- Lets do some reverse engineering …
  - What does the class diagram look like?



State Machine Diagrams

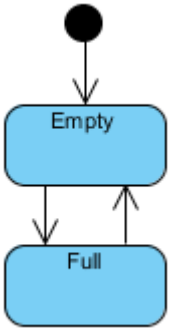
- Guard
  - Sometimes a change of state of the object depends on the exact values of an object's attributes
  - Guard conditions affect the behaviour of a state machine by enabling actions or transitions only when they evaluate to TRUE and disabling them when they evaluate to FALSE.



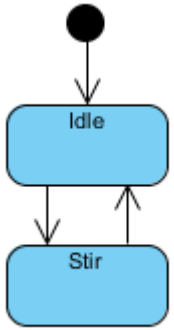
Activity 1: State Machine Diagram

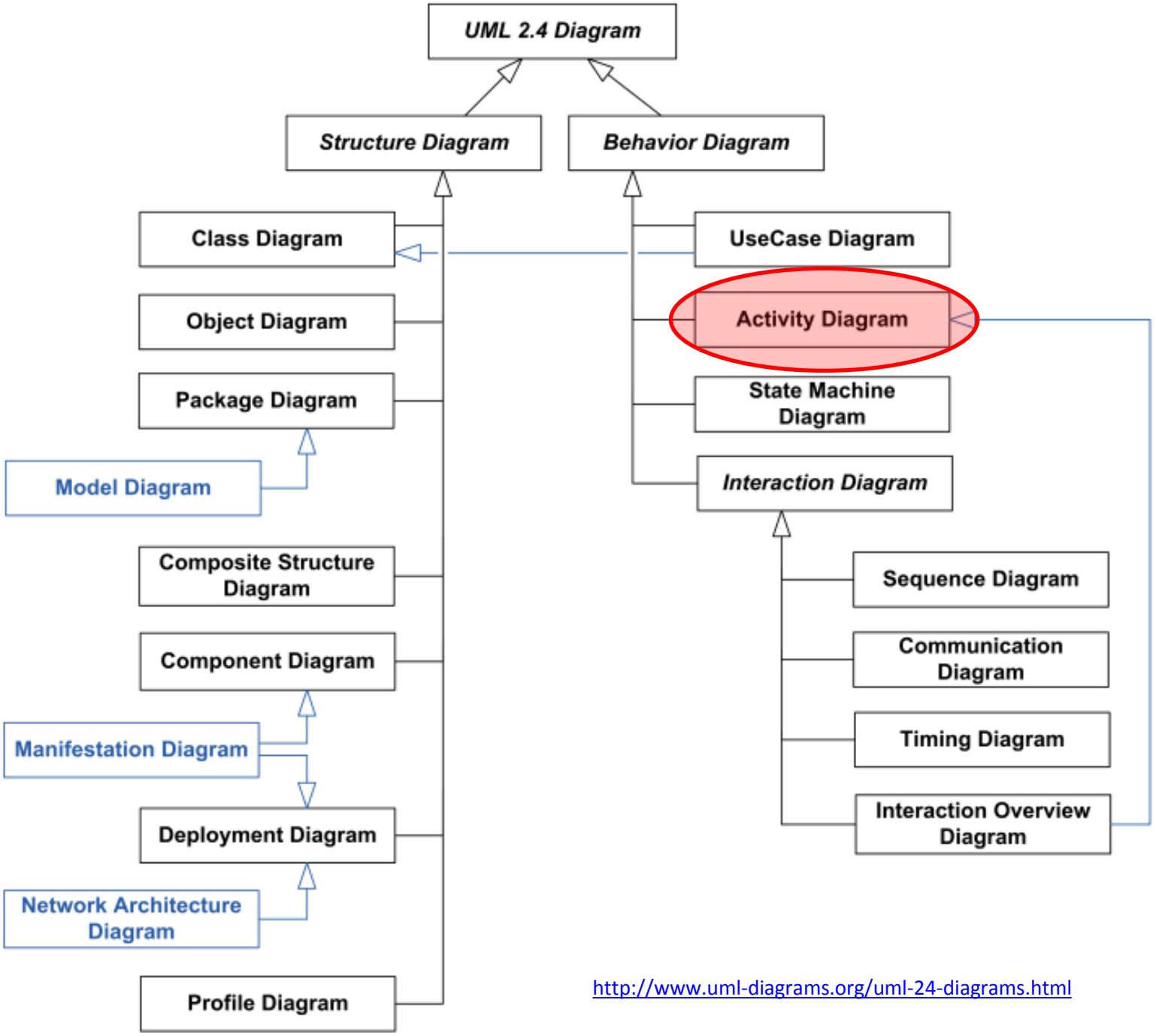
- How to make a decent cup of tea (milk, no sugar) …

- Cup
- Kettle
- Tea box



- Spoon





http://www.uml-diagrams.org/uml-24-diagrams.html

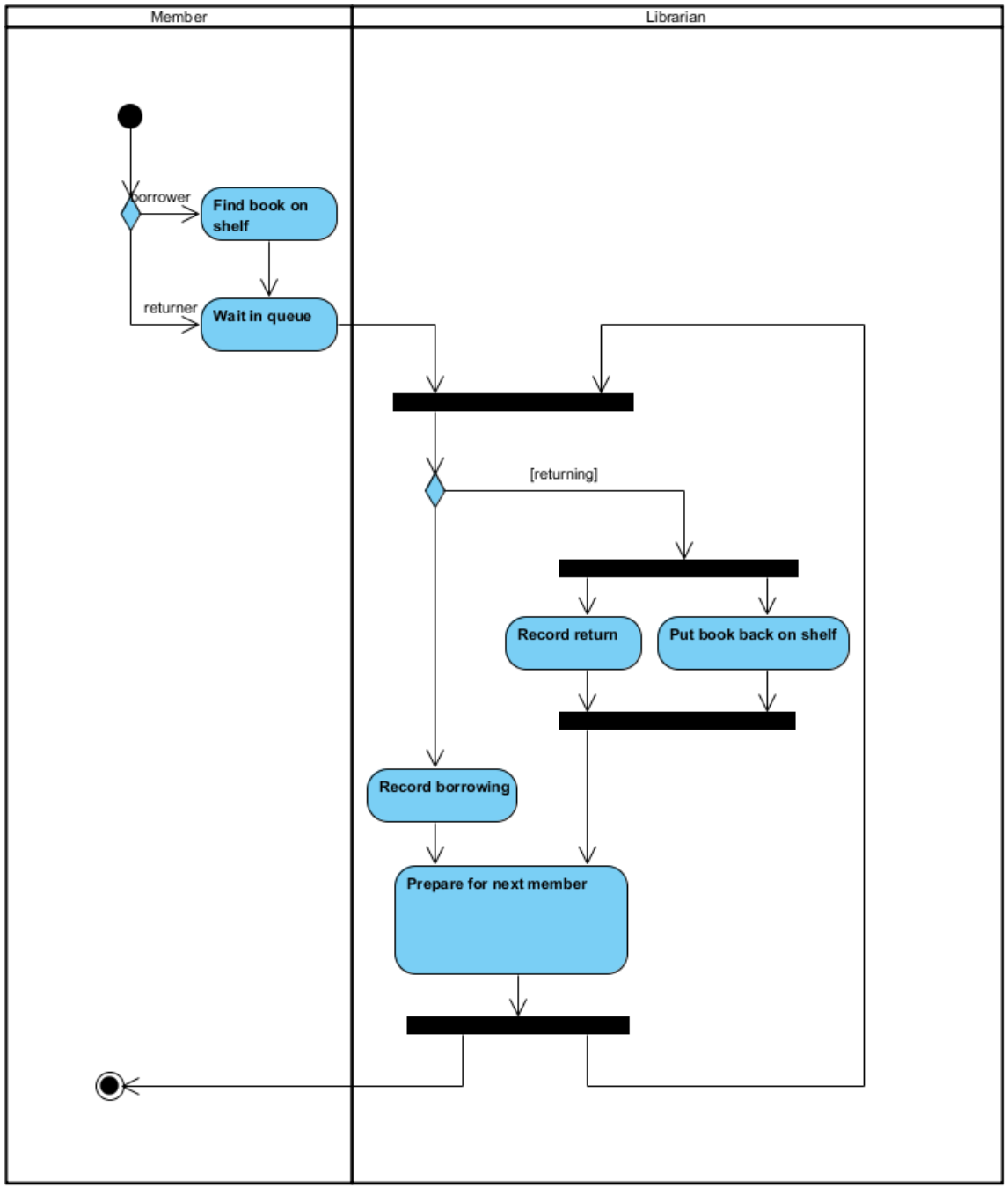
Activity Diagrams

- Activity diagrams describe how activities are co-ordinated
- Support parallel behaviour (unlike flowcharts)

- Activity diagrams are recommended in the following situations:
  - Analysing use case
  - Dealing with multithreaded application
  - Understanding workflow across many use cases

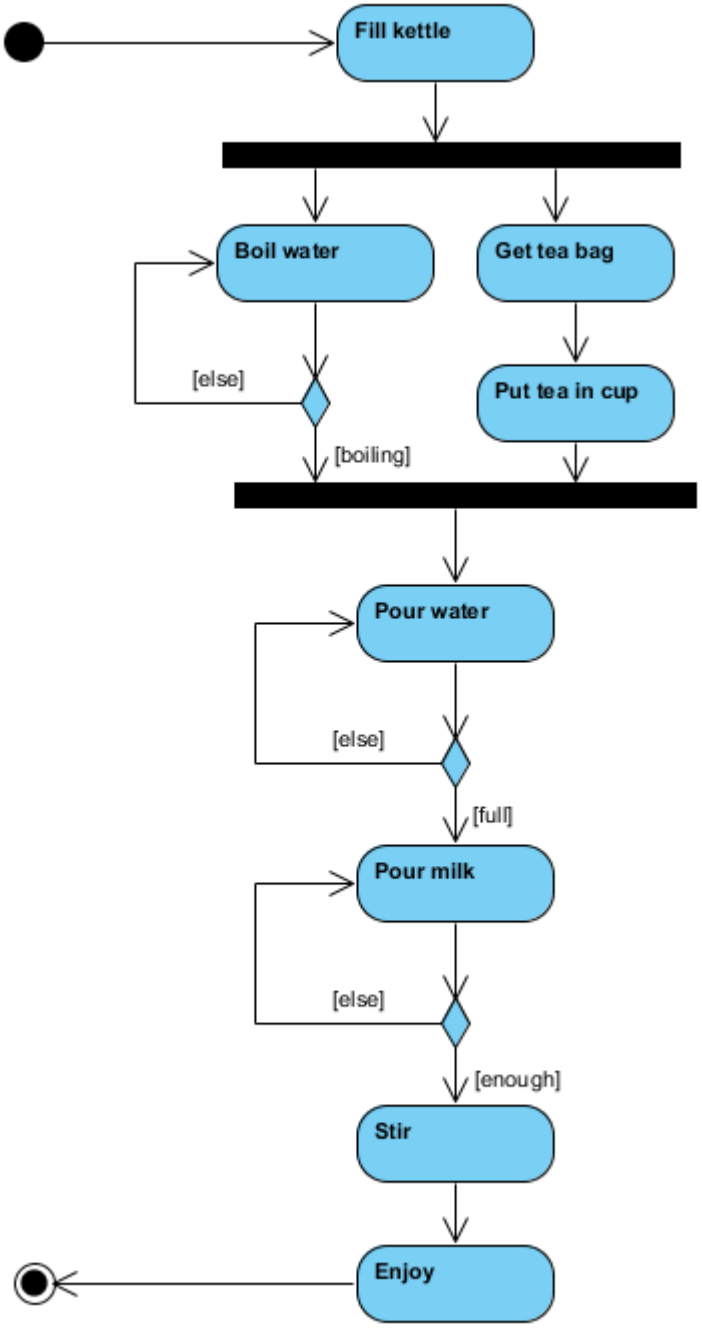
Activity Diagrams

- Activity diagram components
  - Activities: Named box with rounded corners (you can think of an activity as a state that is left once the activity is finished)
  - Activity edge: Arrow (similar to transition but fires only when the previous activity completes)
  - Synchronisation bar: Thick horizontal line describing the co-ordination of activities
  - Decision diamond: Used to show decisions
  - Start and stop markers: Same as in state diagrams


Member
Librarian
borrower
Find book on shelf
returner
Wait in queue
[returning]
Record return
Put book back on shelf
Record borrowing
Prepare for next member

Activity 1: Activity Diagram

- How to make a decent cup of tea …


Fill kettle
Boil water
Get tea bag
[else]
Put tea in cup
[boiling]
Pour water
[else]
[full]
Pour milk
[else]
[enough]
Stir
Enjoy

Summary and Outlook

- Summary
  - UML diagrams are very useful for staying organised during software development and for communication with team members or clients
  - There are many different types of UML diagrams out there but usually people only use a very small subset (most often class and sequence diagrams)

- Next week
  - Principles of Object Oriented Design

Questions / Comments

Bibliography

- Fowler (2004) UML Distilled - 3e
- Booch et al (2007) Object-Oriented Analysis and Design - 3e
- UML 2.4 Diagrams Overview (http://www.uml-diagrams.org/uml-24-diagrams.html)
- Object Management Group (OMG) (http://www.omg.org/)