

COMP5911M

Advanced Software Engineering

3: Review – Modelling Software With UML

Nick Efford

<https://comp5911m.info>

Last Time



UNIVERSITY OF LEEDS

- We reviewed fundamental object-oriented concepts
- We discussed the different types of relationship that can exist between classes
- We explored the ideas of Liskov substitution and polymorphism, using a simple graphics application
- We finished by considering the role played by abstract classes and interfaces in OO systems

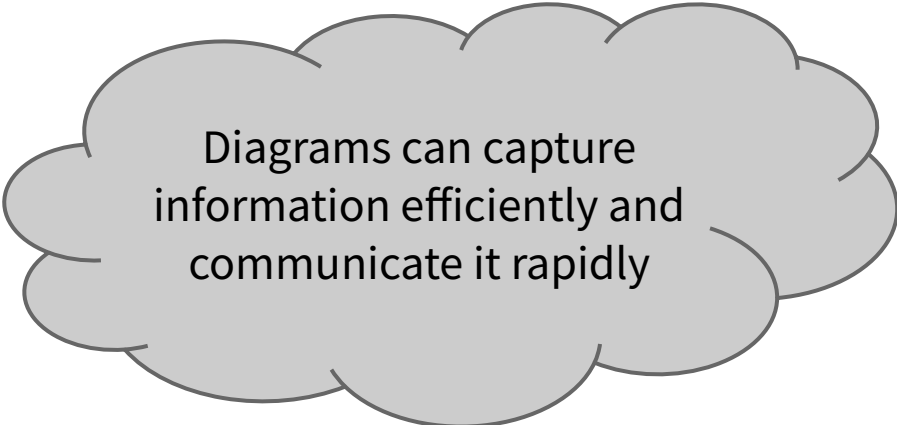
Today's Objectives

- To review the role played by UML in software engineering
- To highlight diagrams that are particularly helpful when developing or improving software architecture
 - Class diagrams
 - Component diagrams

Visual Modelling

“A picture shows me at a glance
what it takes dozens of pages of a
book to expound”

Ivan Turgenev, 1862



Diagrams can capture
information efficiently and
communicate it rapidly



1913 newspaper ad

What Does UML Give Us?

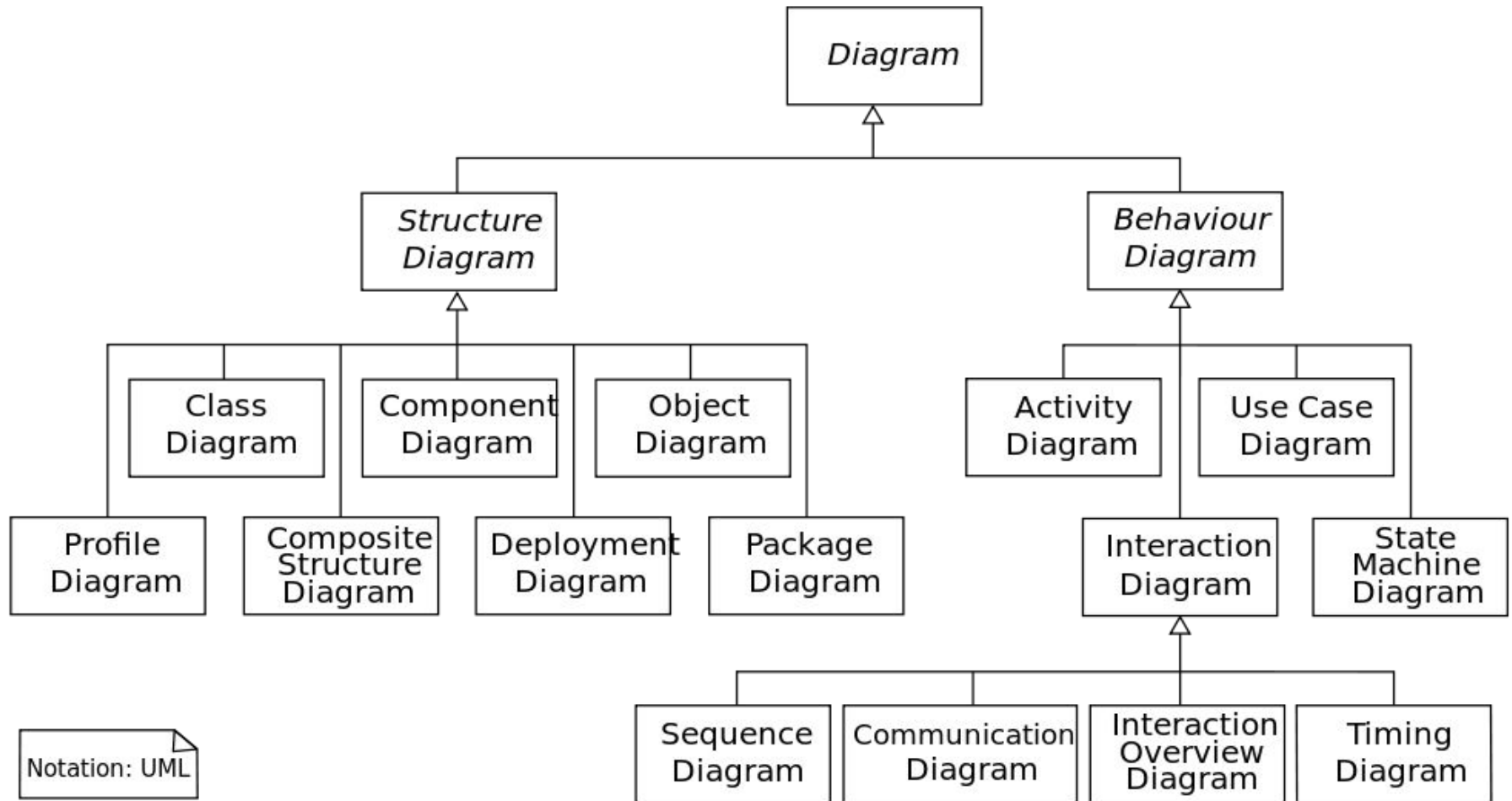
Different **views** into a **model** of a system, aimed at different stakeholders in a software project:

- **Use case view** focuses on scenarios executed by human users and external systems (the ‘what’, not the ‘how’)
- **Design view** focuses on key entities in problem domain and proposed solution, encompassing both structural & behavioural aspects of system
- **Process view** focuses on aspects involving flow & timing
- **Implementation view** focuses on elements that make up the physical system (executables, config files, databases...)
- **Deployment view** focuses on how software components are distributed across hardware

UML Diagrams



UNIVERSITY OF LEEDS



(Source: Wikimedia Commons)

Agile Perspective



UNIVERSITY OF LEEDS

- UML as a standard is too large and too complex; you only need a small subset of it to do useful modelling
- Draw UML diagrams only when they help you move towards working software
- Use only as much detail as you need to help you move forward with design and implementation
- Fancy diagramming tools aren't necessary – sketches on paper or a whiteboard are good enough

Key Structural Diagrams



UNIVERSITY OF LEEDS

- **Class diagram** describes detailed system structure in terms of classes, their attributes and relationships
- **Package diagram** describes **logical** grouping of related classes or use cases, and intergroup dependencies
- **Component diagram** describes system in terms of its higher-level design units, their interfaces & dependencies
- **Deployment diagram** describes how physical artifacts map onto hardware nodes and how nodes communicate

Class Diagrams: Basic Features



- Minimally, a rectangle containing class name
- More detailed representation:
 - Rectangle with three stacked compartments
 - Class name in top compartment
 - **Attributes** listed in middle, **operations** in bottom

Adding Detail

- At the implementation level, attributes become **fields**, operations become **methods**
- Attributes & fields can have types and default values
- Operations & methods can have parameter lists (with types) and can show a return type
- Visibility: + for public, - for private, # for protected

BankAccount
- accountNumber: String - balance: int = 0
+ checkBalance(): int + deposit(amount: int) + withdraw(amount: int)

Relationships



UNIVERSITY OF LEEDS

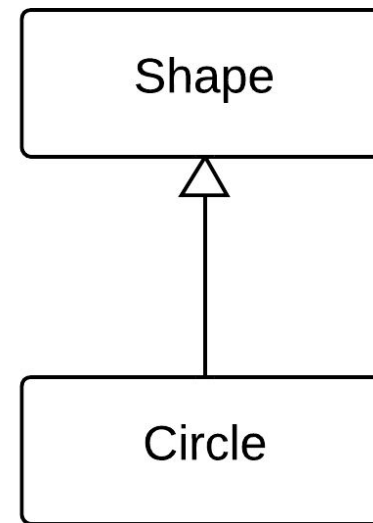
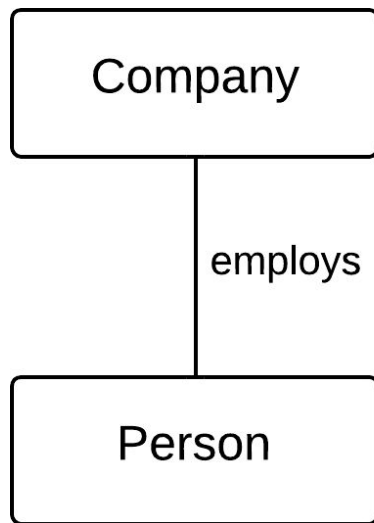
Association

- Class A ‘knows about’ class B
- Implies that an instance of A has access to an instance of B, and can call methods on it (+ *may* imply the reverse)
- Represented as solid line, ideally labelled, between classes
- **Navigability** optionally indicated using a V-shaped arrowhead

Specialisation

- Class B ‘is a kind of’ (specialises / inherits from) class A
- Solid line with unfilled triangular arrowhead, pointing **at the more general class**

Examples



Exercise



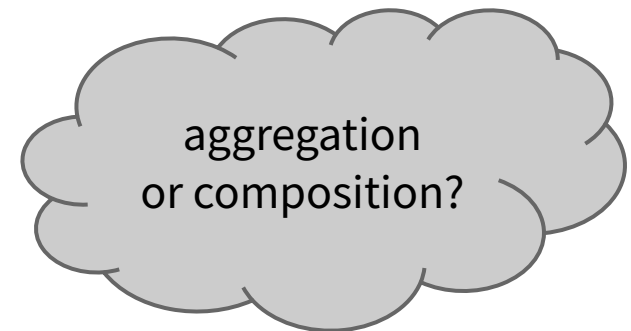
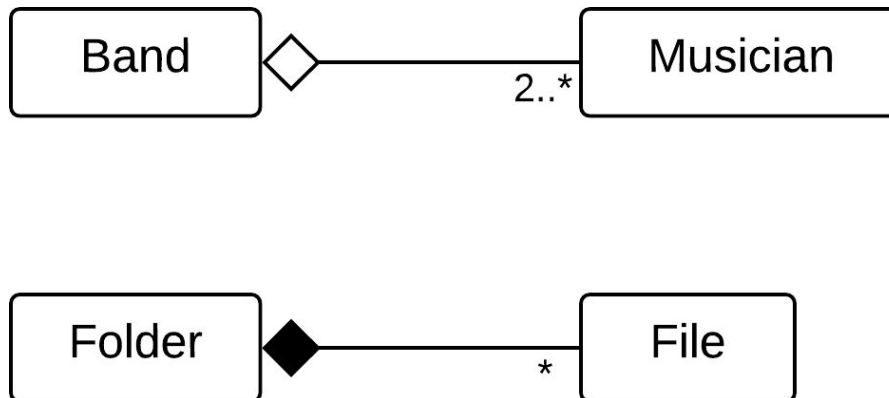
UNIVERSITY OF LEEDS

A university gym has members, who book sessions to use the equipment. Members can be either staff or students. Some rules are the same for these two types of member but others are different.

Draw a class diagram to show the **four** classes implied by the above description, and their relationships.

Aggregation & Composition

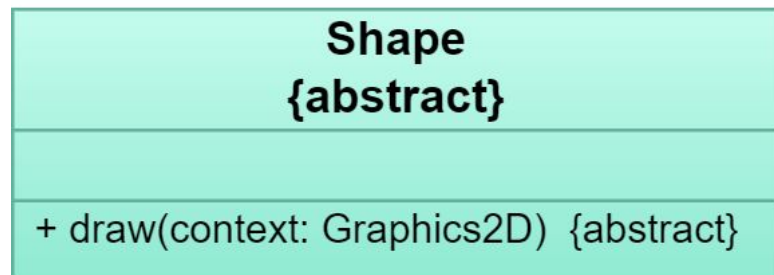
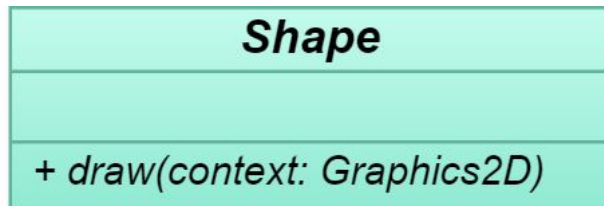
- Used when one class is a composite and the other is a part (e.g., `Inbox` and `EmailMessage`)
- Aggregation allows for sharing of parts, composition indicates no sharing ('strong ownership')



Special Syntax

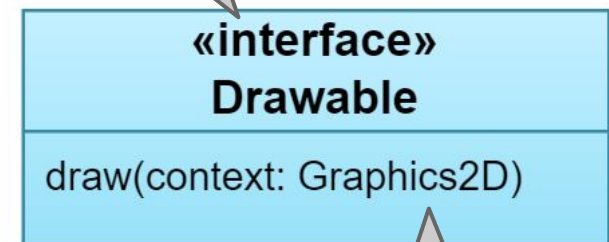


UNIVERSITY OF LEEDS



Indicate abstractness using italics
or {abstract} constraint

Use a **stereotype**
to indicate an
interface

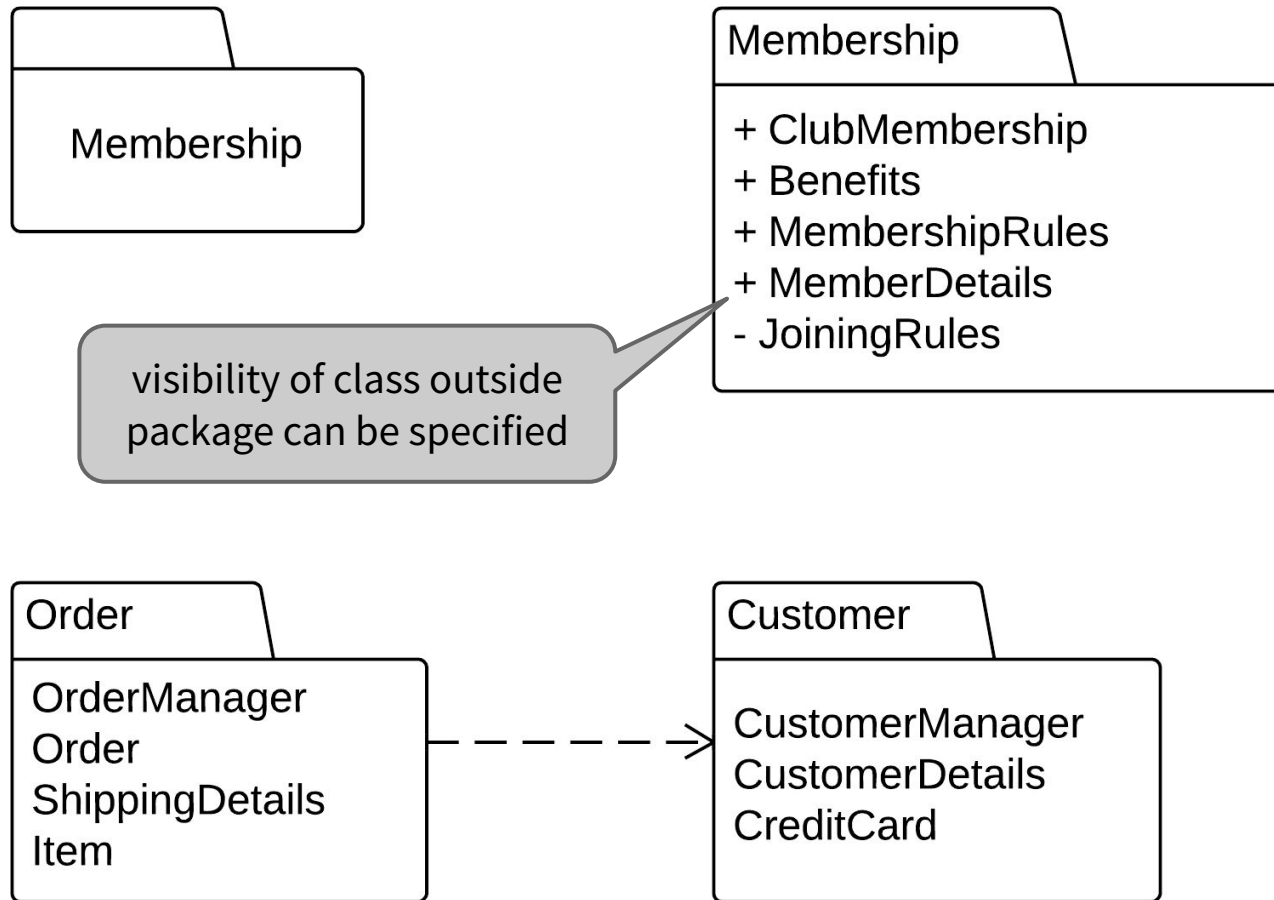


no middle section;
bottom section lists
methods

Package Diagrams

- A **package** is an organisational mechanism for
 - Grouping related elements
 - Defining a ‘semantic boundary’ in the model
 - Providing an encapsulated namespace
- Packages typically contain classes or use cases
- UML symbol for a package is a folder, possibly showing the packaged elements inside
- Package dependencies are shown using a dashed line with a V-shaped arrowhead

Package Examples



Component Diagrams



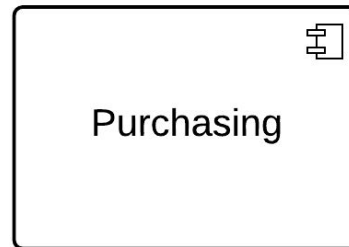
UNIVERSITY OF LEEDS

- **Components** are the high-level design units of a system: the discrete elements that define its architecture
- Components are typically treated as ‘black boxes’, with well-defined **interfaces** through which they interact with each other
- Component interaction can be shown as a dependency (dashed line, V-shaped arrowhead) or using ‘ball and socket notation’

Component Representations



UML 1.x (ideally don't use this!)

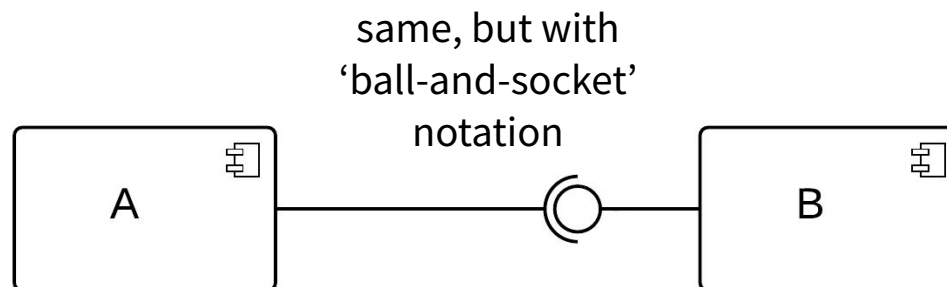
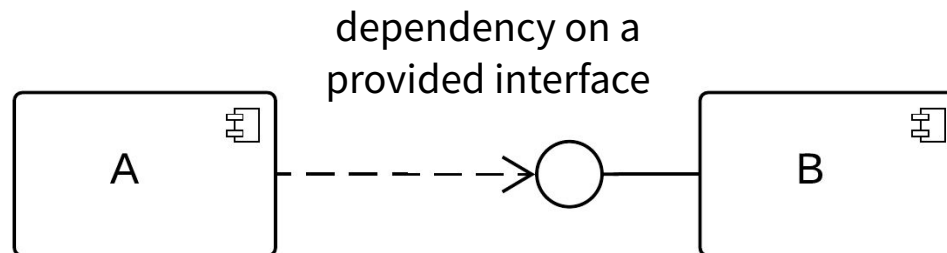
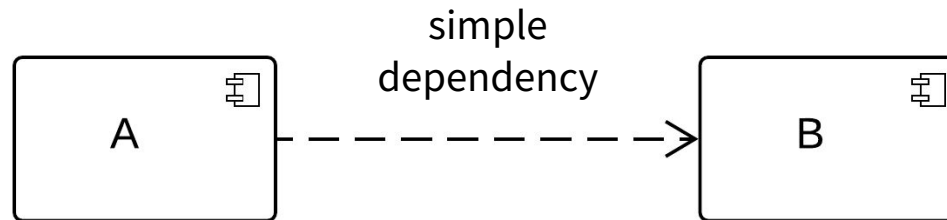


Equivalent options in UML 2

Dependency Syntax



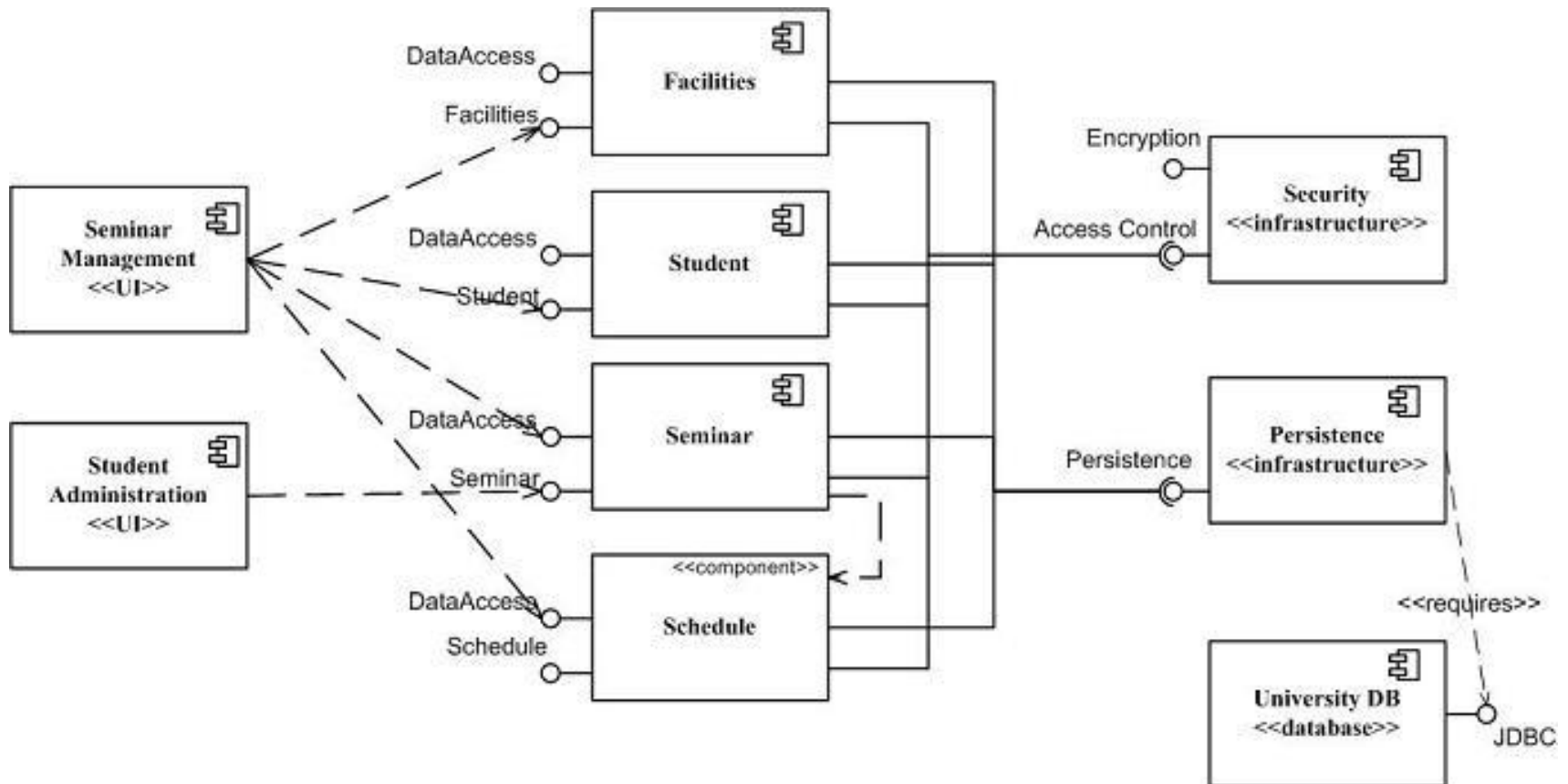
UNIVERSITY OF LEEDS



Example

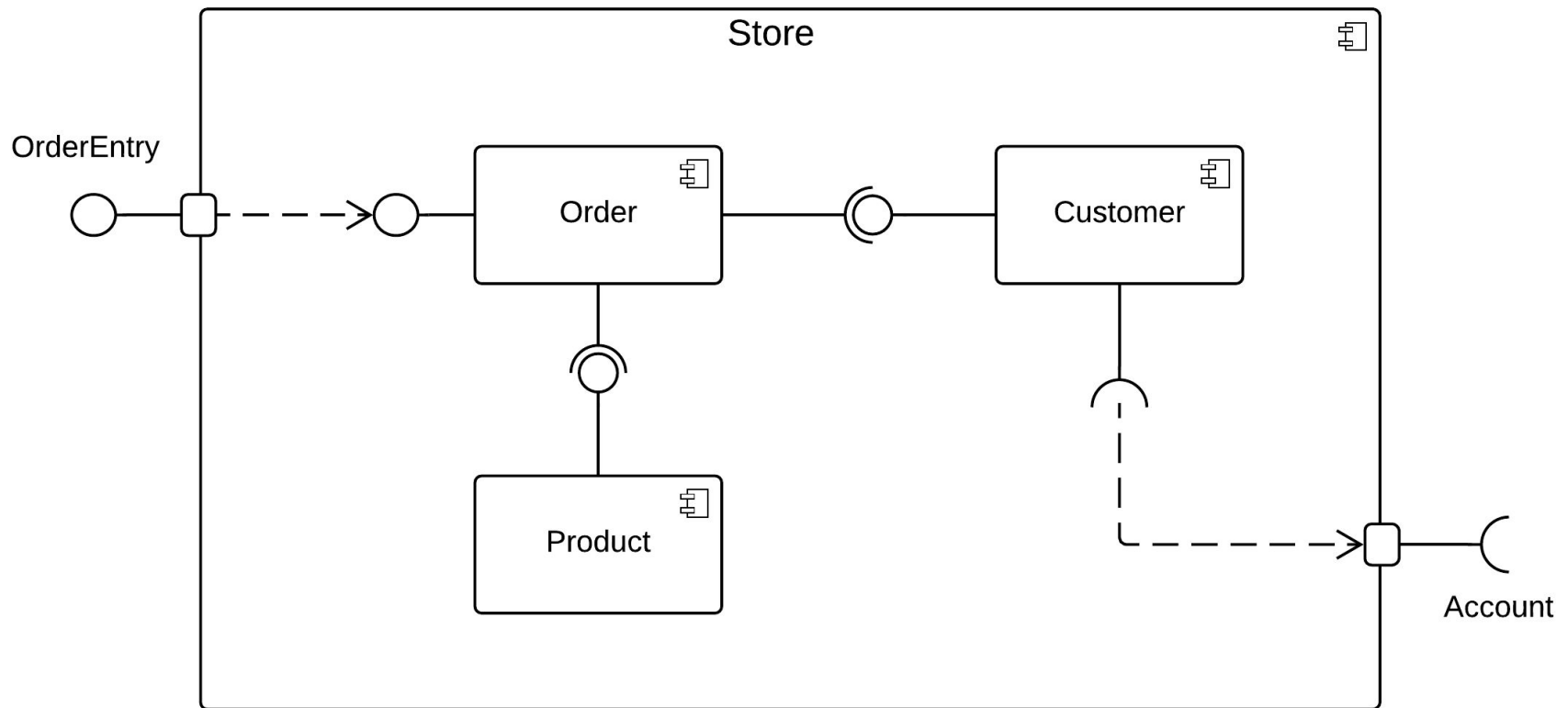


UNIVERSITY OF LEEDS



(by Scott Ambler, <http://www.agilemodeling.com>)

Ports and Subcomponents



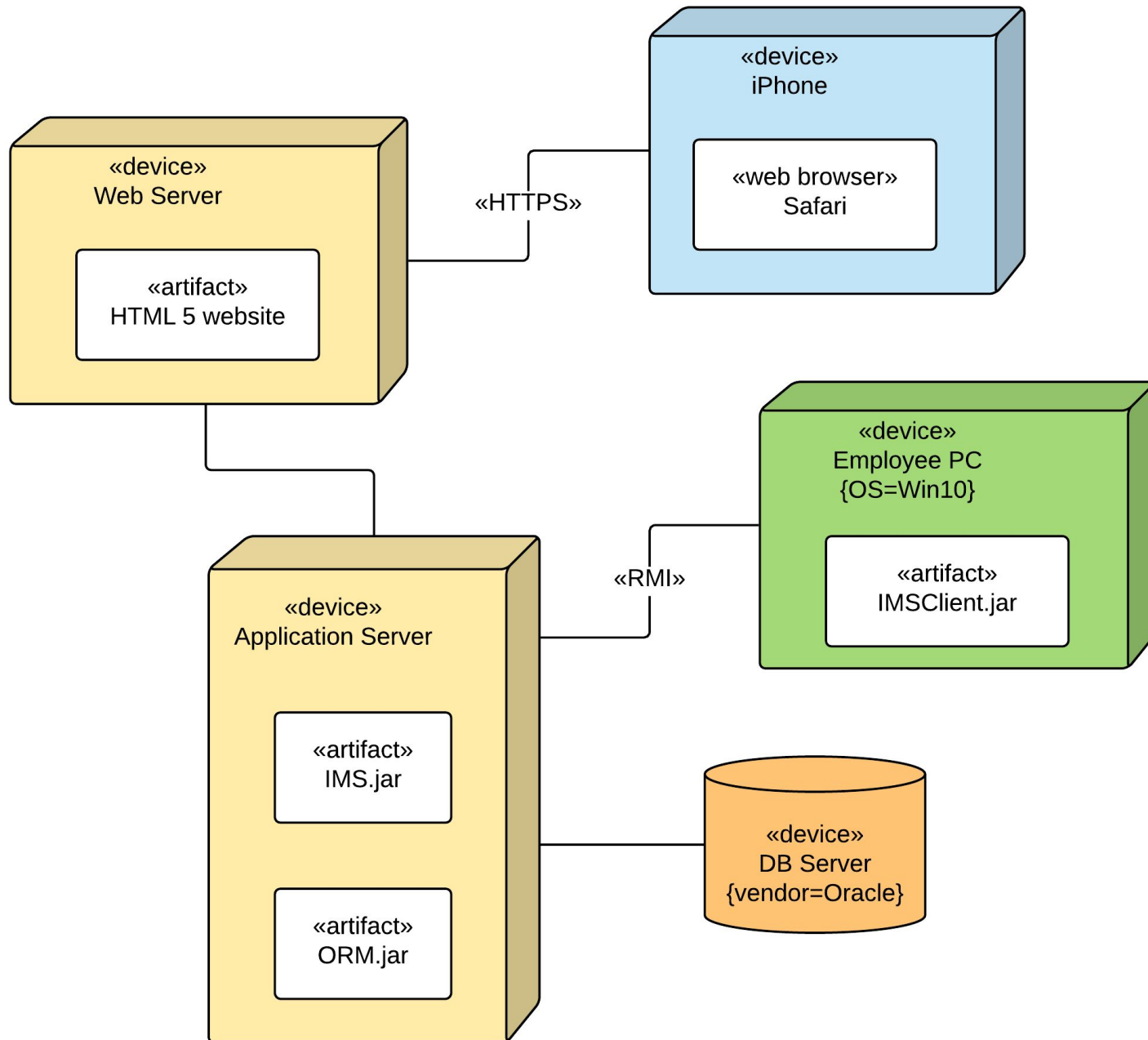
Some provided interfaces or interface requirements are exported via **ports**; others remain private to subcomponents of Store

Deployment Diagrams



UNIVERSITY OF LEEDS

- Specify the physical environment in which a software system will execute, and how the software is deployed in that environment
- Environment consists of **nodes** – which can be physical hardware or an ‘execution environment’ of some kind (browser, VM, etc)
- Nodes are shown as ‘3D boxes’ or as stereotyped icons that resemble the actual hardware
- Components and other model elements manifest as physical **artifacts** (.exe files, JAR files, etc), which are deployed on nodes



Summary



UNIVERSITY OF LEEDS

We have

- Considered why UML is helpful and given an agile perspective on how it can be used in software projects
- Discussed key elements of class diagram syntax
- Noted that the logical grouping of classes can be showing using a package diagram
- Explored the representation of high-level structure via component diagrams
- Seen that components manifest themselves physically as artifacts on a deployment diagram

Follow-Up / Further Reading

- Briefly examine the formal specification for UML 2.5.1 (downloadable from Minerva) – just to get a feel for UML's size and complexity
- Explore Scott Ambler's [Agile Modeling](#) site for more on UML diagrams and how UML can be used in development
- Do [Exercise 1](#), on diagramming tools