COMP2059 Developing Maintainable Software

LECTURE 08 – OBJECT ORIENTATED ANALYSIS/DESIGN WITH UML

Boon Giin Lee (Bryan)

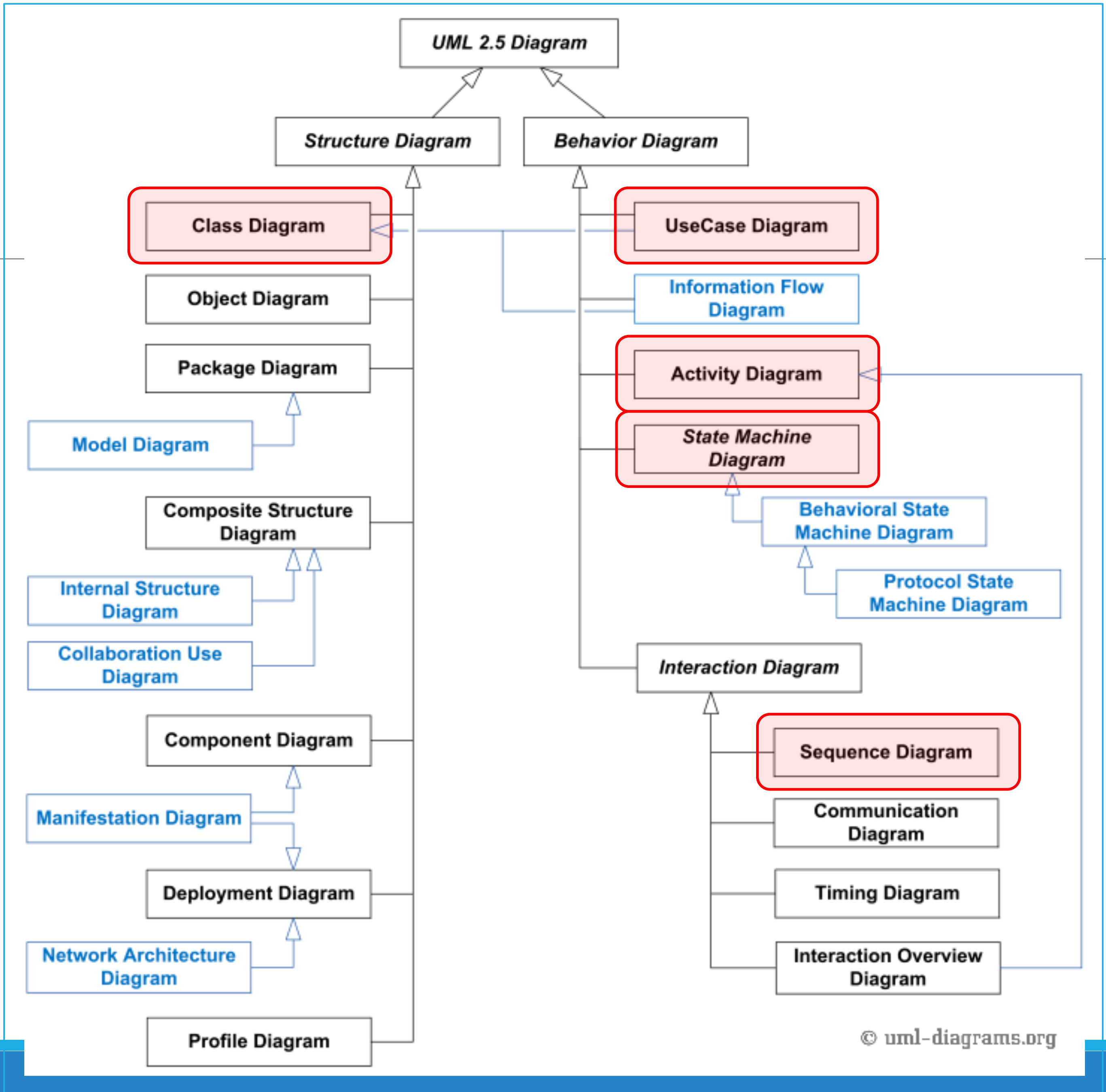
Unified Modelling Language

UML

UML (Unified Modelling Language)

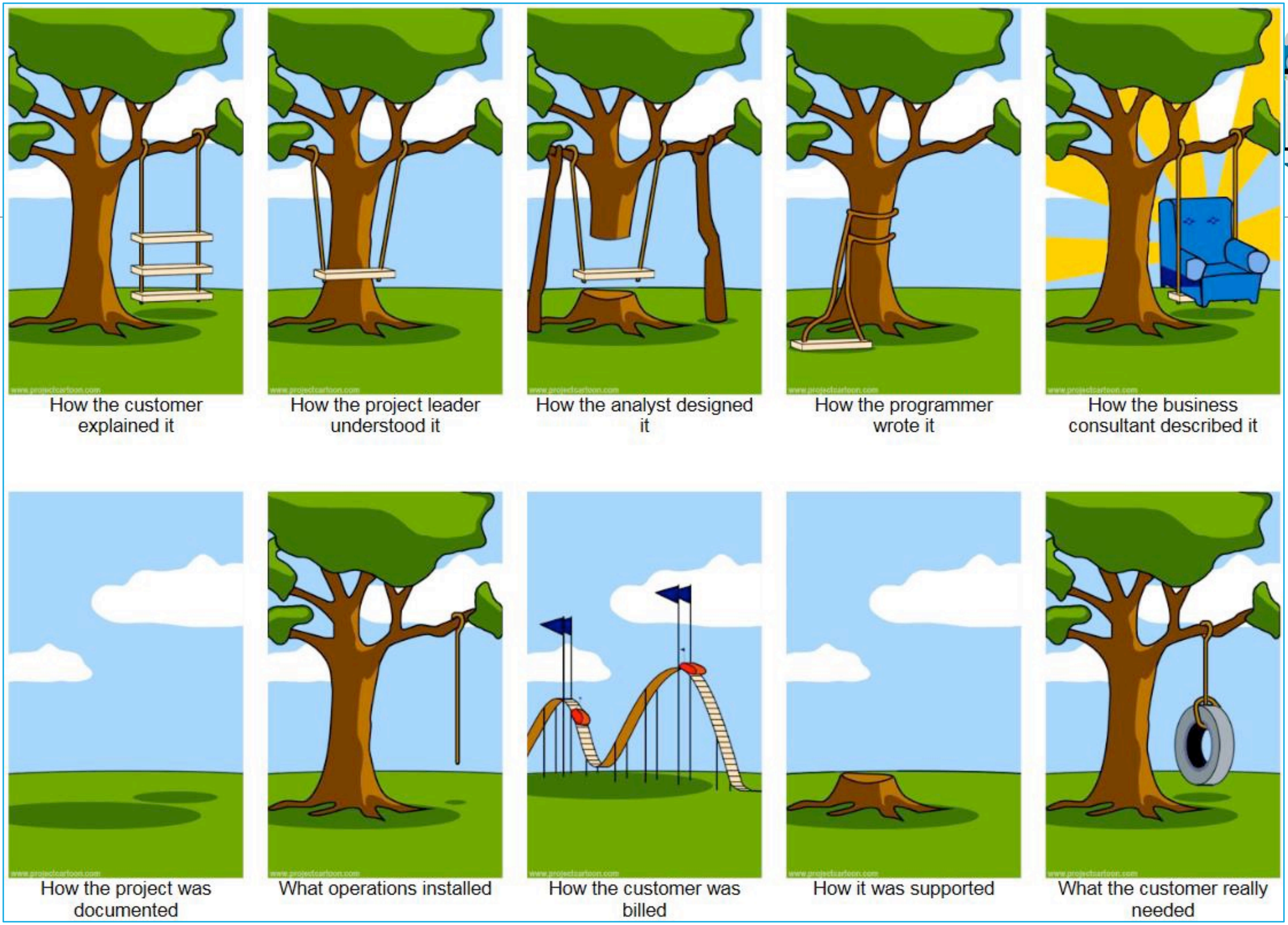
- UML: “A specification defining a graphical language for visualizing, specifying, constructing, and documenting the artifacts of distributed object systems.”
- Latest version: 2.5.1 (Dec 2017)
  - https://www.uml-diagrams.org/
  - https://www.omg.org/spec/UML/About-UML/

UNIFIED
MODELING
LANGUAGE TM


UML 2.5 Diagram
Structure Diagram
Behavior Diagram
Class Diagram
Object Diagram
Package Diagram
Model Diagram
Composite Structure Diagram
Internal Structure Diagram
Collaboration Use Diagram
Component Diagram
Manifestation Diagram
Deployment Diagram
Network Architecture Diagram
Profile Diagram
UseCase Diagram
Information Flow Diagram
Activity Diagram
State Machine Diagram
Behavioral State Machine Diagram
Protocol State Machine Diagram
Interaction Diagram
Sequence Diagram
Communication Diagram
Timing Diagram
Interaction Overview Diagram
© uml-diagrams.org

Why UML?

- Advantages of using UML


How the customer explained it
How the project leader understood it
How the analyst designed it
How the programmer wrote it
How the business consultant described it
How the project was documented
What operations installed
How the customer was billed
How it was supported
What the customer really needed

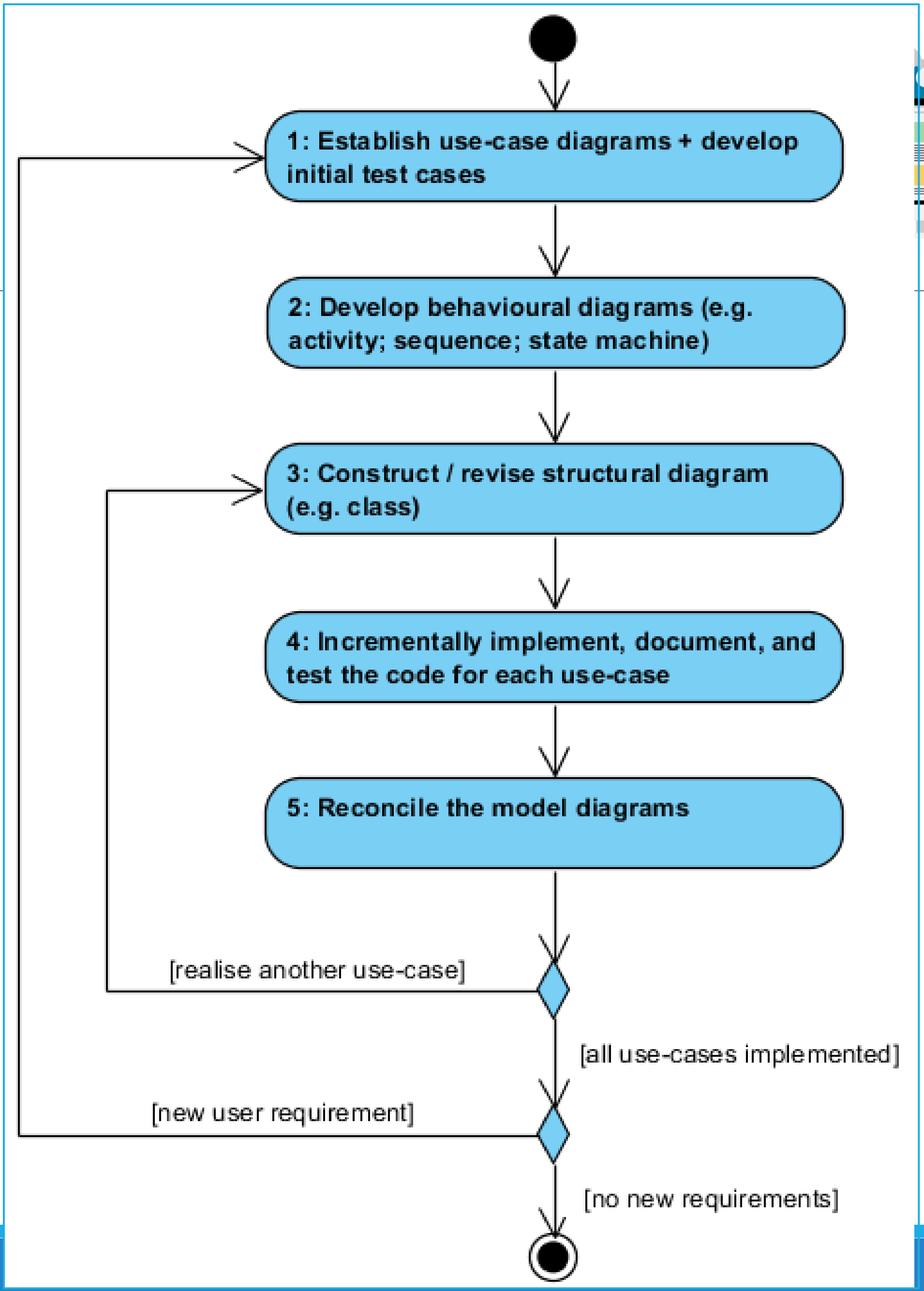
Why UML?

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

Object Oriented Analysis/Design Process

OOA/D PROCESS

“Use Case Driven” OOA/D Process


1: Establish use-case diagrams + develop initial test cases
2: Develop behavioural diagrams (e.g. activity; sequence; state machine)
3: Construct / revise structural diagram (e.g. class)
4: Incrementally implement, document, and test the code for each use-case
5: Reconcile the model diagrams
[realise another use-case]
[all use-cases implemented]
[new user requirement]
[no new requirements]

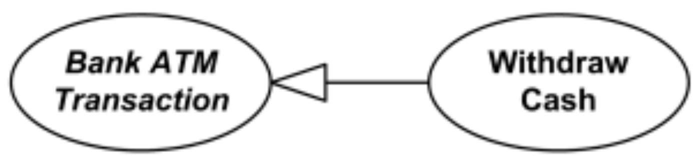
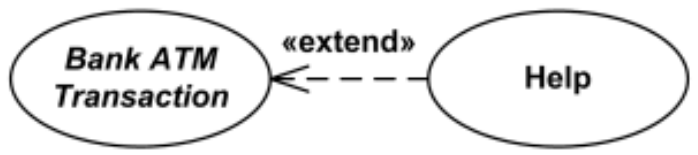
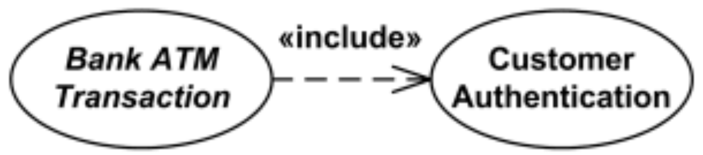
[after Barclay and Savage 2004]

Object Oriented Analysis

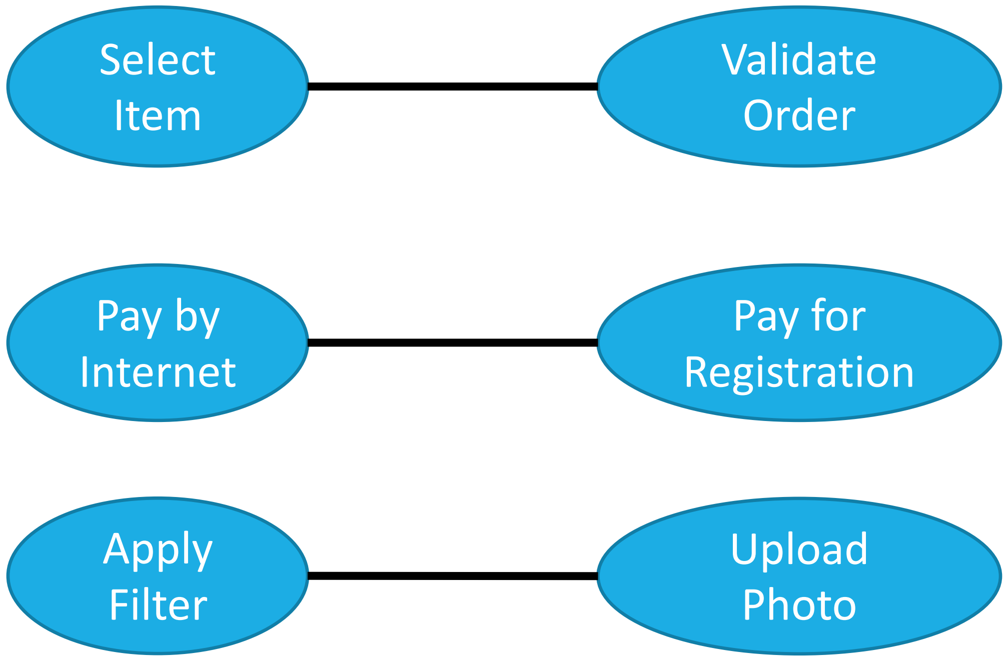
Use Case Diagrams

- Use case diagrams
  - Behavior 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

Generalization	Extend	Include
		 Bank ATM Transaction «include» Customer Authentication
Base use case could be **abstract use case** (incomplete) or concrete (complete).	Base use case is complete (concrete) by itself, defined independently.	Base use case is incomplete (**abstract use case**).
Specialized use case is required, not optional, if base use case is abstract.	Extending use case is optional, supplementary.	Included use case required, not optional.
No explicit location to use specialization.	Has at least one explicit extension location.	No explicit inclusion location but is included at some location.
No explicit condition to use specialization.	Could have optional extension condition.	No explicit inclusion condition.

Self-Test 1

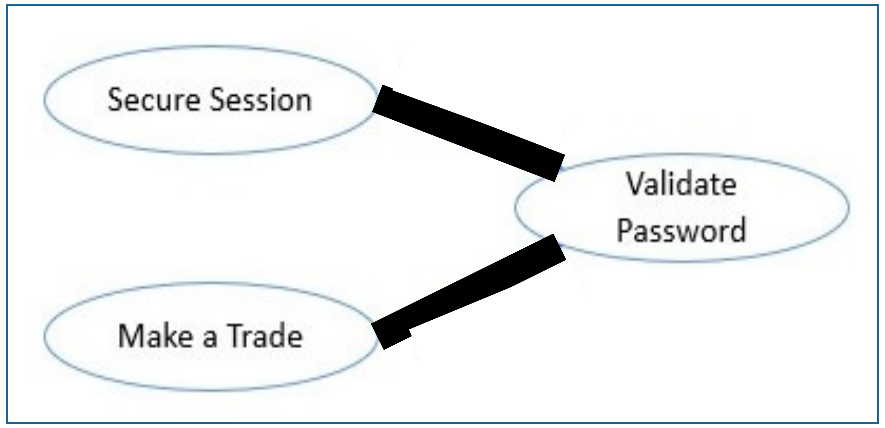


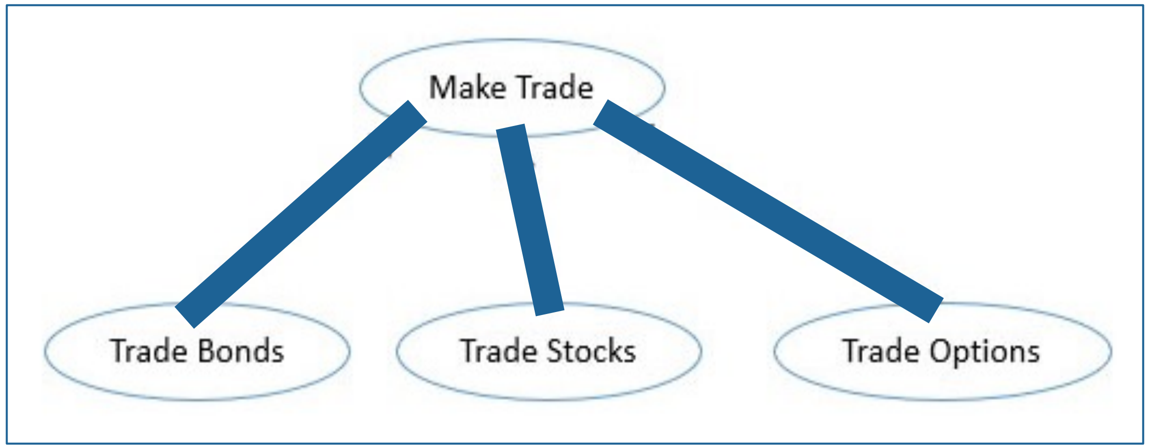
<<include>>

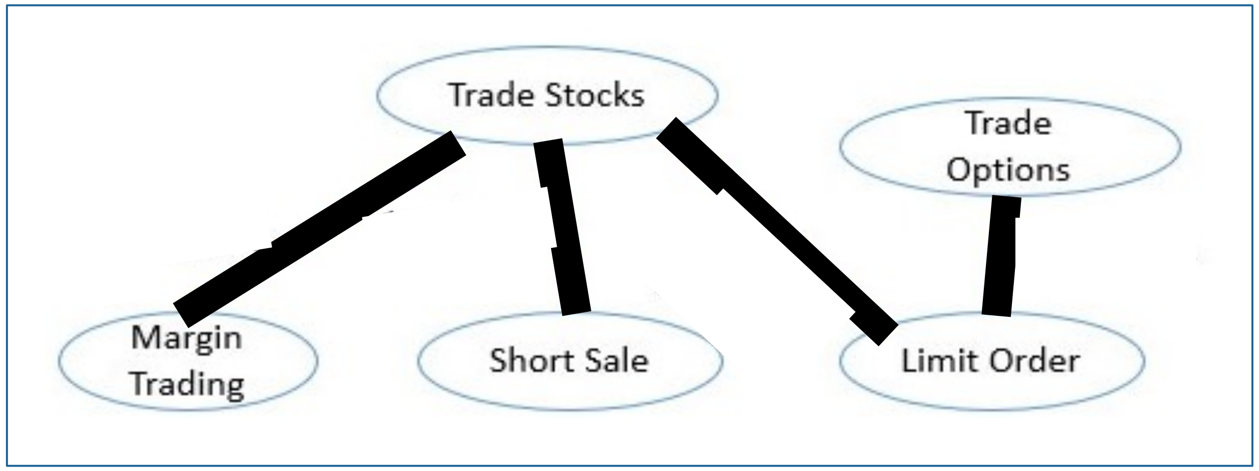
Generalization

<<extend>>

Self-Test 2


Secure Session
Validate Password
Make a Trade


Make Trade
Trade Bonds
Trade Stocks
Trade Options

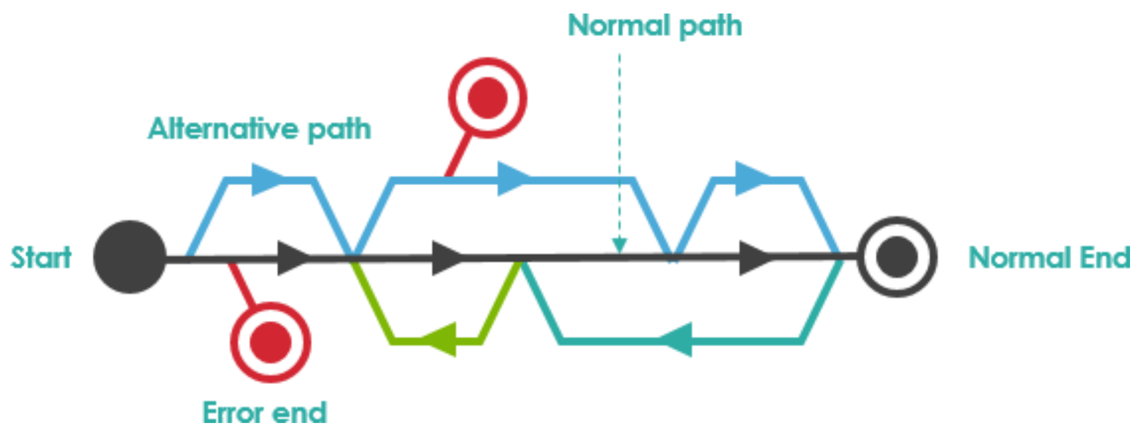

Trade Stocks
Trade Options
Margin Trading
Short Sale
Limit Order

Use Case Specification

- Use case specification elements
  - Use case name.
  - Use case purpose.
  - Pre-conditions(s).
  - Base path (optimistic flow).
  - Alternative paths (pragmatic flows).
  - Post-condition(s).

Use Case Specification

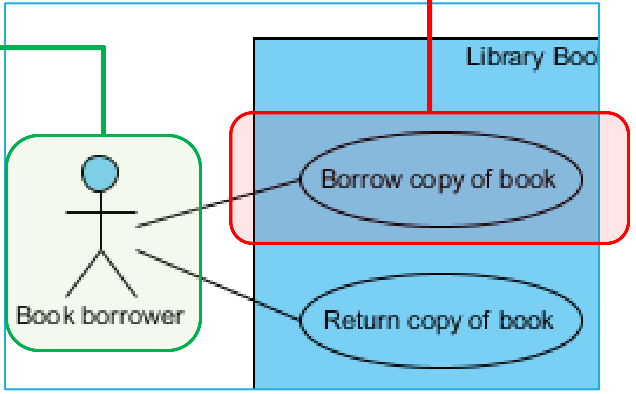
- Base and alternative path:
  - Normal Path (optimistic flow)
    - "Happy Day" scenario.
  - Alternative paths (pragmatic flows)
    - Every other possible way the system can be used.
    - Includes perfectly normal alternative use, but also errors and failures.



Extra: The <<include>> and <<extend>> Relationship in Use Case Models

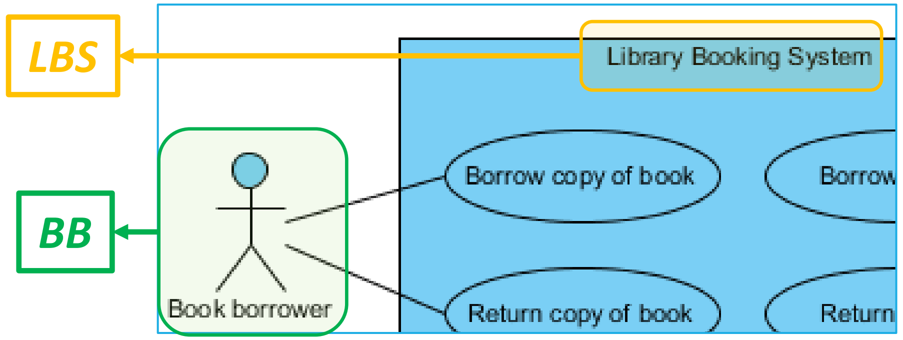
Example: Library Booking System

- Use Case: Borrow copy of book.
  - Purpose:
    - The book borrower borrows a book from the library using the Library Booking System.
  - Pre-condition(s):
    - The book must exist.
    - The book must be available.



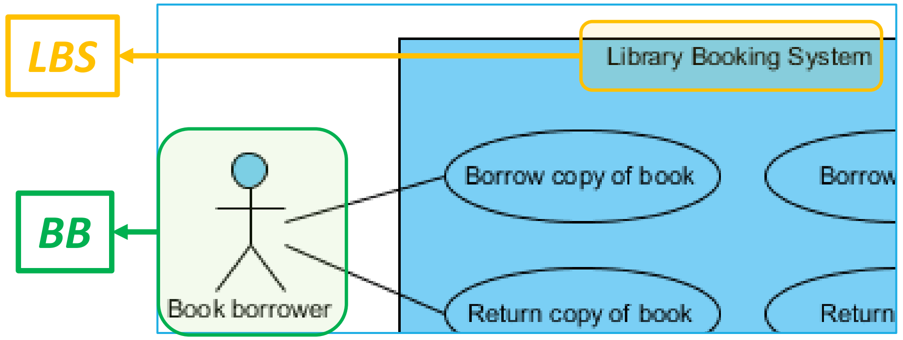
Example: Library Booking System

- Use Case: Borrow copy of book.
  - Base path (optimistic flow)
    1. LBS requests membership card.
    2. BB provides membership card.
    3. BB is logged in by LBS.
    4. LBS checks permissions / debts.
    5. BB presents a book.
    6. LBS scans RFID tag inside book.
    7. LBS updates records accordingly.
    8. LBS disables anti-theft device.
    9. BB is logged out by LBS.
    10. LBS confirms that process has been completed successfully.



Example: Library Booking System

- Use Case: Borrow copy of book.
  - Alternative paths (pragmatic flow)
    - BB's card has expired: Step 3a,
      - LBS must provide a message that card has expired;
      - LBS must exit the use case
    - LBS cannot read membership card: Step 3a,
      - LBS must provide a message that card could not be read correctly;
      - LBS must go back to Step 1.
  - Post-condition(s):
    - The member has successfully borrowed the book.
    - The system is up to date.



https://www.visual-paradigm.com/guide/use-case/what-is-use-case-specification/

Example: ATM Withdraw Case

Use Case Name:	Withdraw Cash
Actor(s):	Customer (primary), Banking System (secondary)
Summary Description:	Allows any bank customer to withdraw cash from their bank account.
Priority:	Must Have
Status:	Medium Level of details
Pre-Condition:	• The bank customer has a card to insert into the ATM • The ATM is online properly
Post-Condition(s):	• The bank customer has received their cash (and optionally a receipt) • The bank has debited the customer's bank account and recorded details of the transaction

Example: ATM Withdraw Case

Basic or Normal Path:	1. The customer enters their card into the ATM 2. The ATM verifies that the card is a valid bank card 3. The ATM requests a PIN code 4. The customer enters their PIN code 5. The ATM validates the bank card against the PIN code 6. The ATM presents service options including "Withdraw" 7. The customer chooses "Withdraw" 8. The ATM presents options for amounts 9. The customer selects an amount or enters an amount 10. The ATM verifies that it has enough cash in its hopper 11. The ATM verifies that the customer is below withdraw limits 12. The ATM verifies sufficient funds in the customer's bank account 13. The ATM debits the customer's bank account 14. The ATM returns the customer's bank card 15. The customer takes their bank card 16. The ATM issues the customer's cash 17. The customer takes their cash

Example: ATM Withdraw Case

Alternative Paths:	2a. Invalid card 2b. Card upside down 5a. Stolen card 5b. PIN invalid 10a. Insufficient cash in the hopper 10b. Wrong denomination of cash in the hopper 11a. Withdrawal above withdraw limits 12a. Insufficient funds in customer's bank account 14a. Bank card stuck in machine 15a. Customer fails to take their bank card 16a. Cash stuck in machine 17a. Customer fails to take their cash • ATM cannot communicate with Banking System • Customer does not respond to ATM prompt

Example: ATM Withdraw Case

Business Rules:	B1: Format of PIN B2: Number of PIN retries B3: Service options B4: Amount options B5: Withdraw limit B6: card must be taken away before dispense of cash
Non-Functional Requirements:	NF1: Time for complete transaction NF2: Security for PIN entry NF3: Time to allow collection of card and cash NF4: Language support NF5: Blind and partially blind support

University of Nottingham
UK | CHINA | MALAYSIA

More Examples

- Team Obiwan – Use Case Specification Project Phase 2
- End-to-End UML: Use Case Specification

Object Oriented Design

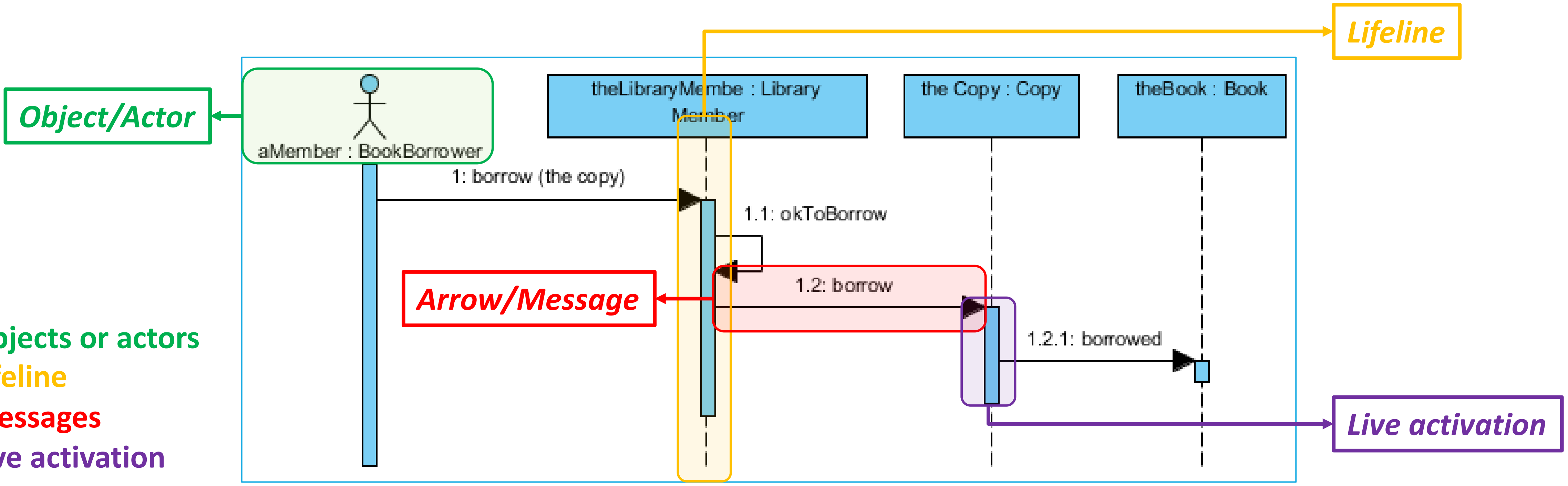
Sequence Diagrams

- 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 lifeline shows a **live activation** of the object.

Example: Library Booking System

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



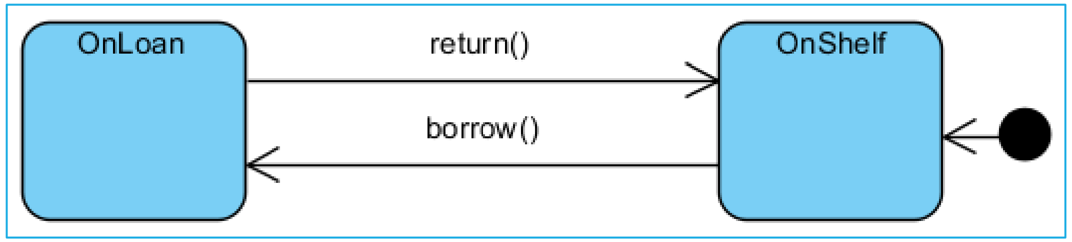
Objects or actors
Lifeline
Messages
Live activation

State Machine Diagrams

- To implement a class, one needs 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

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

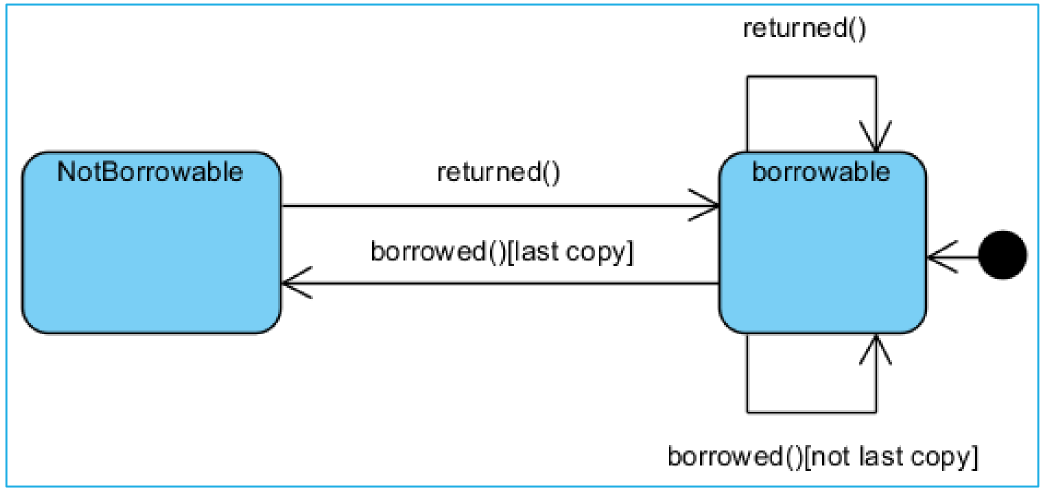


State machine diagram for "Copy".

- 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.

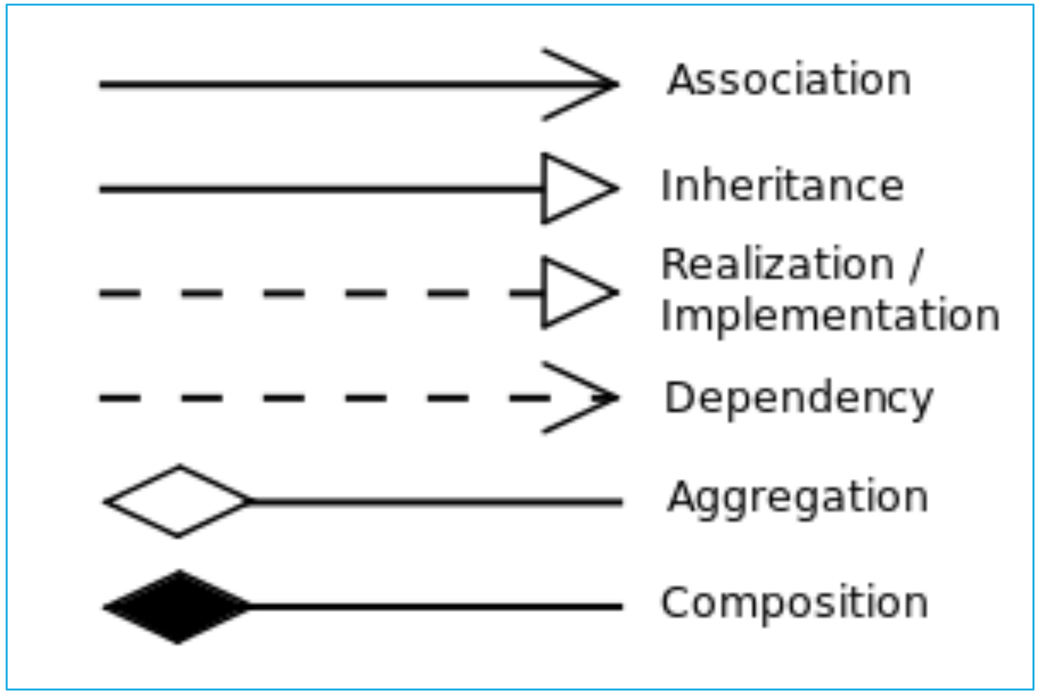
State Machine Diagrams

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



Class Diagrams

- Class diagrams show the existence of **classes**, their **structures** and **relationships** in the logical view of a system.
- Class diagram components:
  - Classes (structure and behaviour).
  - Class relationships.
    - Association.
    - Dependency.
    - Aggregation.
    - Composition.
    - Realisation.
    - Generalisation / Inheritance.
  - Multiplicity and navigation indicators.



Class Diagrams

- What makes a class model good?
  - Able to build a system quickly and cheaply to the satisfaction of the client.
  - Able to build a system that is easy to maintain and easy to extend.

- 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, or 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

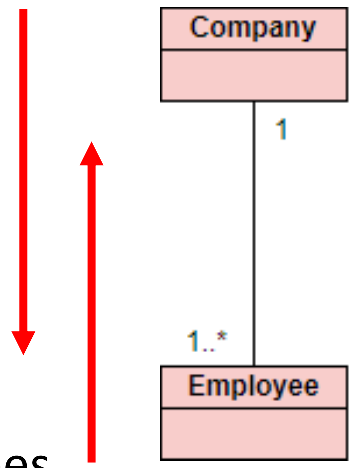
- Associations between classes.
  - Correspond to verbs.
  - Real world association that can be described by a short sentence (reader borrows a book).
  - Classes are associated if some object of class A must know about some object of class B or vice versa.
- Multiplicity.
  - Number of links between each instance of the source class and instances of the target class.

1	Exactly 1
*	Unlimited number (0 or more)
0 .. *	0 or more

1 .. *	1 or more
0 .. 1	0 or 1
3 .. 7	3 or 7

Class Diagrams

Each Employee belongs to a Company.

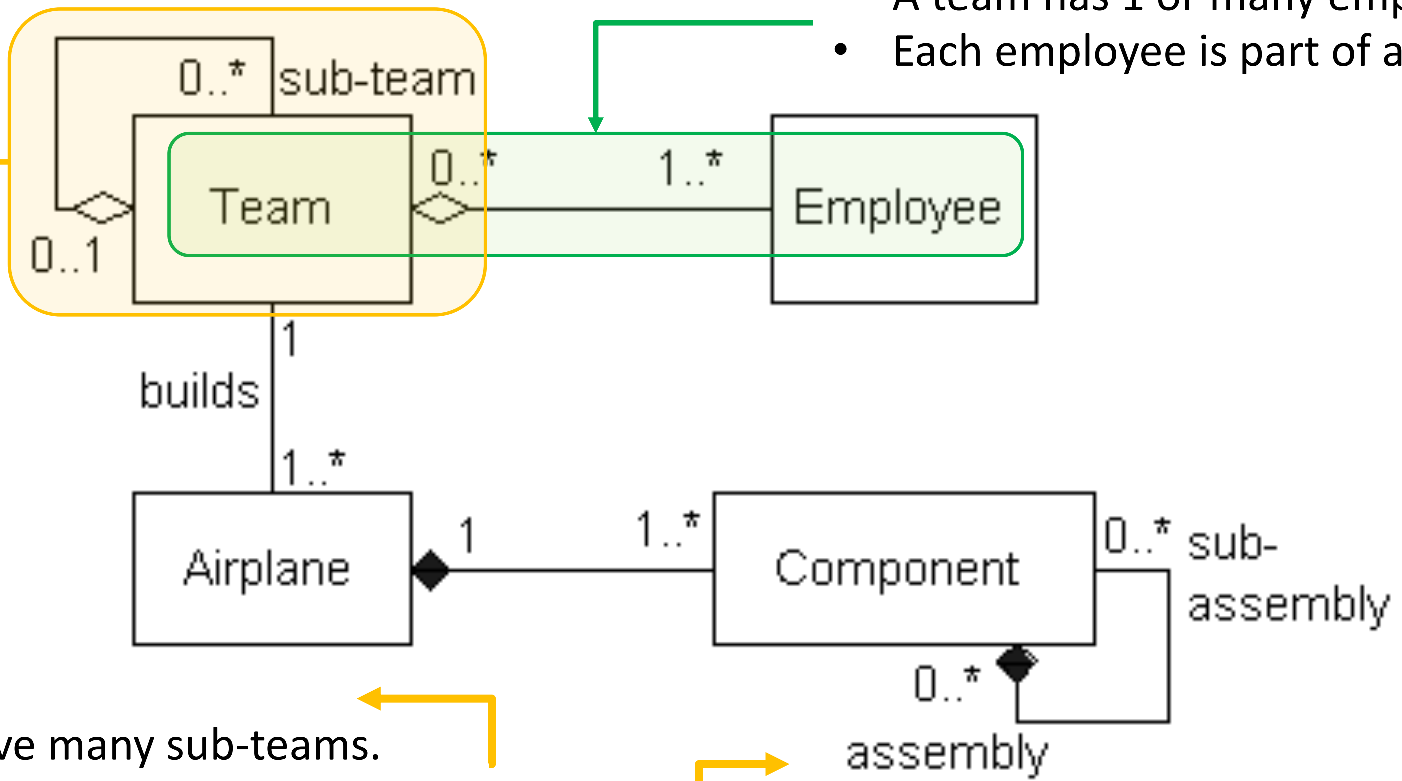


A Company has 1 or many Employees.

Multiplicities examples:

1	Exactly one, no more and no less
0..1	Zero or one
*	Many
0..*	Zero or many
1..*	One or many

Class Diagram



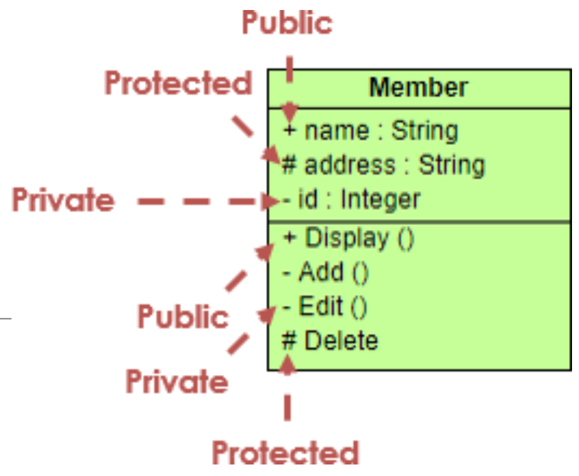
- A team has 1 or many employees.
- Each employee is part of at most 1 team.

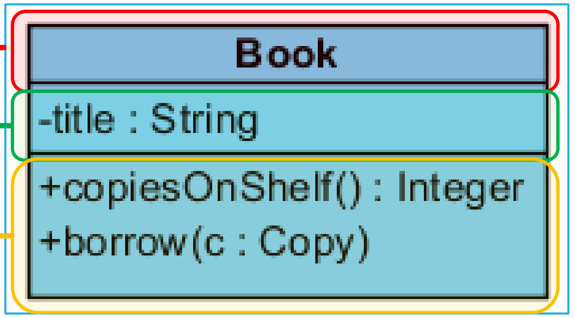
- A team can have many sub-teams.
- A team can be associated with at most one team.

Class Diagrams

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

+	Public
#	Protected
-	Private
/	Derived
$	Static





This is the record that keeps track of the books.

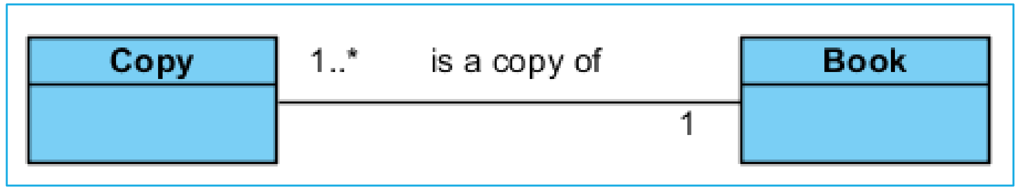
```
public class Book {
    private String title;

    public int copiesOnShelf() { }

    public void borrow(Copy c) { }
}
```

Class Diagrams

- Relationship: **Association**
  - This is the most general type of relationship.
  - It shows bi-directional connection between two classes.
  - It is a weak coupling as associated classes remain somewhat independent of each other.

This is the record that keeps track of the books.

Every Copy is associated only with one Book.



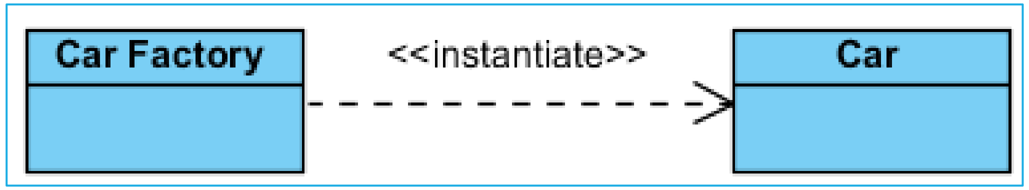
Every Book is associated with one or more Copies.

```
public class Copy {
    private Book books;
}
```

```
public class Book {
    private List<Copy> copy;
}
```

Class Diagrams

- Relationship: **Dependency**
  - A directed relationship which shows that an element or a set of elements require(s), need(s) or depend(s) on other elements for implementation.
  - It is a supplier-client relationship, where supplier provides something to the client, and thus the client is in some sense incomplete while semantically or structurally dependent on the supplier element(s).
  - Modification of the supplier may impact the client elements.



CarFactory class depends on the Car class.

```
public class CarFactory {
    private void manufactureCar(Car car) { }
}
```

University of Nottingham
UK | CHINA | MALAYSIA

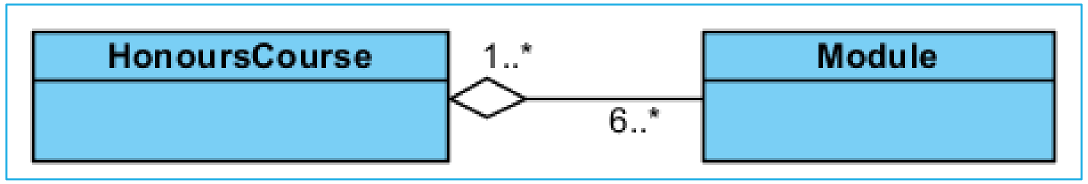
Class Diagrams

- Relationship: **Dependency**
  - Dependency indicates a "**uses**" relationship between two classes.
  - If a class A "uses" class B, then one or more of the following statements generally hold true:
    - Class B is used as the type of a local variable in one or more methods of class A.
    - Class B is used as the type of parameter for one or more methods of class A.
    - Class B is used as the return type for one or more methods of class A.
    - One or more methods of class A invoke one or more methods of class B.

Class Diagrams

- Relationship: **Aggregation ("is part of" relationship)**
  - This is special type of association.
  - 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.

Each HonoursCourse consists of 6 or more Modules.



Each Module could be part of one or more HonoursCourses

```
public class HonoursCourse {
    List<Module> modules;
}
```

```
public class Module {
    List<HonoursCourse> honoursCourseList;
}
```

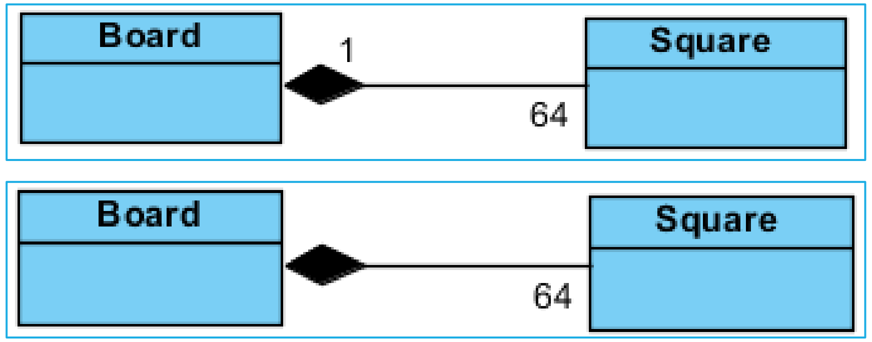
Class Diagrams

- How can we tell if a reference means **aggregation or association**?

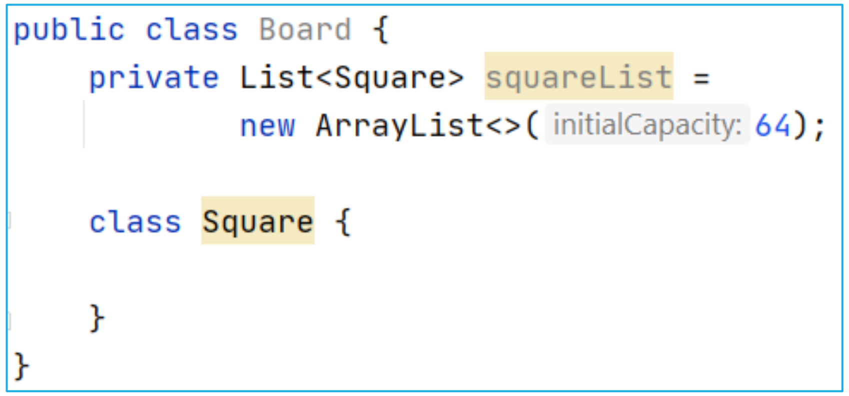
  - Well, we can’t.
  - The difference is only logical – whether one of the objects is part of the other or not.

Class Diagrams

- Relationship: **Composition**
  - This is a strong form of aggregation (physical containment).
  - The multiplicity at the composition end is always 1 as the parts have no meaning outside the whole.
  - It 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.



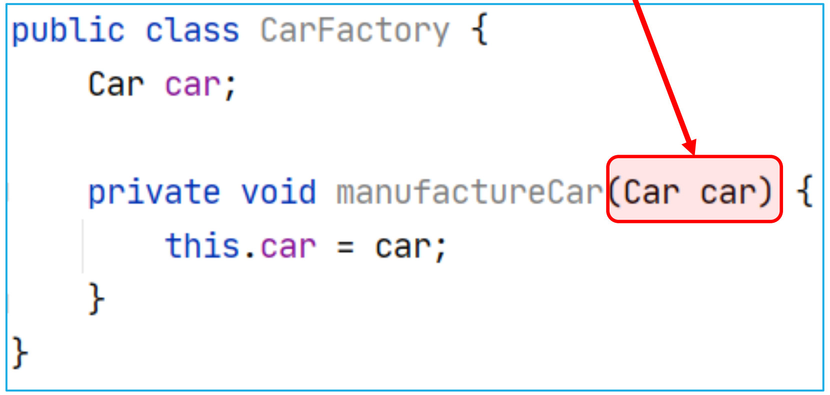
A board has 64 squares; and each square belongs to exactly one board.



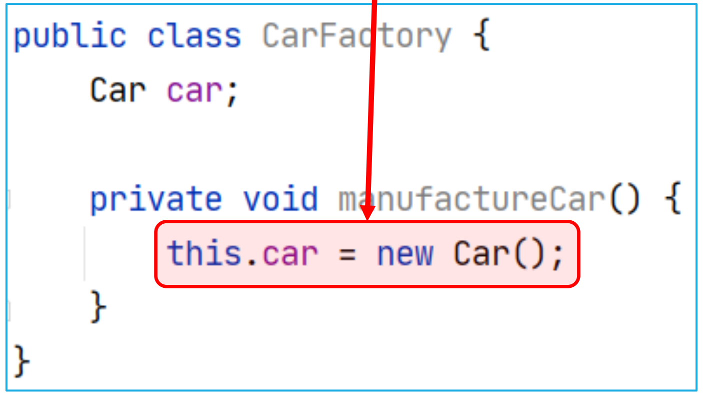
```
public class Board {
    private List<Square> squareList =
            new ArrayList<>( initialCapacity: 64);

    class Square {

    }
}
```

Class Diagrams

Aggregation	Composition
Weaker Association!	Stronger Association! Creating an object of a class `Car` inside class `CarFactory`.
Even if delete class `CarFactory`, car will exist outside (`car` is created outside and passed to class `CarFactory`).	If delete class `CarFactory`, car won't exist (object `car` is created inside `CarFactory` only).
e.g., `Person` has `Car` but `Person` and `Car` exist independently.	e.g., `Liver` can't exist outside `Body`.



```java
public class CarFactory {
    Car car;

    private void manufactureCar(Car car) {
        this.car = car;
    }
}
```



```java
public class CarFactory {
    Car car;

    private void manufactureCar() {
        this.car = new Car();
    }
}
```

University of Nottingham
UK | CHINA | MALAYSIA

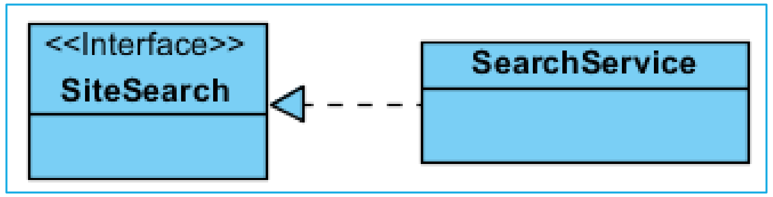
Class Diagrams

- `Automobile` (Parent) and `Car` (Child)
  - If delete the `Automobile`, the child `Car` still exist: Aggregation

- `House` (Parent) and `Room` (Child)
  - `Rooms` will never separate into a `house`: Composition

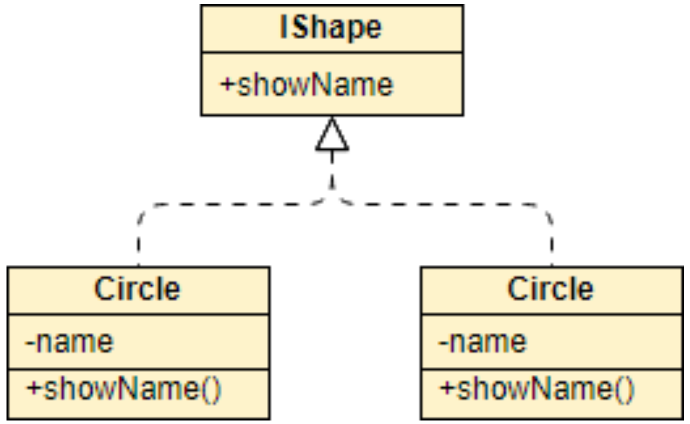
Class Diagrams

- Relationship: **Realisation**
  - A "Realisation" is a specialised abstraction relationship between to sets of model elements, one representing a specification (the supplier), and the other representing an implementation of the latter (the client).



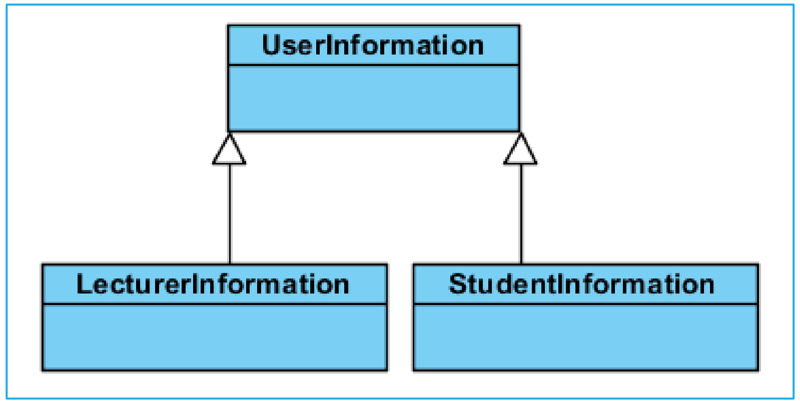
Interface SiteSearch is realized (implemented) by SearchService

```
public class SearchService implements SiteSearch { }
```



Class Diagrams

- Relationship: **Generalisation (“is a” relationship) > Inheritance**
  - A directed relationship between a more general classifier (superclass) and a more specific classifier (subclass).



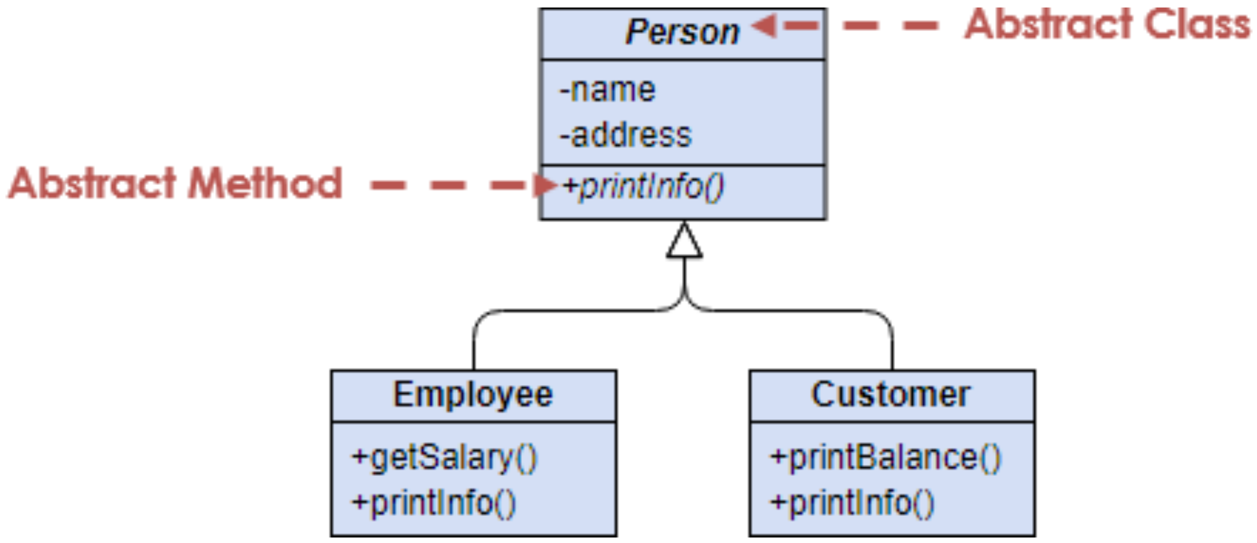
LecturerInformation and StudentInformation are generalised by UserInformation.

```
public class LecturerInformation extends UserInformation { }
```

```
public class StudentInformation extends UserInformation { }
```

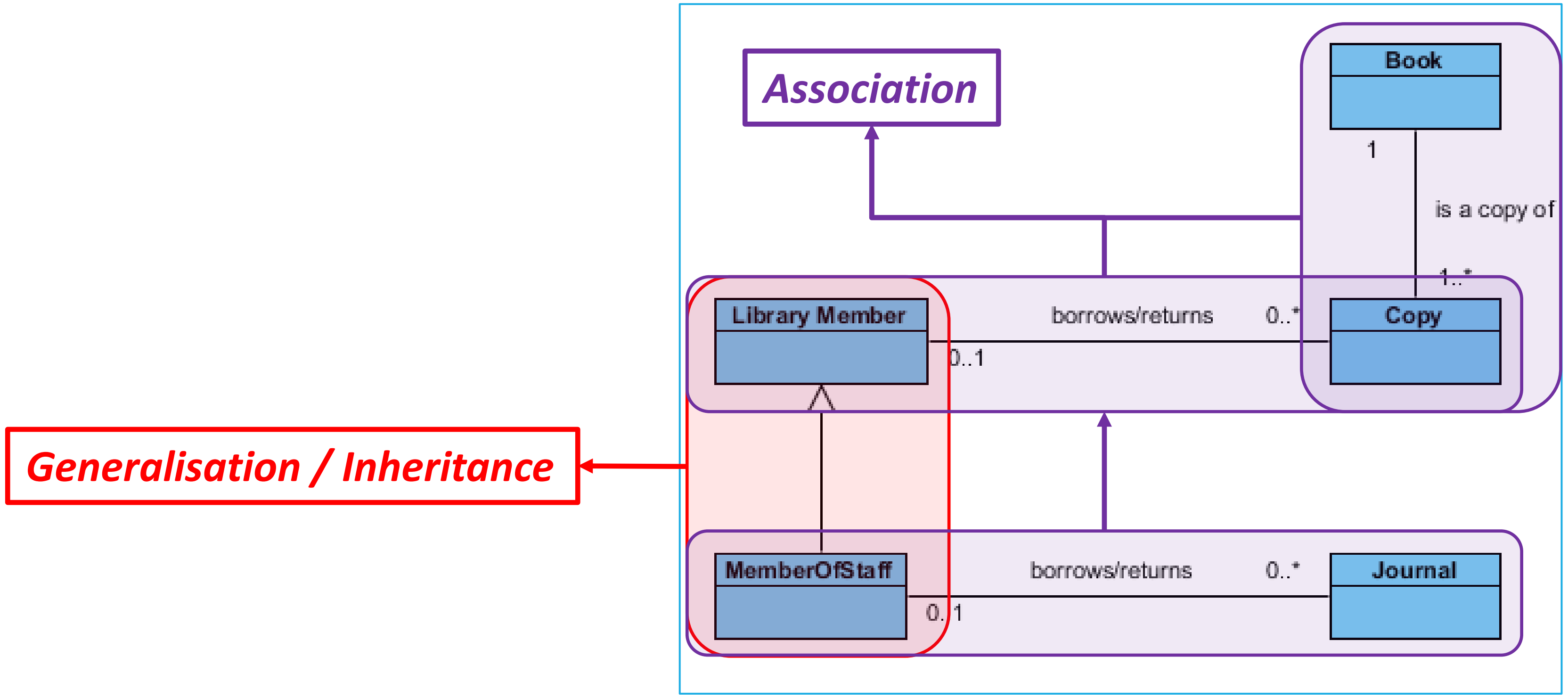
Class Diagrams

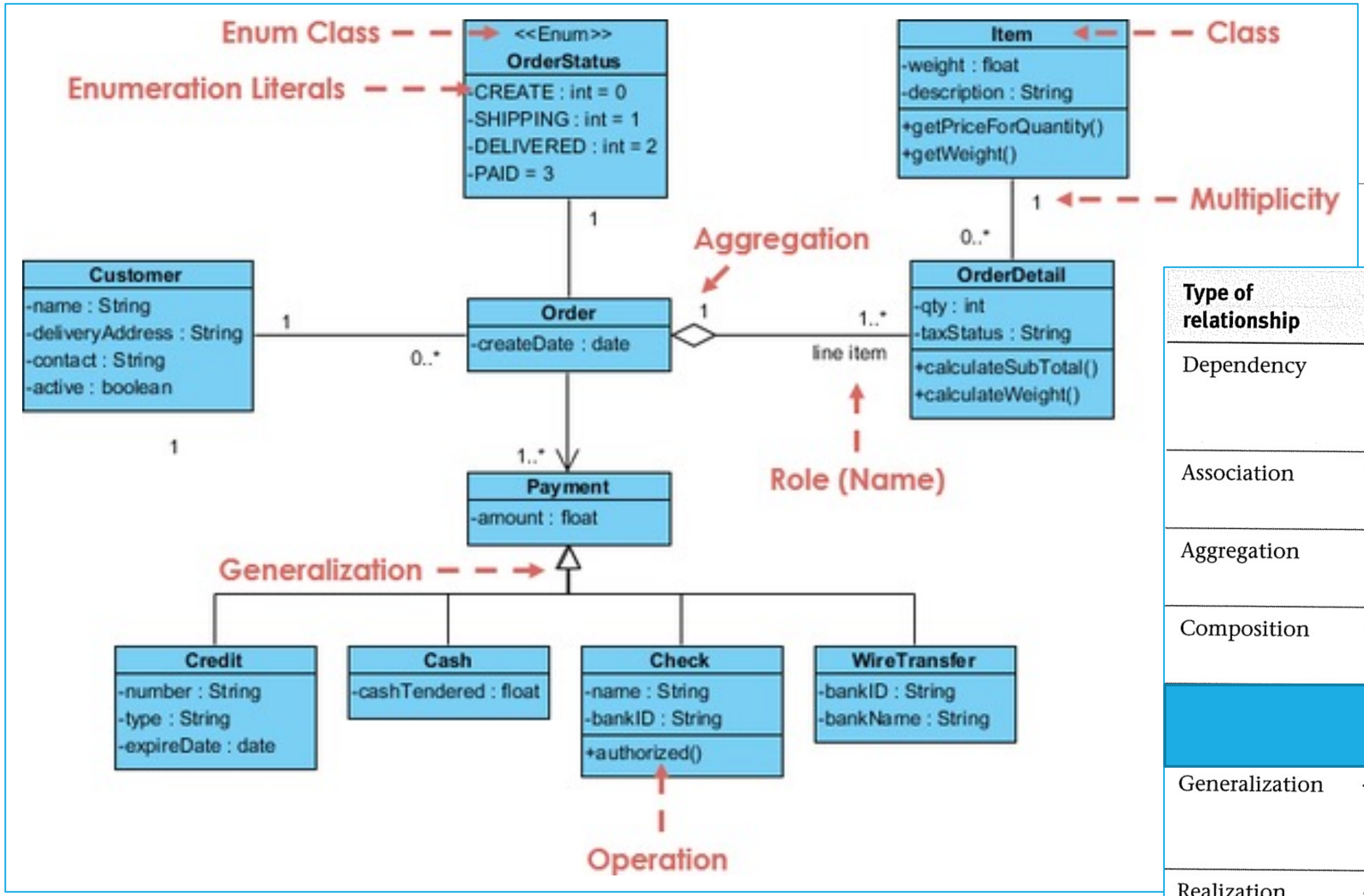
- Relationship: **Generalisation ("is a" relationship) > Inheritance**
  - The name of an **abstract class** is typically shown in italics.
  - An abstract method is a method that **do not have implementation**.

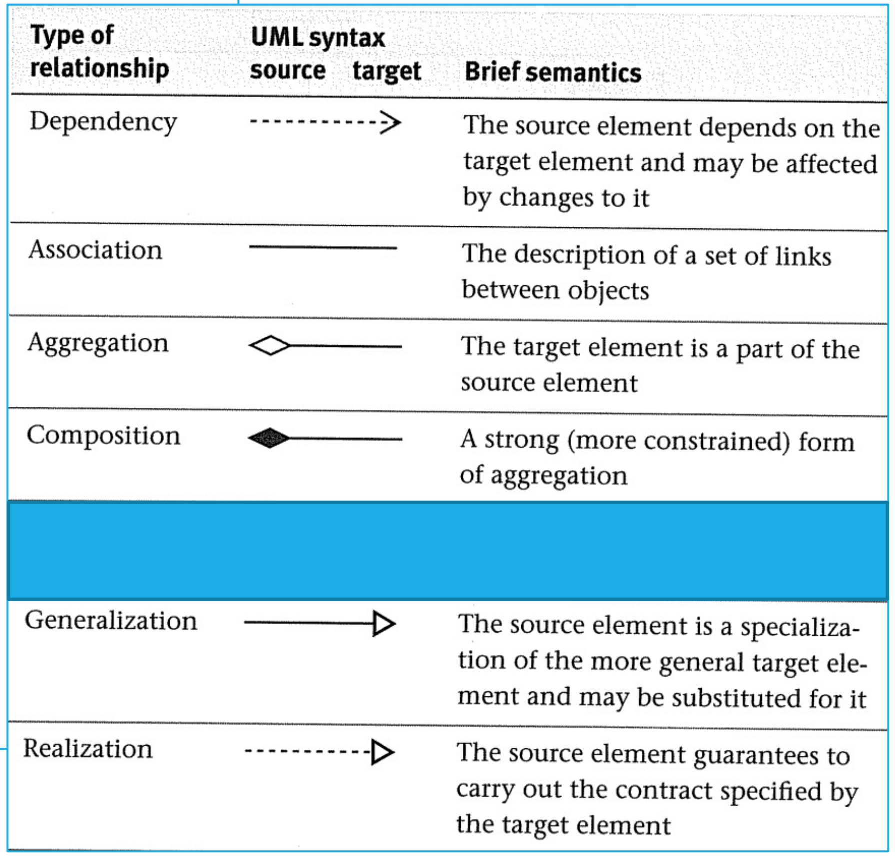


Example: Library Booking System

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



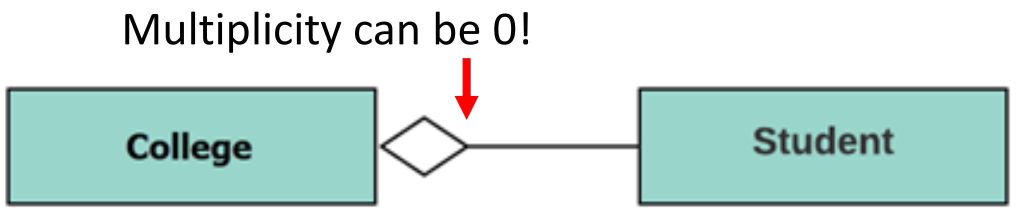




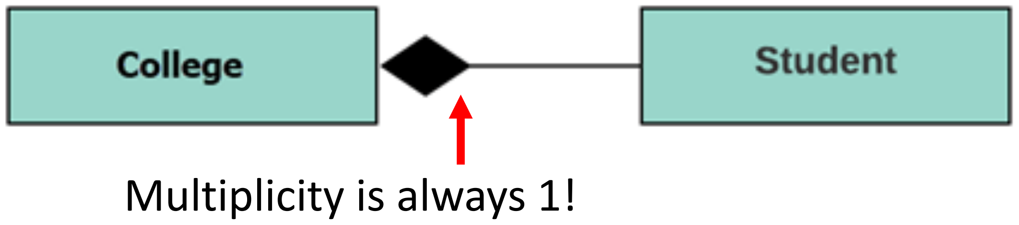
Type of relationship	UML syntax source target	Brief semantics
Dependency	- - - - - - - - - ->	The source element depends on the target element and may be affected by changes to it
Association	————	The description of a set of links between objects
Aggregation	◇————	The target element is a part of the source element
Composition	◆————	A strong (more constrained) form of aggregation
Generalization	————▷	The source element is a specialization of the more general target element and may be substituted for it
Realization	- - - - - - - - - -▷	The source element guarantees to carry out the contract specified by the target element

Sometimes …

- The class college is made up of one or more student.
  - In aggregation, the contained classes are never totally dependent on the lifecycle of the container.
  - The college class will remain even if the student is not available.



- College is composed of classes student.
  - The college could contain many students, while each student belongs to only one college.
  - If college is not functioning all the students also removed.



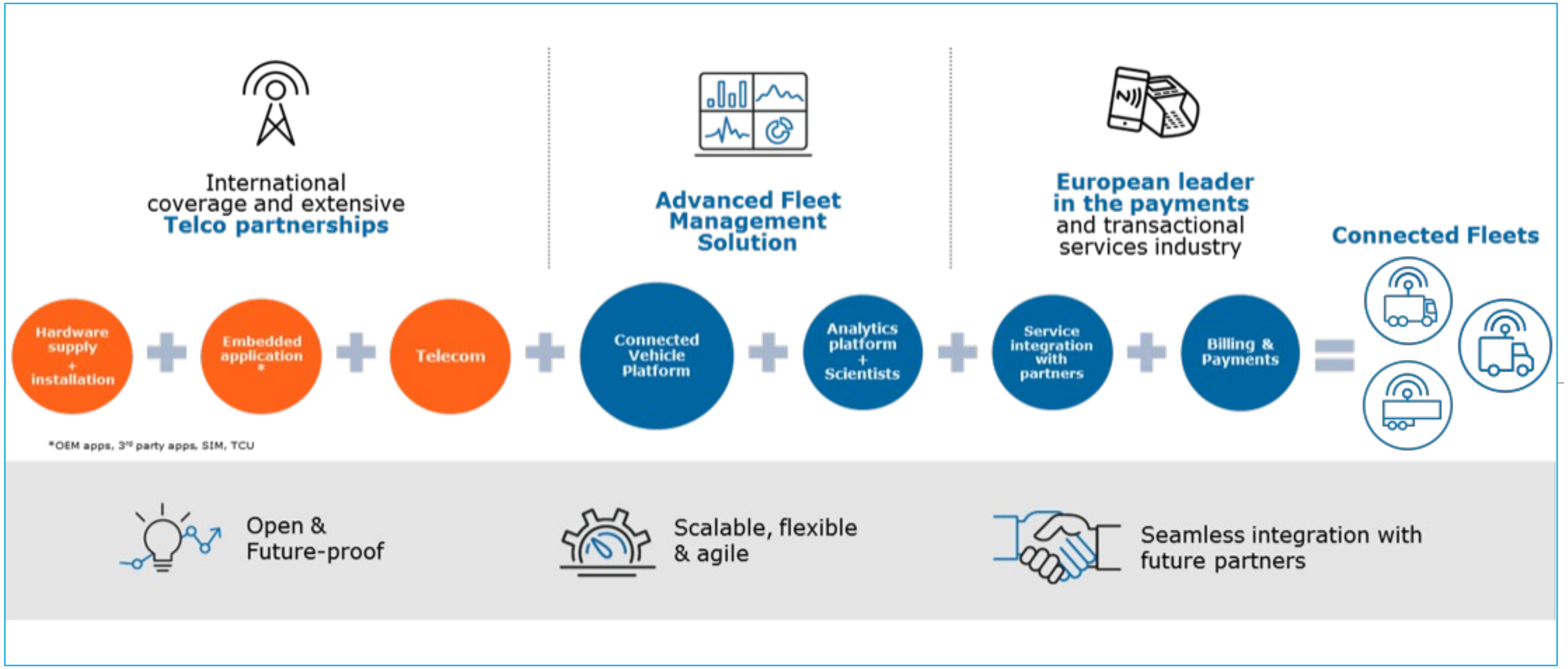
When & Why Draw Class Diagram?

- Most UML diagrams do not have a direct counterpart in object-oriented programming languages, except for class diagrams. In essence, class diagrams can ideally be mapped one-to-one with UML class diagrams.
- Class diagrams prove to be valuable in the following scenarios:
  - Illustrating the system's static view.
  - Representing the interactions among the elements of static view.
  - Documenting & describing the system's functionalities.
  - Developing software applications using object-oriented languages.
  - Performing code forward engineering for the target systems.
  - Categorizing classes or components as reusable libraries for future purposes.

Case Study

FLEET LOGISTIC MANAGEMENT SYSTEM

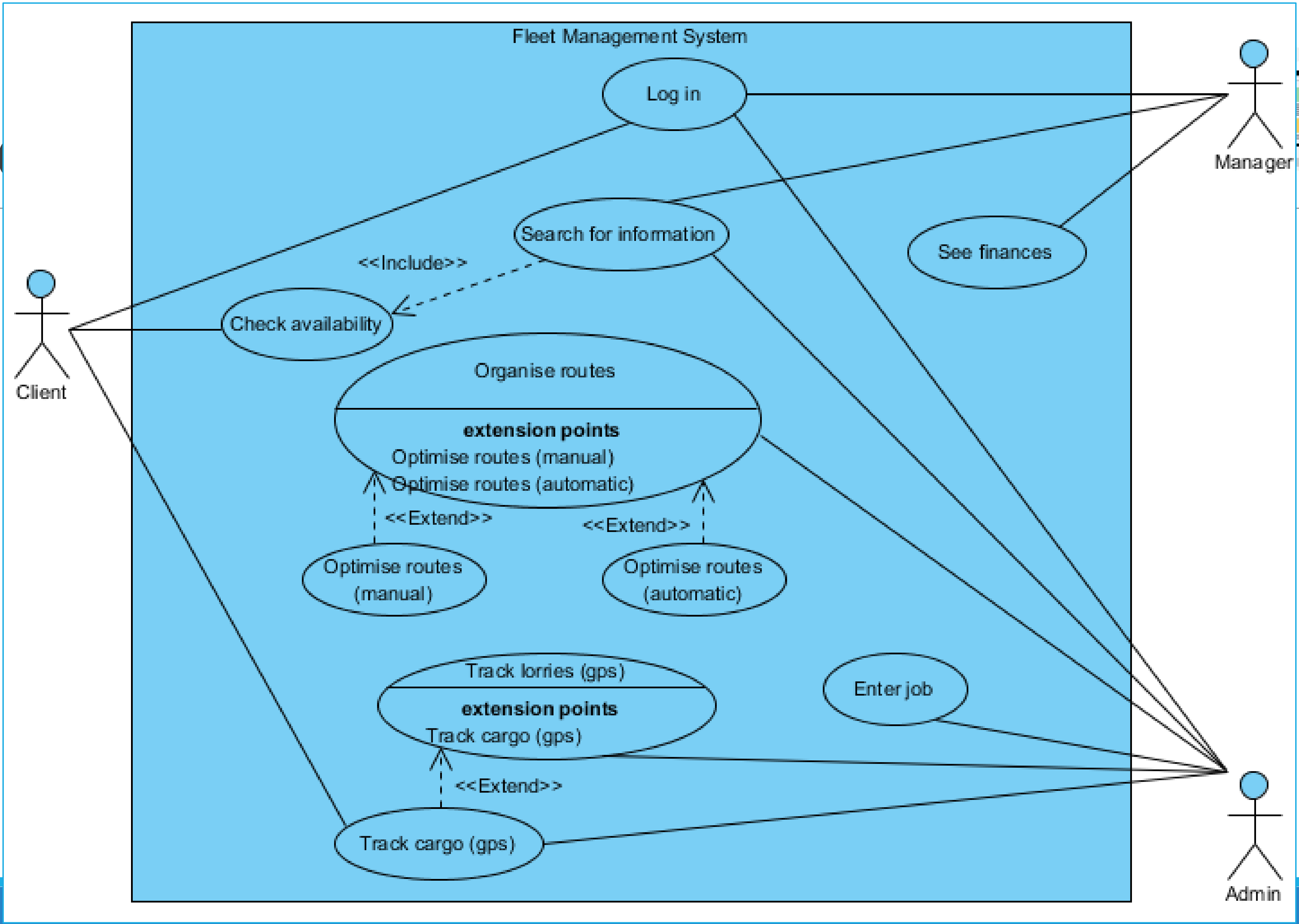
Fleet Logistics Management


International coverage and extensive Telco partnerships
Advanced Fleet Management Solution
European leader in the payments and transactional services industry
Connected Fleets
Hardware supply + installation
+
Embedded application *
+
Telecom
+
Connected Vehicle Platform
+
Analytics platform + Scientists
+
Service integration with partners
+
Billing & Payments
=
*OEM apps, 3rd party apps, SIM, TCU
Open & Future-proof
Scalable, flexible & agile
Seamless integration with future partners

Fleet Logistics Management

- User stories
  - As a client, I want to be able to check availability of lorries.
  - As a client, I want to be able to track cargo.
  - As a manager, I want to be able to see the finances.
  - As an administrator, I want to be able to search for information.
  - As an administrator, I want to be able to organise routes.
  - As an administrator, I want to be able to track lorries and cargo.

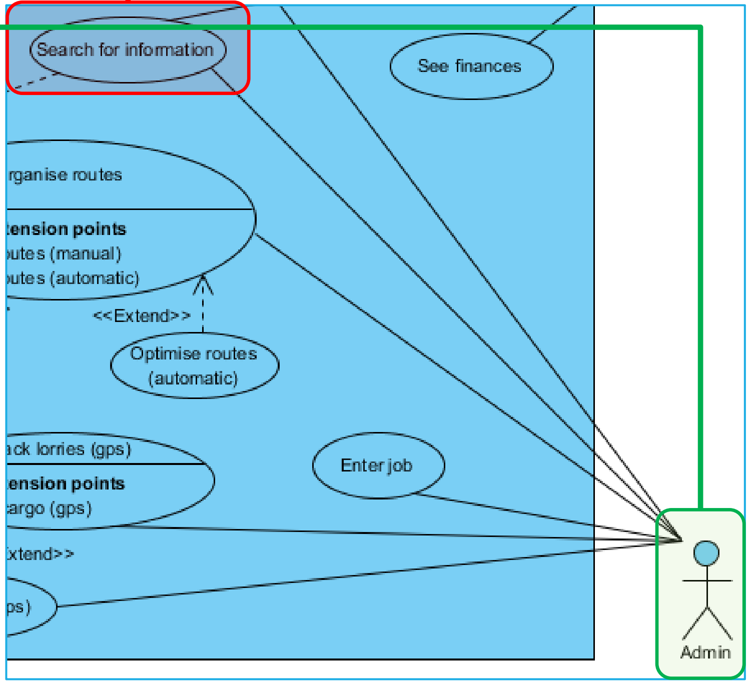
Fl



Fleet Management System
Log in
Search for information
See finances
<<Include>>
Check availability
Organise routes
extension points
Optimise routes (manual)
Optimise routes (automatic)
<<Extend>>
<<Extend>>
Optimise routes (manual)
Optimise routes (automatic)
Track lorries (gps)
extension points
Track cargo (gps)
<<Extend>>
Enter job
Track cargo (gps)
Client
Manager
Admin

Fleet Logistics Management

- Use Case: Search for information.
  - Purpose:
    - Administrator can search the database (DB) for any kind of information related to lorries and jobs.
  - Pre-condition(s):
    - Administrator must be logged in.



Fleet Logistics Management

- Use Case: Search for information.
  - Base path (optimistic flow)

1. Administrator opens search window.
2. Administrator defines query using query editor.
3. Administrator sends query to DB.
4. DB deals with query: finding results.
5. DB deals with query: organising them by relevance.
6. DB sends results back.
7. DB requests confirmation that results are sufficient.
8. Administrator confirms that results are sufficient.
9. DB closes search window.

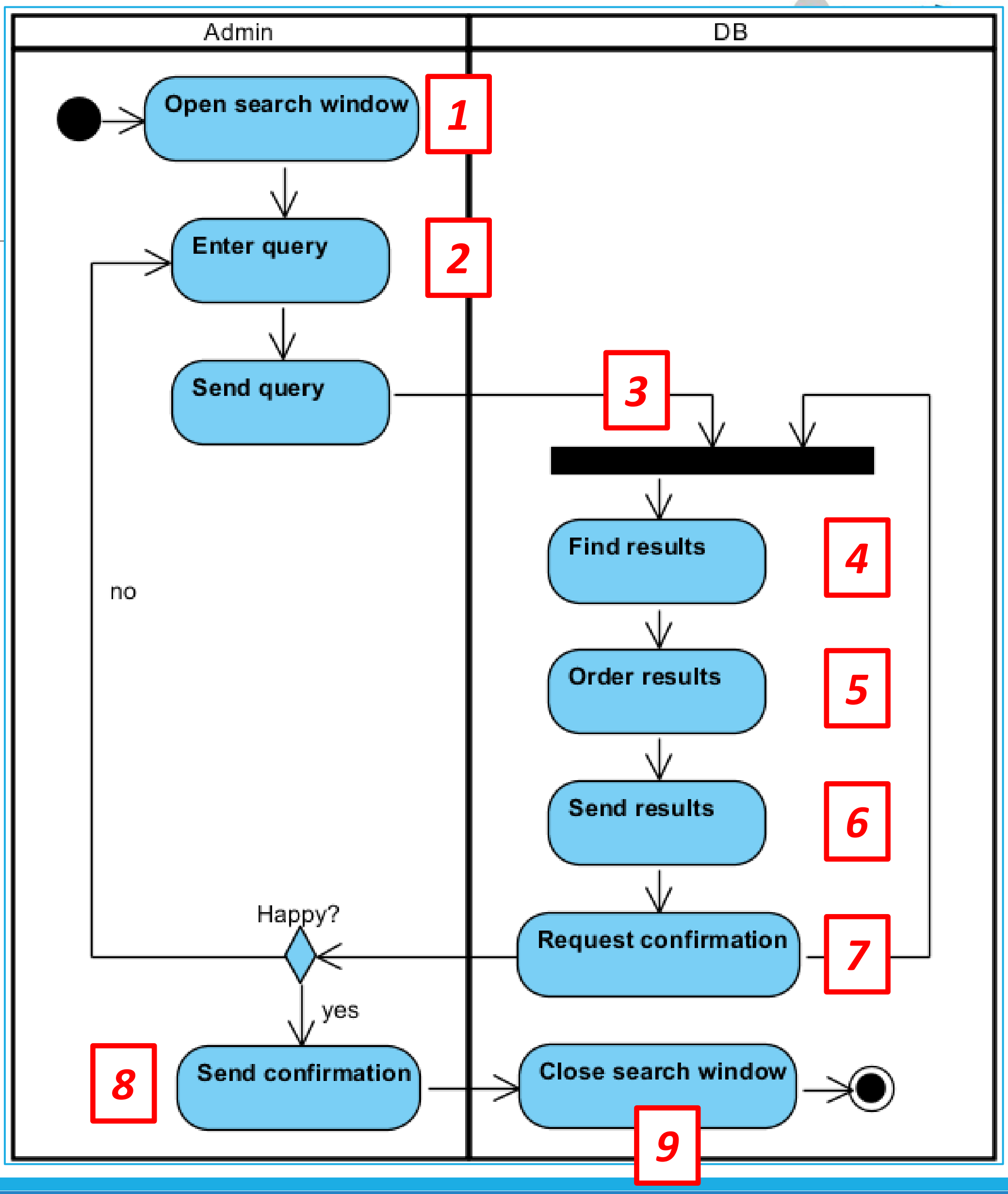
Fleet Logistics Management

- Use Case: Search for information.
  - Alternative paths (pragmatic flow)
    - Administrator has not received the required information: Step 7a,
      - Administrator denies that results are sufficient;
      - Administrator must go back to Step 2.
    - DB is not accessible: Step 3a,
      - DB returns warning message that DB is not accessible;
      - Use case needs to be quit.

  - Post-condition(s):
    - The administrator has retrieved the required information.

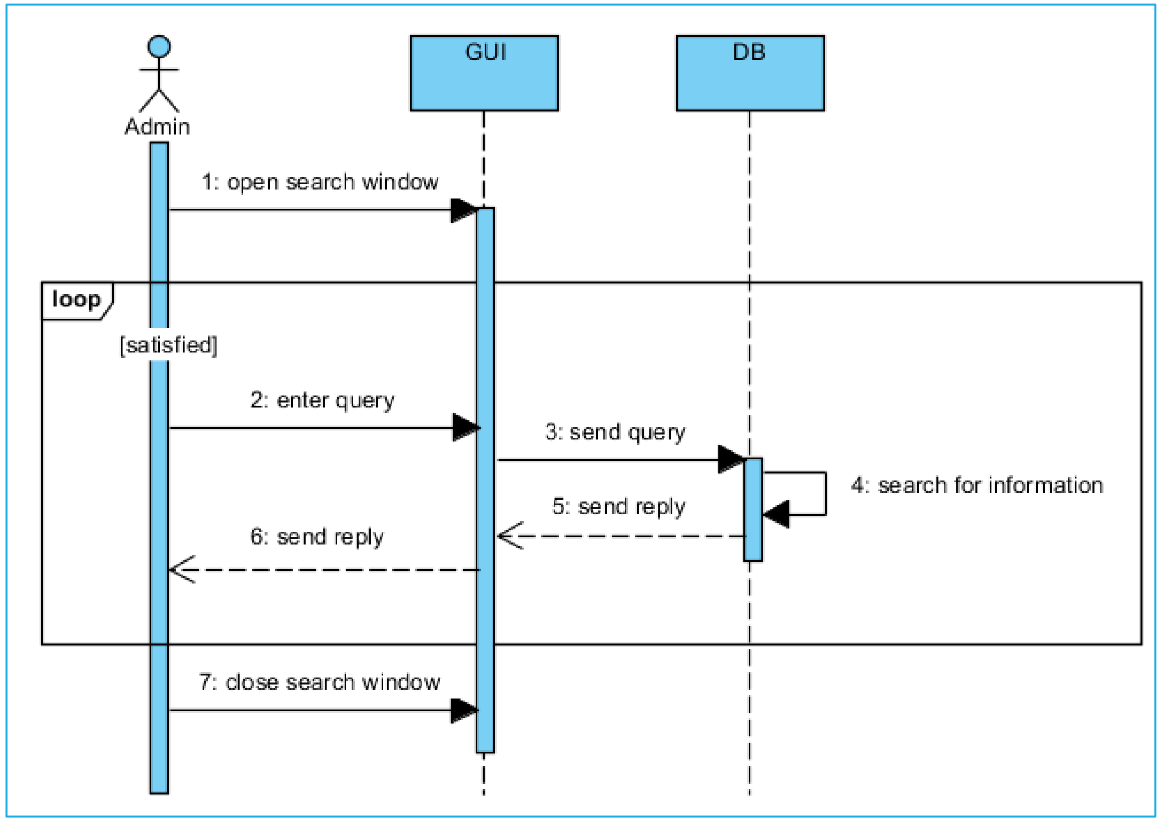
Fleet Logistics Manag

- Activity diagram for use case "Search for information":
  1. Administrator opens search window.
  2. Administrator defines query using query editor.
  3. Administrator sends query to DB.
  4. DB deals with query: finding results.
  5. DB deals with query: organising them by relevance.
  6. DB sends results back.
  7. DB requests confirmation that results are sufficient.
  8. Administrator confirms that results are sufficient.
  9. DB closes search window.



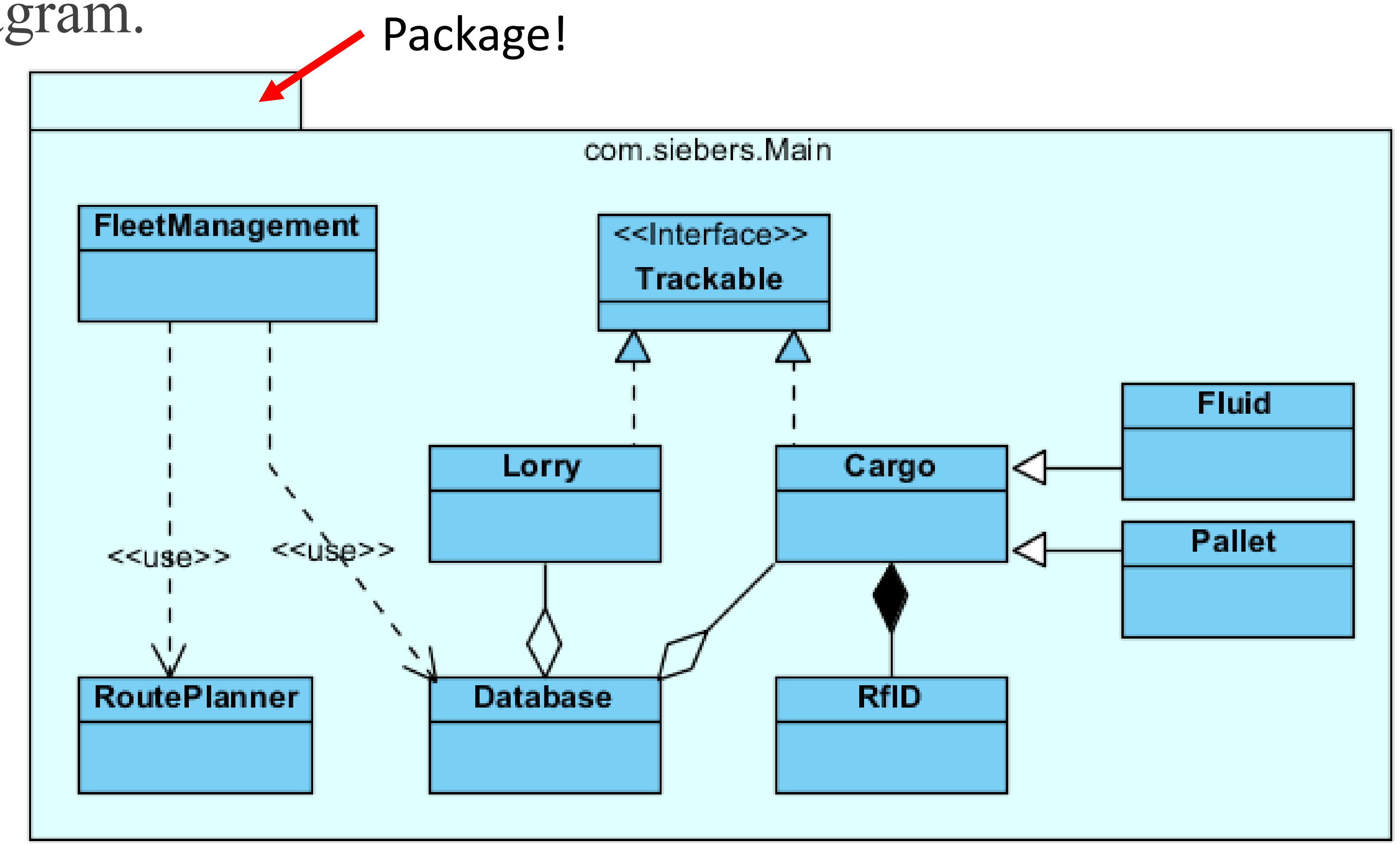
Fleet Logistics Management

- Sequence diagram for use case "Search for information":
  1. Administrator opens search window.
  2. Administrator defines query using query editor.
  3. Administrator sends query to DB.
  4. DB deals with query: finding results.
  5. DB deals with query: organising them by relevance.
  6. DB sends results back.
  7. DB requests confirmation that results are sufficient.
  8. Administrator confirms that results are sufficient.
  9. DB closes search window.



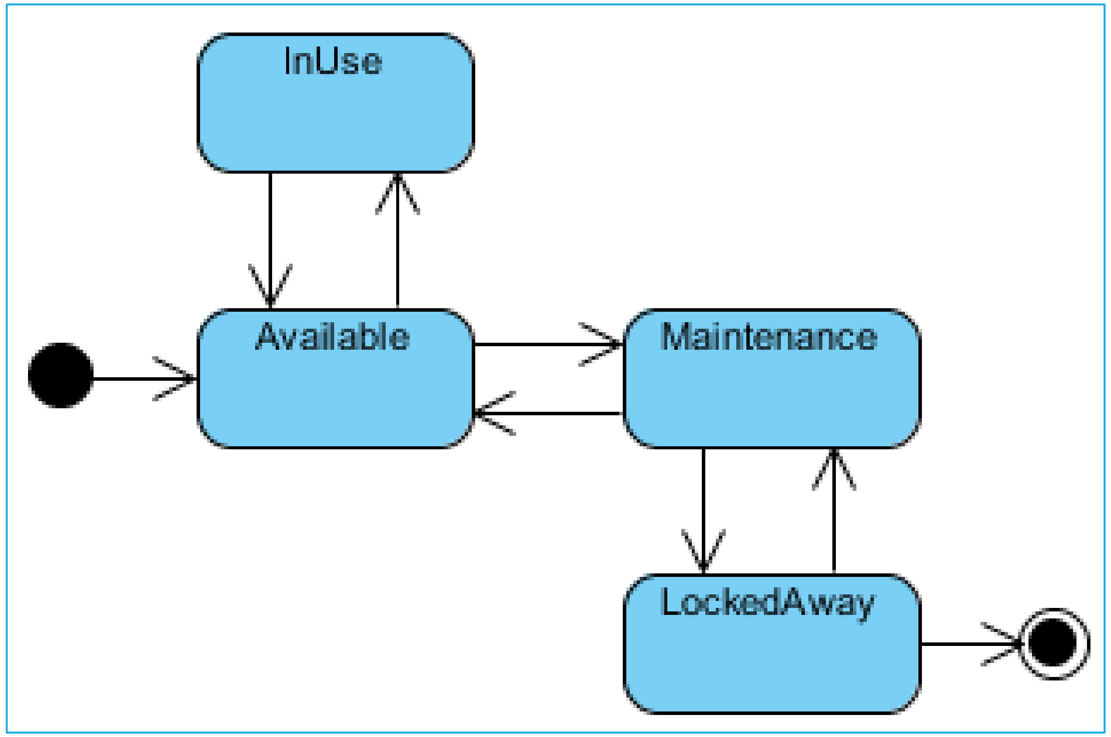
Fleet Logistics Management

- Class diagram.



Fleet Logistics Management

- State Machine diagram for "`Lorry`" class.



Useful Resources

FOR STUDYING UML IN MORE DEPTH

University of Nottingham
UK | CHINA | MALAYSIA

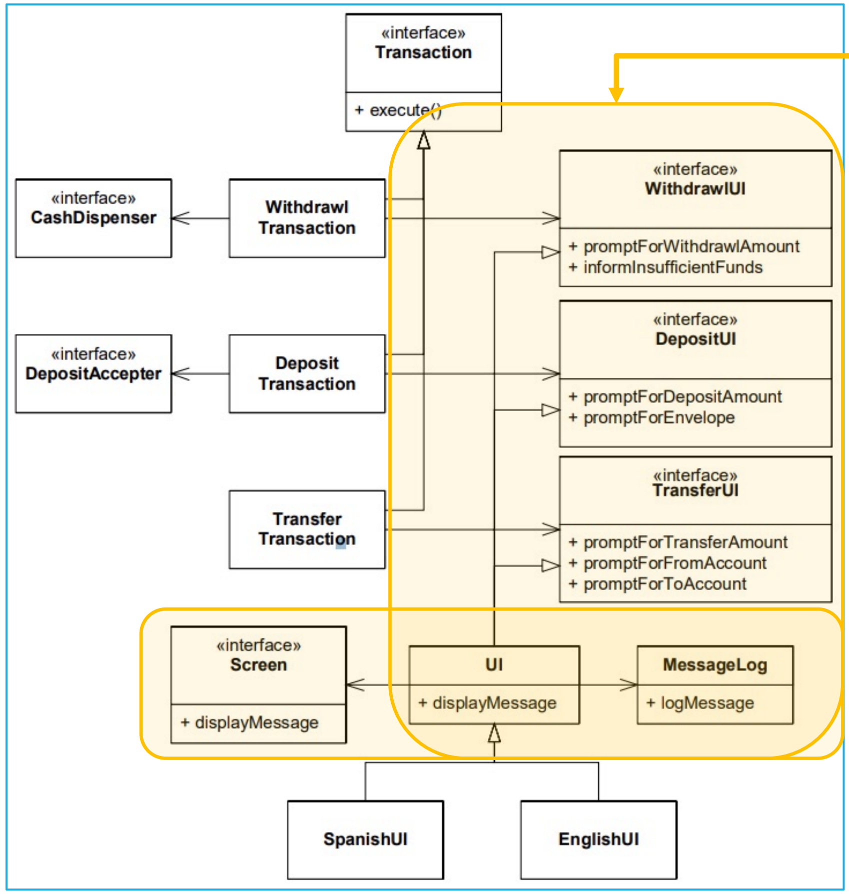
References

- Barclay and Savage (2004) – Object-Oriented Design with UML and Java.
- Some other UMLs - https://www.lucidchart.com/blog/types-of-UML-diagrams
- smartdraw - https://www.smartdraw.com/uml-diagram/
- UML – Standard Diagrams - https://www.tutorialspoint.com/uml/uml_standard_diagrams.htm
- UML for Java Programmer (Chapter 1 to 6) - https://www.csd.uoc.gr/~hy252/references/UML_for_Java_Programmers-Book.pdf
- The <<include>> and <<extend>> Relationship in Use Case Models - https://karonaconsulting.com/downloads/UseCases_IncludesAndExtends.pdf

Some Examples

CLASS DIAGRAM

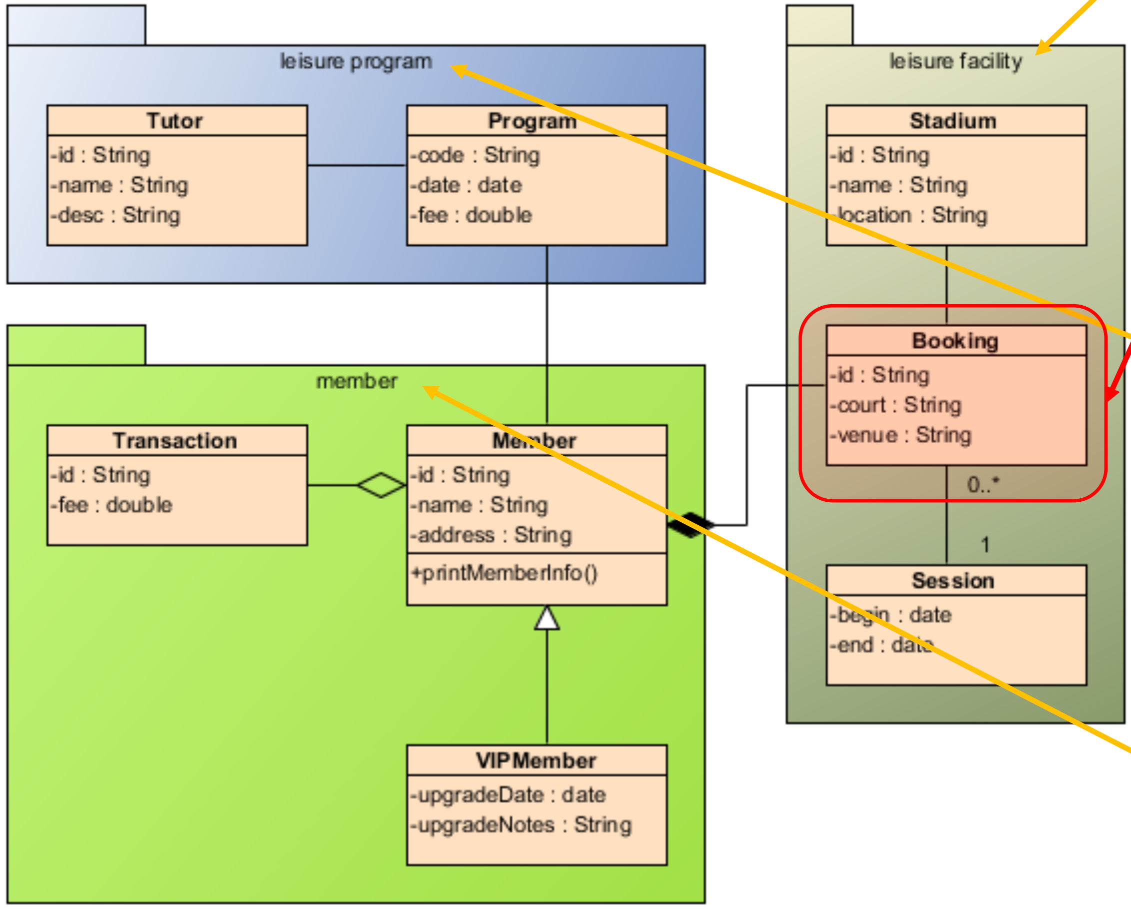
Class Diagram

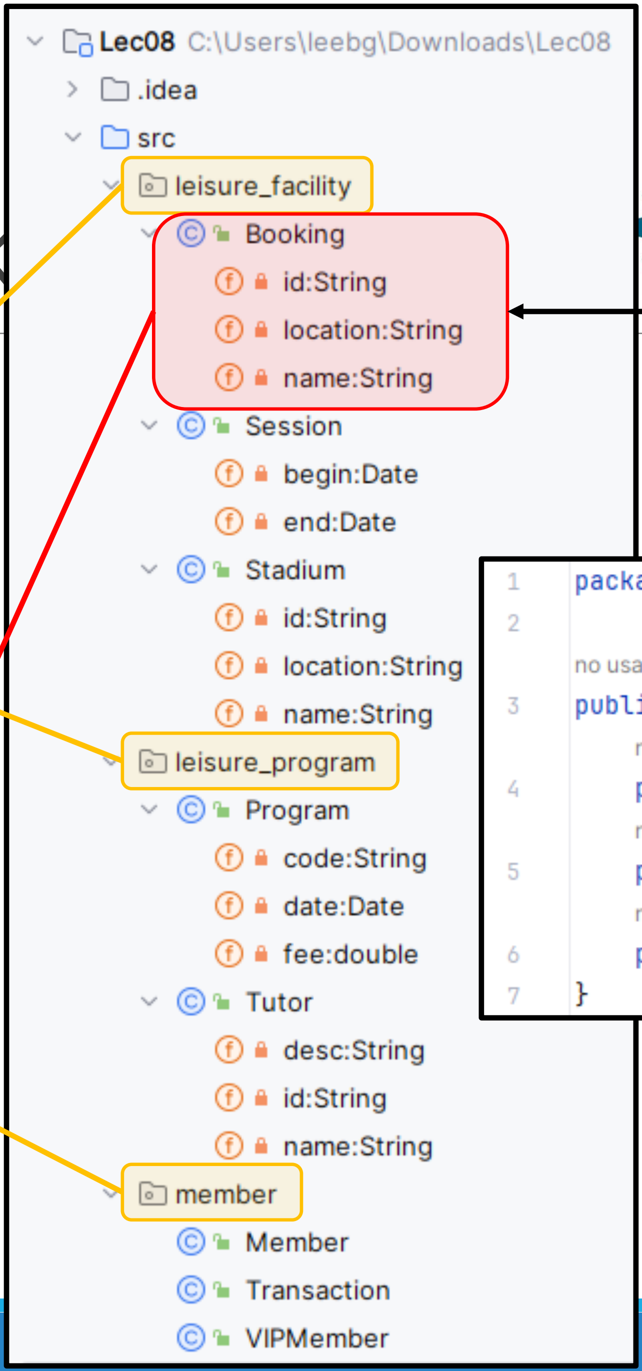


```java
public class UI implements WithdrawlUI, DepositUI, TransferUI {
    private Screen itsScreen;
    private MessageLog itsMessageLog;

    public void displayMessage(String message) {
        itsMessageLog.logMessage(message);
        itsScreen.displayMessage(message);
    }
}
```

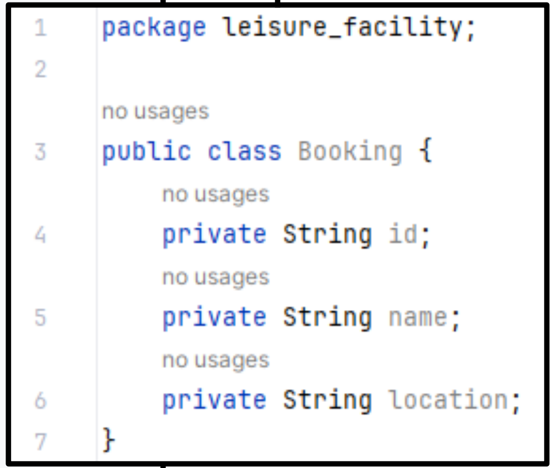
For more examples, see page 36 of UML for Java Programmer
Chapter 3: Class Diagrams

Class Diagram with Pack



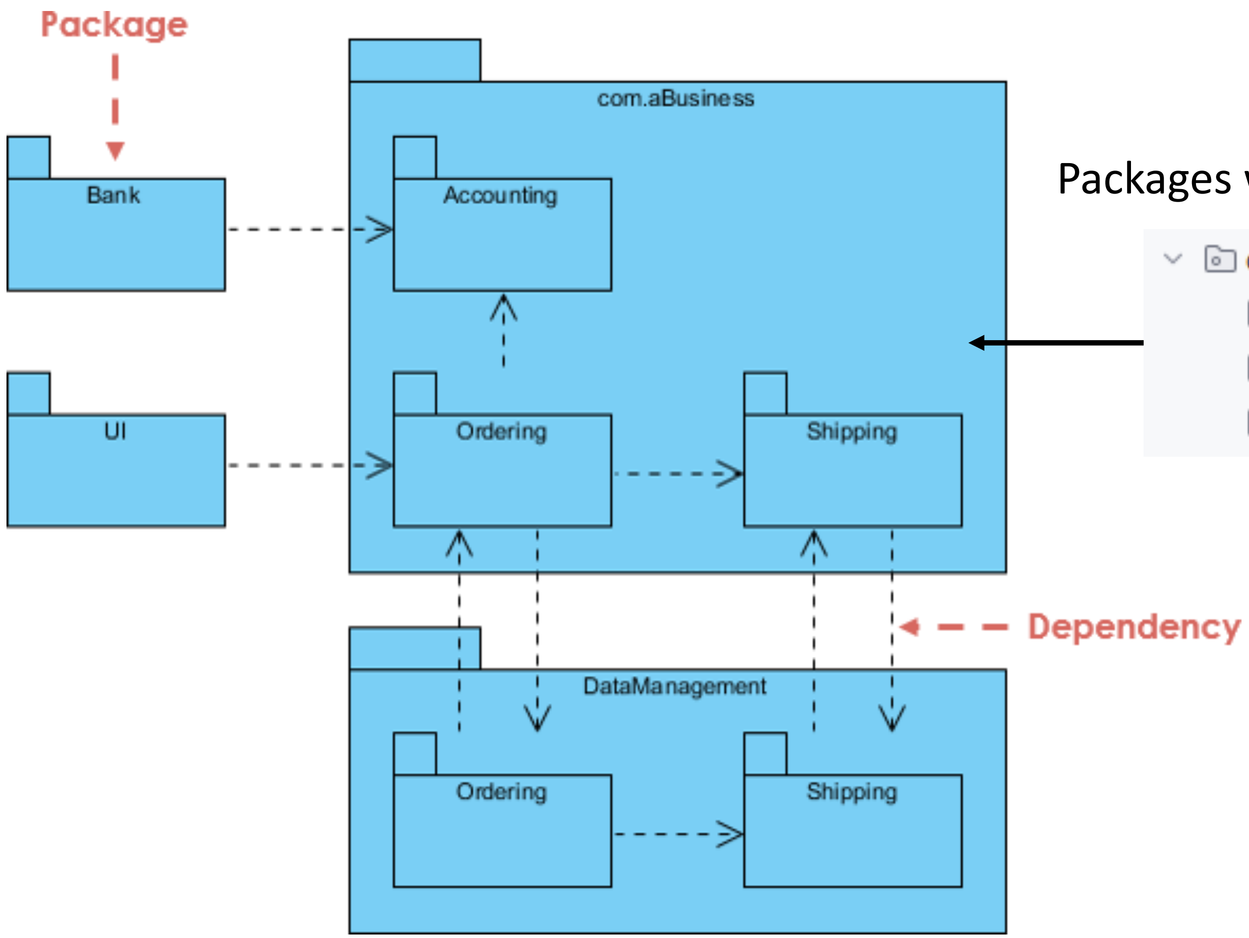


```
Lec08  C:\Users\leebg\Downloads\Lec08
  .idea
  src
    leisure_facility
      Booking
        id:String
        location:String
        name:String
      Session
        begin:Date
        end:Date
      Stadium
        id:String
        location:String
        name:String
    leisure_program
      Program
        code:String
        date:Date
        fee:double
      Tutor
        desc:String
        id:String
        name:String
    member
      Member
      Transaction
      VIPMember
```

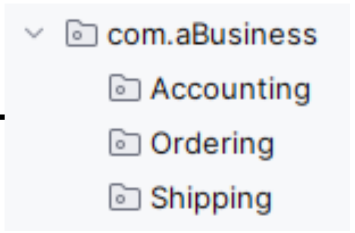


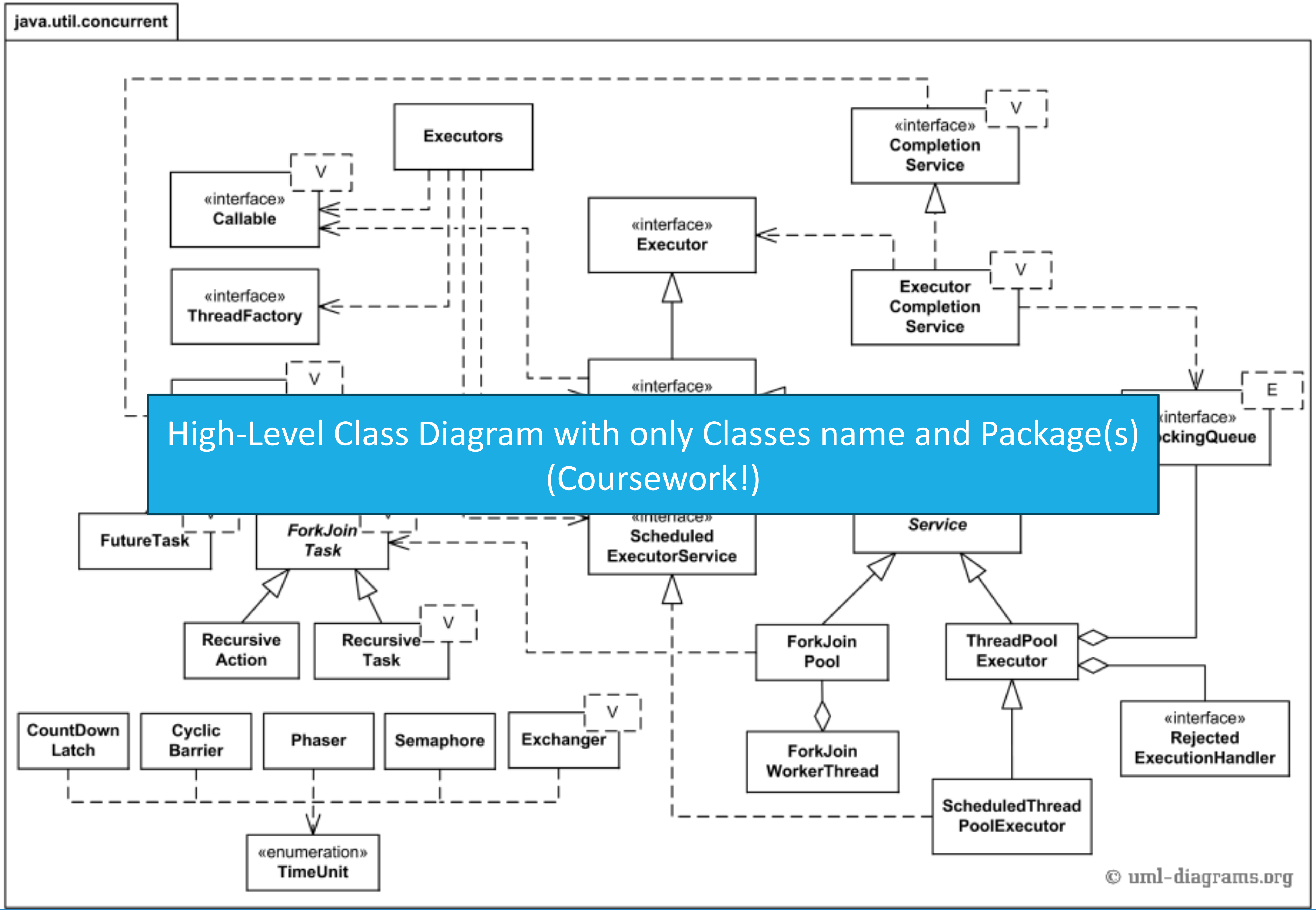
```java
package leisure_facility;

no usages
public class Booking {
    no usages
    private String id;
    no usages
    private String name;
    no usages
    private String location;
}
```

High-Level Class Diagram (Packages)



Packages within a package



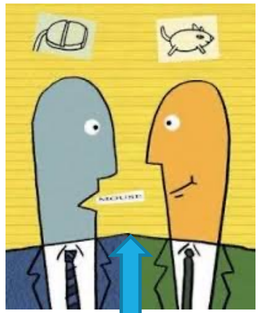


High-Level Class Diagram with only Classes name and Package(s) (Coursework!)

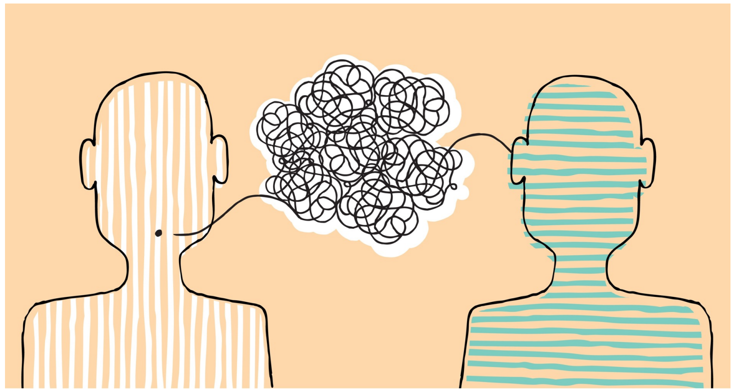
Summary

- Look into usage of UML associated with object-oriented design.
- 5 basic UMLs you have learnt and revised:
  - Use Case Diagram (FSE & DMS)
  - Activity Diagram (FSE)
  - Class Diagram (DMS)
  - Sequence Diagram (FSE)
  - State Machine Diagram (FSE)

Importance of UML to Object-Oriented Design/Analysis



This is fried rice you ordered!



When I said fried rice, it does not mean deep fried the rice!
OMG, you are killing me!
Why you tortured the rice, Uncle Roger heart broken!

Put Your Mind in Maintenance Mode