NANYANG POLYTECHNIC

IT2755 & IT1573

Software Engineering

2019 S1
Module Revision

Exam Time and Venue

- Date: 21 August 2019 (Wed)
- Time: 4:00- 6:00 pm
- Venue: G221 (Z3)
- Be early!
- No entry to exam venue once 30 mins have lapsed
- No unauthorised materials

Exam Format

- 2 hours
- 10 MCQ questions
- 4 structured questions

On your exam answer booklet
Write the questions attempted as follow :
Eg.

1) Q1-10 (MCQ)
2) Q11
3) Q12
4) Q13
5) Q14

Topics

- Overview of Software Engineering
- Intro to Project Management
- Intro to Agile and Scrum Methodology
- Requirements Gathering
- UML Modelling : Use Case Model
- UML Modelling : Use Case Description
- Domain Modelling & Domain Class Diagrams
- Moving into Design: Sequence Diagram I
- Moving into Design: Sequence Diagram II

Topics

- Moving into Design: Solution Class Diagrams
- Software Testing 1
- Software Testing 2
- Software Quality 1
- Software Quality 2
- User stories

Key Learning Outcomes [1/2]

- To describe the aims of software Engineering, SDLC and agile methodology
- To describe the concepts of user stories & SCRUM.
- To identify and differentiate between functional and non-functional requirements.
- To identify the concepts of scenario in the use case description.
- To explain the various class relationships for the domain model.

Key Learning Outcomes [2/2]

- To explain the various class relationships with respect to object oriented concepts.
- Describe the differences between Analysis and Design and the models created in each of these.
- To distinguish the key concepts in developing sequence diagrams.
- To explain the connections between system qualities and system development activities covered in software engineering.
- To identify and develop test cases.
- To explain the key concepts of software quality.

NANYANG POLYTECHNIC

IT2755 & IT1573

Software Engineering

Topic : Software Engineering, SDLC & Agile / SCRUM

2019 S1

Software Engineering, SDLC & Agile / SCRUM

- Software Engineering
- SDLC
- Agile development (SCRUM framework)

What is Software Engineering?

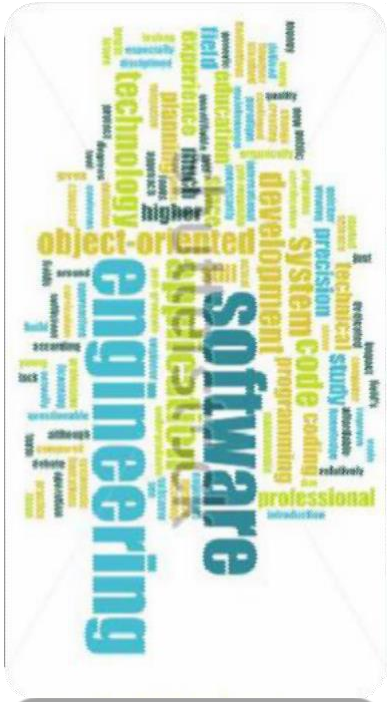
Software Engineering is a discipline whose goal is the production of:

- **quality** software,
- delivered **on time** and
- **within budget**, that
- **satisfies the user's needs**

It encompasses:

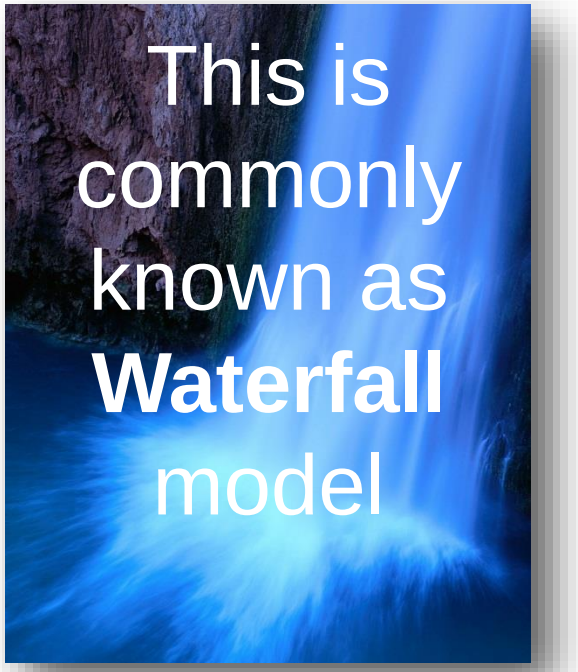
- **processes**,
- **management techniques** and
- **technical methods**

to achieve the software engineering goals.



Software Development Life Cycle (SDLC)

- **SDLC**: process of building, deploying, using, and maintaining a system
  - **Five activities or phases** in a project
    1. **P**lanning,
    2. **A**nalysis,
    3. **D**esign,
    4. **I**mplementation,
    5. **S**upport



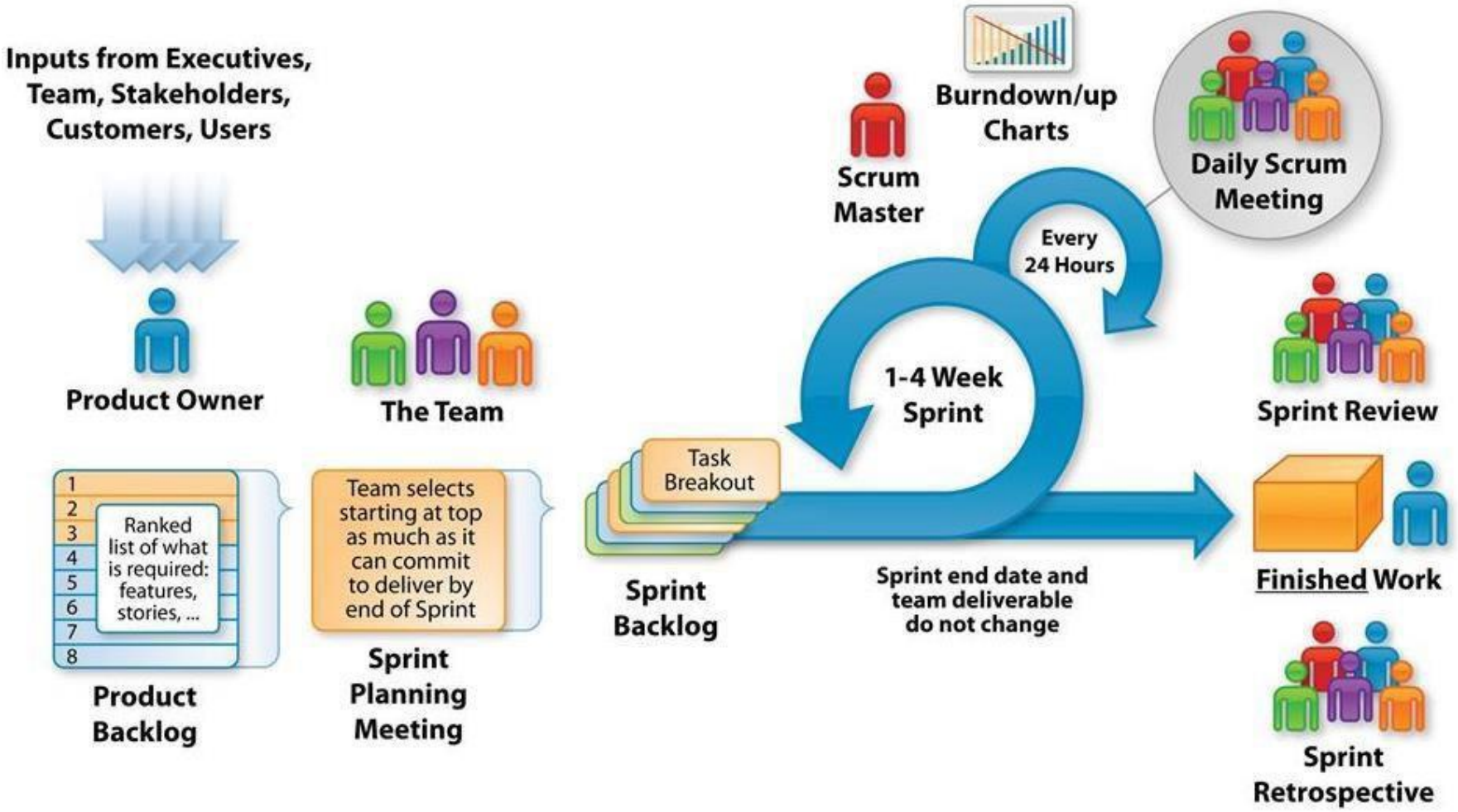
SDLC Phases and their Objectives

SDLC PHASE	OBJECTIVE
Project Planning	To identify the scope of the new system, ensure that the project is feasible, and develop a schedule, resource plan, and budget for the remainder of the project
Analysis	To understand and document in detail the business needs and the processing requirements of the new system
Design	To design the solution system based on the requirements defined and decisions made during analysis
Implementation	To build, test, and install a reliable information system with trained users ready to benefit as expected from use of the system
Support	To keep the system running productively initially and during the many years of the system's lifetime

Scrum

- **Scrum** is one of the **Agile** methodologies which allow the project teams to prioritise the requirements in consultations with customers to frequently deliver the system **part-by-part** and adapt to changing circumstances.
- Iterative and incremental
- 3 roles, 3 artefacts, 3 ceremonies
- What do we do during sprint?

Agile Methodology - Scrum



User stories

- **User stories** are informal description of system features from end to end slicing through all layers of architecture and delivers value. They focus in what is the feature, who needs it and why it is needed.
  - 3Cs approach – Card, Conversation, Confirmation
  - User stories can be evaluated by INVEST –

Independent
Negotiable
Valuable
Estimable
Small
Testable

NANYANG POLYTECHNIC

IT2755 & IT1573

Software Engineering

Topic : Project Management

2019 S1

Project Management

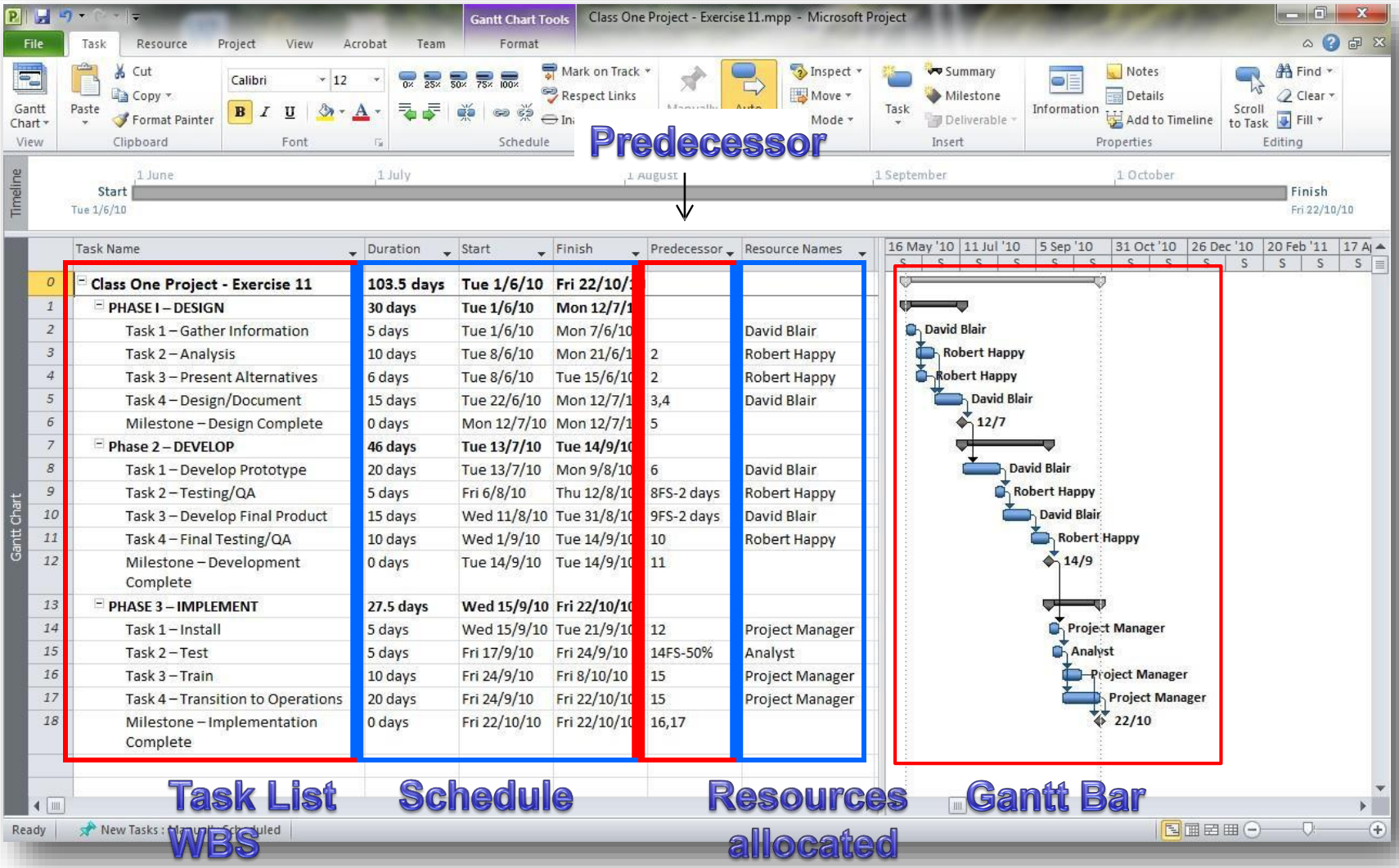
- 3 common constraints
- Gantt chart:
  - Construct Gantt chart
  - Identify predecessor tasks
  - Identify critical path
  - Identify slack (float)
  - Baseline

Project Constraints

- Triple Constraints:
  - **Scope**
  - **Cost (budget, resources)**
  - **Time (schedule, resources)**

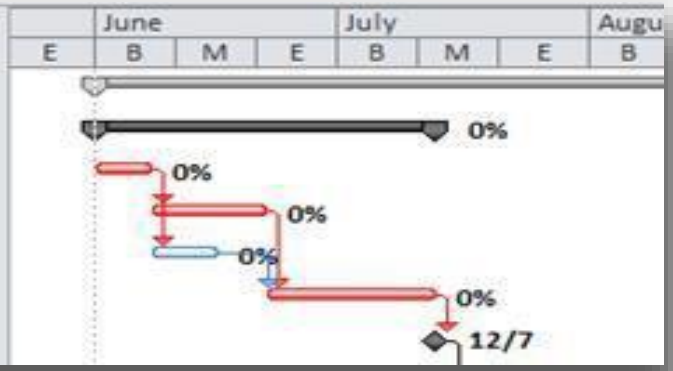
- One side can not be adjusted or altered without affecting the other sides.

An example of a Gantt chart (using Microsoft Project 2010)



Critical path & milestones

		Task Name	Duration
0		**Class One Project - Exercise 11**	**103.5 days**
1		**PHASE I – DESIGN**	**30 days**
2		Task 1 – Gather Information	5 days
3		Task 2 – Analysis	10 days
4		Task 3 – Present Alternatives	6 days
5		Task 4 – Design/Document	15 days
6		Milestone – Design Complete	0 days



- The critical path :

  Task 1 -> Task 2 -> Task 4

- because any delay completion of these tasks will affect the final project completion date of 12 July.

- A Gantt chart (project schedule) will need to be reviewed with client and approved as a baseline for use as a reference to compare with the actual project progress.

NANYANG POLYTECHNIC

IT2755 & IT1573

Software Engineering

Topic : Requirements Gathering

Requirements Gathering

- Functional vs Non-Functional Requirements
- 7 primary techniques for gathering requirements
  - Situations to use
  - Pros & Cons
- Challenges - Scope Creep

Techniques for Information Gathering

A. Review reports, forms, procedure, descriptions
B. Conduct interviews and discussions with the users
C. Observe business processes
D. Building effective prototypes
E. Distribute and collect questionnaires
F. Conduct Joint Application Design sessions (JAD)
G. Research vendor solutions

NANYANG POLYTECHNIC

IT2755 & IT1573

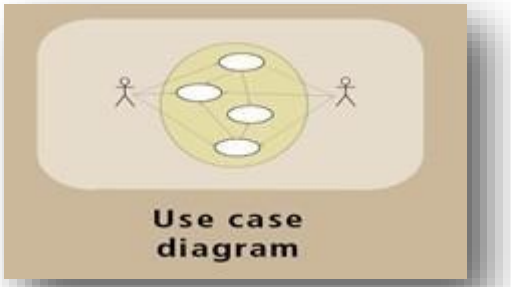
Software Engineering

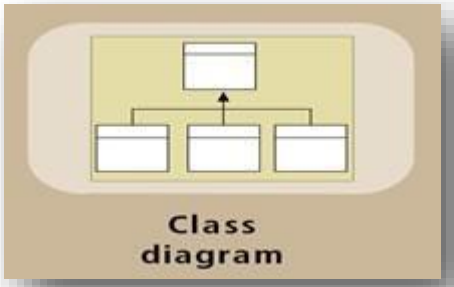
Topic: Use Case Model & Description

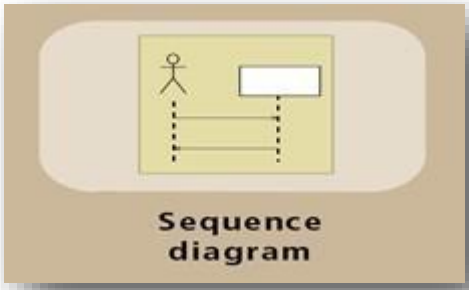
Use Case Modelling

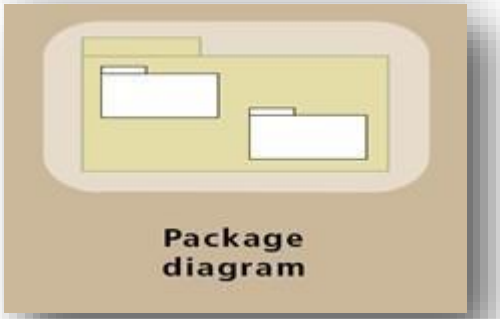
- Use Case Model/ Diagram
- Use Case Description

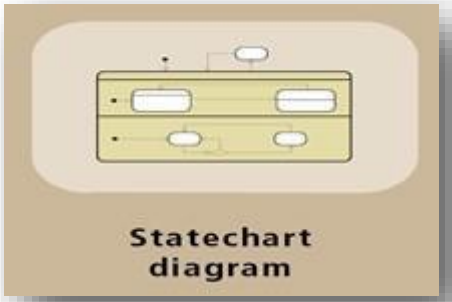
UML Diagrams used for Modeling

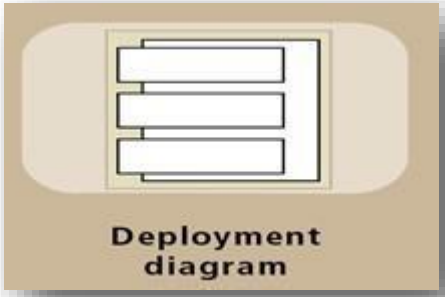




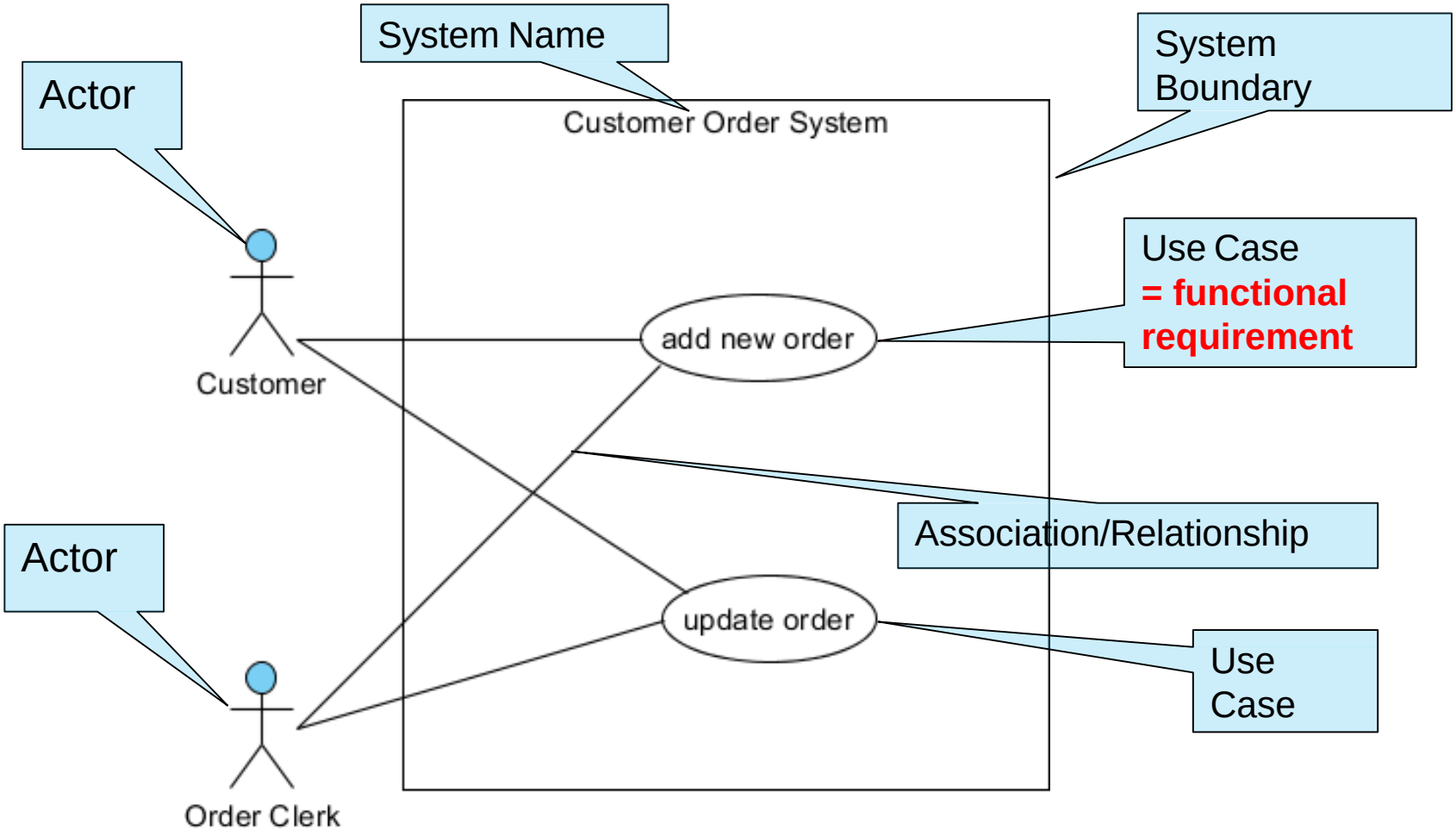








A Use Case Diagram of the Order System with System Boundary



Identifying Actors

- **Actor**
  - Person or thing that directly interacts with the system
  - Primary actor – initiates or triggers the use cases to achieve the actor's goals
  - Secondary actors – participates in the use cases to assist the primary actor.
    - External systems, such as email service, sms service, paypal service etc.
  - Normally described by **nouns**
  - **Example:** *A online bookstore allows customer to buy books and payment is handled by an external payment system (e.g. Paypal or eNets).*

Identifying Use Case

- **Use Case**
  - Something (a goal) that the actors wants the system to do or achieve.
  - Always triggered by actor(s)
  - Identified from the point of view of the actor
  - **Actions phrases or verbs**
  - Eg: A online bookstore allows customer to buy books and payment is done through an external payment system (e.g. Paypal or eNets).
- **Use Case Name:** use a short verb phrase
  - E.g. Add orders, Cancel membership, Print reports

Use case diagram example
2013 S2 Exam

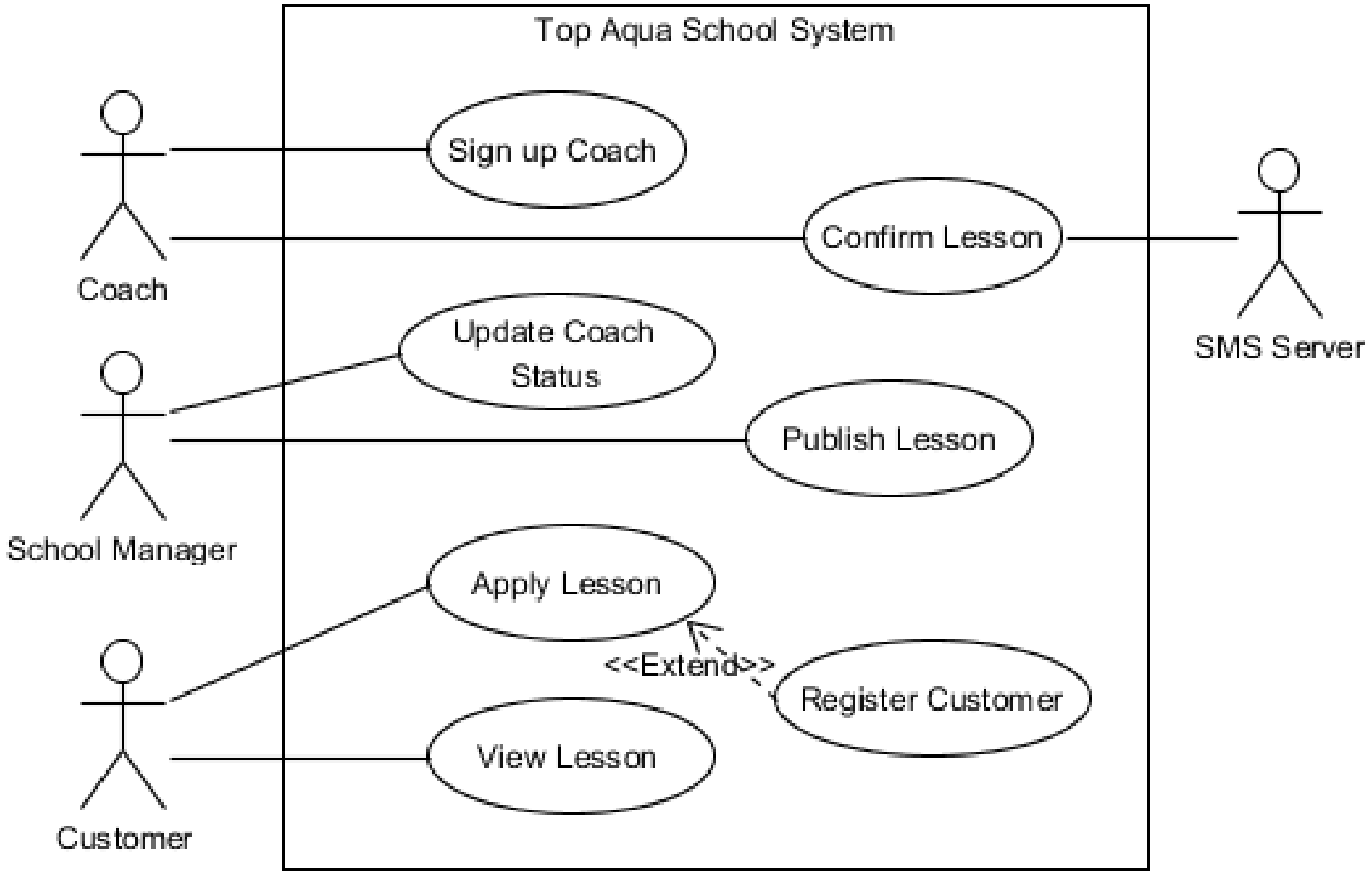
Top Aqua School is a private swimming school that provide customised training program to customers in condos all over the island. Currently the company maintain a list of coaches and customers using Microsoft Excel, and lesson arrangement is done manually. As their business is expanding, it is necessary to have an information system to manage the coaches and customers.

The School wants to expand their pool of coaches by allow coaches to sign up through the system. The school manager will interview the coach, and verify his/her qualification. After the interview the school manager needs to update the coach's status from "newly registered" to "qualified". Only qualified coaches can conduct swimming lessons in locations they prefer. School manager also needs to discuss with the coach to plan out the lesson and publish the lessons on the web site.

Customer will be able to view the lesson schedule published on the web site, and apply for lessons. During the application, customers are asked for their user ID. If customer has not registered, the system will redirect the customer to register with the system. Once the customer registered successfully, the system will bring the customer back to continue with the lesson application.

The coach will confirm the lesson and an SMS will be sent to the customer.

Use case diagram example



Guidelines for use case description

- Use Simple Grammar, eg:
  - Librarian enters book ID
  - System displays book details
- Show clearly who is in control at point in time
  - Always start with:
    - The *actor* …. (replace actor with the actor's name in the use case)
    - The system …
- Write the interaction between the actor and the system.
- Do not impose constraints on user interface
  - User ~~clicks on OK button~~ to submit personal details X
  - User submits personal details to system
- Alternative flow numbering follows the main flow.

Use Case Description Example - 2013 S2 Exam

Main Flow:

1. Technical support staff enters purchase code.
2. System validates purchase code and displays details of the purchase.
3. Technical support staff confirms that all the models delivered are correct.
4. Technical support staff assigns Laptop ID.
5. System verifies that laptop ID is valid and not duplicated.
6. Technical support staff indicates everything complete.
7. System displays successful creation of new laptop models in inventory listing.

Alternative Flow:

2A Purchase code invalid, technical support staff rejects the delivery.Use case ends.

3A Laptop models delivered does not match purchase order, technical support staff rejects delivery. Use case Ends.

5A Duplicate Laptop ID found, System display a message.

NANYANG POLYTECHNIC

IT2755 & IT1573

Software Engineering

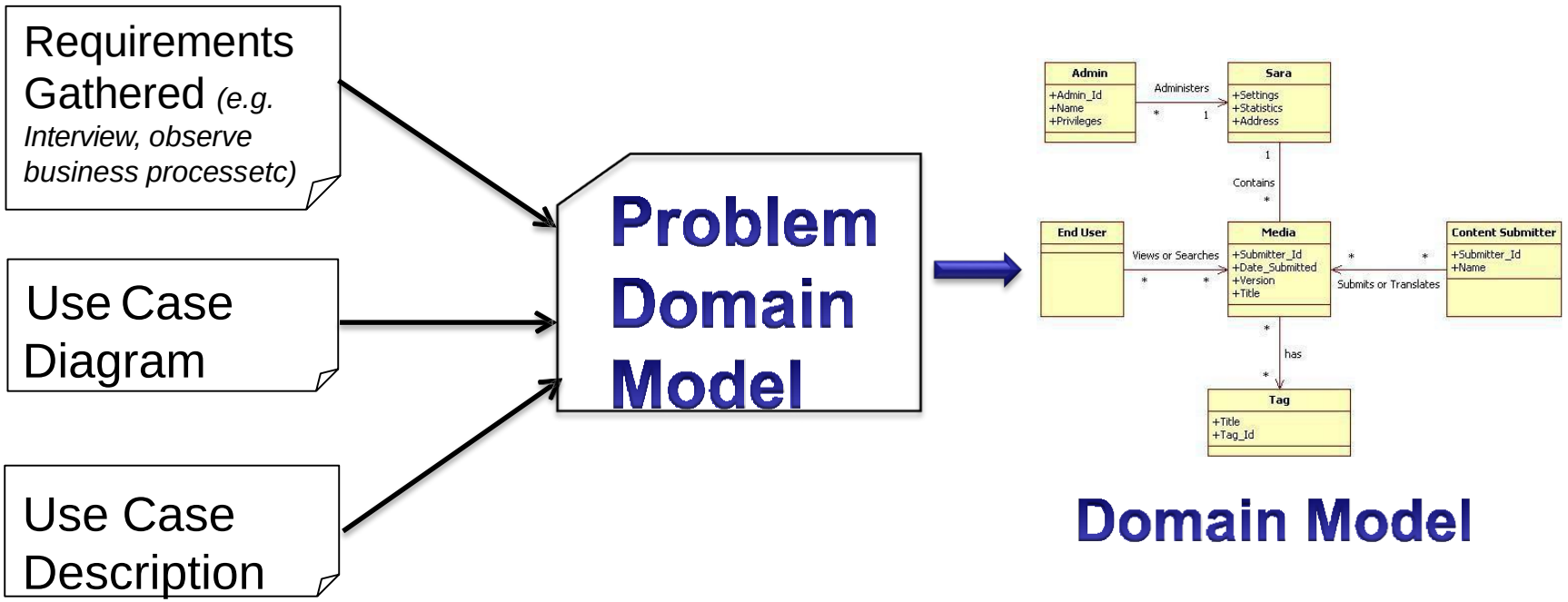
Topic: Domain Modelling

(Problem) Domain Model and Class Diagrams

- Class diagrams are Object Oriented approach to representing:
  - Things in a problem domain (**Domain** class diagram)
  - Objects that interact in the system (**Solution/Design** class diagram)
- Domain Class Diagrams model set of important data representation within a problem domain.
  - Examples: products, orders, invoices, customers
- Solution Class Diagrams model (software) objects within the system that interact to either process or track information. (Java or C# classes)

Source of potential things

- Use cases and their description, system events, external agents , system requirements, triggers and responses, interview notes etc.



Types of Domain Class Objects

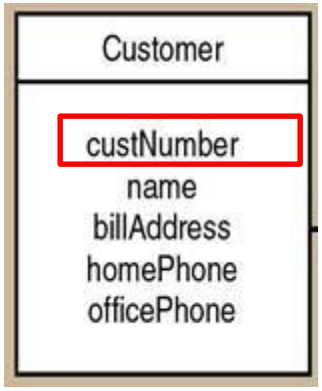
- **Identified** through users sharing "things" regarding their work routine.
  - Include information from **all types of users**
- Separate the **tangible** from the **intangible**
  - **Tangible:** product, an item in the product catalog
    - **Nouns** users mention when discussing system
  - **Intangible:** an order
    - Ask questions about **nature of event**
    - "What interactions should be acknowledged and recorded by the system?"

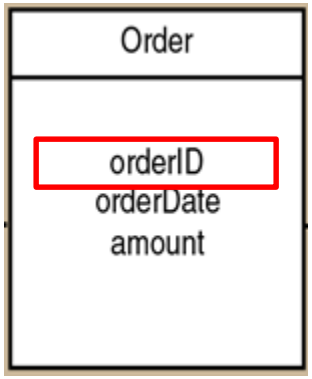
E.g. Customer places an order.

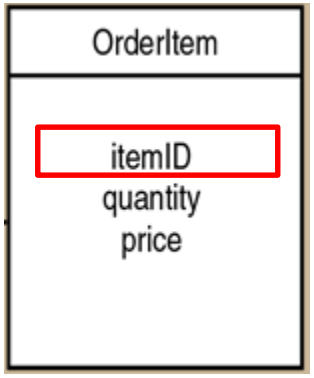
Order is an intangible object arising from customer interaction with item in inventory

Attributes of classes

- Specific details of classes are called attributes
  - Attributes of classes dictate the data that need to be stored with these classes of objects
  - Identifier (key): attribute uniquely identifying class objects
    - Examples: Social Security number, vehicle ID number, or product ID number

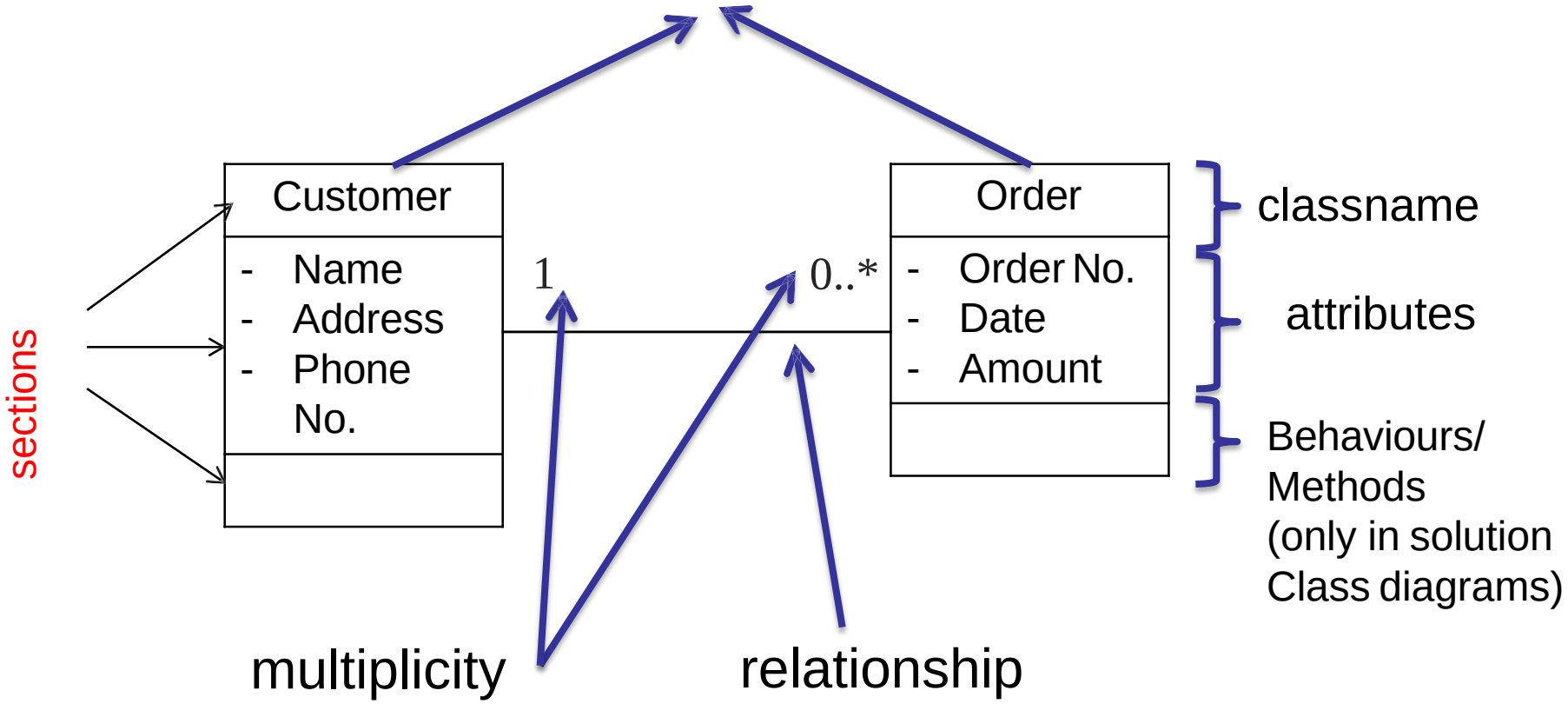




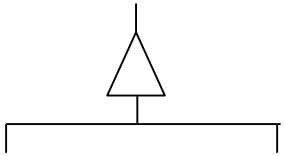
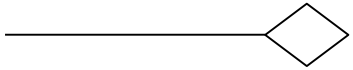
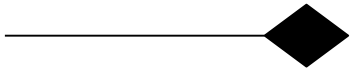


(UML) Class Diagram Notation

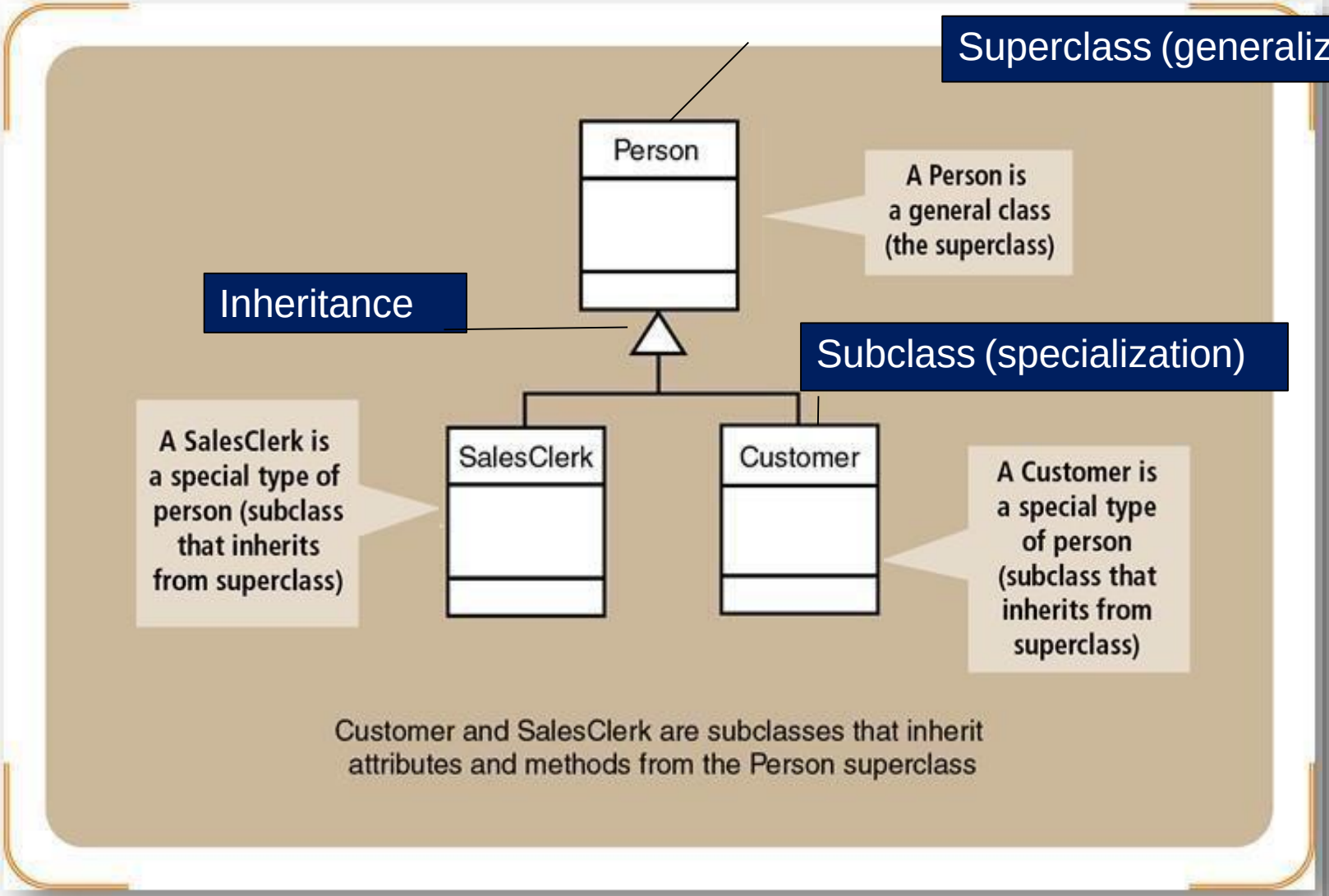
class symbol



Types of Relationship that need Multiplicity

Relationship	Notation	Multiplicity
1. Association		
i. Bi-directional		✓
ii. Uni-directional		✓
2. Generalisation/Specialisation		
i. a.k.a Inheritance		X
ii. "Is a kind of"		
3. Aggregation		✓
i. "Consist of"		
4. Composition		✓
i. "made up of"		

A Generalization/Specialization Hierarchy Notation

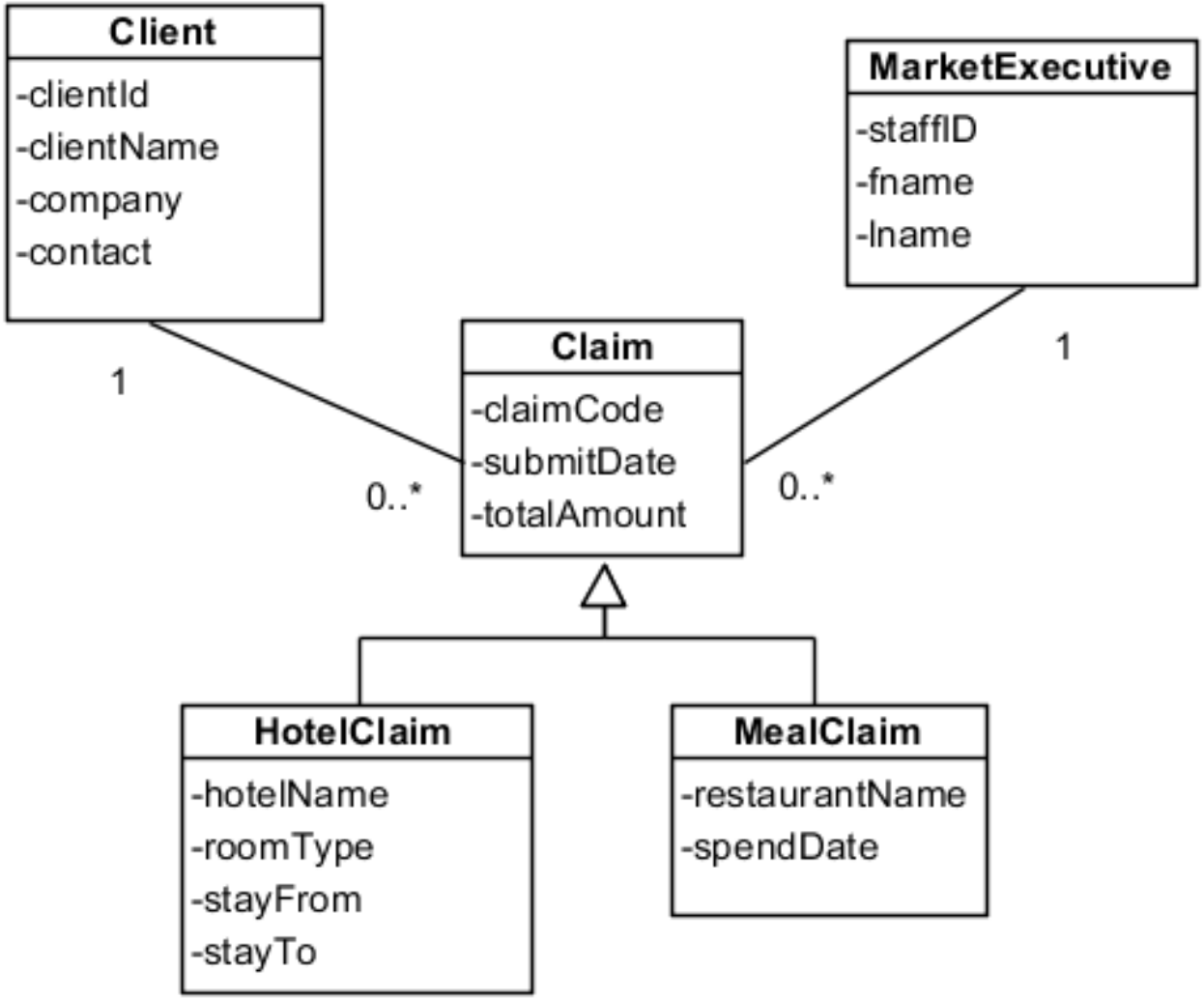


Domain model Example

For the Bliss Claim System, market executives can submit claims for the expenses incurred during the course of their work. The claims can be for hotel accommodation while on oversea assignment or for meals when networking and entertaining potential or existing clients. Each marketing executives can submit any number of claims. For each claim, the executive must specify the total amount of the claim. For accommodation claims, the executive must specify the period of the stay by indicating the start and end dates, the name of hotel and the type of room which the executive stayed in. For meal claims, the name of the restaurant must be entered as well as the date the meal took place. The system will generate a unique claim code for each of the claim submitted and record the date when each claim is submitted.

For each claim, the marketing executives have to indicate who the client is. Each client is identified by a unique client identifier assigned by the system. Other details the system must maintain about each client are the client name, the client company and the client contact number. Over a period of time, a marketing executive can submit many claims for the same client to build a closer relationship.

Solution


Client
-clientId
-clientName
-company
-contact
MarketExecutive
-staffID
-fname
-lname
Claim
-claimCode
-submitDate
-totalAmount
1
0..*
0..*
1
HotelClaim
-hotelName
-roomType
-stayFrom
-stayTo
MealClaim
-restaurantName
-spendDate

NANYANG POLYTECHNIC

IT2755 & IT1573

Software Engineering

Topic: Design Discipline & Sequence Diagram

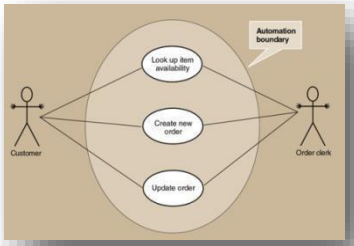
Design Discipline & Sequence Diagram

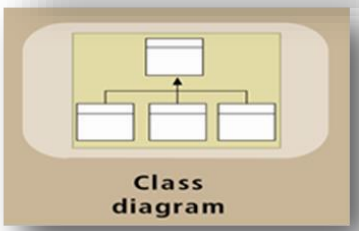
- OOAD
  - OOA vs OOD
- Sequence Diagram
  - Elements of Sequence Diagram
  - Software models required
  - Guidelines for constructing Sequence Diagram

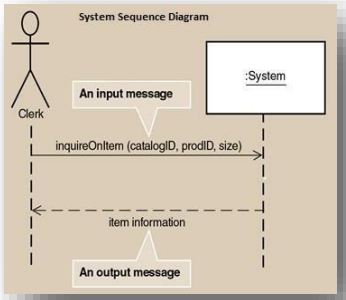
Object Oriented Analysis (OOA)

- In O-O Analysis we try to understand **WHAT** the problem is.

  - Use case diagrams and descriptions ✔
  - Problem domain class diagram ✔
  - System sequence diagrams ✔
  - Statechart diagram
  - Activity diagrams





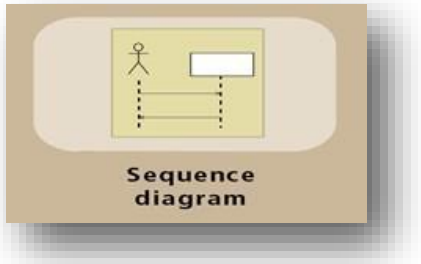


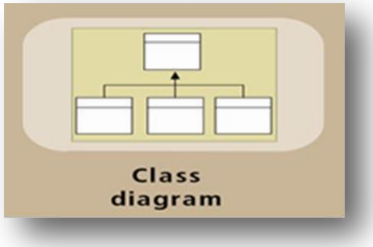
promote understanding

Object Oriented Design (OOD)

- In O-O Design, we focus on **HOW** to solve the problem.

  - Sequence diagram ✔
  - Design class diagram ✔
  - Package diagram
  - Network diagram
  - Database schema

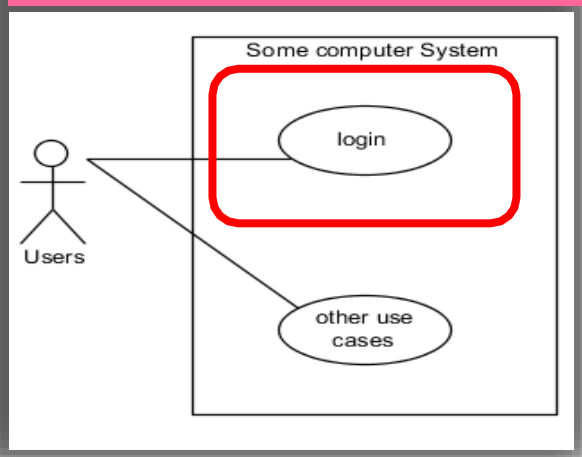




move project closer to implementation

Design Use Case Realisation – Login use case example

The design of the software that implements each use case to make them executable

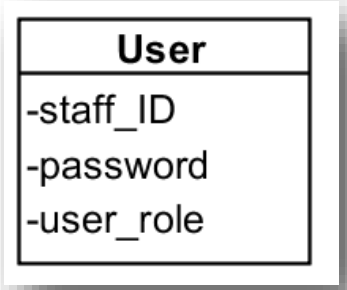


Use case

Use Case ID:	ACC_UC_1
Use Case Name:	login
Created By:	Joe Blogs
Date Created:	1-1-2012
Description:	This use case allows user to login into the system to access the relevant functions according to the user's role. The various user roles are staff, admin staff, system administrator, manager and department head. To login to the system, all users have to enter their unique staff id which is their NRIC number and a valid current password. The users have a maximum of 3 attempts to login after which their account are locked and they will have to contact the system administrator to unlock their account upon successful login the system will display the relevant user's home page.
Primary Actor:	User
Secondary Actor:	None
Preconditions:	1. User has to have a valid account
Postconditions:	1. The system displays the relevant homepage
Main Flow:	1. The user enters the staffID and password 2. The user submits the staffID and password 3. The system validates the staffID and password 4. The system verifies the staffID and password 5. The system displays the user's homepage 6. The use case ends

Use case description

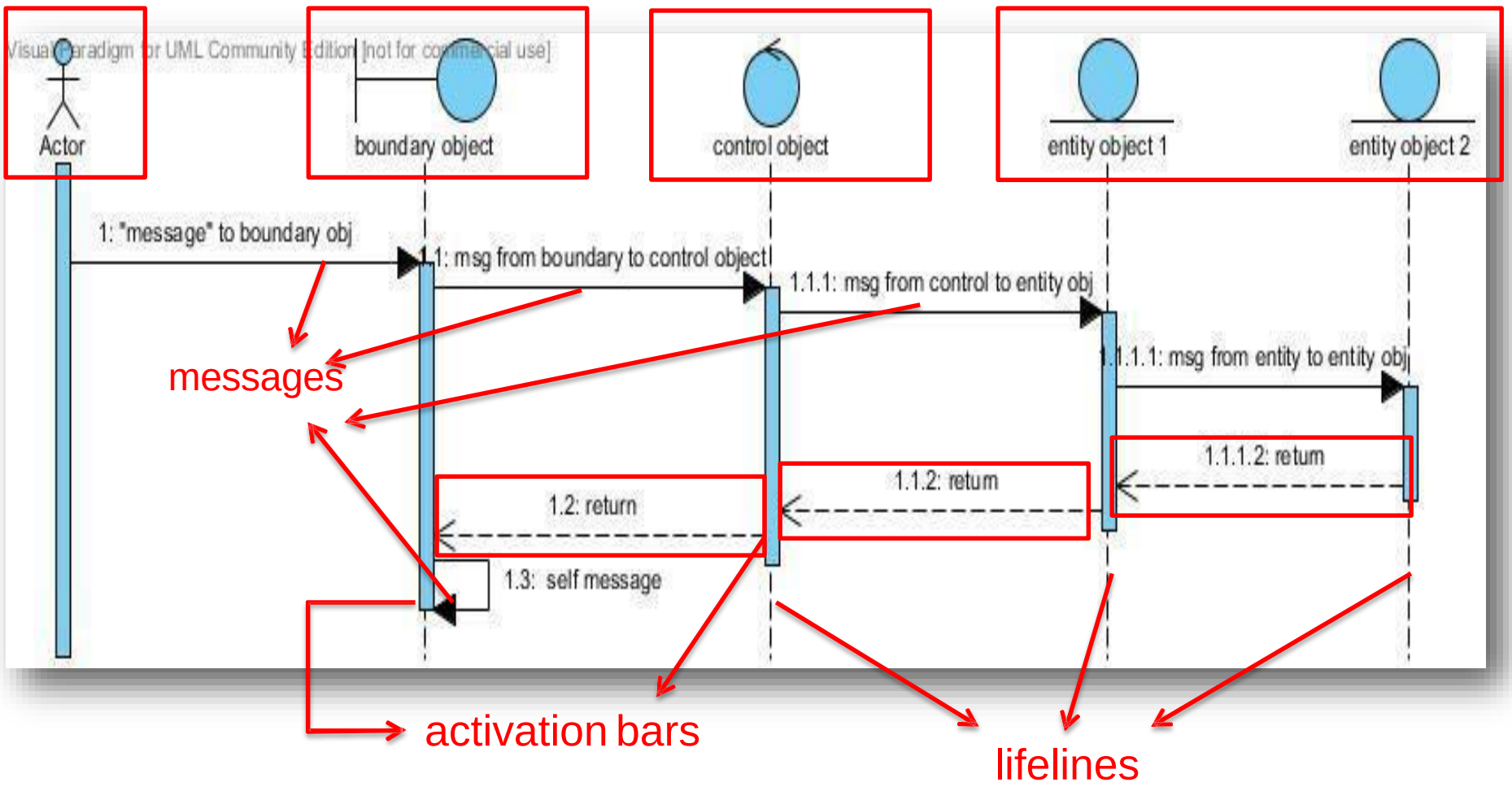
Domain Class diagram



Sequence Diagram

- Drawing Sequence diagram is the process of discovering responsibilities of each object that participates in a use case.
  - Sequence Diagram shows the interactions of different analysis objects when use cases execute.
  - The interactions are depicted as messages in the diagram
  - Messages are used to delegate responsibilities to the various analysis objects
  - These responsibilities are used to derive the methods of participating classes

Elements of a Sequence diagram



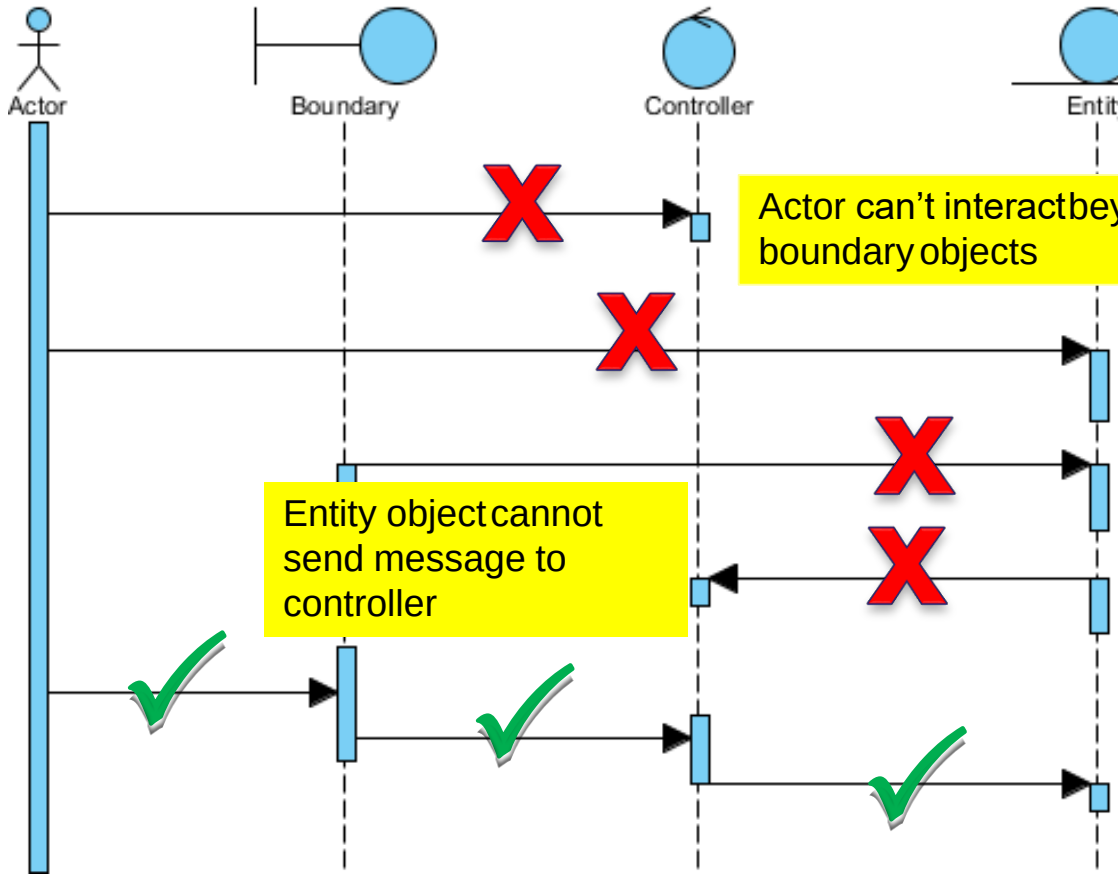
Elements of a Sequence diagram

- Only one boundary object
  - Normally just append the word "form" to the use case name
  - Only one control object
    - Normally just append the word "controller" to the use case name
  - One or more entities objects
    - Identified from the domain class diagram

Interactions between analysis objects in a Sequence diagram

- Actor can only interacts (i.e. send messages to) with the boundary object
  - Boundary object can only interacts (i.e. send messages to) with controller object
  - Control object can only interacts (i.e. send messages to) with entity objects
    - Control object can only return to boundary object
  - Entity objects can only interact (i.e. send messages to) with other entity objects
    - Entity objects can only return to other entity objects or control object

Common mistakes in constructing sequence diagrams



Actor can't interact beyond boundary objects

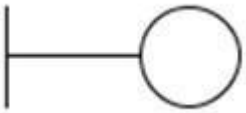
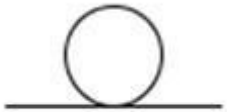
Actor can't interact beyond boundary objects

Entity object cannot send message to controller

Its not a good design to have boundary objects access to entity objects directly.
Let controller object accesses to entity objects.

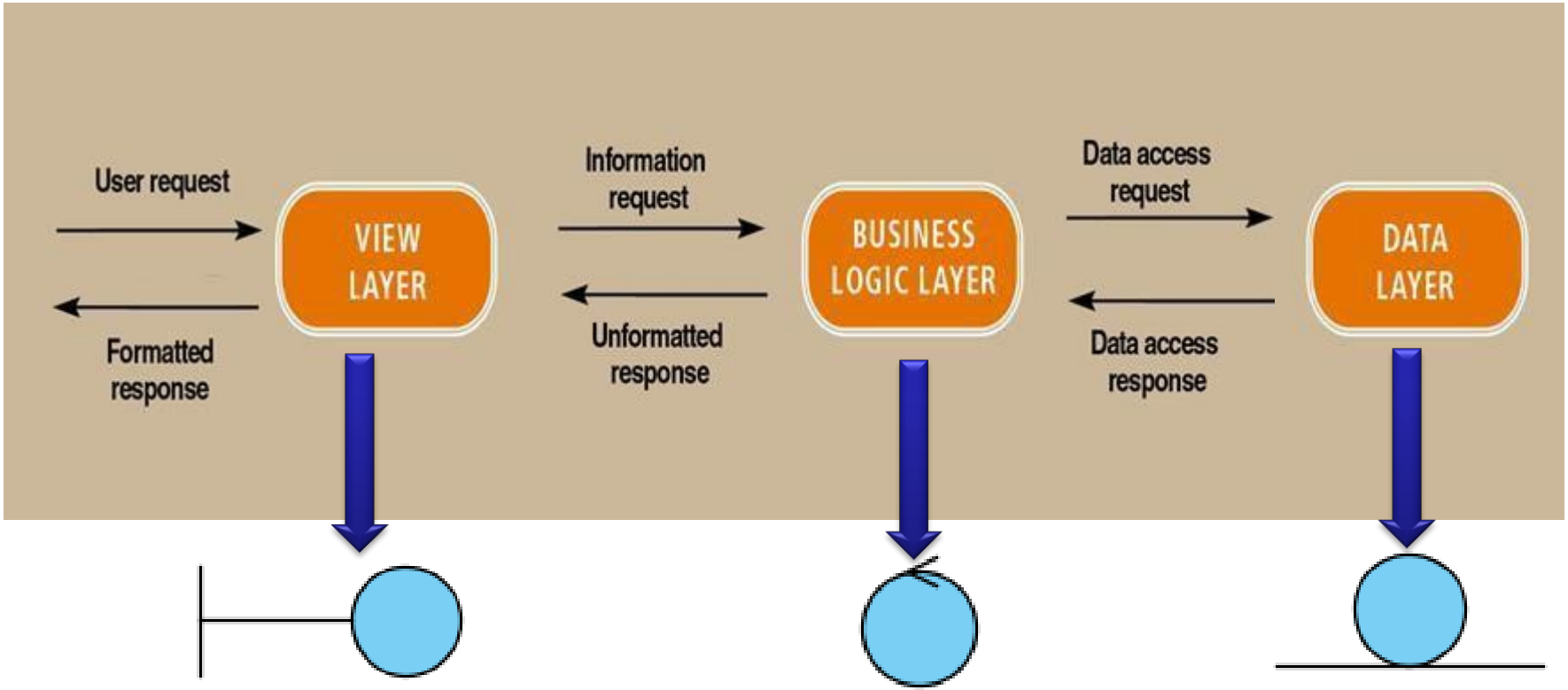
Sequence Diagram

3 types of analysis objects:

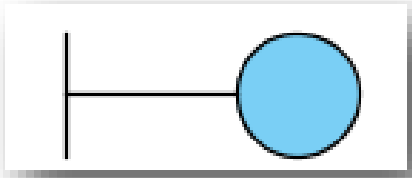
UML Notation	Analysis object Name	UML Class with stereotype << >>	3 – tier architecture	Purpose
	boundary	<< Boundary >> loginForm	presentation layer	Handle user input & system output
	control	<<Control>> loginController	Business logic layer	Handle business process
	entity	<<Entity>> Login	Data layer	Handle data access

3-Tiers/Layers Software Architecture

Divides application software into 3 “independent” layers



View layer (boundary object) responsibilities



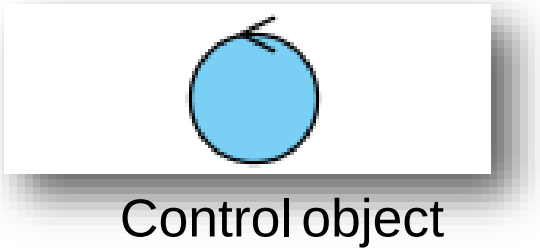
Boundary object

- Accepts and validates submission from user
- Relay service requests to business logic layer
- Receives unformatted response from business logic layer
- Format and displays responses to the user

Business Logic layer (control object) responsibilities

process

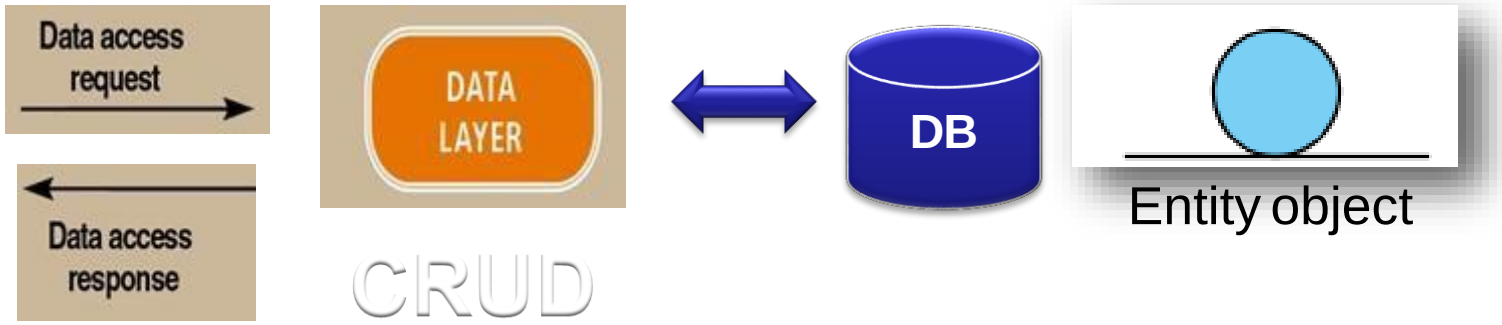
process



Control object

- Receives service request from the view layer
- Process & Relay data access request to the data layer
- Receives & process data access response from the data layer
- Return unformatted response to view layer

Data layer (entity objects) responsibilities



CRUD

- Receives data access requests (insert, retrieve, update, delete) from the business logic layer
- Carry out the data access requests (insert, retrieve, update, delete) in the database
- Return an appropriate data access response to the business logic layer

Advantages of the 3 Layer Architecture

- Separation of concerns.
- Promotes code reuse
- Easy to implement changes
- Enables parallel development of the different tiers of the application.
- Improved data integrity
- Improved security

NANYANG POLYTECHNIC

IT2755 & IT1573

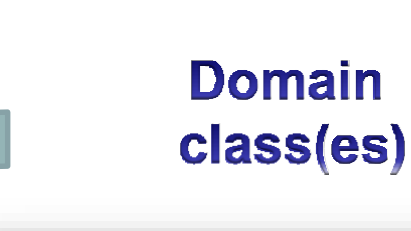
Software Engineering

Topic: Solution Class Diagram

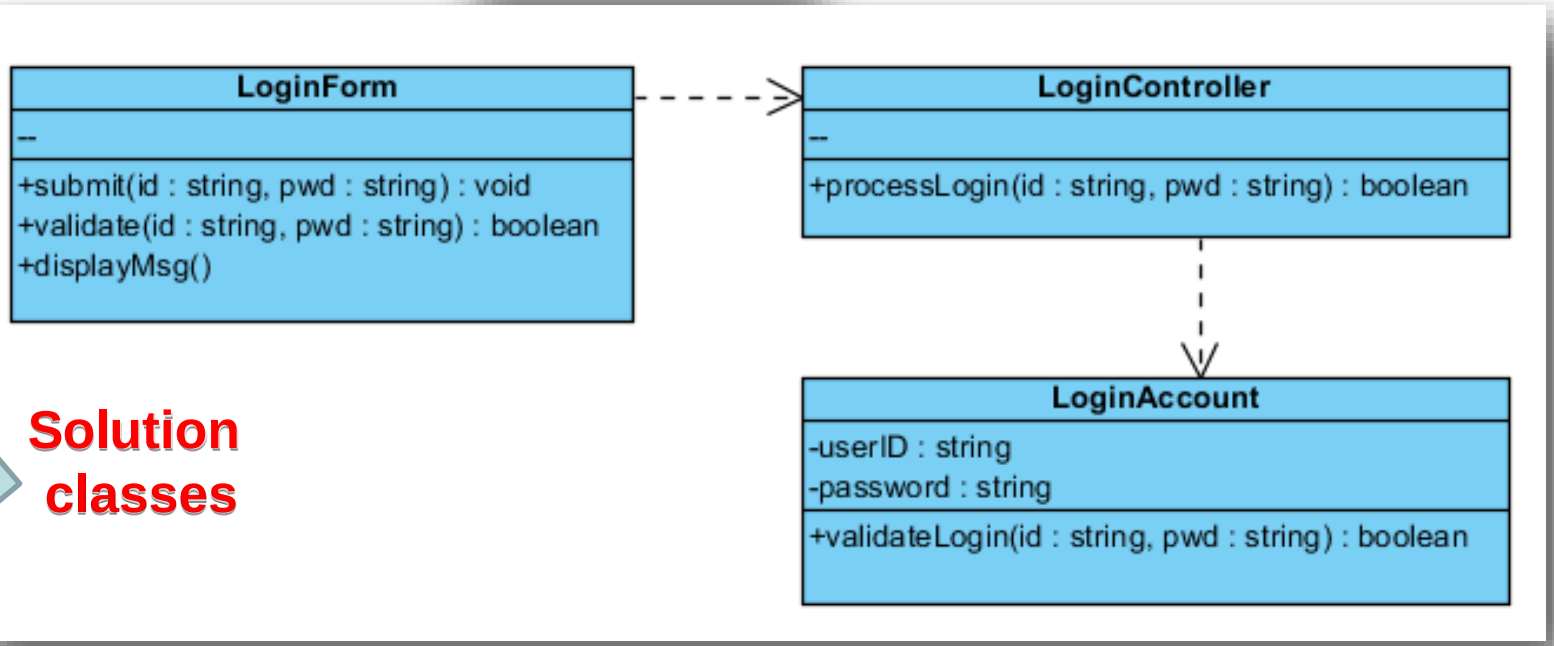
From Domain class diagram to Design/Solution class diagram

code

Domain class(es)

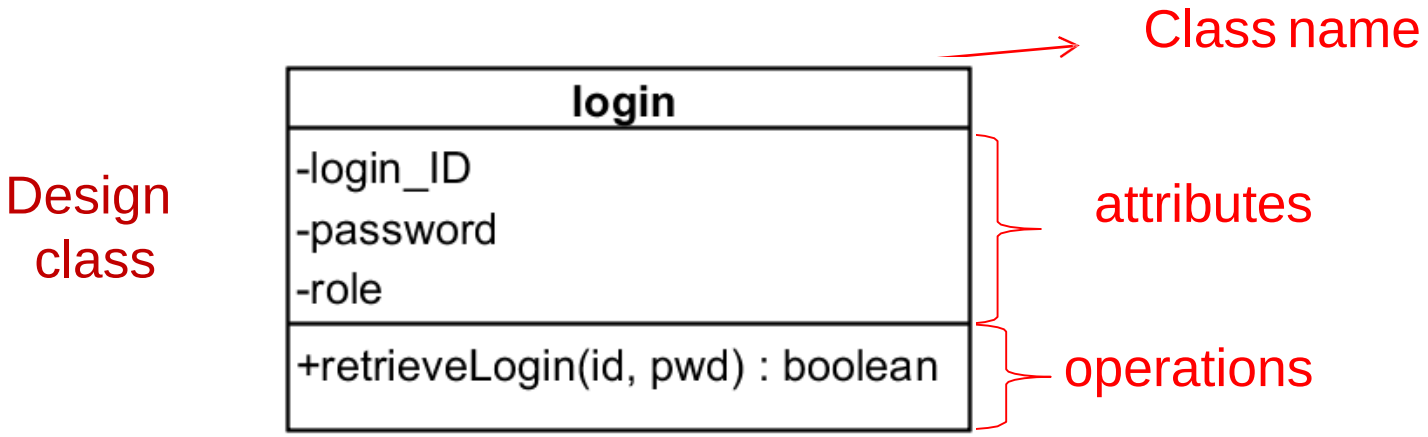


HOW?



Solution classes

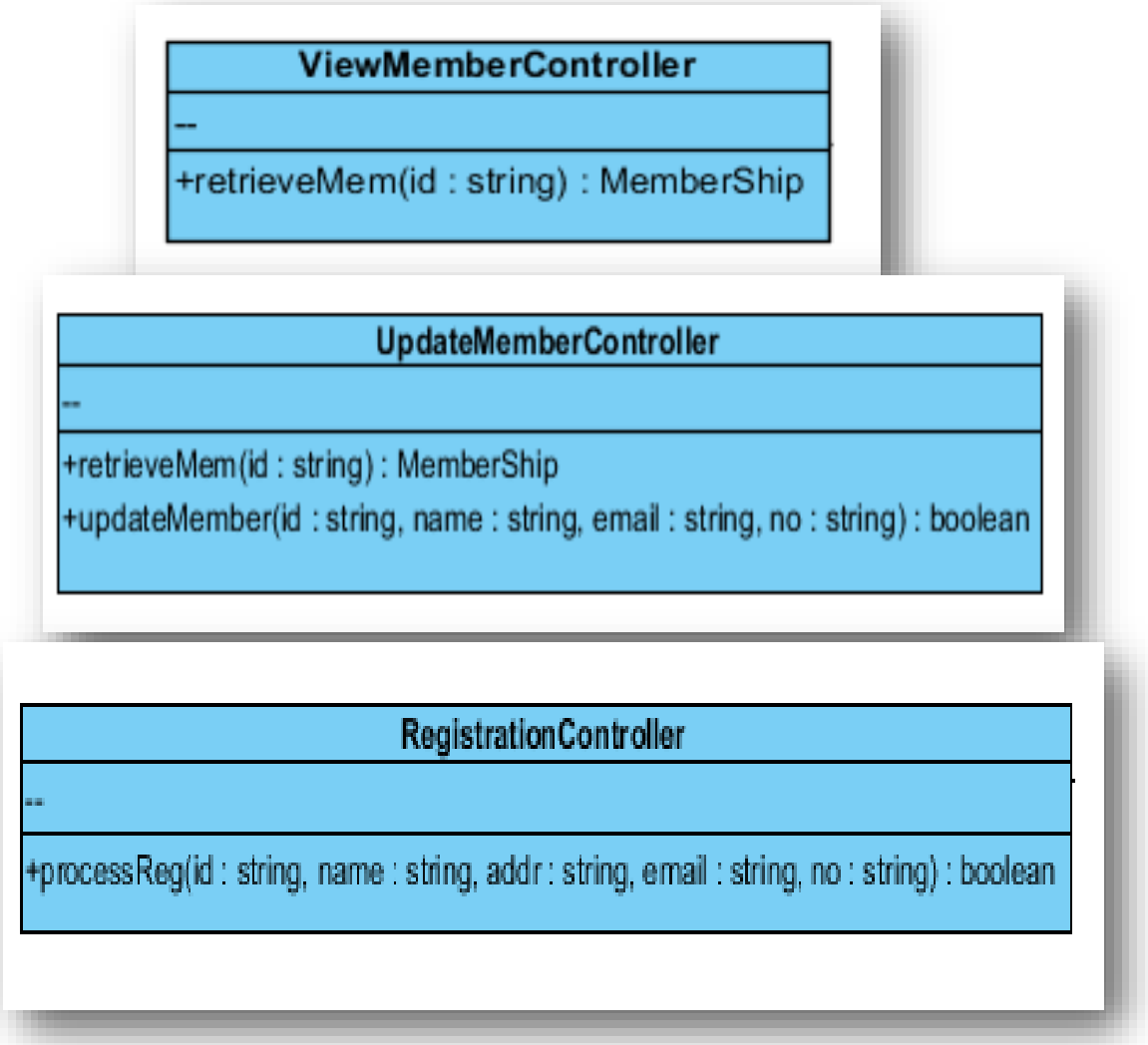
UML Design/Solution Class



- A design class consists of : class name, its attributes and
  its operations (methods)
- The domain and design models show the definition of a class and how it is related to other classes
  (i.e. the structural aspect of the system)

D – Design Principles

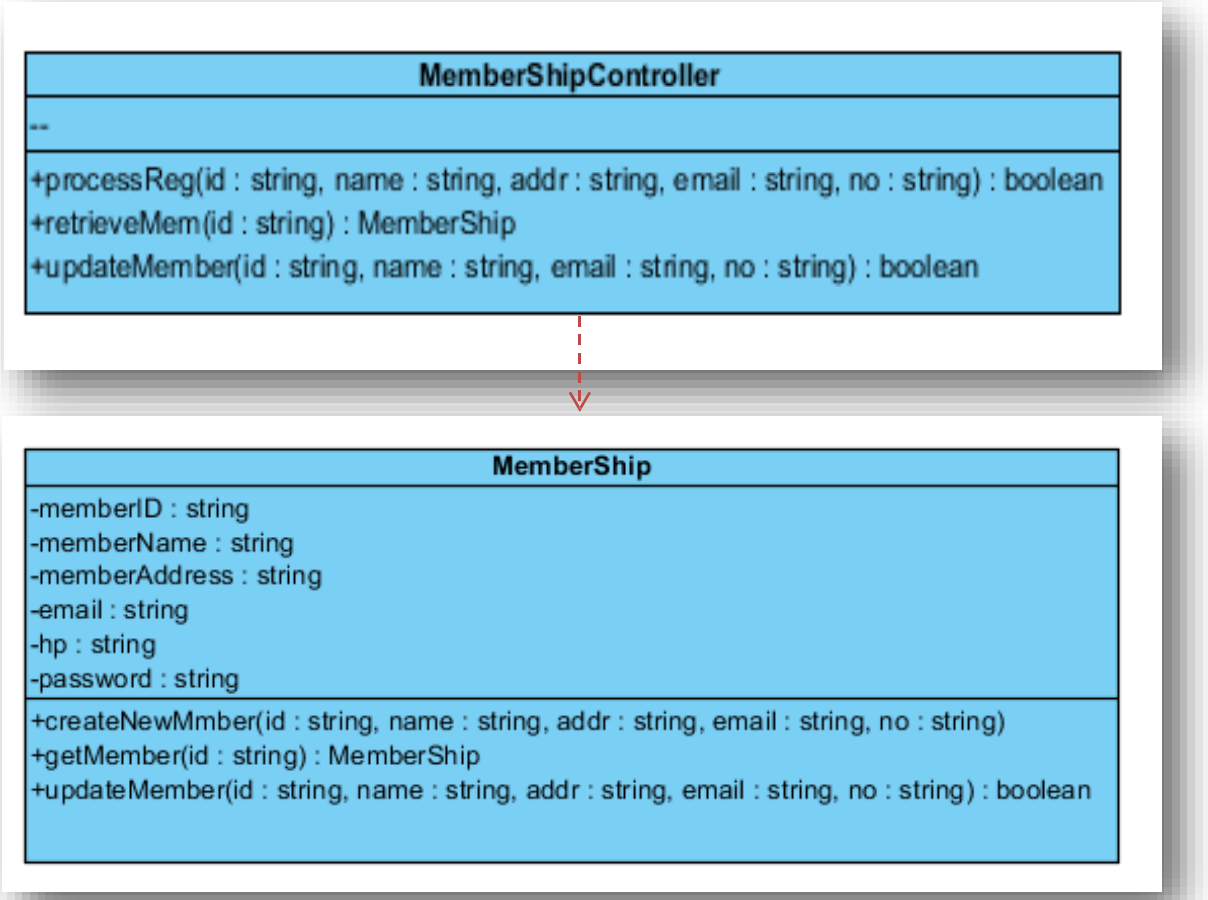
Step 3 – Considering 2 fundamental design principles : cohesion and coupling



Suggestion:
Instead of having 3 controller classes each handling *related operations* such as registration, updating and retrieving, we have just ONE controller class to handle all of them.

This helps us to achieve low coupling. So instead of depending on 3 classes, now we have one class to manage the membership records For registration, updating and retrieving.

Revised solution class diagram



Advantage?

Low coupling & High cohesion

This improvement also helps to achieve high cohesion for the Membership controller class as we focus all membership operations in this class.

Apply 2 fundamental design principles : cohesion and coupling

Cohesion:

Measures how focused a class is, that means; is the class doing what it is supposed to do? The higher the functional cohesiveness, the better is the class design.

Considering 2 fundamental design principles : cohesion and coupling

Coupling:

Coupling indicates how one class is dependent on other classes.
The more dependent a class is on other classes, the higher the coupling.
High coupling creates ripple effect when changes are introduced which is **NOT** desired.

We want to achieve
HIGH cohesion and **LOW coupling**.

Separation of concerns (responsibilities)

The process of separating the design of classes into distinct features that minimalize overlap in functionality as much as possible.

Concerns (responsibilities) refers to features or behaviours of a class.

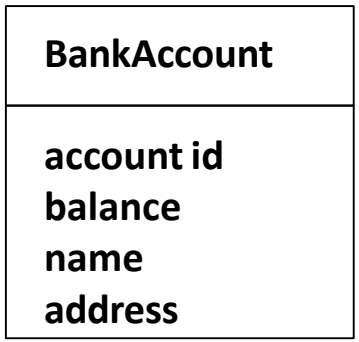
An example of separation of concerns is the 3-tier system architecture:

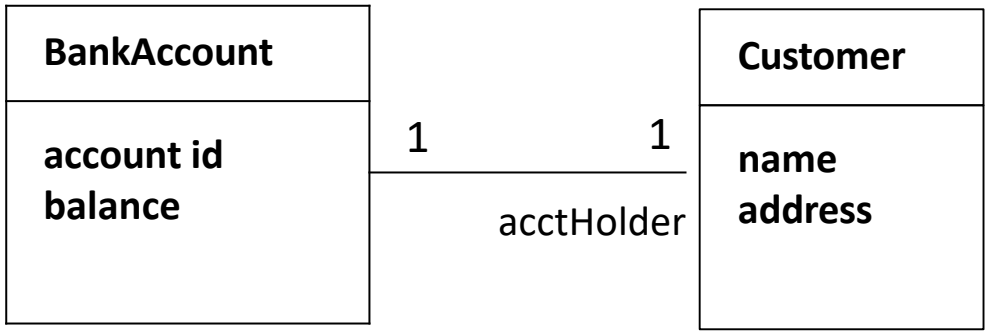
Presentation layer : concerns of this layer : user interface

Business logic layer: concerns of this layer- processing logic

Data layer: concerns of this layer is handing data manipulation

Example of High Cohesion





Low cohesion (bad design, why?)

High cohesiveness (better design)
Don't worry about how to know who owns the account.
It is taken care of by the association relationship.

Another good reason for achieving low coupling

A class with high (or strong) coupling relies on many other class. It is not desirable because:

a) Changes in related classes force local changes
b) Harder to understand in isolation
c) Harder to reuse because its use requires the additional presence of the classes on which it is dependent

But it does not mean no coupling is good.

in OO system, objects are supposed to interact with one another to execute the tasks in a system.

NANYANG POLYTECHNIC

IT2755 & IT1573

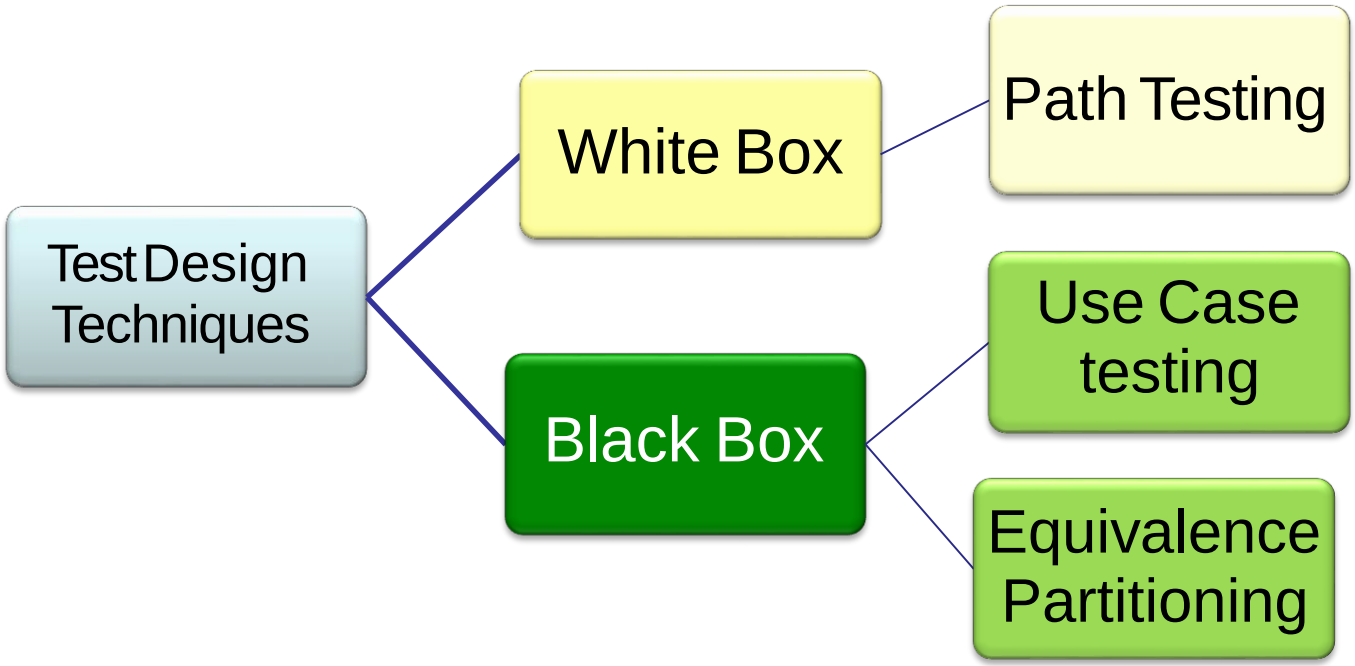
Software Engineering

Topic: Software Testing

Learning Outcomes

1. Explain the aims of white box test
2. Explain the aims of black box testing
3. Identify test cases for black box testing
4. Identify the test class partitions using equivalence partitioning
5. Identify test cases using white box testing
6. Calculation of **cyclomatic complexity**

Test design techniques – focusing on how test cases can be derived



Test Design Techniques

Black Box Testing

- Focus to test the specified functions that a product has been designed to perform without knowing its code and the programming logic

•Normally used in Integration & System testing

White Box Testing

- Focus to test the code and the programming logic against design specifications.
- Normally used in Unit testing

Example of developing test cases (Similar to Tutorial 11)

An application has been developed for the calculation of parking charges for vehicles at a country club.

The charges are given as follow:

- For members, $2.00/hr if vehicle parks for less than and equals to 2 hours. $1.5/hr otherwise.
- For non-members, $3.00/hr if vehicle parks for less than and equals to 2 hours. $2.00/hr otherwise.
- If parking duration is more than 5 hours, a flat rate of $15/day is charged.

Test case ID	Scenario		Input test data to be used (membership type, duration)	Expected result
	Membership type	Parking duration		
1	Member	2 hrs >= duration	Member, 2 hrs	$4
2	Member	2 hrs < duration <=5 hrs	Member, 4 hrs	$6
3	Non-Member	2 hrs >= duration	Non-Member, 2 hrs	$6
4	Non-Member	2 hrs < duration <=5 hrs	Non-Member, 4 hrs	$10
5	Either type	5 hrs <= duration	Either type, 10 hrs	$15

NANYANG POLYTECHNIC

IT2755 & IT1573

Software Engineering

Topic: Software Quality

2018 S1
2019 S1

What is Software Quality

- Quality is defined as the degree to which a system, computer or process meets customer or user needs or expectations.
- Functional quality
  - Fitness
  - How well software complies/conforms to design specification
  - How software compares to competition
- Structural quality
  - How software meets non-functional requirement (robustness, maintainability
  - Degree to which the software was produced correctly

3 Stakeholder Views On Software Quality

- System user view quality as : fitness for use
  - Reliability, Usability
  - System developer view quality as compliance to requirements
    - Testability, Portability
  - Maintenance team view quality as how easy to change the system
    - Maintainability, Efficiency

Software Quality Attributes

Attributes	Description
Functionality	The capability of the software product to provide function which meet stated and implied needs
Reliability	The capability of the software product to maintain a specified level of performance
Usability	The capability of the software product to be understood, learned, used, and attractive to the user
Efficiency	The capability of the software product to provide appropriate performance, relative to the amount of resources used
Maintainability	The capability of the software product to be modified. Changes may include corrections, improvements or adaption to different environment and functional requirements
Portability	The capability of the software product to be transferred from one environment to another

CMMI

- CMMI : Capability Maturity Model – Integration
  - A process improvement framework based on the best practices from successful companies. Aims to built up a quality culture for organizations.
  - Developed by Software Engineering Institute of Carnegie Mellon University.
  - Software development processes are grouped and characterized as one of the 5 maturity levels evolve towards achieving a mature level for the whole organization.
  - To achieve a certain maturity level, a number of process areas (PA) must be in place. These process areas indicate important issues that have to be addressed in order to reach a given level.

Good Luck !

Be honest!