Lab 2

1. Use Case diagram
2. Use Case descriptions
3. Class Diagram
4. Sequence Diagram
5. Dialog Map

Use Case descriptions

1. Check If description is matched with the diagram
2. Extended Use Case
3. Describe the detailed process of how user conduct this function.
4. Do not develop too simple function. Otherwise you omit a lot of steps or it is not necessary to build an independent use case.

Class Diagram

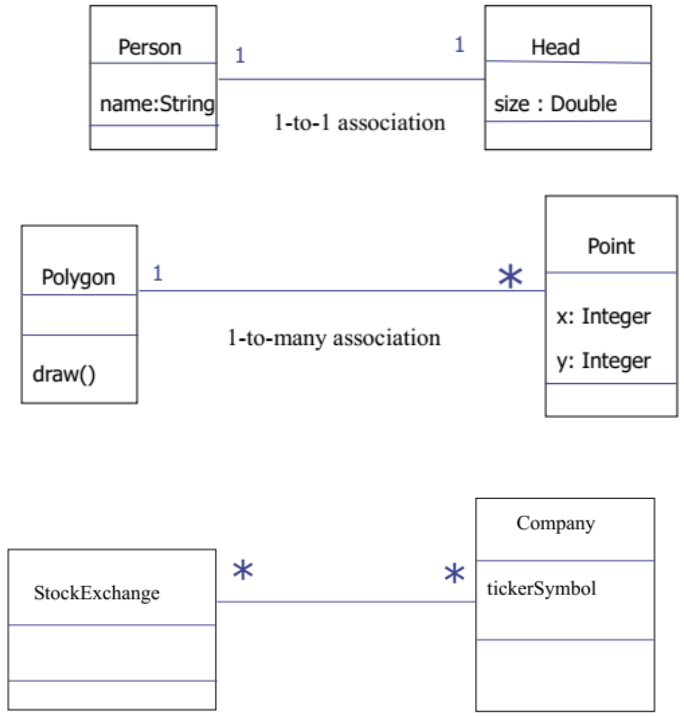
1. Entity: Objects representing system data
2. Boundary: Objects that interface with system actors (e.g. a user or external service)
3. Controls: Objects that mediate between boundaries and entities.

	Boundary	**Controls**	**Entity**
<Actor>	√	×	×
Boundary	×	√	×
Controls	√	√	√
Entity	×	√ (be called)	√ (association)

Class Diagram

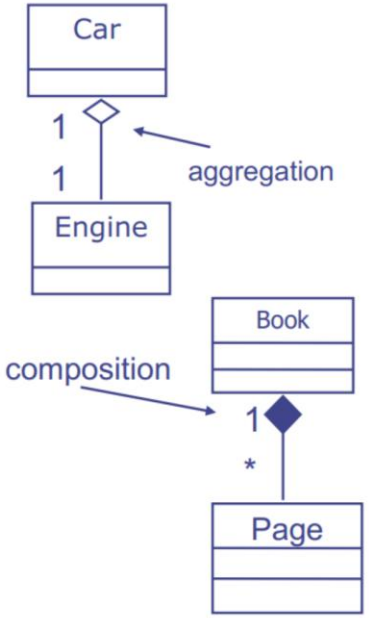
1. Relationship lines

Association: association / Aggregation / Composition



Class Relationships

- **aggregation**: "is part of"
  - symbolized by a clear white diamond
- **composition**: "is entirely made of"
  - stronger version of aggregation
  - the parts live and die with the whole
  - symbolized by a black diamond



Class Diagram

1. **Relationship lines**

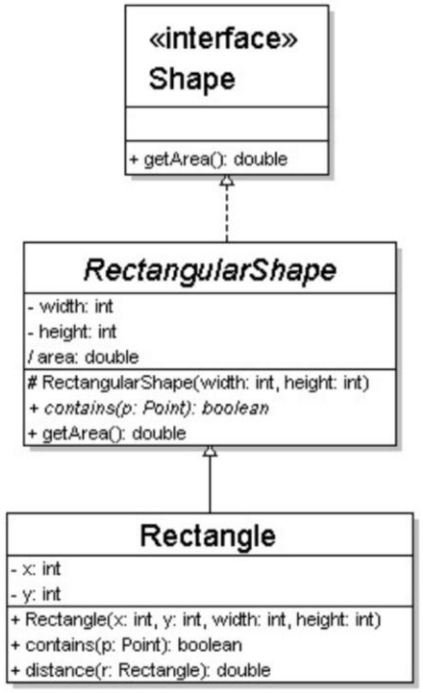
   Association: association / Aggregation / Composition

   Generalization: inheritance / interface

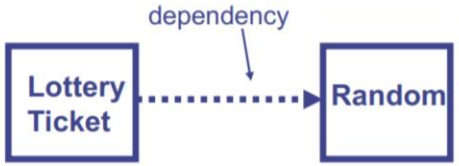
   dependency

Class Relationships

- generalization (inheritance) relationships
  - hierarchies drawn top-down with arrows pointing upward to parent
  - line/arrow styles differ, based on whether parent is a(n):
    - class:
      solid line, black arrow
    - abstract class:
      solid line, white arrow
    - interface:
      dashed line, white arrow
  - we often don't draw trivial / obvious generalization relationships, such as drawing the Object class as a parent



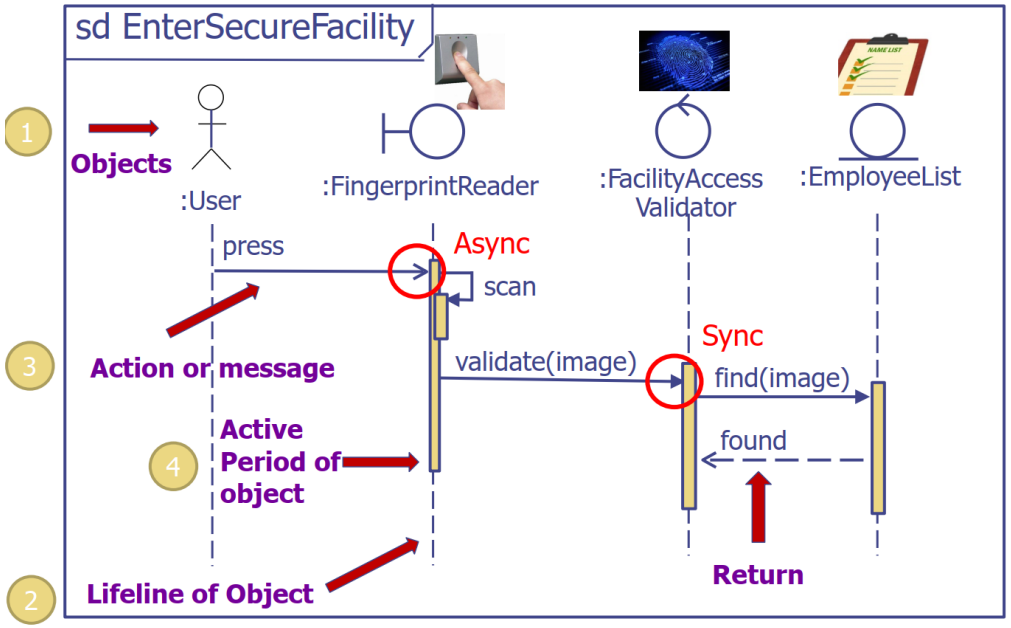
- **dependency**: "uses temporarily"
  - symbolized by dotted line
  - often is an implementation detail, not an intrinsic part of that object's state



Class Diagram

1. Relationship lines
    - Association: association / Aggregation / Composition
    - Generalization: inheritance / interface
    - Dependency
2. You can add the class type upon the name of class for clarify.

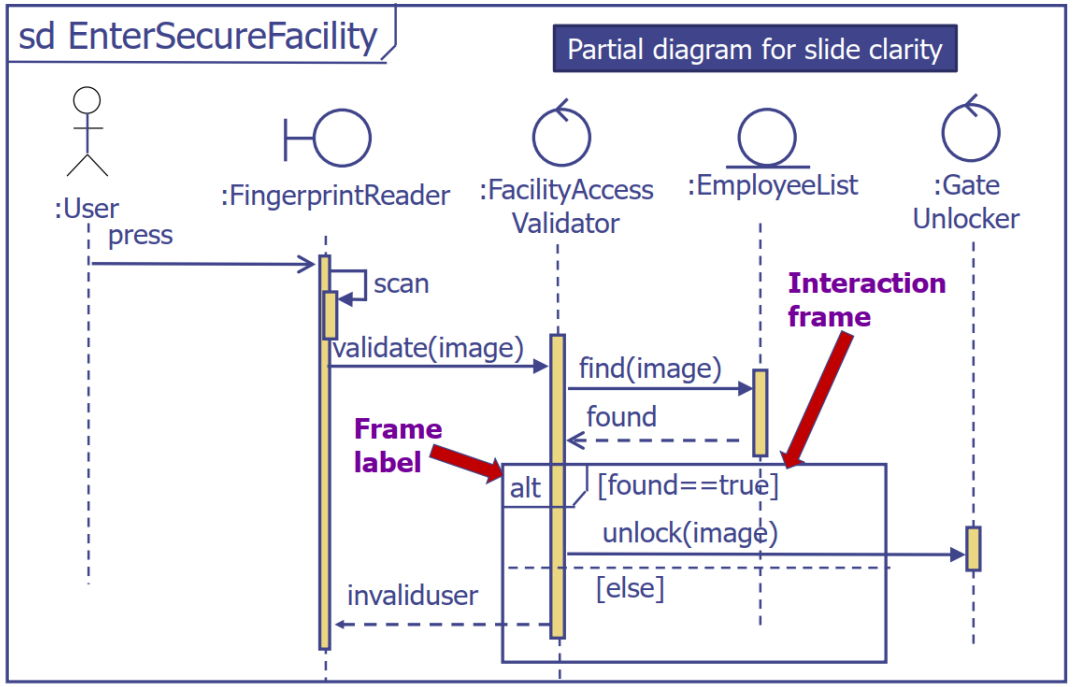
Sequence Diagram



1. Asynchronous vs Synchronous
   1)Arrow
   2)Life bar
   3)Do not send parallel synchronous message
2. Position of the alt Frames.
3. The logic of sequence should be continuous.
4. Role of three types of classes
5. Implement sequence for each use case
6. The class definitions of the class diagram and the sequence diagram need to be consistent.

Sequence Diagram

Example: EnterSecureFacility Sequence Diagram



1. Asynchronous vs Synchronous
   1)Arrow
   2)Life bar
   3)Do not send parallel synchronous message
2. Position of the alt Frames.
3. The logic of sequence should be continuous.
4. Role of three types of classes
5. Implement sequence for each use case
6. The class definitions of the class diagram and the sequence diagram need to be consistent.

Dialog Map

1.Dialog Map is a state machine Diagram.

Dialog Map≠UML Activity Diagram

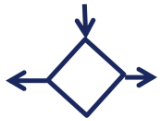
UML Activity Diagram

- Activities are the execution of one or more basic operations. Represented by a rounded rectangle.

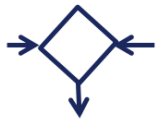
- Control flow between activities represented by arrows.

- Forks are when control flow has one input and two or more outputs (go to concurrent activities).

- Joins are when two or more control flows come together into a single control flow.



- Decision boxes (branches) are when control flow can go in one of several directions depending on a condition.



- Merge several branches into one branch

These items are used for activity diagram, not dialog map.

Dialog Map

1.Dialog Map is a state machine Diagram of system UI.

A dialog map shows the state of your system / where the user locates,

and how these states are changed between each other.

Do not need to display actions.

Dialog Map≠UML Activity Diagram

2. Dialog Map is not a series of separate diagram for each use case!

3. Each state must have entry and exit.

4. Each state change should contain condition