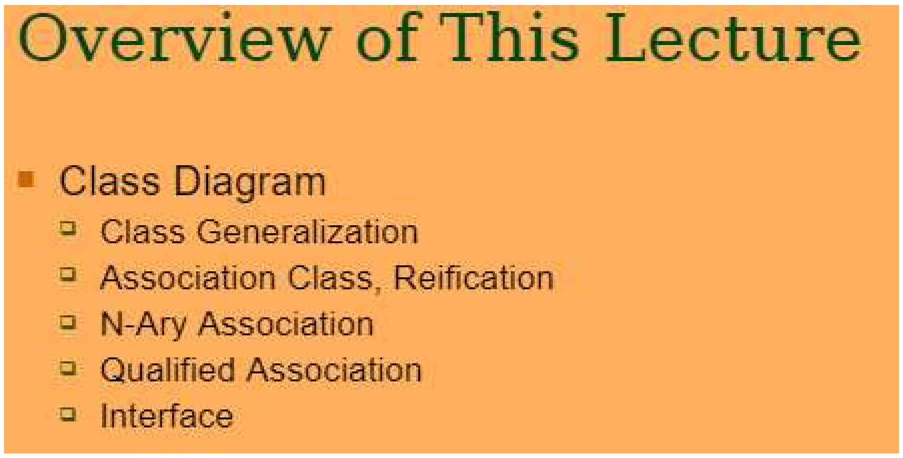
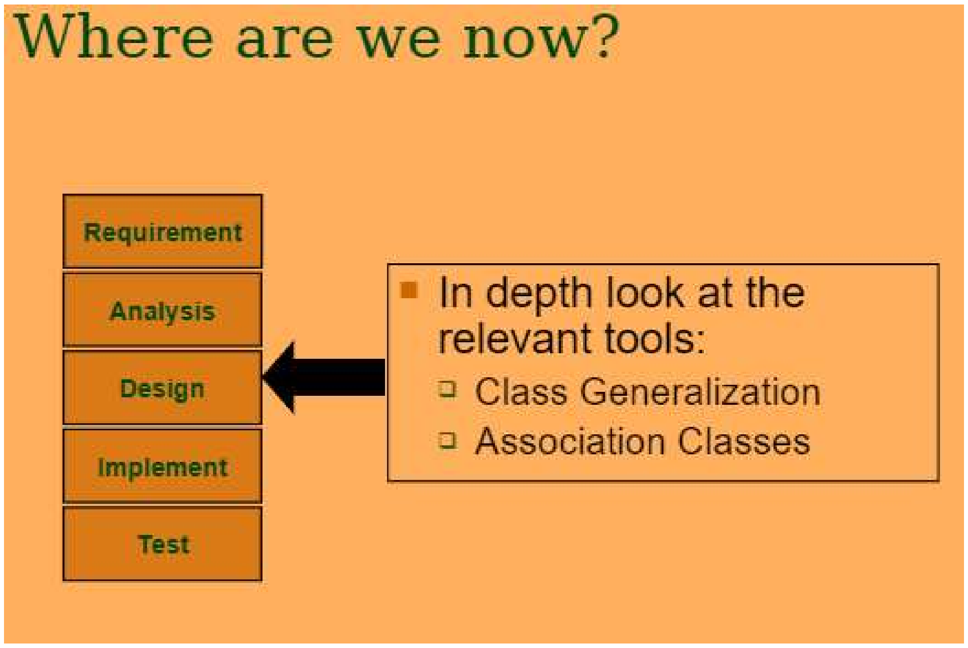
Lecture 6

Thursday, February 23, 2023 9:40 AM



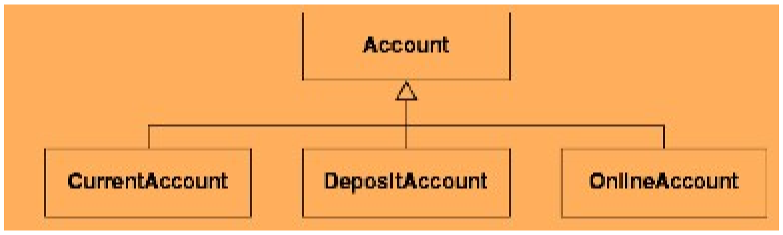
Overview of This Lecture

- Class Diagram
  - Class Generalization
  - Association Class, Reification
  - N-Ary Association
  - Qualified Association
  - Interface

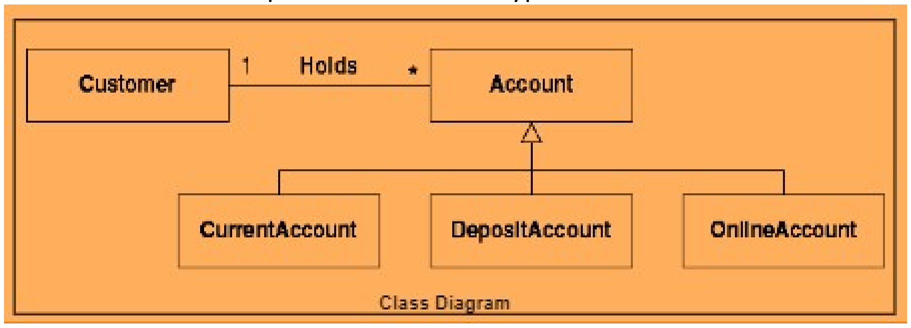
- 

Where are we now?

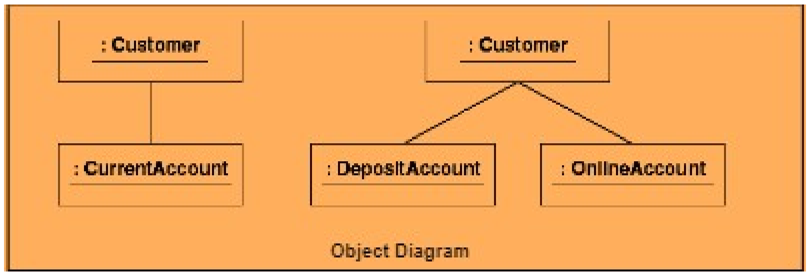
- In depth look at the relevant tools:
  - Class Generalization
  - Association Classes

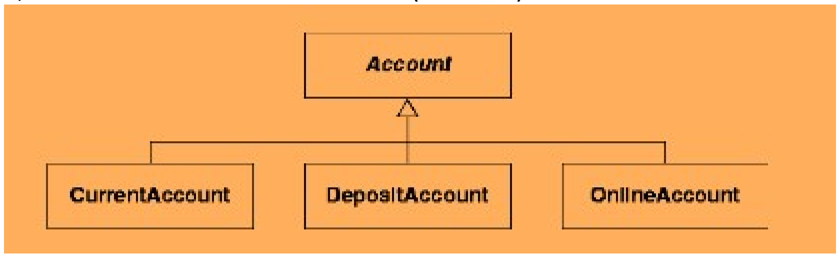
- Class generalization
  - Common btwn classes (shared attributes and operations) can be extracted (generalized) into a superclass
  - Ex: Bank accounts
    - 3 types of accounts: current account, Deposit account, and Online account
    - Use super class Account to capture similarity:
    - 

- Meaning of Generalization
  - In UML, generalization means substitutability- where instance of superclass is expected, an

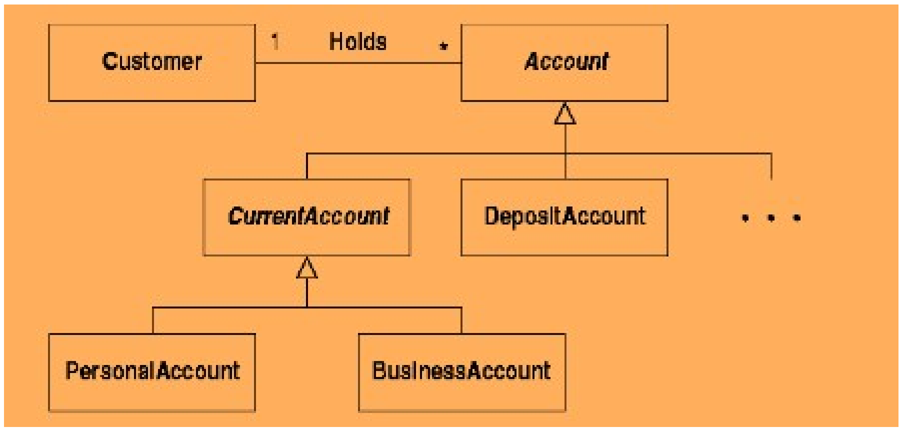
instance of a subclass can be substituted for it without any probs
- Ex: Customer can hold multiple accounts of diff types:
  - 

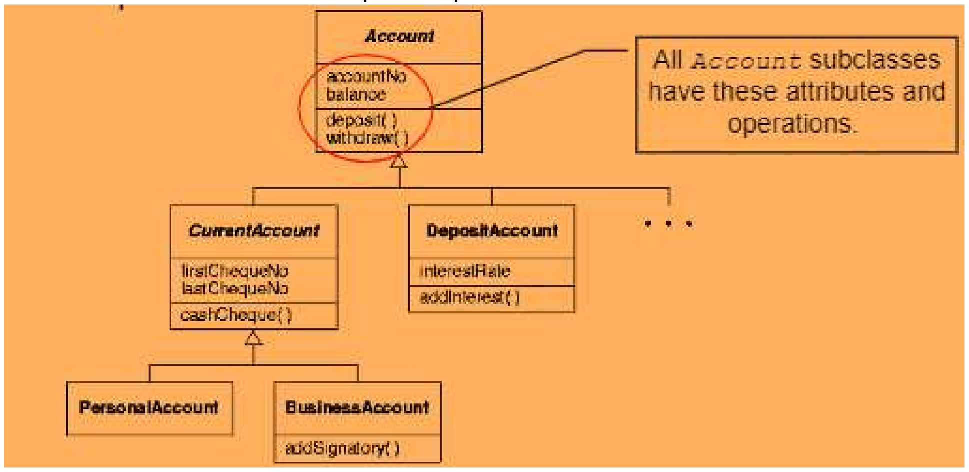
- Where instance of superclass expected, instance of subclass can be substituted for it w/out any probs
  - Association in pic (^) implies that instances of Customer and Account classes can be linked at run-time
  - Bc of substitutability, Account instance can be replaced by instance of any of the subclasses of Account

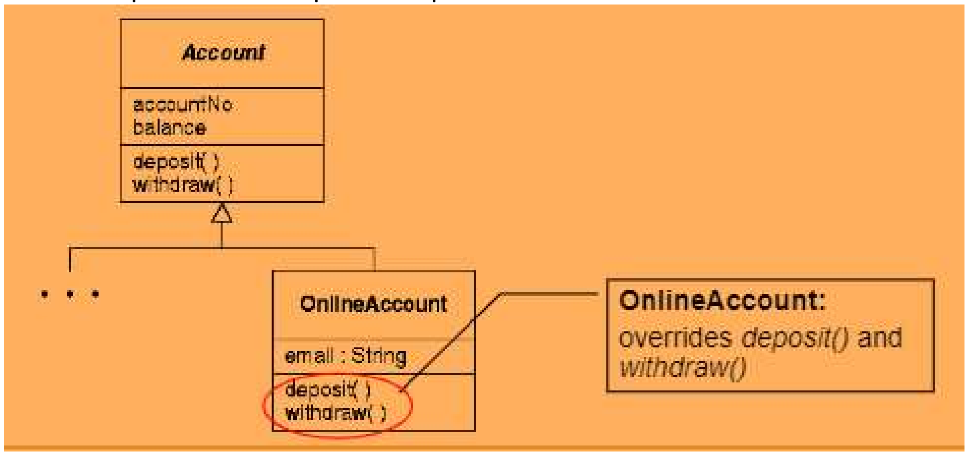
- Substitutability
  - Model connects Customers to Accounts, but instance of any subclasses can be substituted for an account object (links demonstrates polymorphism)
  - Ex: links btwn Customer and diff Accounts objects rep instances of the association btwn Customer and account classes
    - 

- Abstract Class
  - Superclasses often defined solely to group together shared features:
    - May not make sense to have an instance of a superclass
    - If ^ happens, define the class as abstract
  - In UML, abstract classes written in italic (*Account*)
    - 

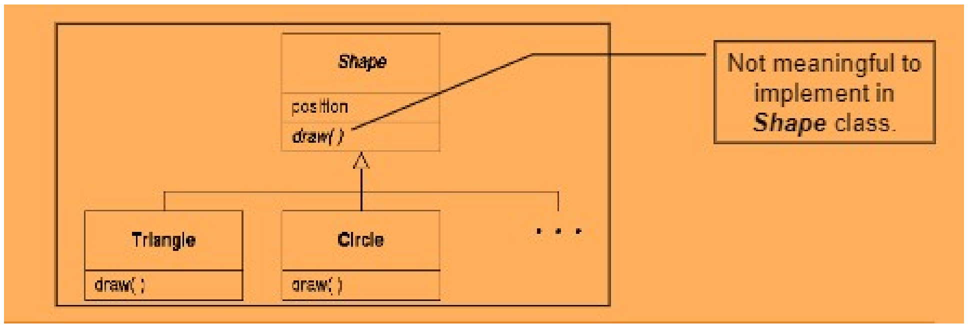
- Generalization Hierarchies
  - Generalization process can be carried out @ diff lvl if need
  - hierarchy can dev to as many lvls as req

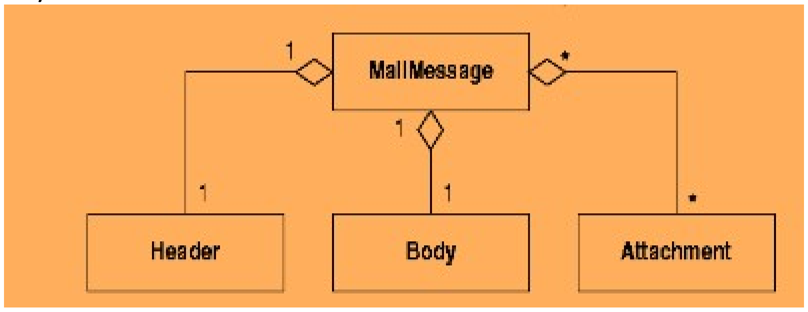
- 

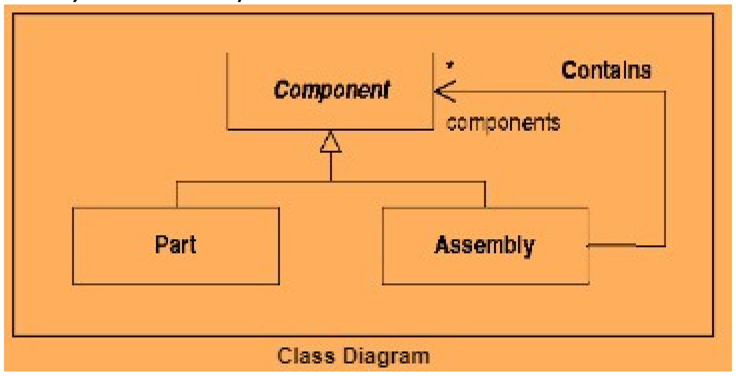
- Inheritance
    - Subclass inherits all attributes and ops of superclass
        - 

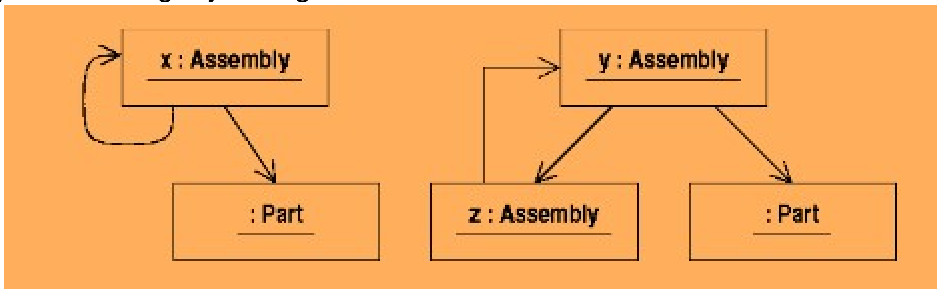
- Modifying Subclasses
    - Subclasses can:
        - Add features to model special properties
        - Override operations to implement specialized behaviour
        - 

- Abstract Operations
    - Some ops can't be implemented in abstract classes (just define them as abstract and override them in subclasses)
    - Use italic for operation name

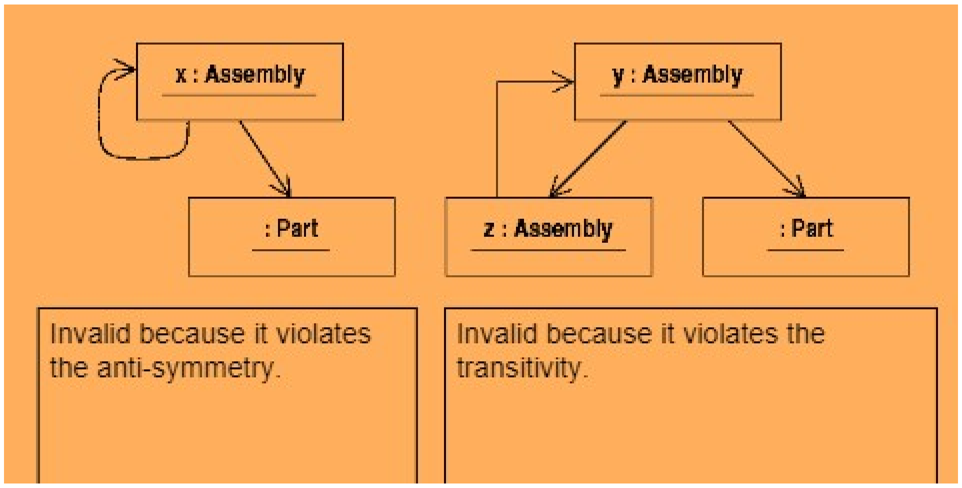
- 

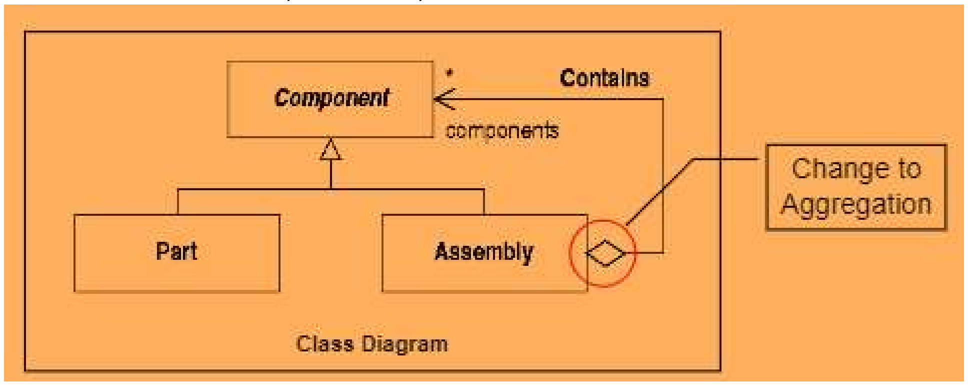
- Association: Aggregation
  - Aggregation is a sub-case of an association:
    - Informal model: the 'whole-part' relationships
    - Object is 'part of' another object
    - Standard association annotation is still applicable
  - Shown by hollow diamond
    - 

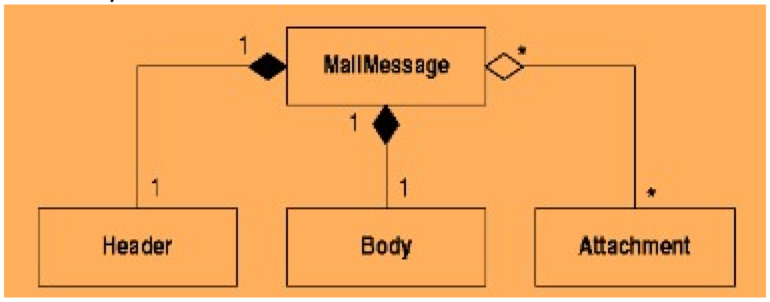
- Cyclic object Structures
  - Aggregation useful for ruling out invalid cyclic object structures
  - Ex: *Assembly* can have any numb of *Parts* and other *Assemblies*
    - 

  - Pic (^) lets following object diagrams
    - 

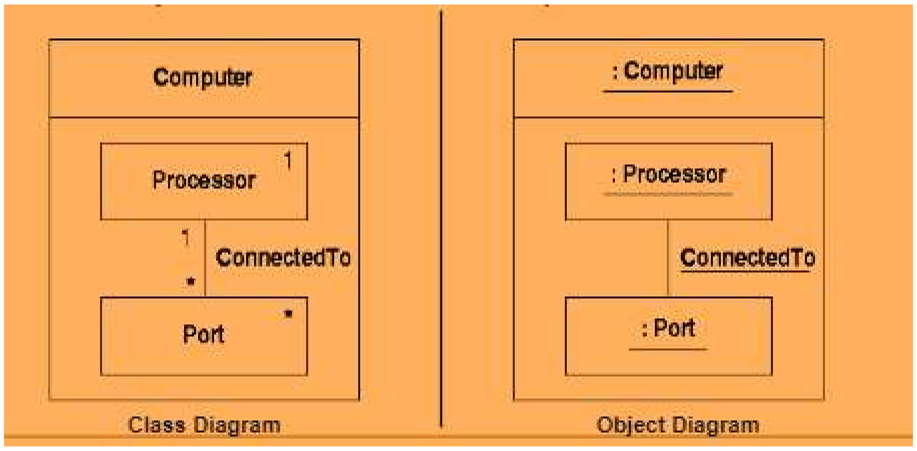
  - Don't like it tho bc
    - not reflective of the real world
    - Traversing the hierarchy can cause infinite loops
  - Two object diagrams are invlalid

- ▪ 
x : Assembly
: Part
y : Assembly
z : Assembly
: Part
Invalid because it violates the anti-symmetry.
Invalid because it violates the transitivity.

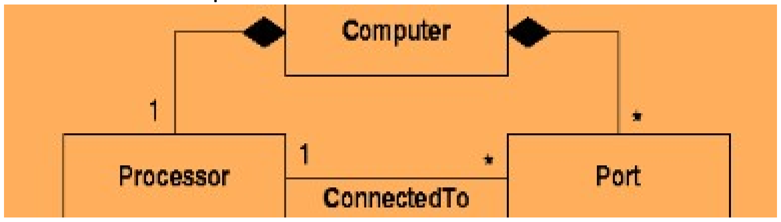
- Property of Aggregation
    - Aggregation forbids cycling structure bc its
        - Anti symmetric- object can't link to self
        - Transitive- if a links to b, and b to c, then a links to c
        - 

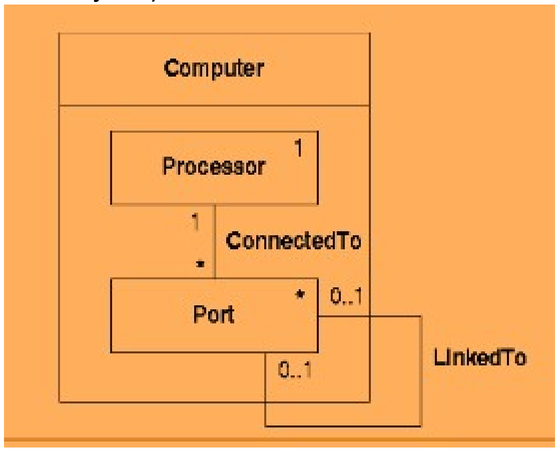
- Association: Composition
    - Composition stronger form of aggregation
        - Pts only belong to one composite @ a time
        - Pts destroyed (deleted) when composite destroyed
        - Shown by filled diamond
        - 

    - Alternative notation lets association to be defined in composites
    - Multiplicities associated w/ composition relationships now class multiplicities

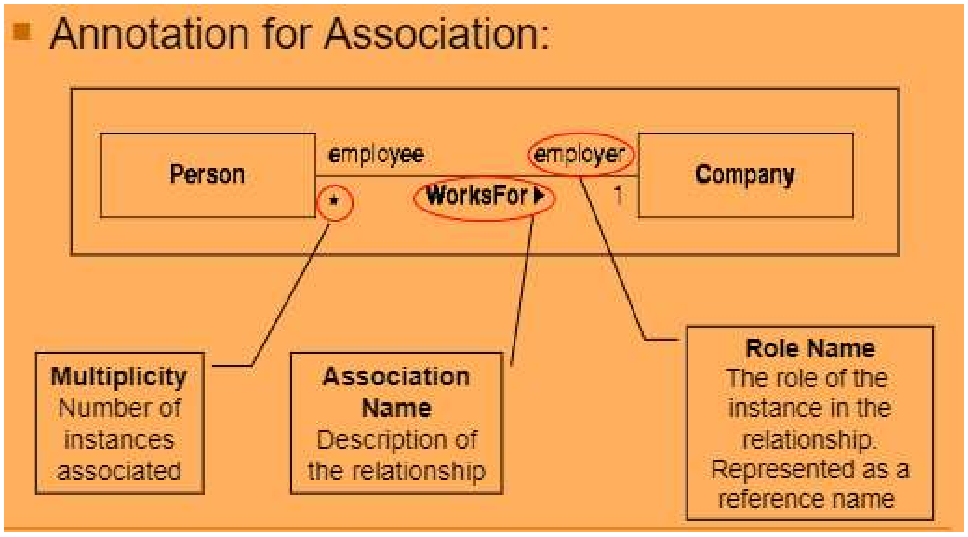
            - 

    -

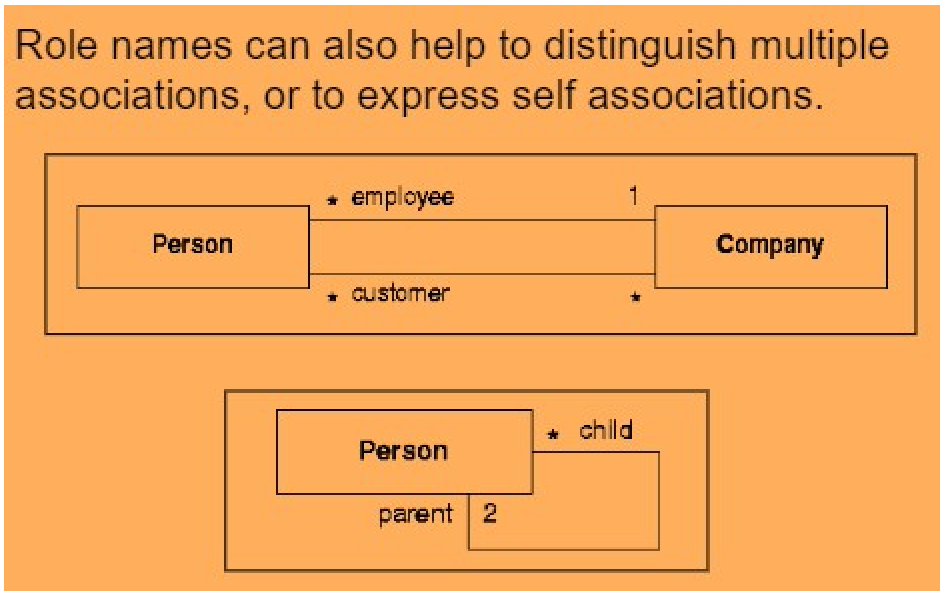
- Component Relationship
    - 

    - (^) Model fact the:
        - Processor must belong to a computer
        - Port must also belong to a computer
- Composite Boundary
    - Associations can cross boundary to link objects in diff composites
    - Ex: network modelling (ports linked by the instances of the association can belong to diff composite objects)
        - 

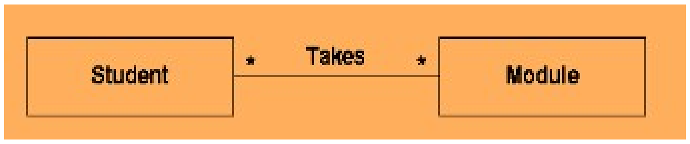
- Association Review
    -

- 

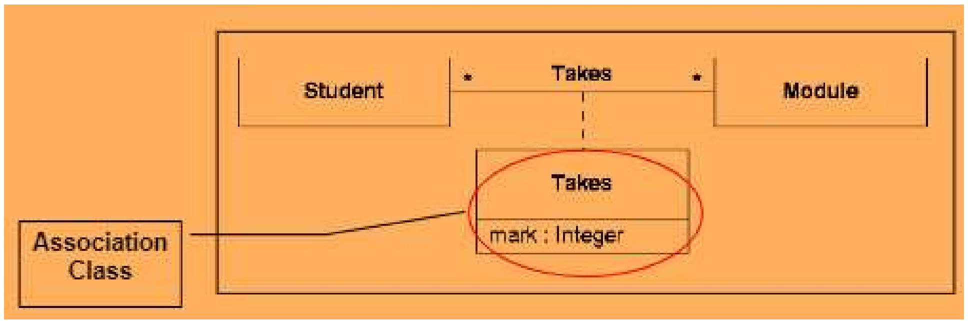
  Annotation for Association:

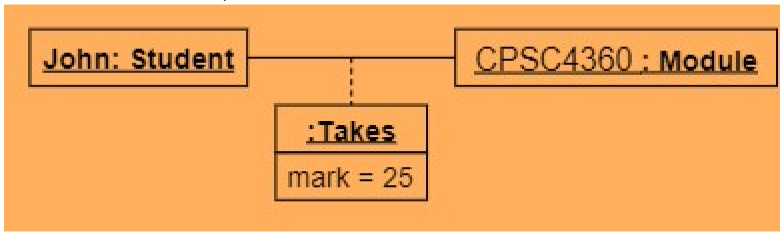
  Person
employee
employer
Company
*
WorksFor ▸
1
Multiplicity
Number of instances associated
Association Name
Description of the relationship
Role Name
The role of the instance in the relationship. Represented as a reference name
-
- 

  Role names can also help to distinguish multiple associations, or to express self associations.

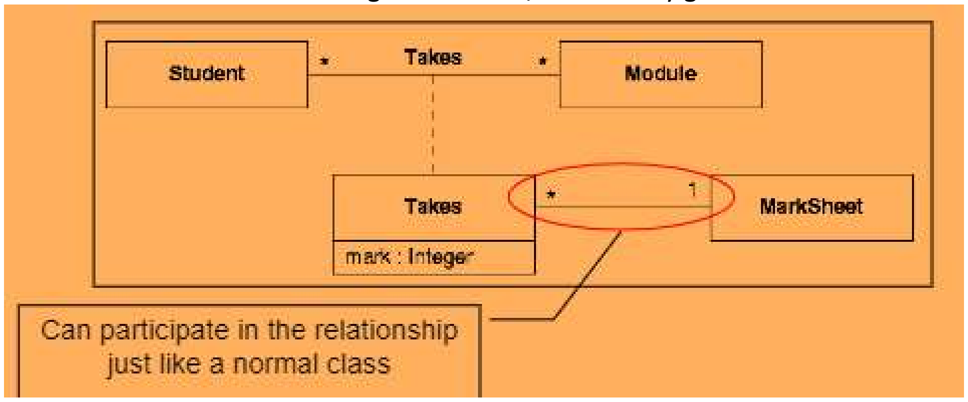
- Property of Links
  - Cases where info abt links needs to be kept
  - Ex:
    - 

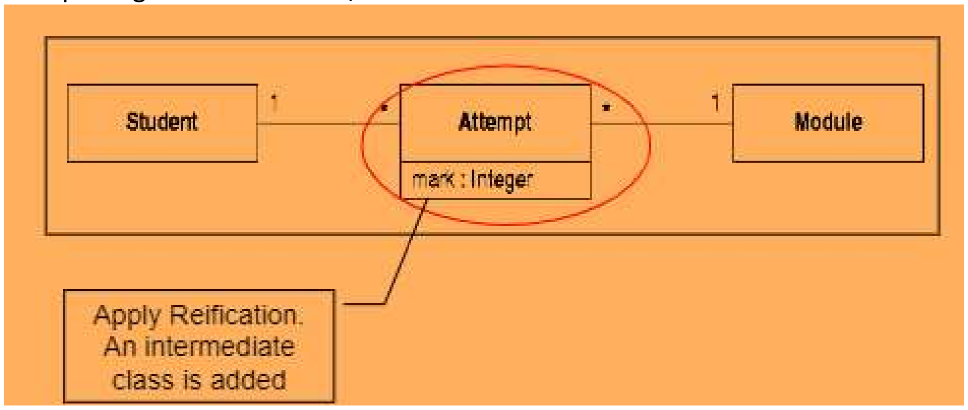
    - Student takes a module and gets a mark for it
    - Mark only makes sense if we know the student and the module
    - Mark not simply an attribute of either class
- Association Class
  - Association Classes can be added
  - Combo of both association and class:
    - Association: can connect 2 classes
    - Class: allow attribute to be stored
  - Syntax: dashed line btwn association and class icon

- 

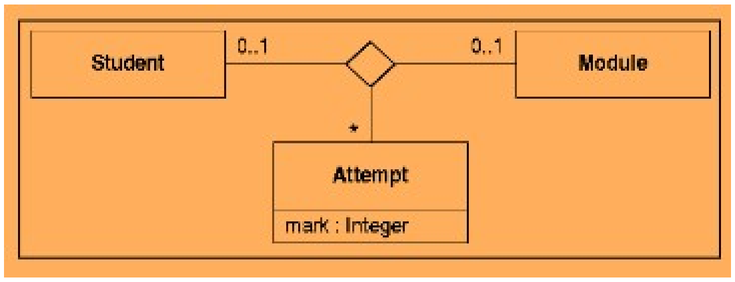
  - Ex: John took CPSC 4360, scored 25 marks
    - 

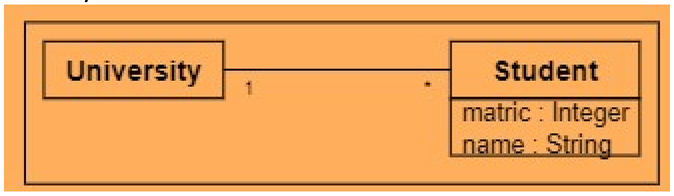
    - Takes instance is uniquely defined by both John:Student and CPSC4360:Module

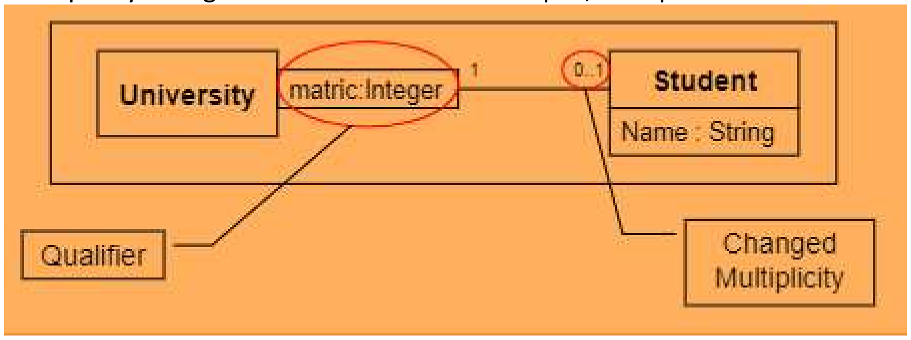
- Association Class Property
  - Their classes, so can participate in associations
  - Ex: MarkSheet lists all students taking a module w/ marks they got
    - 
Student
Takes
Module
Takes
mark : Integer
MarkSheet
*
*
*
1
Can participate in the relationship just like a normal class

- Reification
  - Another technique to capture association info is reification
  - Means "Replacing an association w/ a class"
    - 

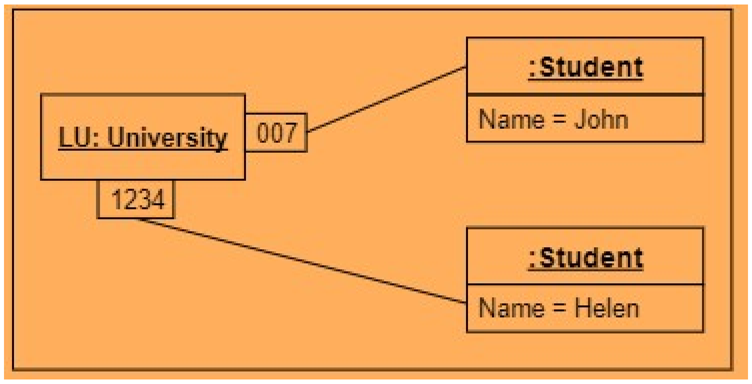
  - Reification has property of allowing students to take a module more than once, in case they failed first time
  - Property not possible for the association classes, when there is exactly one link btwn a student and a module
  - Ex: Using reification, we can now design object diagrams which model the following situation:
    - John takes CPSC4360 and score 25

            - John retakes CPSC4360 and score 99
- N-ary Association
    - Associations can connect more than 2 classes:
        - 

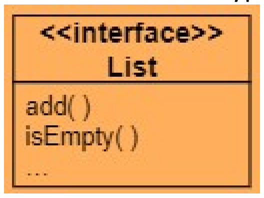
    - 3-way association could be used to store marks
    - Multiplicity @ given association end defines the numb of instances of the class @ that end that can be linked to a tuple of instances, on from each of the other ends of association
    - Ex: '*' specifies that each pair of student and module object can be linked to 0 or more Attempt objects
- Qualified Associations
    - In real world, 'objects' usually id by a unique value (aka key)
    - Ex: University uses Matriculation Number to id a student
        - 

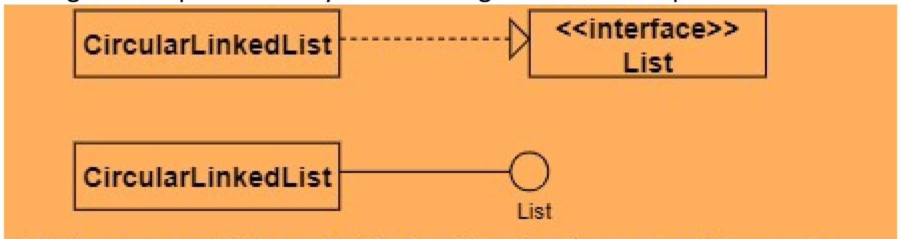
    - Problems:
        - Is matric number unique? Is it repped as such?
        - How do we know the matric can be used as a key?
    - Syntax:
        - Key is known as qualifier
        - Qualifier written in a square box attached to the class used to id the other party
        - Multiplicity changed to reflect the relationship w/ the qualifier instead of the class
        - 

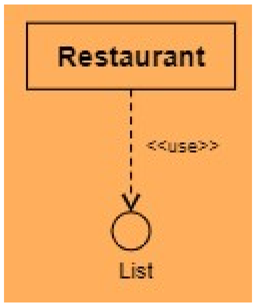
    - Semantics
        - Set of qualifiers is unique in context of attached class (like the University has a set of unique matric vals)
        - The multiplicity reflects the relationship btwn the id class and the qualifier
            - Like a Student is given exactly one matric in the University
            - Like a University object will use matric to uniquely id 0 or 1 Student (why 0?)
    - Ex:

- 

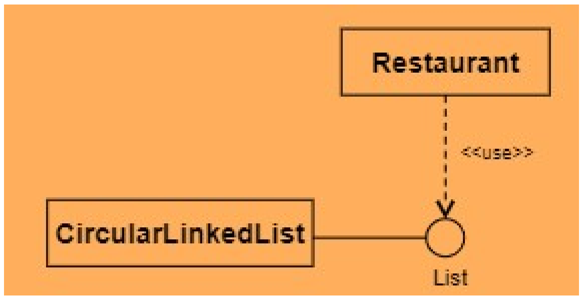
- Here is object diagram that is an instance of previous qualified association diagram:
  - Student John in LU university with matric no. 007
  - Student Helen in LU university w/ matric no. 1234

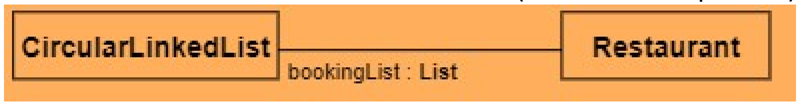
- Interface
  - UML interface is a named set of operations
  - Interfaces are used to charize the behavior of an entity
    - Shown as a stereotyped class
    - 

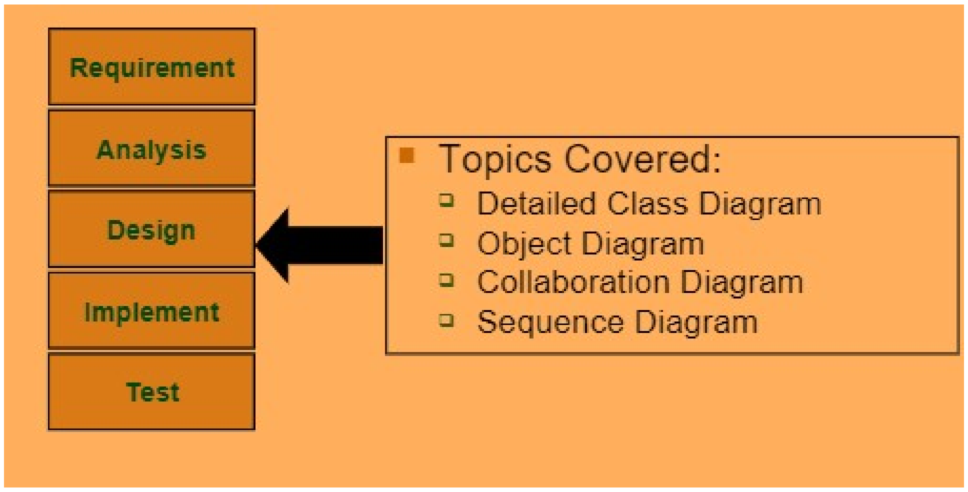
  - Generalization can be defined btwn interfaces
- Realizing an Interface
  - Class realizes an interface if it gives implementations of all the operations (like implements keywork in Java)
  - UML gives 2 equivalent ways of denoting the relationship
  - 

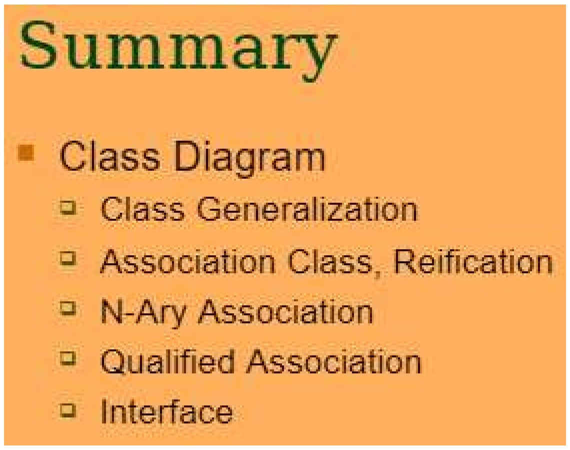
    Both represent "**CircularLinkedList** implements all operations defined by the **List** interface".
- Interface dependency
  - Class can be dependent on an interface- means that it makes use of the operations defined in that interface
  - Ex: Restaurant class makes use of the List interface
    - 

    - If the class realizing the interface is known, the dependency can be specified by

- 

- Info can also be attached to the role name (aka interface specifier)
- 

- Evo of Class diagram
  - Class diagram evolves w/ dev process:
    - Starts out as Domain Model:
      - Classes w/ name and attribute only
    - Refined to an Analysis Class Model:
      - Classes w/ name, attribute, and simplified operation
    - Refined to a Design Class Model:
      - Class with name, attribute, and operation
      - Type info & scope defined for attribute and operation
-
- 

  - Topics Covered:
    - Detailed Class Diagram
    - Object Diagram
    - Collaboration Diagram
    - Sequence Diagram
-
- 

Summary

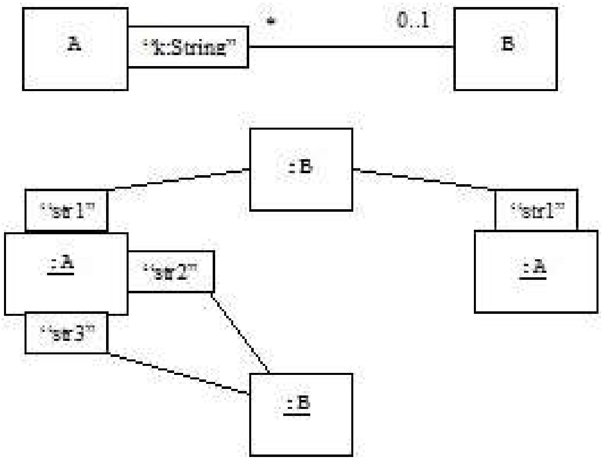
  - Class Diagram
    - Class Generalization
    - Association Class, Reification
    - N-Ary Association
    - Qualified Association
    - Interface
-
  -

L6 Assessment

Thursday, February 23, 2023 5:56 PM

Question 1
10 Points

Let us consider below a class diagram and an object diagram. Is the object diagram valid according to the given class diagram?



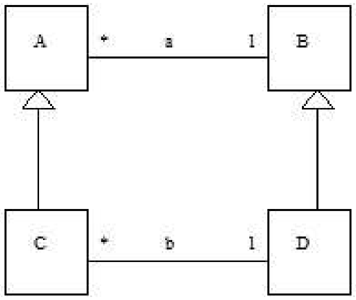
Class diagram

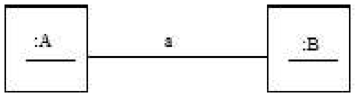
Object diagram

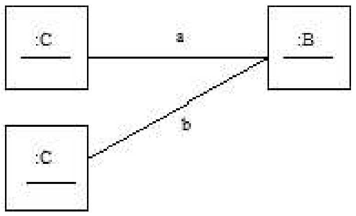
A: In the class diagram, we have an association of class A and class B. There is a qualifier on A, namely the string k. The class A can have 0 or more k's. The class B will use k to identify 0 or 1 of class A.
In the object diagram, we have 4 different objects. Two objects named B both use a string to identify another object A. The object A on the left accepts 3 different strings. The object A on the right accepts 1 string, but it is the same as one that was accepted by the object A on the left.
The object diagram does not accurately describe the class diagram as the same key is used twice to identify more than one object, unlike what was specified in the class diagram.

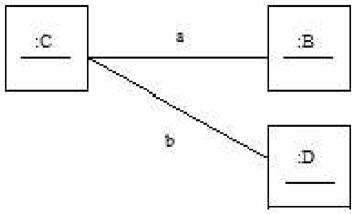
Question 2
10 Points

Given the below UML class diagram, which of the following objects diagrams are valid?



1. 

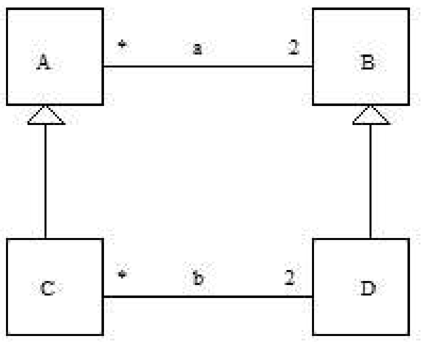
2. 

3. 

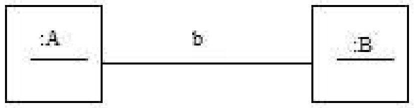
all of them;
only 1 and 2;
only 3;
only 1 and 3;
None of the above.

Question 3
10 Points

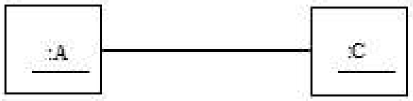
Given the below UML class diagram, which of the following objects diagrams are valid?



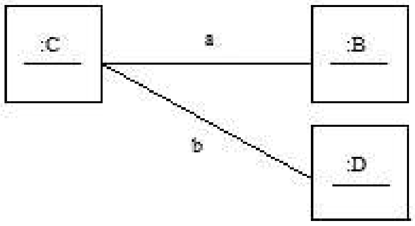
1.



2.



3.



all of them;
only 1 and 2;
only 3;
only 1 and 3;
None of the above.

Question 4
10 Points

Which of the following is false about the aggregation relationship?

Aggregation is a stronger form of association.
Aggregation is anti symmetric.
Aggregation is informal meaning of the "whole-part" relationship.
Aggregation is not transitive.
The most important usage of the aggregation is to disallow any cyclic object linkage.