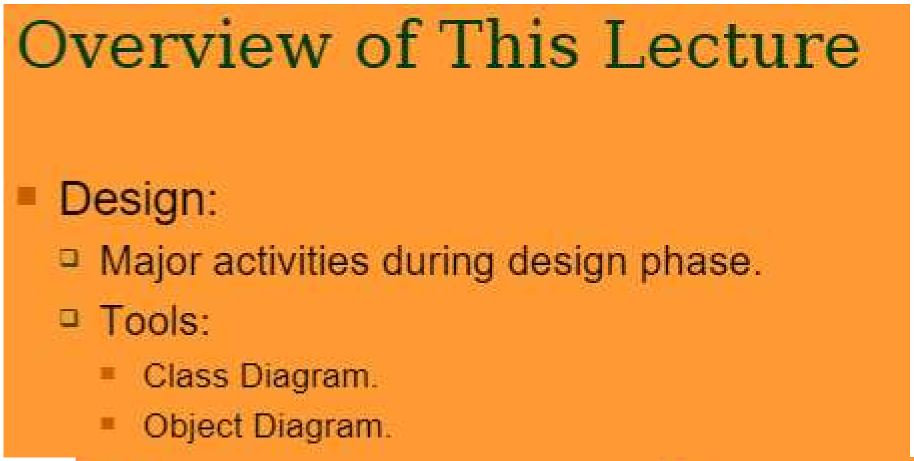
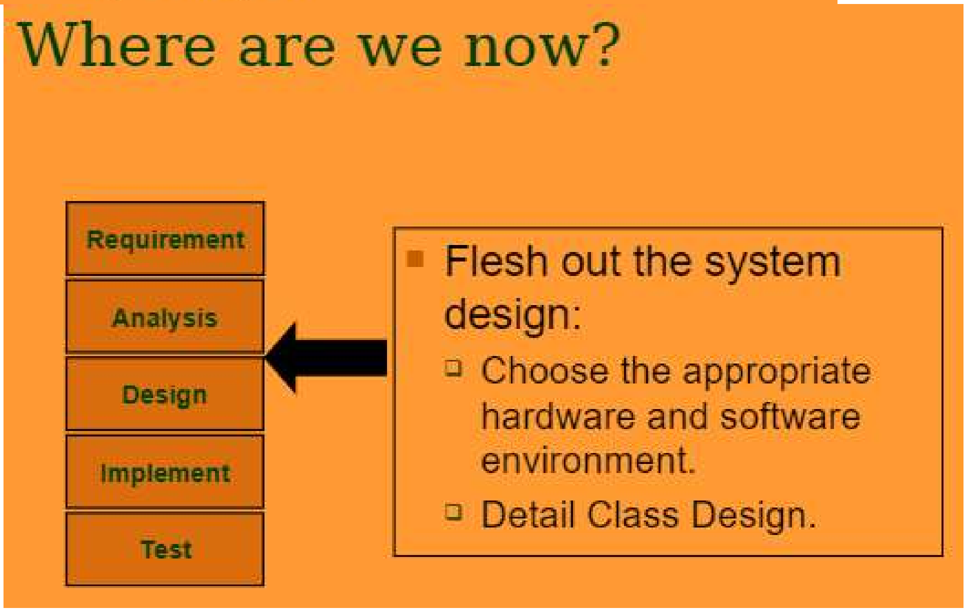
# Lecture 5

Wednesday, February 15, 2023 8:06 PM



## Overview of This Lecture

- Design:
  - Major activities during design phase.
  - Tools:
    - Class Diagram.
    - Object Diagram.

- 

  ## Where are we now?

  Requirement

  Analysis

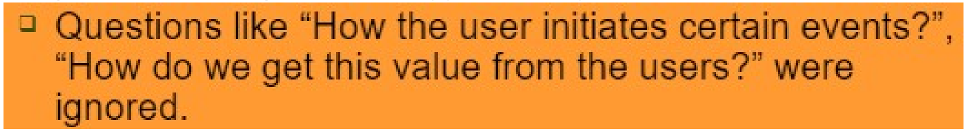
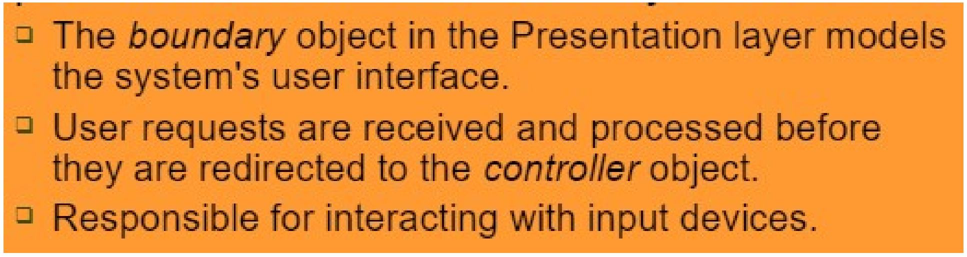
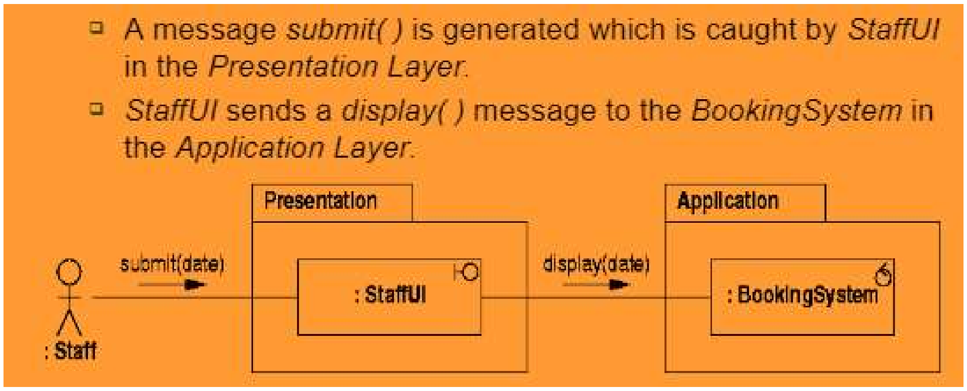
  Design

  Implement

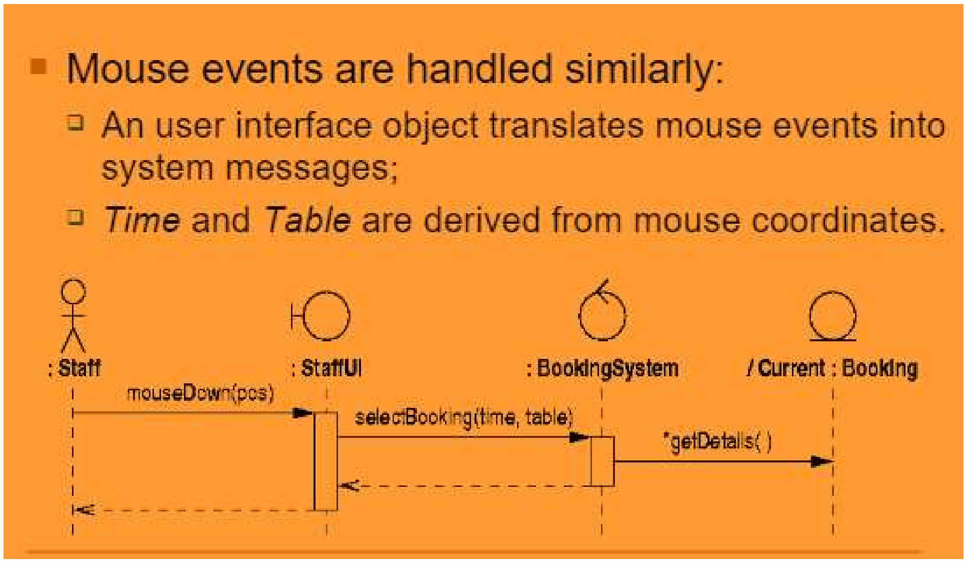
  Test

  - Flesh out the system design:
    - Choose the appropriate hardware and software environment.
    - Detail Class Design.

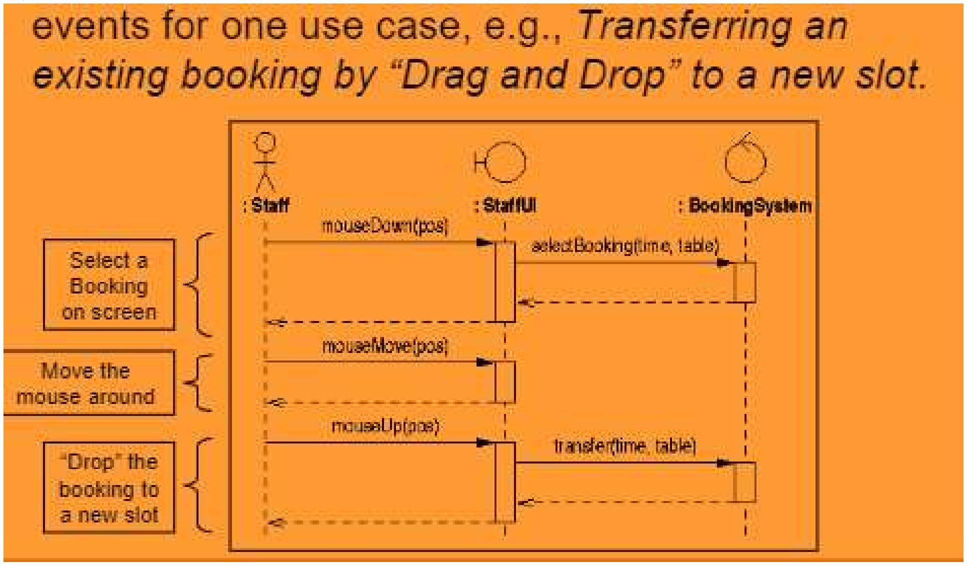
- Design: Overview
  - Inputs: software Architecture & Analysis Class Diagram
  - Activities:
    - Decide & design appropriate classes for other layers in the software architecture
    - Refine Analysis Class Diagram:
      - Produce detailed class Diagram ready for implementation
      - Collaboration Diagrams to study interaction btwn classes
      - State Chart to study behavior of a class
    - Apply Desing Pattern
- Design & Layered Architecture
  - Analysis show how business functions can be implemented in application layer
  - Design extends this lvl of modelling to the other layers
    - Ex: Presentation & Storage layers:
    - Studies interaction using the sequence diagrams
    - Infers info abt the classes
- Design: Presentation Layer
  - Decide the suitable UI and design the classes to support

    - Details abt designing good user interface not aim of this course
    - Important pts independent of the chosen UI:
        - Sep btwn UI and application
        - Relationship btwn UI and application
- Presentation Layer Revisited
    - Sys messages shown arriving @ ctrler in Application layer:
        - 
          - Questions like "How the user initiates certain events?", "How do we get this value from the users?" were ignored.
    - Messages pre-processed in presentation layer:
        - 
          - The *boundary* object in the Presentation layer models the system's user interface.
          - User requests are received and processed before they are redirected to the *controller* object.
          - Responsible for interacting with input devices.
- Case Study 1: Graphical UI
    - Assumption: window-based application (GUI) used
    - Possible sequence of initiating the Display Booking use case:
        - Select appropriate menu option
        - Dialogue box appear
        - User enters date and clicks ok button
    - Should concentrate on the steps that trigger ops in the Application layer (last step above)
- Case Study 1: Presentation Layer
    - Packages included to show interaction btwn the layers
    - After click OK:
        - 
          - A message *submit( )* is generated which is caught by *StaffUI* in the *Presentation Layer*.
          - *StaffUI* sends a *display( )* message to the *BookingSystem* in the *Application Layer*.

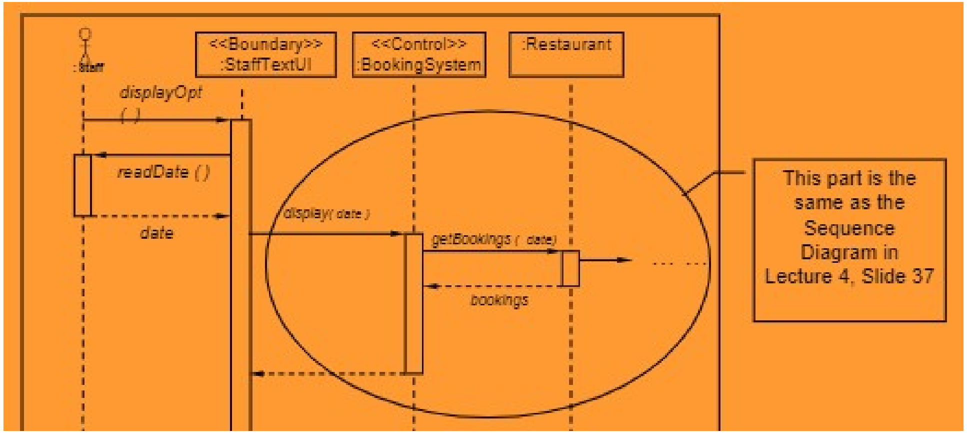
- Case Study 1: Mouse Event
    -

- 

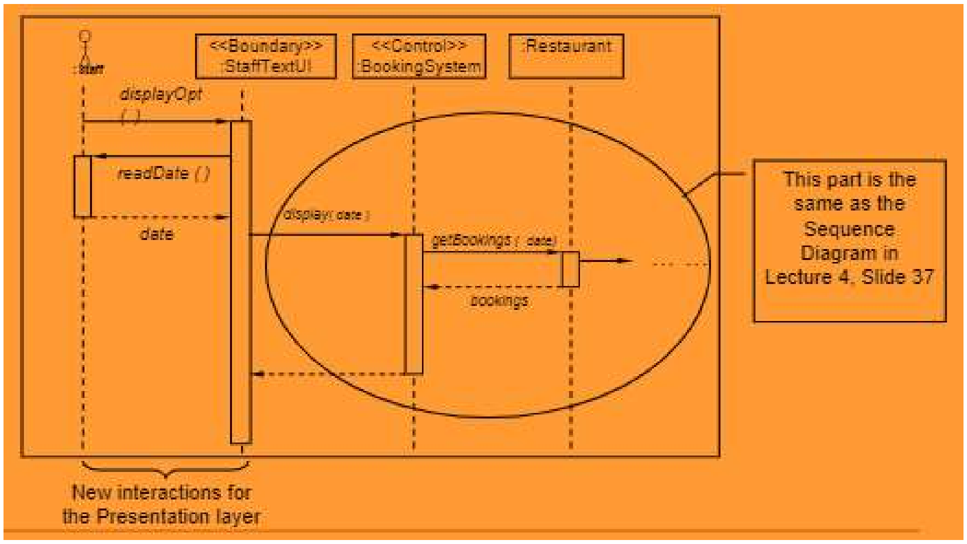
  - Mouse events are handled similarly:
    - An user interface object translates mouse events into system messages;
    - *Time* and *Table* are derived from mouse coordinates.

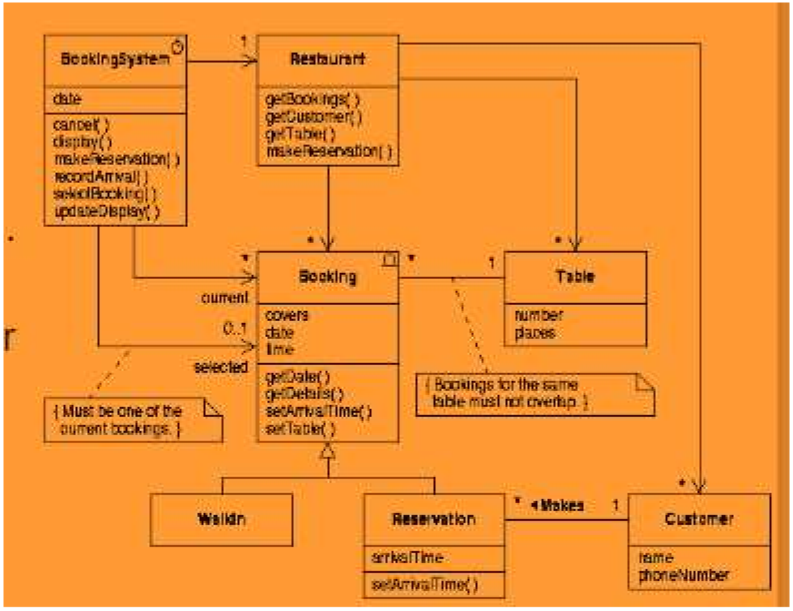
- Case Study 1: Inferring Class Information
  - For GUI, exists series of user made events for 1 use case
    - 

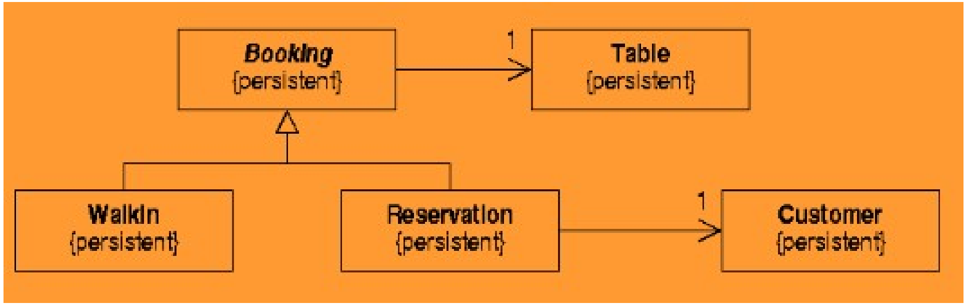
      events for one use case, e.g., *Transferring an existing booking by "Drag and Drop" to a new slot.*

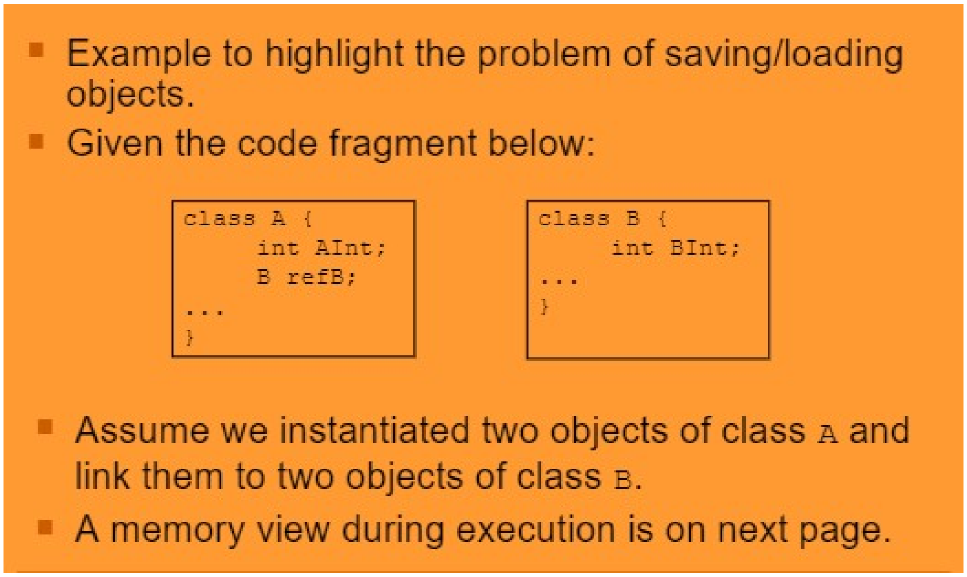
  - User-made events might not corresponde w/ sys message
  - Boundry objects may need to keep extra info to remember state of current interaction, deduce when to send a sys message
  - In table transfer ex. , GUI has to remember that Table Transfer is being even when user moves mouse around, and send transfer() message when mouse button is released
- Case Study 1: Txt based UI
  - Tho interface is diff, activities similar: design UI, study interaction, design the UI classes
  - Txt base UIs slightly easier to make bc order of events well defined
  - 

    This part is the same as the Sequence Diagram in Lecture 4, Slide 37

- 

- Presentation Layer: Summary
  - Activities during design phase for Presentation Layer:
    - Decide the appropriate type of UI
    - Draft the UI
    - Study the interaction to perform each of the use cases: modifying existing sequence diagrams derived during analysis
    - Derive the classes needed: messages, class attributes, association w/ classes in the application layer
- Storage Layer: Revisited
  - Most sys need persistent data- not lost when sys closes down
  - Few possible choices for it
    - Relational databases (RDBMS)
    - Object Serialization (Java)
    - Files I/o
  - Designer needs to:
    - Id what data needs to be persistent
    - Id ways to save/load the data according to the storage choice
- Designing Persistent Classes
  - Case Study 1:
    - All booking info must be saved, except current data, or selected bookings
    - 

    - In UML, classes are unit of persistence, persistent class denoted by a tagged value
        - 

- Saving and Reloading an Object
    - Object only exist during pgrm execution, no clear counterpart in static storage (can't save object directly to file, attributes of an object are saved)
    - When reloading, attributes are read and a object is recreated using same attributes, illusion
    - Works well for objects w/ simple vals, what abt reference?
- Preserving Association
    - Association btwn objects operates using reference 9mem address): mem address of object can be diff when reloading as object is reallocated, id info to locate object needed to restore the association
    - Simple way is to assign a unique id to each object
- Simple Ex
    -
    - 

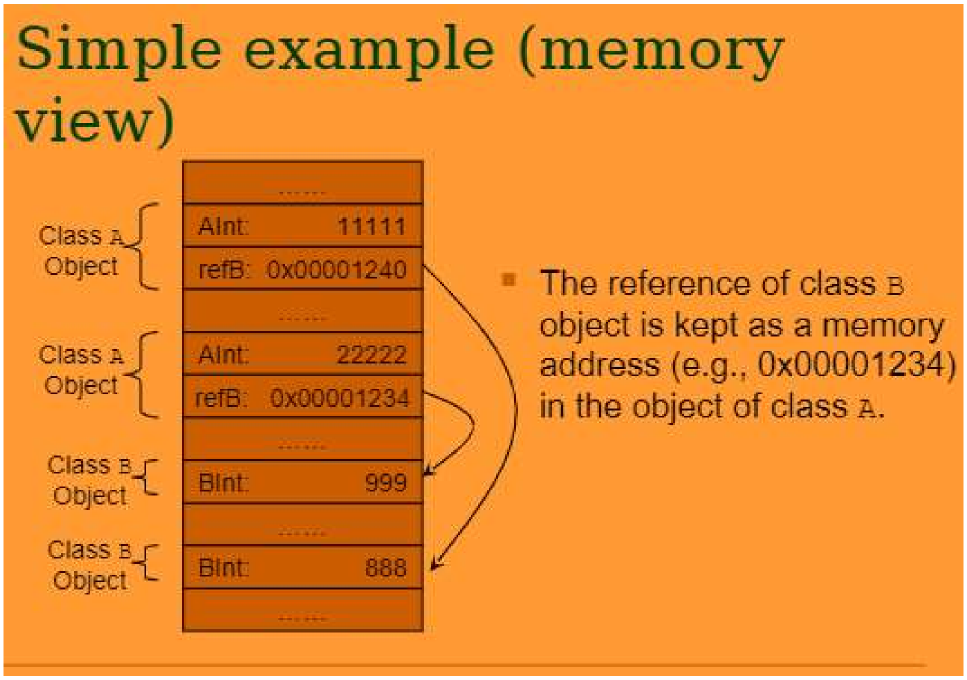
- Example to highlight the problem of saving/loading objects.
- Given the code fragment below:

```
class A {
      int AInt;
      B refB;
...
}
```

```
class B {
      int BInt;
...
}
```

- Assume we instantiated two objects of class `A` and link them to two objects of class `B`.
- A memory view during execution is on next page.

    -



## Simple example (memory view)

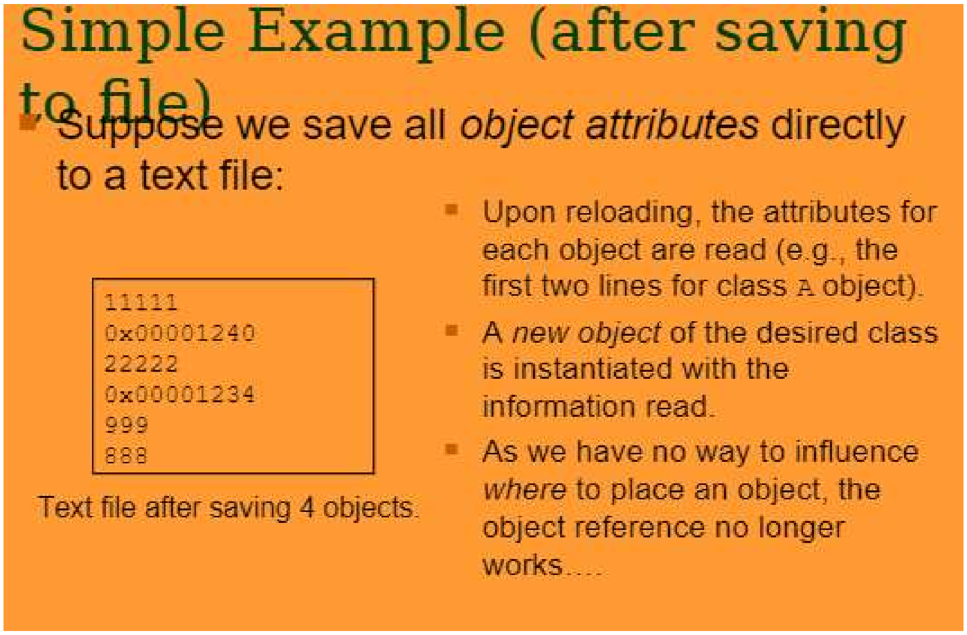
Class A Object
- AInt: 11111
- refB: 0x00001240

Class A Object
- AInt: 22222
- refB: 0x00001234

Class B Object
- BInt: 999

Class B Object
- BInt: 888

- The reference of class B object is kept as a memory address (e.g., 0x00001234) in the object of class A.



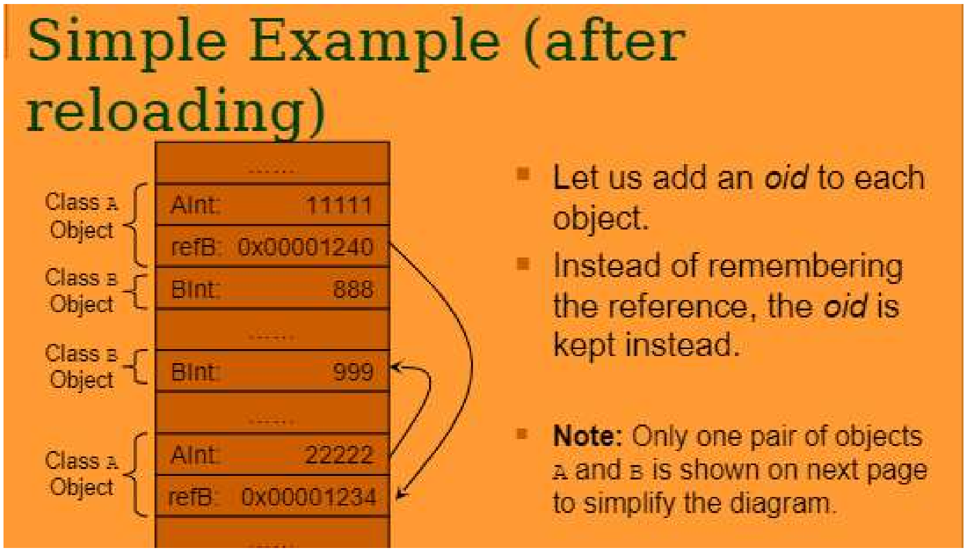
## Simple Example (after saving to file)

- Suppose we save all *object attributes* directly to a text file:

```
11111
0x00001240
22222
0x00001234
999
888
```

Text file after saving 4 objects.

- Upon reloading, the attributes for each object are read (e.g., the first two lines for class A object).
- A *new object* of the desired class is instantiated with the information read.
- As we have no way to influence *where* to place an object, the object reference no longer works….



## Simple Example (after reloading)

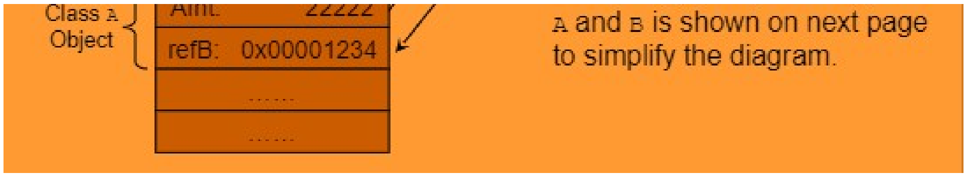
Class A Object
- AInt: 11111
- refB: 0x00001240

Class B Object
- BInt: 888

Class B Object
- BInt: 999

Class A Object
- AInt: 22222
- refB: 0x00001234

- Let us add an *oid* to each object.
- Instead of remembering the reference, the *oid* is kept instead.
- **Note:** Only one pair of objects A and B is shown on next page to simplify the diagram.



Class A Object

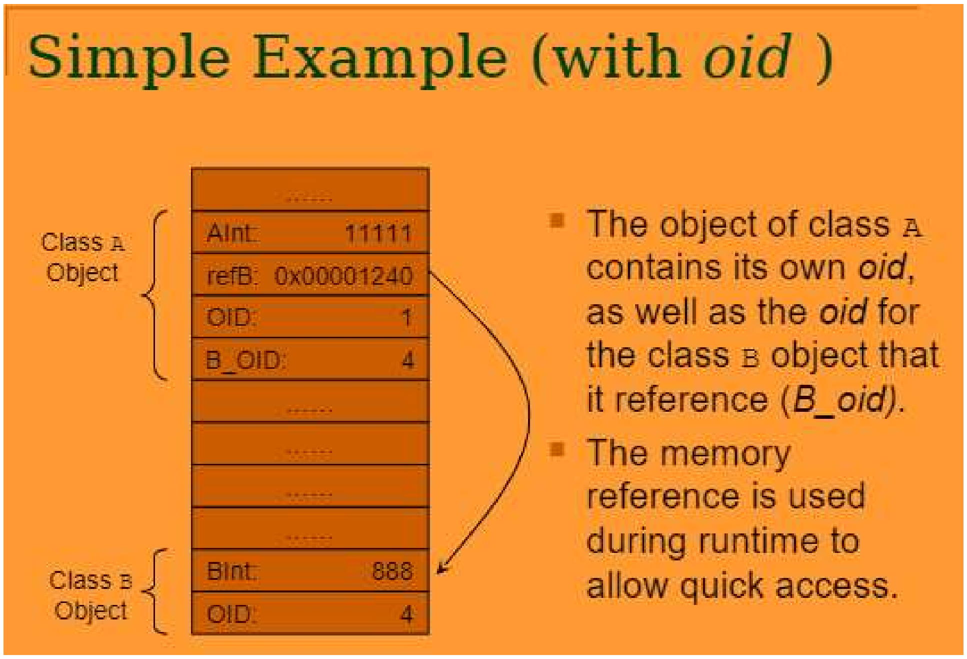
AInt: 22222

refB: 0x00001234

......

......

A and B is shown on next page to simplify the diagram.



## Simple Example (with *oid* )

Class A Object

......

AInt: 11111

refB: 0x00001240

OID: 1

B_OID: 4

......

......

......

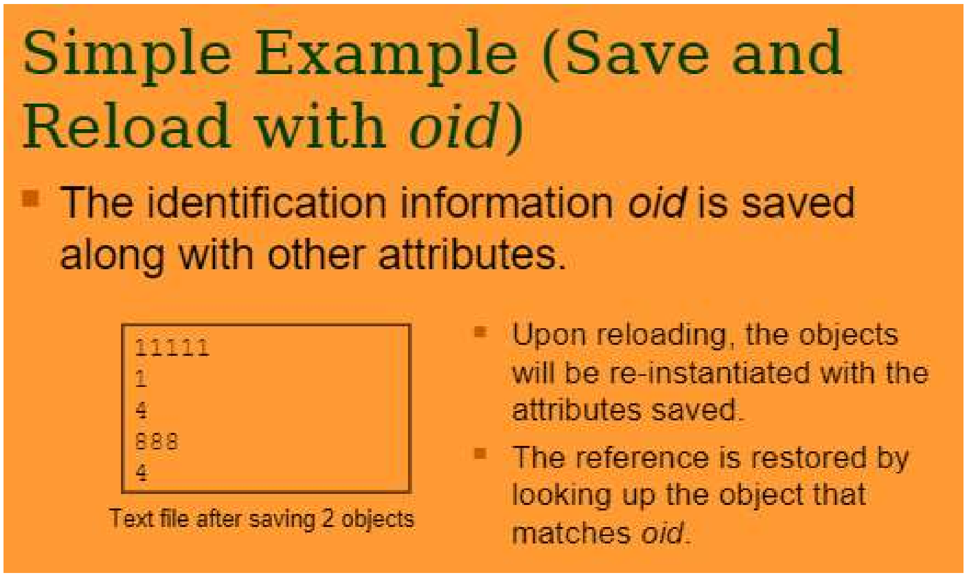
......

Class B Object

BInt: 888

OID: 4

- The object of class A contains its own *oid*, as well as the *oid* for the class B object that it reference (*B_oid)*.
- The memory reference is used during runtime to allow quick access.



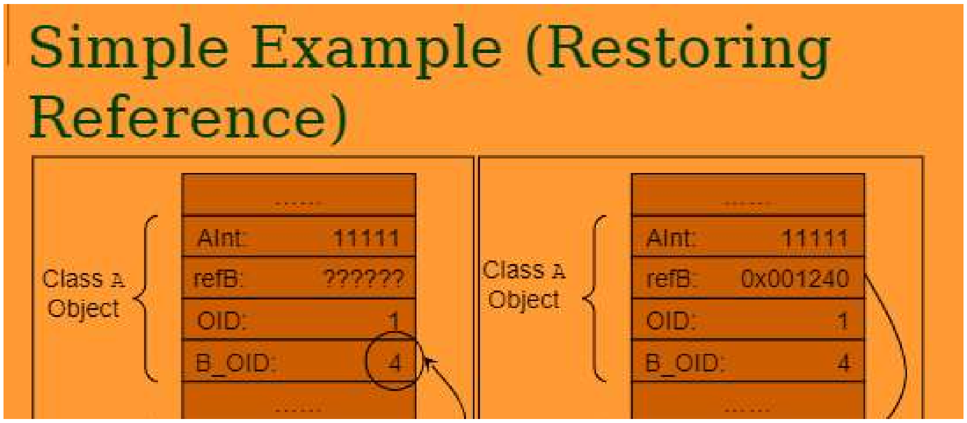
## Simple Example (Save and Reload with *oid*)

- The identification information *oid* is saved along with other attributes.

```
11111
1
4
888
4
```

Text file after saving 2 objects

- Upon reloading, the objects will be re-instantiated with the attributes saved.
- The reference is restored by looking up the object that matches *oid*.



## Simple Example (Restoring Reference)

Class A Object

......

AInt: 11111

refB: ??????

OID: 1

B_OID: 4

......

Class A Object

......

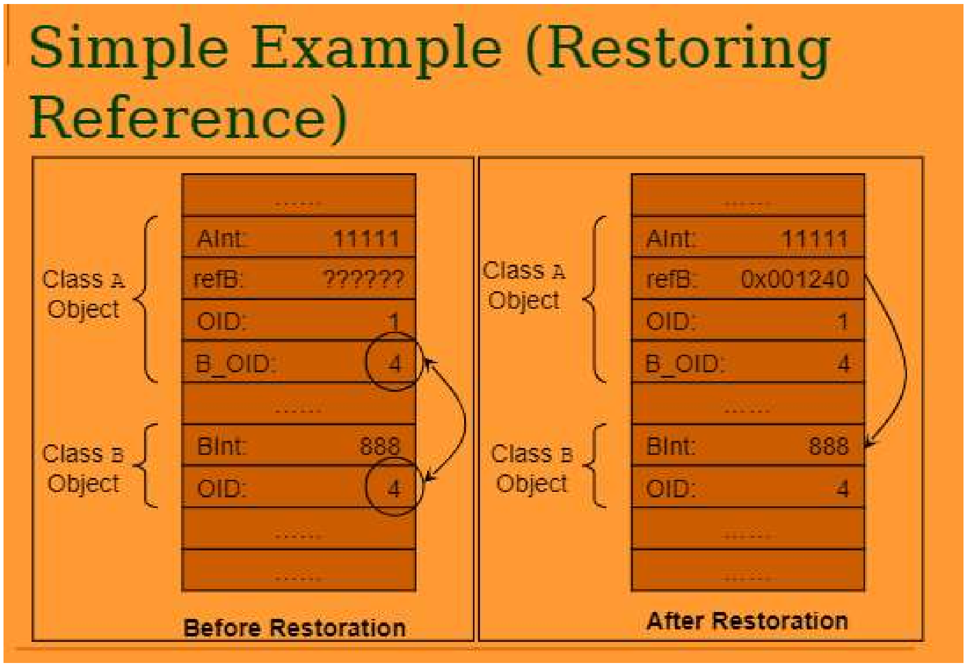
AInt: 11111

refB: 0x001240

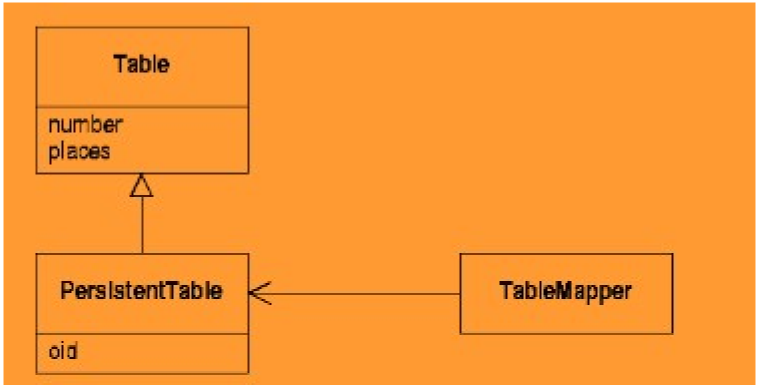
OID: 1

B_OID: 4

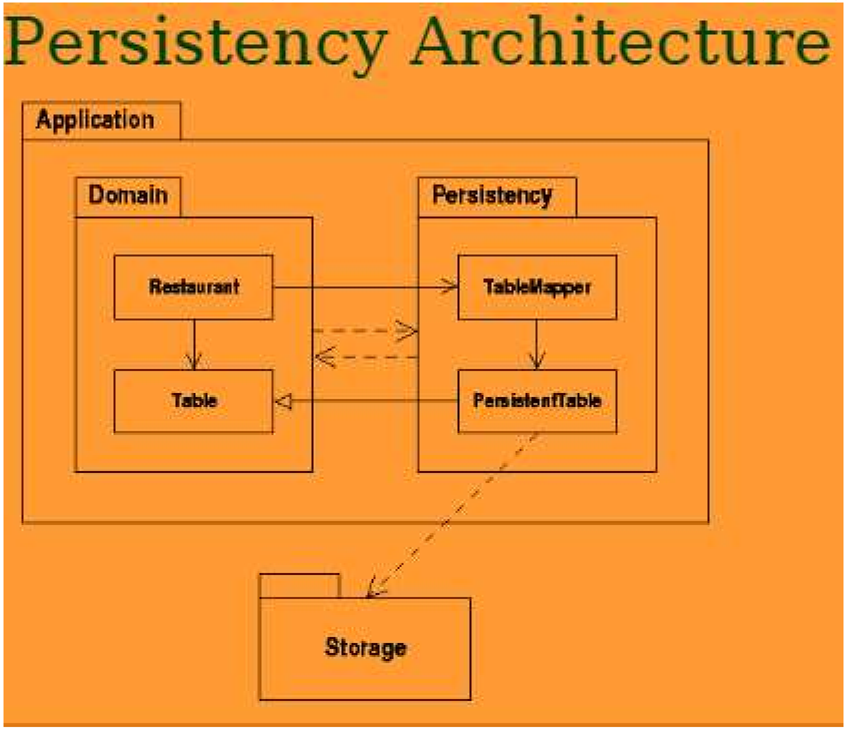
......

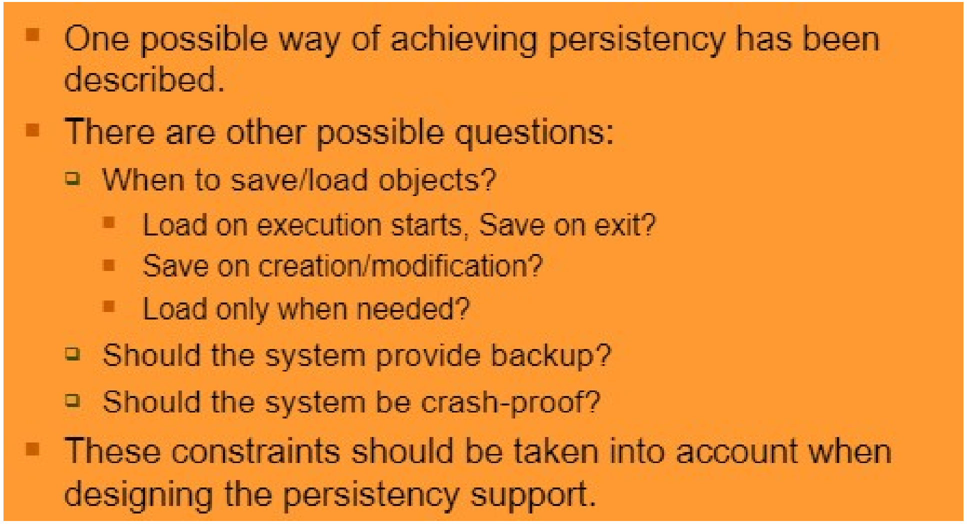
- ○ 

  Simple Example (Restoring Reference)

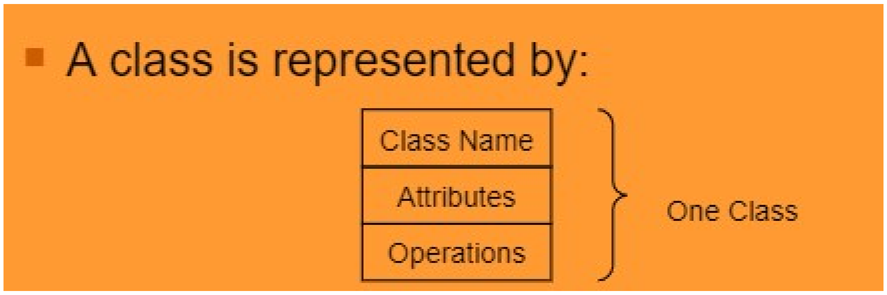
- Observations
  - ○ Mapping class needed: to map oid to actual mem reference & to assign oid to objects during new object creation
  - ○ Oid behaves just like key in database record: whole process is equally applicable when using DBMS unstead of an I/o file, using an oid to id other objects is same to use a foreign key to id other record
  - ○ DBMS beyond scope of this course, but process/steps still valuable insights
- Case Study 1: Adding Persistency Support
  - ○ Take table class as ex:
    - ▪ subclass is added to incorporate the oid
    - ▪ Mapper class is added
    - ▪ 

    - ▪ Persistency table subclass added so that other classes that depend on the table class won't be affected
    - ▪ Incorporating oid directly into Table class would limit possible choice of persistency support
    - ▪ TableMapper class is responsible for making objects of the PersistentTable class and restoring them from persistent storage if req
    - ▪ Apply same idea to other classes that req persistency support: booking, customer, etc...
- Case Study 1: Persistency Architecture
  - ○ Persistency subclasses and mapper classes depend on the class they are supporting, so must be in application layer
  - ○ Restaurant class is dependent on mapper classes, Restaurant uses the mapper class to make

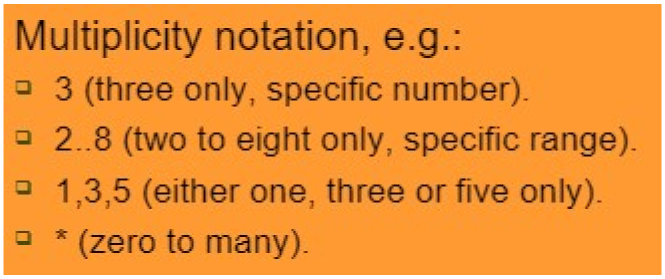
    an object w/ persistency support
  - To preserve layered architecture: split the application layer into 2 subpackages (domain sub package for all classes directly related to problem domain, persistency subpackage for persistency support classes has min effect on Domain sub-package), and also changing the persistency sub-package will have a min impact on domain sub-package
  - 
    Persistency Architecture

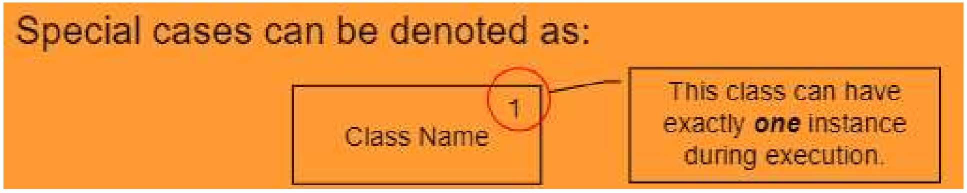
- Persistency Sub-Package: Possible Questions
  - 
    - One possible way of achieving persistency has been described.
    - There are other possible questions:
      - When to save/load objects?
        - Load on execution starts, Save on exit?
        - Save on creation/modification?
        - Load only when needed?
      - Should the system provide backup?
      - Should the system be crash-proof?
    - These constraints should be taken into account when designing the persistency support.
- Storage Layer: Summary
  - Activities during design phase for storage layer:
    - Id classes that req persistency support
    - Choose appropriate persistency service
    - Intro classes to work with the chosen persistency service
    - Add sequence diagram to capture the ops needed
- Design Phase: Rest of the journey
  - Basic technique/activity during design phase coverd, later lectures look closer look @ tools and one important technique to enhance the design
- Detailed Class Design
  - Make detailed class specs that could be used as basis for implementation
    - Start w/ analysis class model

        - Collect messages from all realizations:
            - Check redundancy, inconsistency, etc.
            - Define the op interface of that class
            - Specify detailed param and return types

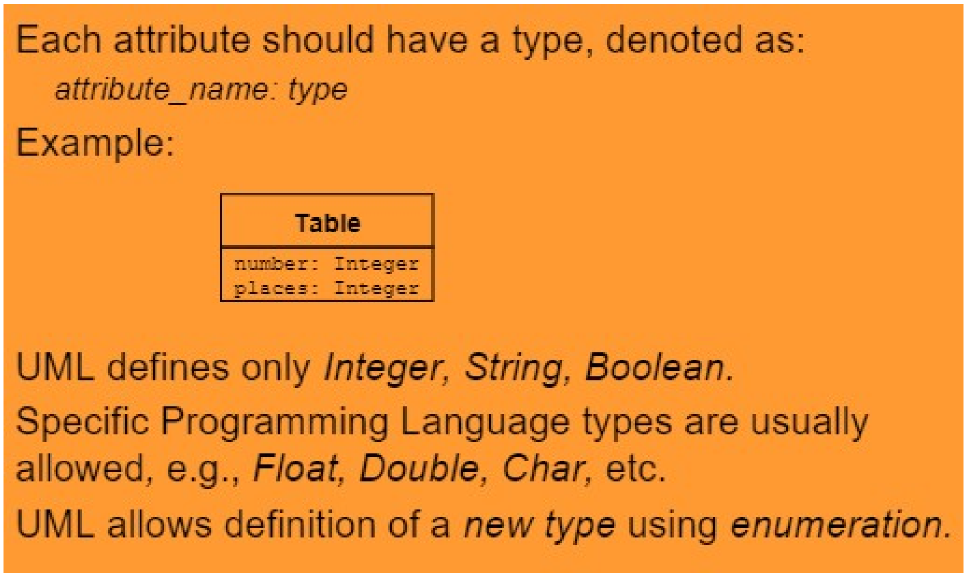
- Class Diagram: Review
    - ○
    - 

      A class is represented by:

    - A relationship btwn classes reped by association w/ : Association name, Role name, Multiplicity, Nagivability (Direction)
- Class Multiplicity
    - ○
    - 

      Multiplicity notation, e.g.:
      - 3 (three only, specific number).
      - 2..8 (two to eight only, specific range).
      - 1,3,5 (either one, three or five only).
      - * (zero to many).
    - Can be applied to class, known as class multiplicity, determine numb of instances for a class
    - Default class multiplicity is *
    - 

      Special cases can be denoted as:

      This class can have exactly ***one*** instance during execution.
- Attribution Type
    - ○
    - 

      Each attribute should have a type, denoted as:

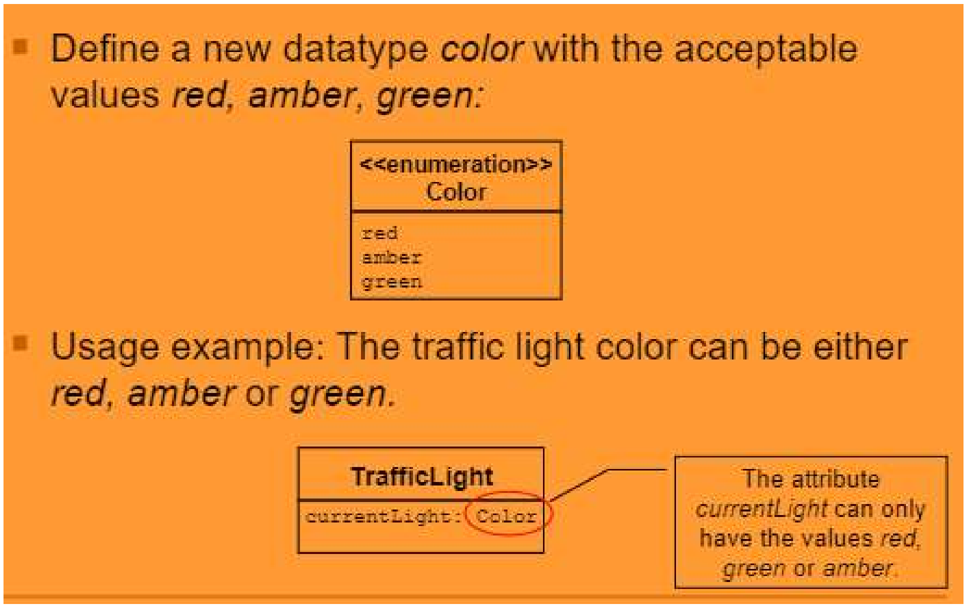
      *attribute_name: type*

      Example:

      UML defines only *Integer, String, Boolean.*

      Specific Programming Language types are usually allowed, e.g., *Float, Double, Char,* etc.

      UML allows definition of a *new type* using *enumeration.*
- Enumeration

- 

  ▪ Define a new datatype *color* with the acceptable values *red, amber, green:*

  <<enumeration>>
  Color

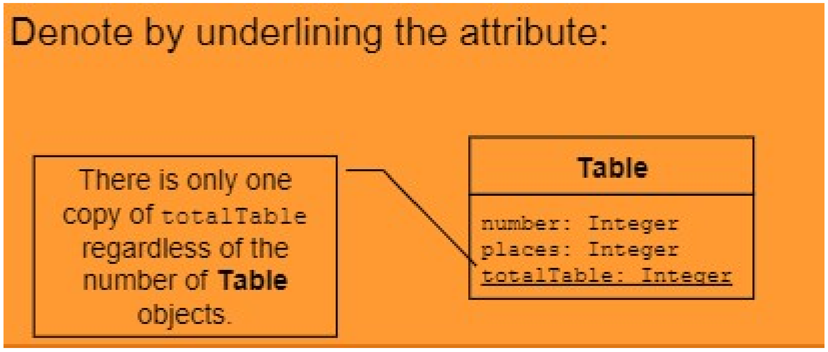
  ```
  red
  amber
  green
  ```

  ▪ Usage example: The traffic light color can be either *red, amber* or *green.*

  **TrafficLight**

  ```
  currentLight: Color
  ```

  The attribute *currentLight* can only have the values *red, green* or *amber.*

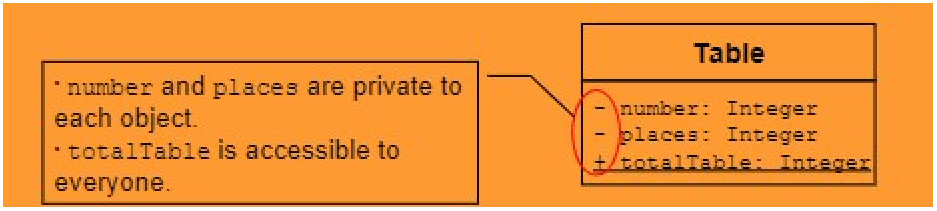
- Attribute Scope
  - Attribute of a class has an instance scope by default: (ex. Each instance (object) of the class has an independent copy of the attribute)
  - There are cases where class scope is needed: all instances share one copy of the attribute (aka class attribute in Java)
  - 

    Denote by underlining the attribute:

    There is only one copy of `totalTable` regardless of the number of **Table** objects.

    **Table**

    ```
    number: Integer
    places: Integer
    totalTable: Integer
    ```

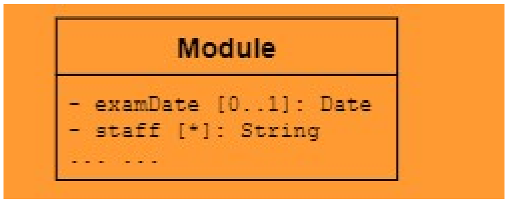
- Attribute Visibility
  - Equivalent to the accessibility lvl in languages like C++, Java
  - UML defines 4 lvls of accessibility
    - Public (+): visible to all classes
    - Protected (#): visible to instances of subclasses
    - Private (-): visible only in the same class
    - Package (~): visible to classes in the same package
    - 

      · `number` and `places` are private to each object.
      · `totalTable` is accessible to everyone.

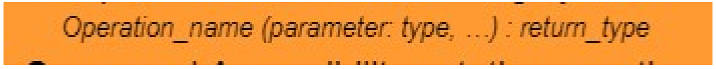
      **Table**

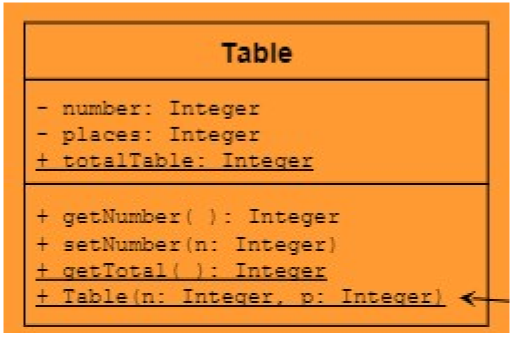
      ```
      - number: Integer
      - places: Integer
      + totalTable: Integer
      ```

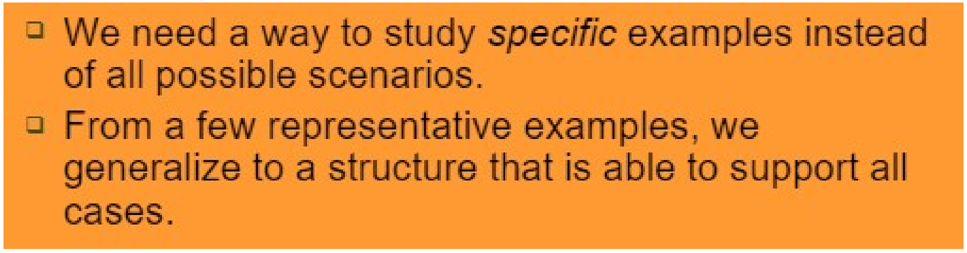
- Attribute Multiplicity
  - Defines how many values an object stores for an attribute
    - Default is exactly 1
    - Optional multiplicity shows possible null vals
    - Arrays modelled by * (many) multiplicity

    - 

      ```
      Module
      - examDate [0..1]: Date
      - staff [*]: String
      ... ...
      ```

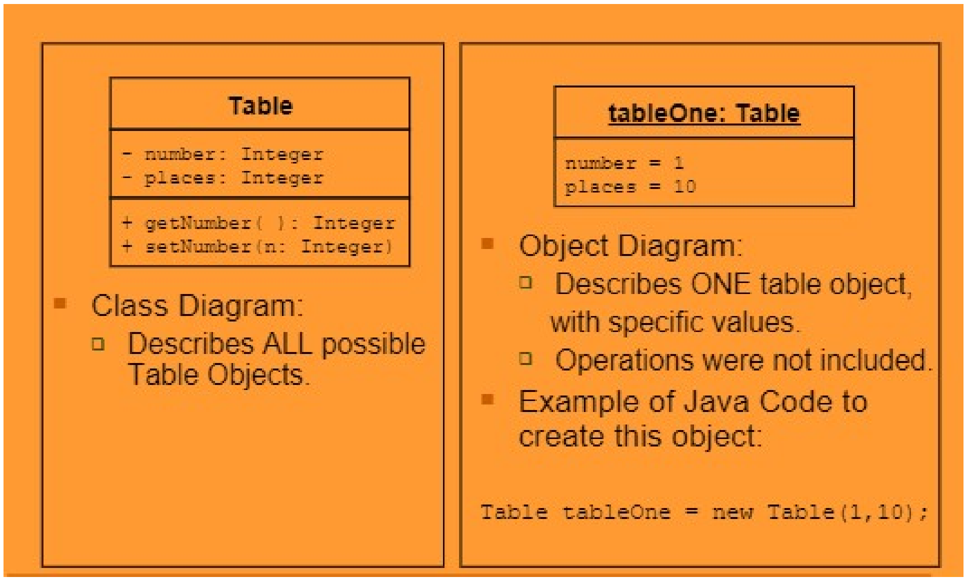
- Operations
  - An op uses following syntax:
  - 

    Operation_name (parameter: type, ...) : return_type
  - Scope and accessibility notation are same as attributes
  - Constructor is denoted as Class Scope operation in UML
  - 

    ```
    Table
    - number: Integer
    - places: Integer
    + totalTable: Integer
    + getNumber( ): Integer
    + setNumber(n: Integer)
    + getTotal( ): Integer
    + Table(n: Integer, p: Integer)
    ```
- Constructing Class Diagram
  - Study relevant structures (usually real world entities) & generalize to class diagram
  - Can be difficult in complicated cases
    - 

      ❑ We need a way to study *specific* examples instead of all possible scenarios.

      ❑ From a few representative examples, we generalize to a structure that is able to support all cases.
  - Object diagram intro to study specific ex instead of general cases
- Object Diagram
  - Classes in a pgrm describe all possible instances (objects), during pgrm execution, only a subset of all possible objects are instantiated
  - Similarly, UML defines Object Diagrams to examine instances (possible scenarios) of Class Diagrams:
    - Notation similar to class diagram
    - Slight diff in terminology
    - Can help understand and refine a class diagram
  - First look:

- 

  Table

  ```
  - number: Integer
  - places: Integer
  + getNumber( ): Integer
  + setNumber(n: Integer)
  ```

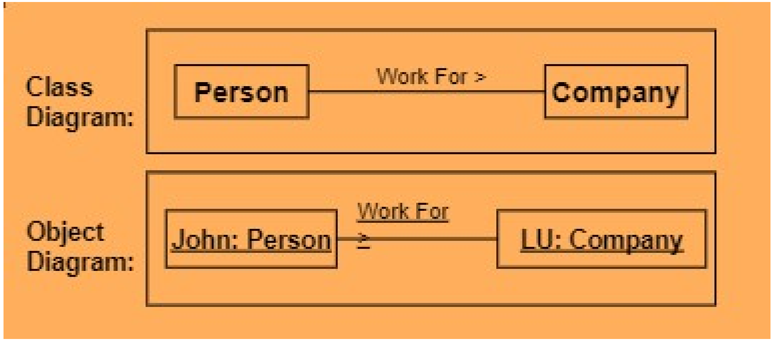
  - Class Diagram:
    - Describes ALL possible Table Objects.

  tableOne: Table

  ```
  number = 1
  places = 10
  ```

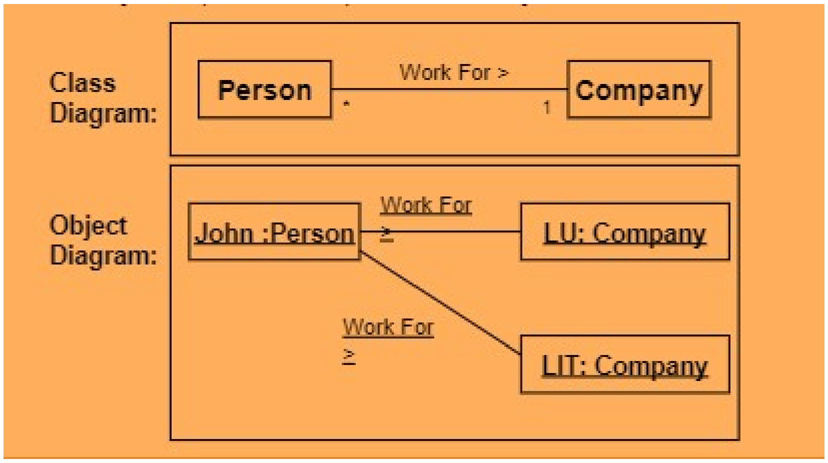
  - Object Diagram:
    - Describes ONE table object, with specific values.
    - Operations were not included.
  - Example of Java Code to create this object:

  ```
  Table tableOne = new Table(1,10);
  ```

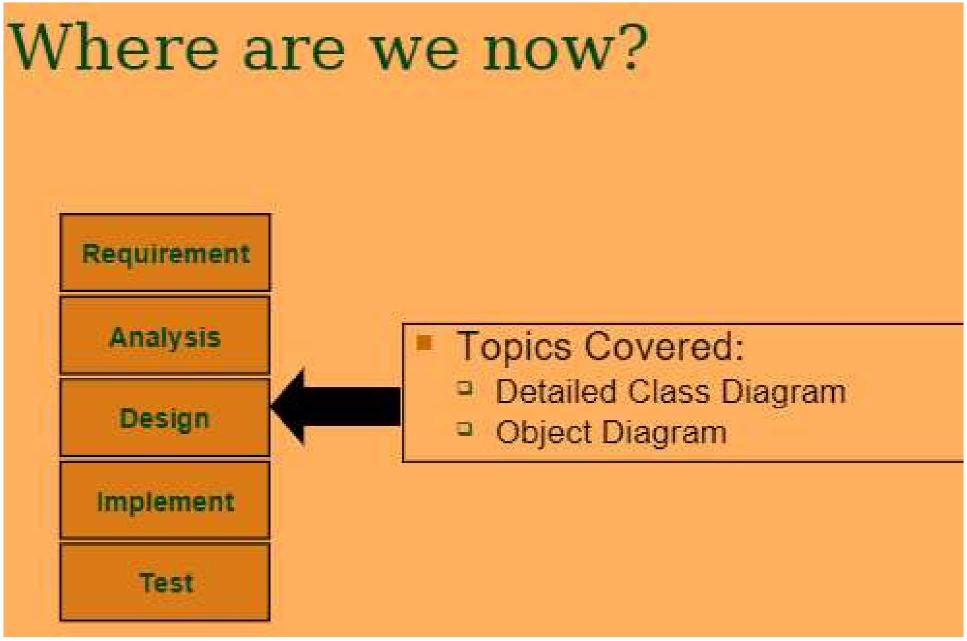
- Closer Look
  - Object given an object name:
    - Rep id of an object, easy understand from diagram
    - Usually go w/ var name in pgrm construct for convenient sake: object var in java is a reference, multiple object vars can refer to the same object, but one object can have only 1 object name in UML
  - Denoted by : objectName: ClassName
  - Object name can be omitted when  we not interested
  - Attributes of object have specific vals, denoted by: attribute = value.
  - Ops of object not shown as all objects of same class that same set of operations
- Link
  - Association in a class diagram describes general relationship btwn classes
  - Instance of association (link) connects 2 specific objects in a Object Diagram
  - Association Names are underlined in Object Diagram to rep a link
  - 

    Class Diagram: Person — Work For > — Company

    Object Diagram: John: Person — Work For > — LU: Company
- Correctness
  - Object diagrams has to repect the constraints laid out by the corresponding class diagram, that is, a multiplicity "many-to-one" for the association from Person to Company
  - Object Diagram below not valid w/ class diagram bc there exist 2 objects (LU and LIT) linked to object John instead of one

- 

- Usefulness
  - Use object diagram to sketch real world scenarios:
    - Can check  whether the class diagram can correctly support possible cases
    - Ex if a person is allowed to take 2 jobs as in the last object diagram, then the class diagram needs to be moded
  - Can be used to visualize the execution of a code fragment

- 

## Where are we now?

Requirement

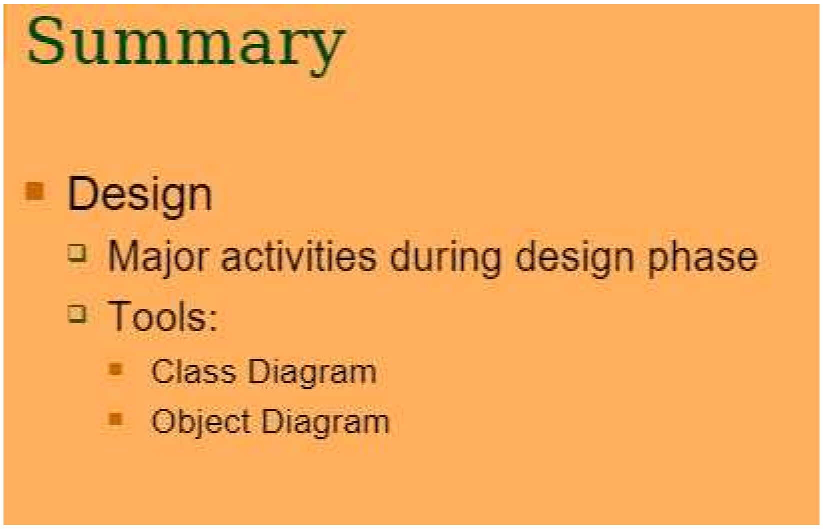
Analysis

Design

Implement

Test

- Topics Covered:
  - Detailed Class Diagram
  - Object Diagram

- 

## Summary

- Design
  - Major activities during design phase
  - Tools:
    - Class Diagram
    - Object Diagram

# Assessment 5

Friday, February 17, 2023 8:07 PM

**Question 1**
**10 Points**

**(Exercise 2.4 of [Priestley, 2004])** On a single diagram, illustrate the following using the UML notation for objects, links and messages.

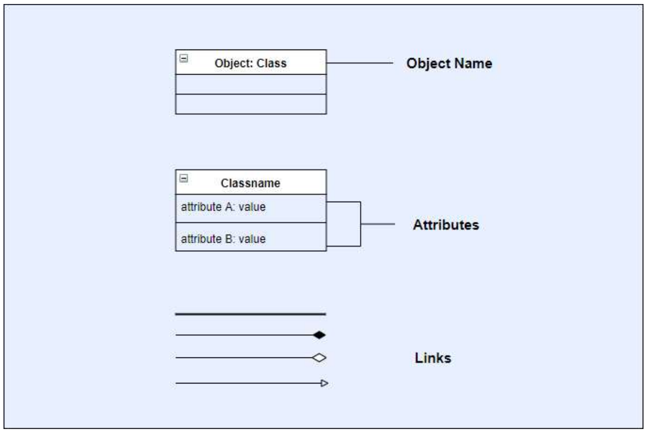
(a) An object of class Window, with no attributes shown;

(b) An object of class Rectangle with attributes length and width. Assume that the Rectangle class supports an operation to return the area of a rectangle object.

(c) A link between the window and rectangle objects, modeling the fact that the rectangle defines the screen co-ordinates of the window.

(d) The window object sending a message to the rectangle asking for its area.

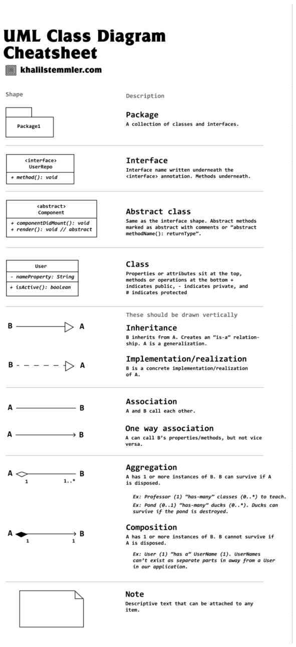
Draw a class diagram showing Window and Rectangle classes with the properties mentioned in this question.



**Question 2**
**10 Points**

Why persistency support is important for a software system? How to design it?

A: Persistency support is important for a software system because some data should not be lost when the system closes down. This is designed by both identifying what data needs to be persistent, and by identifying ways to save/load the data according to the storage choice.



**UML Class Diagram Cheatsheet**

khalilstemmler.com

| Shape | Description |
|---|---|
| Package1 | **Package**<br>A collection of classes and interfaces. |
| <interface><br>UserRepo<br>*+ method(): void* | **Interface**<br>Interface name written underneath the <interface> annotation. Methods underneath. |
| <abstract><br>Component<br>*+ componentDidMount(): void*<br>*+ render(): void // abstract* | **Abstract class**<br>Same as the interface shape. Abstract methods marked as abstract with comments or “abstract methodName(): returnType”. |
| User<br>*- nameProperty: String*<br>*+ isActive(): boolean* | **Class**<br>Properties or attributes sit at the top, methods or operations at the bottom + indicates public, - indicates private, and # indicates protected |
|  | These should be drawn vertically |
| B ——▷ A | **Inheritance**<br>B inherits from A. Creates an “is-a” relationship. A is a generalization. |
| B - - - - -▷ A | **Implementation/realization**<br>B is a concrete implementation/realization of A. |
| A —— B | **Association**<br>A and B call each other. |
| A ——> B | **One way association**<br>A can call B’s properties/methods, but not vice versa. |
| A ◇—— B<br>1 1..* | **Aggregation**<br>A has 1 or more instances of B. B can survive if A is disposed.<br>*Ex: Professor (1) “has-many” classes (0..*) to teach.*<br>*Ex: Pond (0..1) “has-many” ducks (0..*). Ducks can survive if the pond is destroyed.* |
| A ◆——> B<br>1 1 | **Composition**<br>A has 1 or more instances of B. B cannot survive if A is disposed.<br>*Ex: User (1) “has a” UserName (1). UserNames can’t exist as separate parts in away from a User in our application.* |
|  | **Note**<br>Descriptive text that can be attached to any item. |

**Question 3**
**10 Points**

**(Exercise 8.2 of [Priestley, 2004])** Define multiplicities for the following associations:

**(a)** ‘Has a loan’, linking people to books in a library system (hint: a person can have zero or more books on loan, and each book can be borrowed by at most one person at any given time);

A: The books "Has a loan" association has a default multiplicity (zero or more).

**(b)** ‘Has read’, linking people to books;

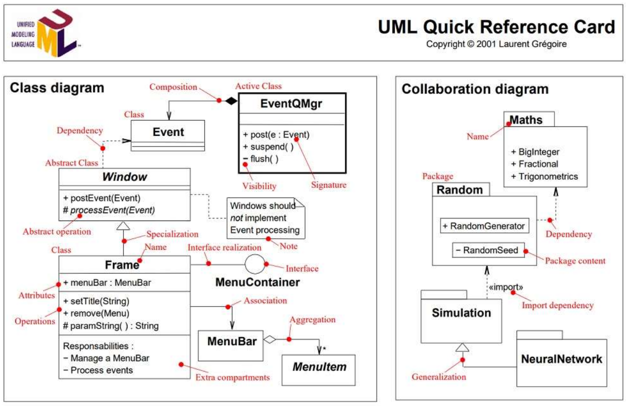
A: The 'Has read' association has a default multiplicity (zero or more).

**(c)** ‘Is occupying’, linking chess pieces to squares on a chess board;

A: The 'Is occupying' association has a multiplicity of 6 (six different types of chess pieces).

**(d)** ‘Spouse’, linking class ‘Person’ to itself.

A: The 'Spouse' association has a multiplicity of 1.



UML Quick Reference Card

Copyright © 2001 Laurent Grégoire

**Question 4**
**10 Points**

**(Exercise 2.5 of [Priestley, 2004])** Suppose that one particular environmental monitoring station contains three sensors, namely a thermometer, a rain gauge, and a humidity reader. In addition, there is a printer for printing purpose of the data read by the monitoring station.

Draw an object diagram to represent this scenario.
Generalize (i) for a class diagram that describes monitoring station in

general.

**Question 5**
**10 Points**

**(Exercise 8.5 of [Priestley, 2004])** Companies may employ many people, and people may work for many companies. Each company has a managing director and every employee in a company has a manager, who may manage many subordinate employees.

Draft a class diagram to capture the structure described. You can ignore attributes and operations for the classes. Just concentrate on the classes and associations.

A:

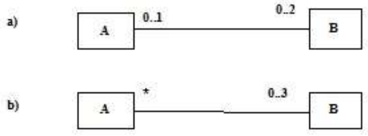
**Question 6**
**10 Points**

From the following description, identify the main objects and their links. Draw a class diagram that specify the below description. Be sure to indicate the multiplicity, role and name of each association. Also, draw an object diagram that shows these objects and links. Justify your choices.

“This description is about mobile phone companies. The companies VerizonTel, TexTel have coverage in Texas and, LouisTel and LouisMobile have coverage in Louisiana. There are many customers of the above mobile phone companies. A customer can be a client of more than just one mobile phone company.”

**Question 7**
**10 Points**

Describe what the two class diagrams below indicate?

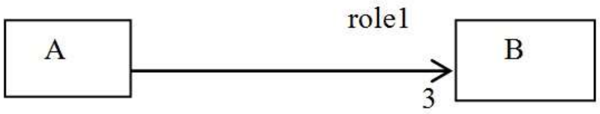


A: For a), the 0...1 means that there is either 0 or 1 instances of A. The 0...2 means that there is in between 0-2 instances of B. So in total, there can be 0-1 instances of A that has 0-2 instances of B.

For b) , the * means 0 or more instances of A. The 0...3 means that there is between 0-3 instances of B. So in total, there can be 0 or more instances of A that has 0-3 instances of B.

**Question 8**
**10 Points**

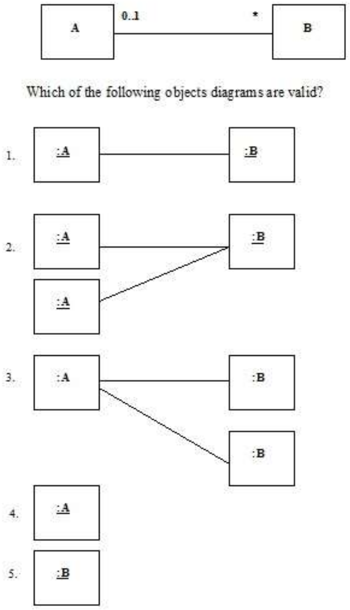
Given the below class diagram, which of the following statements is true?



Class A may have three objects defined “role1”;
There will be exactly three links at run-time from one object of A to three objects of B labeled with “role1”;
There will be exactly three links at run-time from B to A labeled with “role1”;
The multiplicity “3” is incorrect, as only 1 and * are allowed as multiplicity specification;
None of the above.

**Question 9**
**10 Points**

Let us consider the below UML class diagram:



Which of the following objects diagrams are valid?

only 1, 2, 3 and 5;
only 1, 3, 4 and 5;
only 1, 2, 4 and 5;
all five of them;
None of the above.