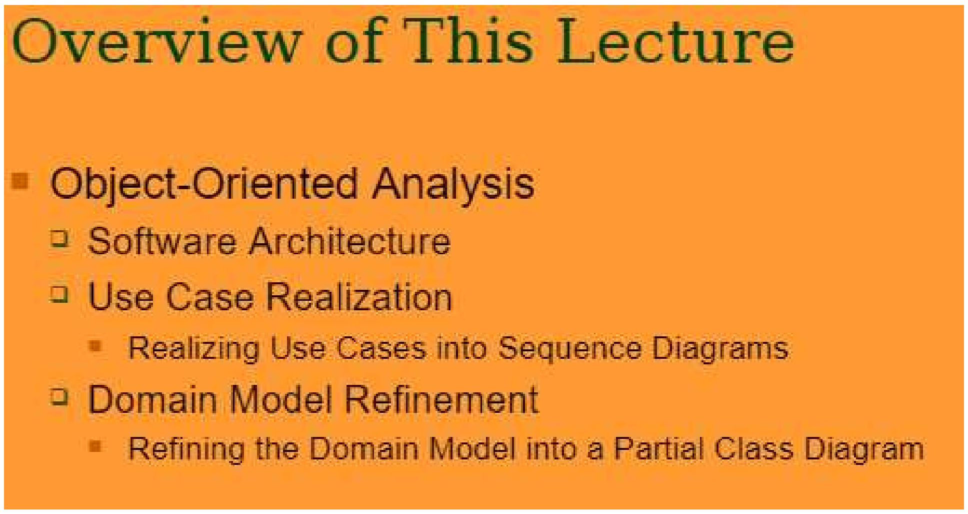
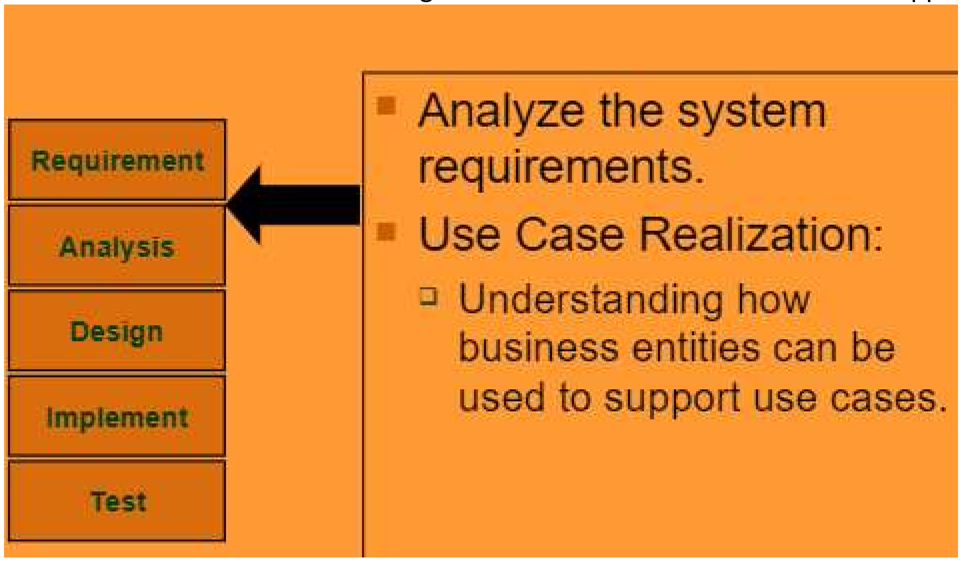
Lecture 4

Friday, February 10, 2023 8:05 PM

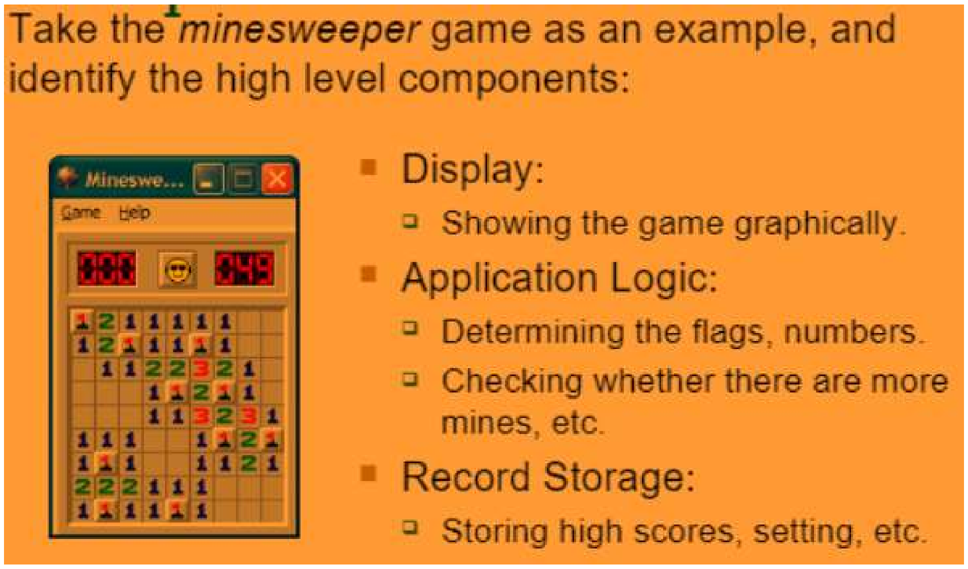


Overview of This Lecture

- Object-Oriented Analysis
  - Software Architecture
  - Use Case Realization
    - Realizing Use Cases into Sequence Diagrams
  - Domain Model Refinement
    - Refining the Domain Model into a Partial Class Diagram

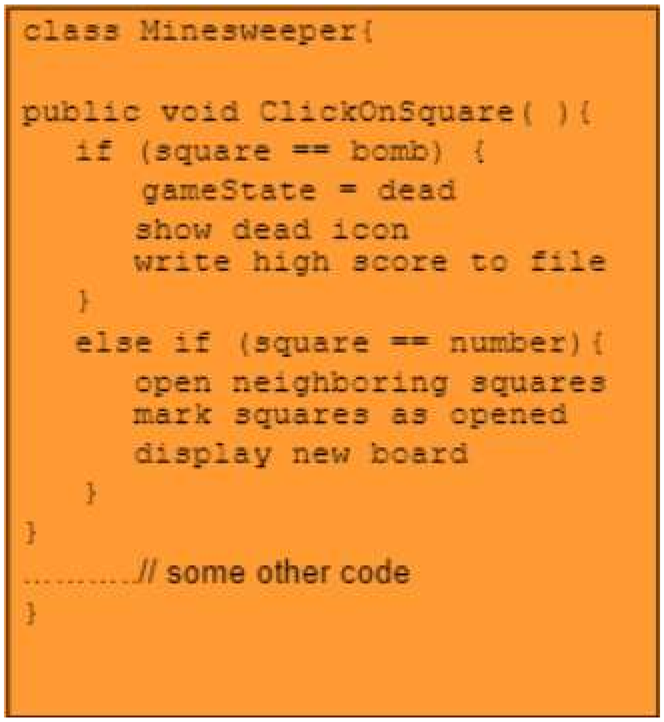
- Where We @ rn
  - Analyze the sys req
  - Use case Realization: understanding how business entities can be used to support use cases
  - 

    - Analyze the system requirements.
    - Use Case Realization:
      - Understanding how business entities can be used to support use cases.
- Analysis: Overview
  - Inputs:
    - Use cases
    - Domain model
  - Activities:
    - Draft Software Architecture
    - Realize Use Cases into Sequence Diagrams
    - Refine Domain Model into Analysis Class Diagram
  - 1st step bridging external view to internal view (sys behavior to sys implementation)
  - Req (use cases) give external view
  - Domain model gives structural info about business entities
  - Analysis performed to understand how business' can be implemented and support use cases via *Use Case Realization*
    - Adding ops to the business entities

        - Move closer to an actual class in pgrm
- Diff btwn Analysis & design
    - Usually both seen as 1 activity, made worse by same UML diagram
    - Analysis: descrbe strucure of real-world application, focus on req of sys
    - Design: describes the structure of a proposed software sys, focuses on the software structure that implements the sys
    - Analysis model to id relevant objects in real-world sys, design model for adding certain details
    - "do the right thing (analysis), then do the right thing (design)"
- Software Architecture
    - High lvl desciption of overall sys: top level structure of subsys, role & interaction of the subsys
    - Grouping of classes to improve cohesion and reduce coupling
        - Cohesion- set of things that work well together
        - Coupling- inter-dependency btwn 2 entities
    - Ex:
        - 

Take the *minesweeper* game as an example, and identify the high level components:

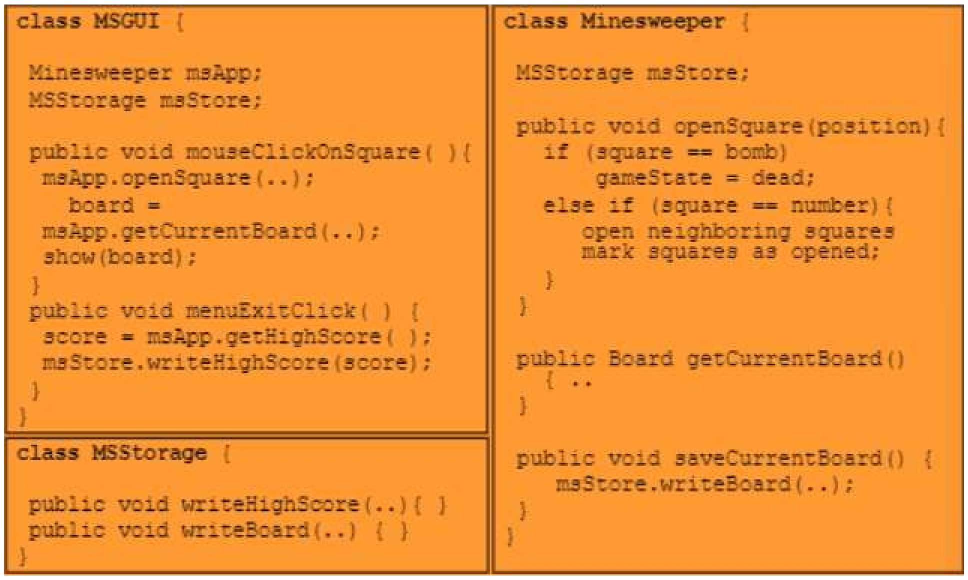
- Display:
    - Showing the game graphically.
- Application Logic:
    - Determining the flags, numbers.
    - Checking whether there are more mines, etc.
- Record Storage:
    - Storing high scores, setting, etc.

        - Sys 1:
            - 

```
class Minesweeper{

public void ClickOnSquare( ){
   if (square == bomb) {
       gameState = dead
       show dead icon
       write high score to file
   }
   else if (square == number){
       open neighboring squares
       mark squares as opened
       display new board
    }
}
..........// some other code
}
```

- Bad cohesion:
    - Display function all over place
    - Application logic buried under other operations
    - Storage functions not clearly separated
- Lost coupling:
    - No interdependency bc only 1 class

▪ Sys 2:

- 

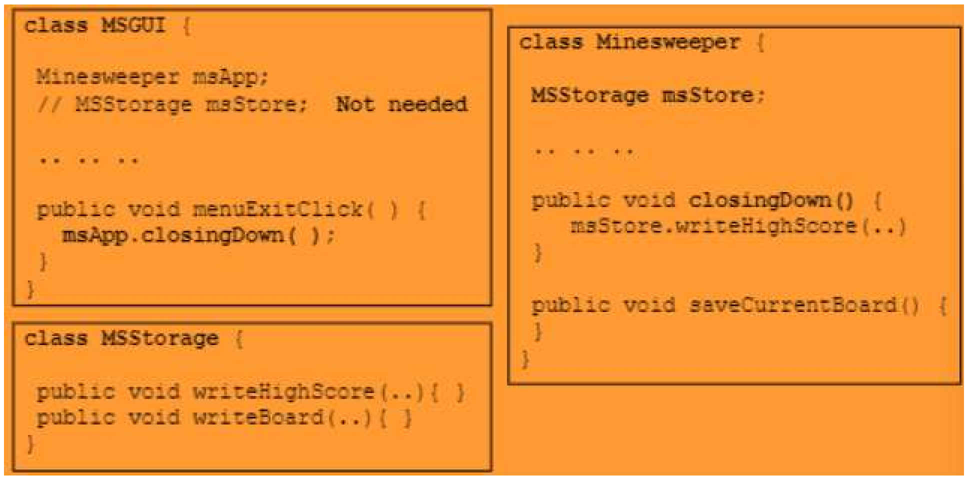
```
class MSGUI {

 Minesweeper msApp;
 MSStorage msStore;

 public void mouseClickOnSquare( ){
  msApp.openSquare(..);
    board =
  msApp.getCurrentBoard(..);
  show(board);
 }
 public void menuExitClick( ) {
  score = msApp.getHighScore( );
  msStore.writeHighScore(score);
 }
}
```

```
class MSStorage {

 public void writeHighScore(..){ }
 public void writeBoard(..) { }
}
```

```
class Minesweeper {

 MSStorage msStore;

 public void openSquare(position){
   if (square == bomb)
       gameState = dead;
   else if (square == number){
       open neighboring squares
       mark squares as opened;
   }
 }

 public Board getCurrentBoard()
   { ..
 }

 public void saveCurrentBoard() {
    msStore.writeBoard(..);
 }
}
```

- Cohesion: Each class groups logically similar functionality together
    - Class MSGUI: UI, I/o to screen
    - Class minesweeper: computation and logic
    - Class MSStorage: file operations
- Coupling
    - Some interdependencies, can be improved
    - Observe the MSGUI uses MSStorage directly
    - If we sub another UI, the high score saving functionality need to be recorded

▪ Sys 3:

- 

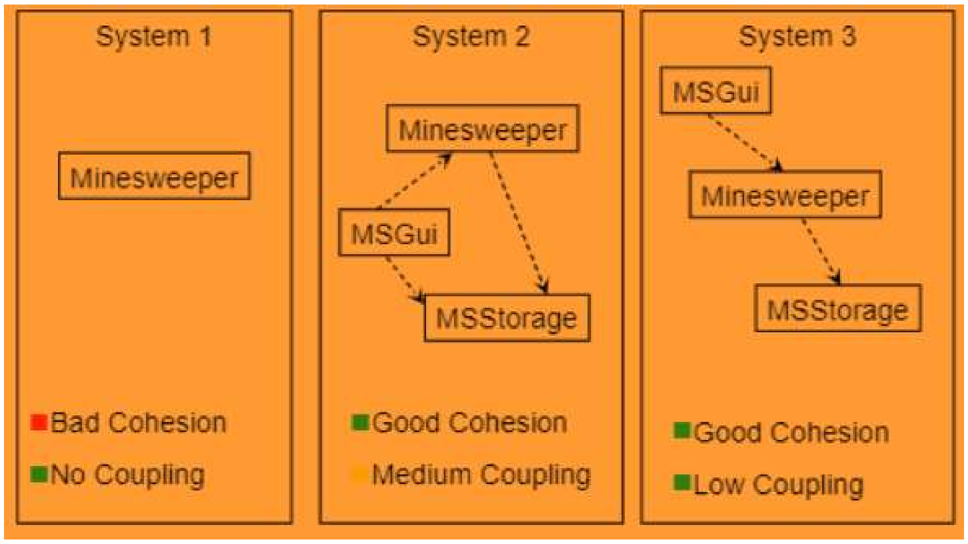
```
class MSGUI {

 Minesweeper msApp;
 // MSStorage msStore;   Not needed

 .. .. ..

 public void menuExitClick( ) {
   msApp.closingDown( );
 }
}
```

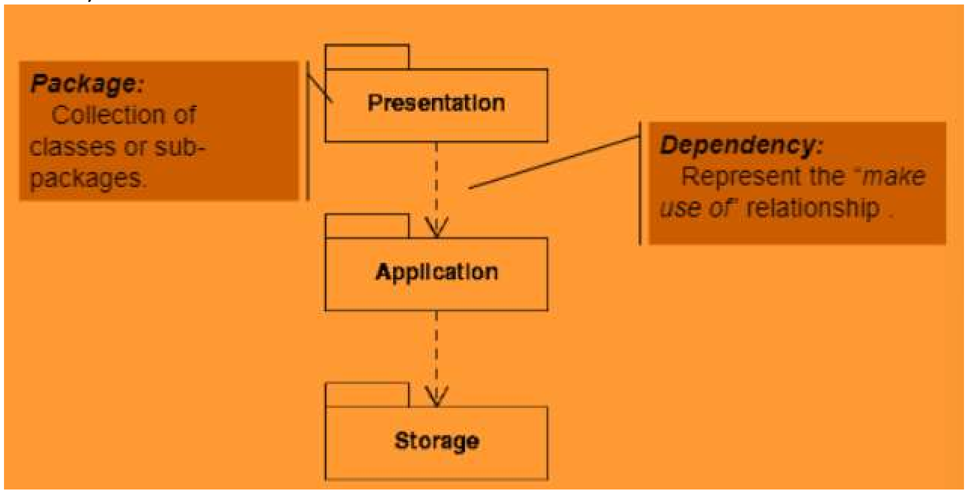
```
class MSStorage {

 public void writeHighScore(..){ }
 public void writeBoard(..){ }
}
```

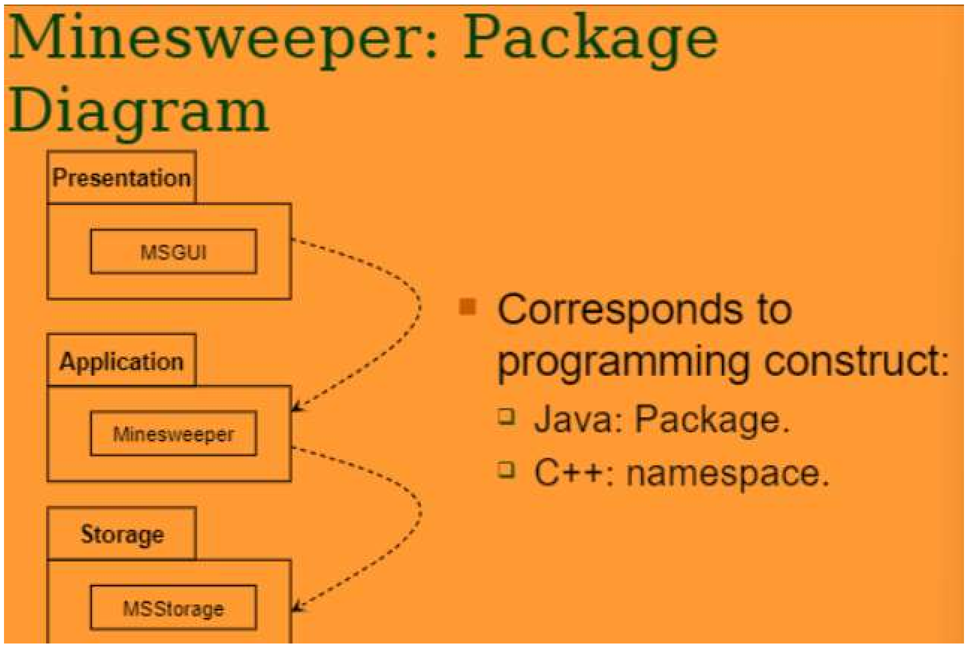
```
class Minesweeper {

 MSStorage msStore;

 .. .. ..

 public void closingDown() {
    msStore.writeHighScore(..)
 }

 public void saveCurrentBoard() {
 }
}
```

- Coupling: lowered, MSGUI depends on Minesweeper only, which depends only on MSStorage
- Low coupling lets easy maintenance, so
    - Changing MSGUI to MSTextUI won't affect main application at all
    - Can swap in another storage class

▪

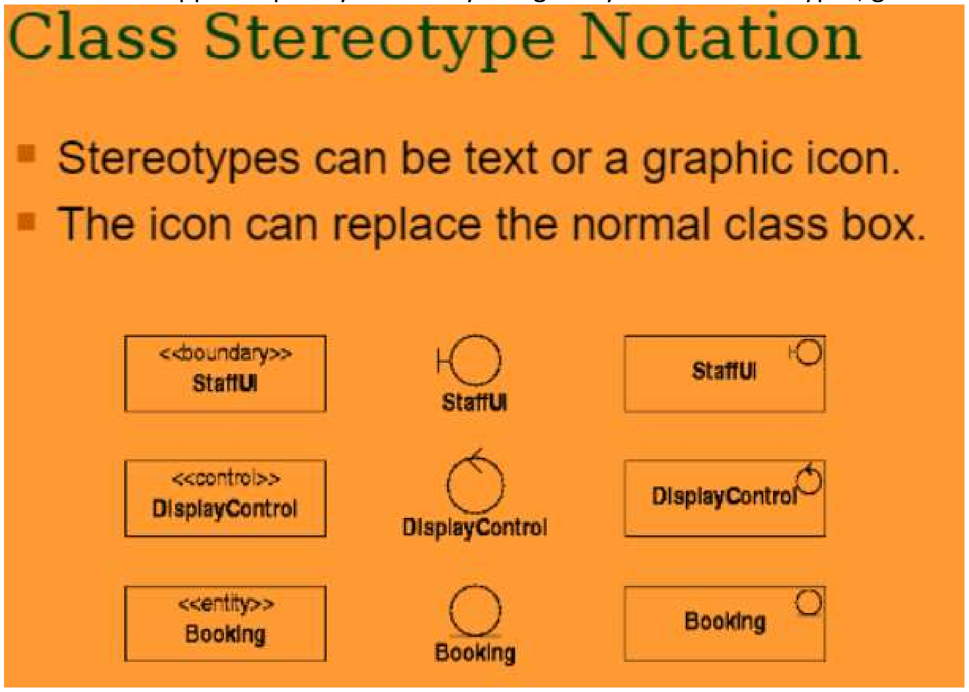
            - 

            - Observations:
                - Trade btwn cohesions and coupling
                - 3 catefories of functionality is widely used and better acceptance
                - Help shaping software architecture
- Layered architecture
    - Oldest idea in SE, split into 3 layers
        - Presentation layer- UI
        - Application Layer- underlying logic, implements functionality of sys
        - Storage Layer- deals w/ data storage: files, database, etc.
    - Also layers higher lvl abstraction: each can have many classes, or many packages (group of classes)
    - 
Package: Collection of classes or sub-packages.
Presentation
Application
Storage
Dependency: Represent the "make use of" relationship.
    -

- 

  ## Minesweeper: Package Diagram

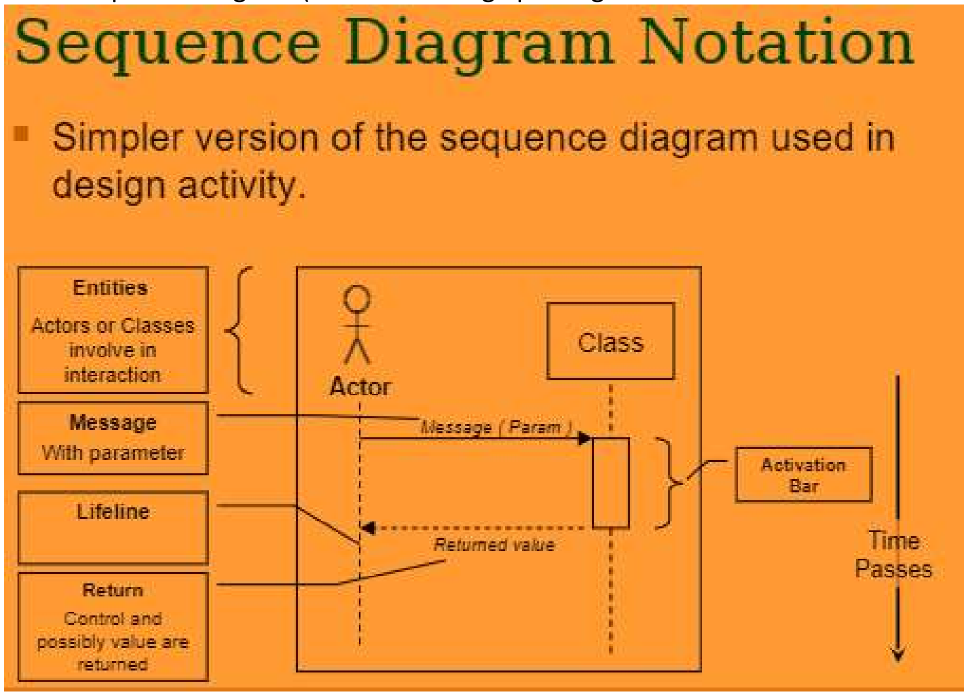
  - Corresponds to programming construct:
    - Java: Package.
    - C++: namespace.

- Layered architecture Advantages:
  - Layers aim to insulate a sys from the effects of change
  - Ex:
    - Ui often change, but app lay don't use presentation layer
    - So changes to sys should be restricted to presentation layer classes
  - Similarly, details of persistent data storage are sep from app logic
- Analysis Class Stereotypes
  - W/in this architecture, objects can have diff typical roles:
    - Boundry objects: interact w/ outside actors
    - Ctrl objects: manage use case behavior
    - Entity objects: maintain data
  - These repped explicitly in UML by using analysis class stereotypes, give extra info for objects
- 

  ## Class Stereotype Notation

  - Stereotypes can be text or a graphic icon.
  - The icon can replace the normal class box.

- Use Case Realizations: Overview
  - Implementation of use case
  - Steps:

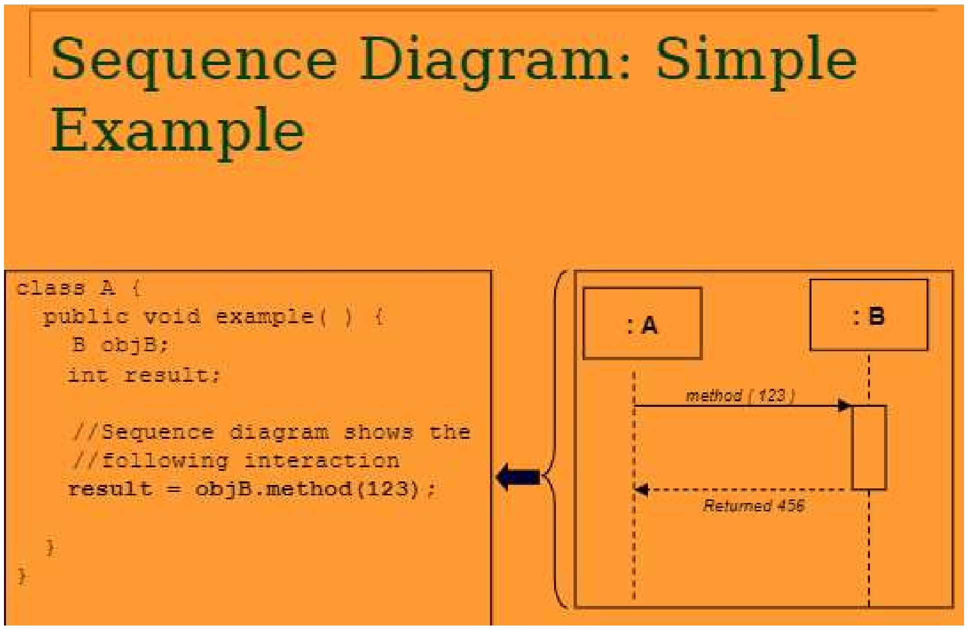
        - Go thru each use case
        - Id interaction using UML sequence diagram
        - Refine domain class diagram
    - Possible refinements
        - Adding new class/data
        - Define association navigability (direction) and role name
        - Add operation to an existing class
    - Focus is on sys functionality during analysis phase
        - Ignore other layer (presentation and storage)
        - Concentrate on the application layer
- Sequence Diagram
    - UML Diagram to:
        - Emphasize the interaction in the form of messages.
        - Identify the party involved in the interaction.
        - Outline the timing of the message.
        - Be mainly used in Analysis and Design activities.
    - Analogy help understand
        - Oop, method invocations can be modeled as messages passing btwn objects, and sequence diagram (records message passing
    - 

      Sequence Diagram Notation

      Simpler version of the sequence diagram used in design activity.

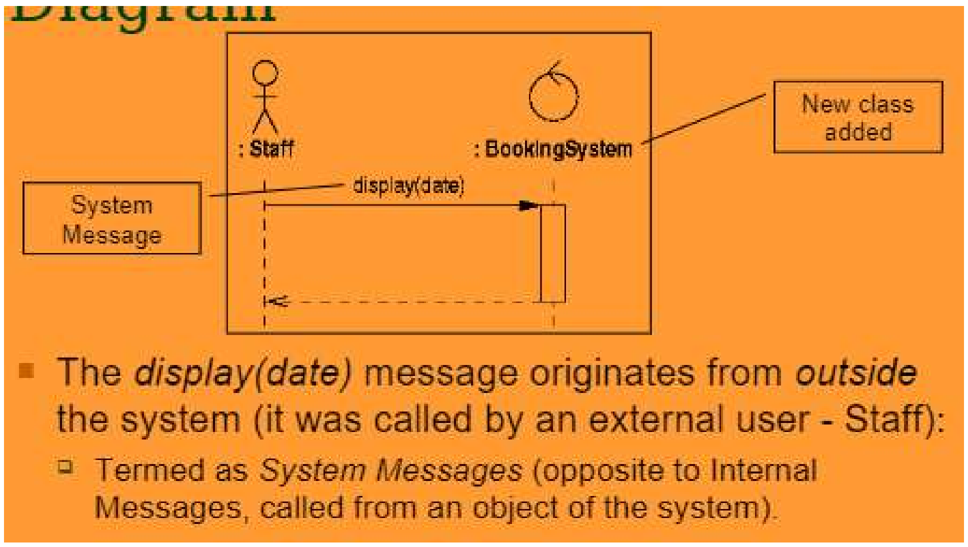
        - Time passes from top to bottom
        - Classes and actors @ top:
            - Only show those participating in this interaciton
            - Each instance has a lifeline
        - Messages shown as arrows btwn lifelines:
            - Labelled w/ op names and parameters
            - Return messages (dashed) show return of ctrl
            - Activations show when the receiver has ctrl
        - Ex:

          Sequence Diagram: Simple Example

- 

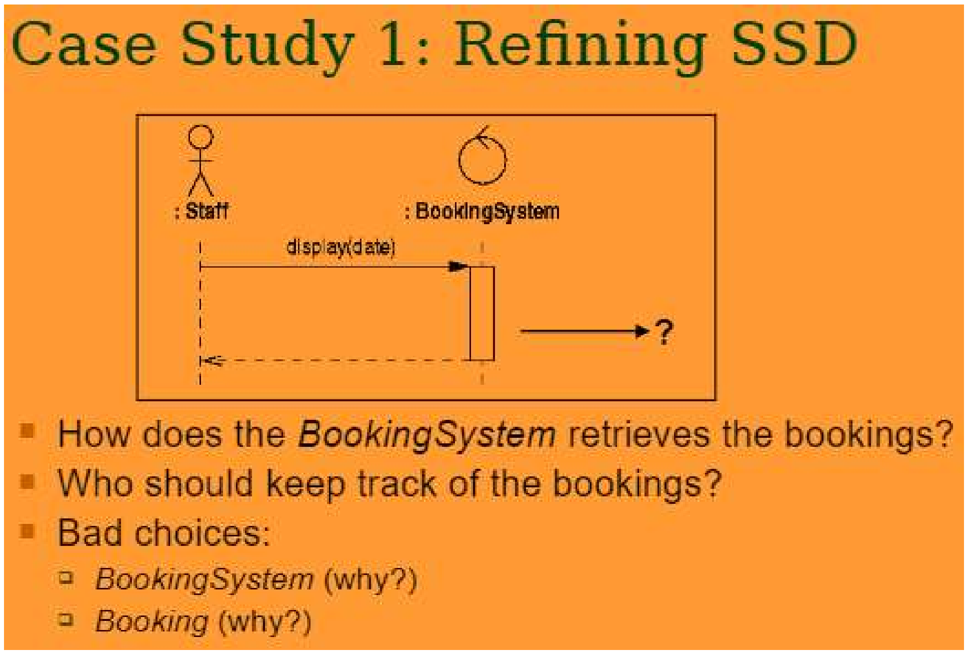
Sequence Diagram: Simple Example

```
class A {
  public void example( ) {
    B objB;
    int result;

    //Sequence diagram shows the
    //following interaction
    result = objB.method(123);

  }
}
```

- Case Study 1: Use case realization: Take display booking use case as ex
  - Initial realization has: staff actor, system, message(s) passed btwn them
  - Domain model only concentrate on entities in real world
  - New class bookingSystem added:
    - Ex of domain model refinement
    - Categorized as a ctrl stereotype- gets high lvl request from the user, contacts the relevant classes to fulfill the request
    - Boundry stereotype not appropriate (emphasizes on the use case behavior, no I/o)
    -
    - 

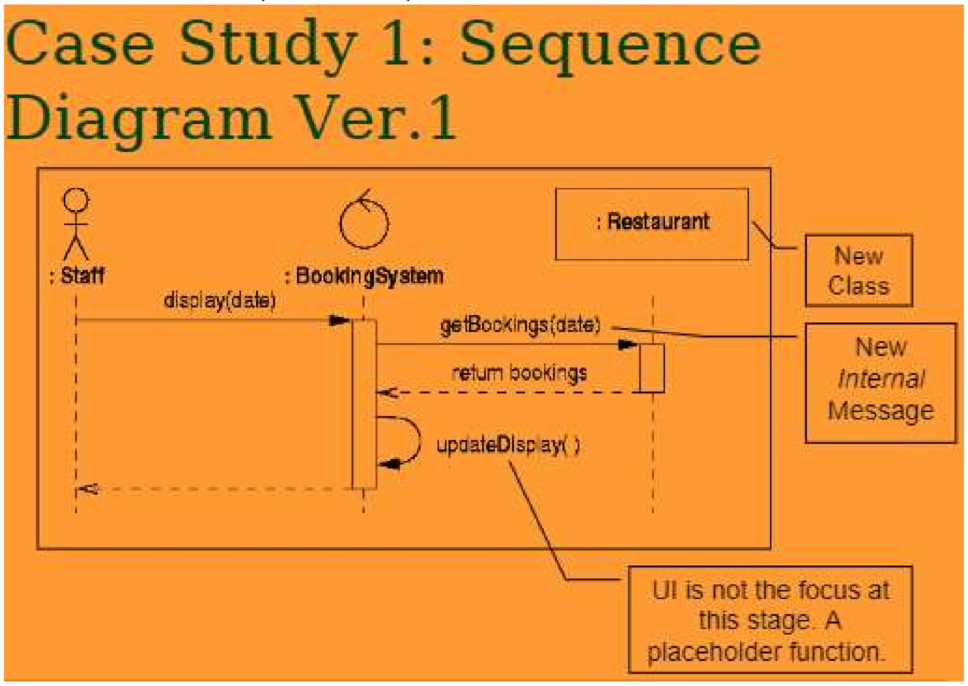
- The *display(date)* message originates from *outside* the system (it was called by an external user - Staff):
  - Termed as *System Messages* (opposite to Internal Messages, called from an object of the system).

- Sys sequence diagram
  - Shows only sys messages: informal sybtype of sequence diagram, concentrates on user-sys exchange only, graphical rep of a use case, serves as a starting pt for more complete seq diagram
- Case Study 1: Refining SSD
  -

- 

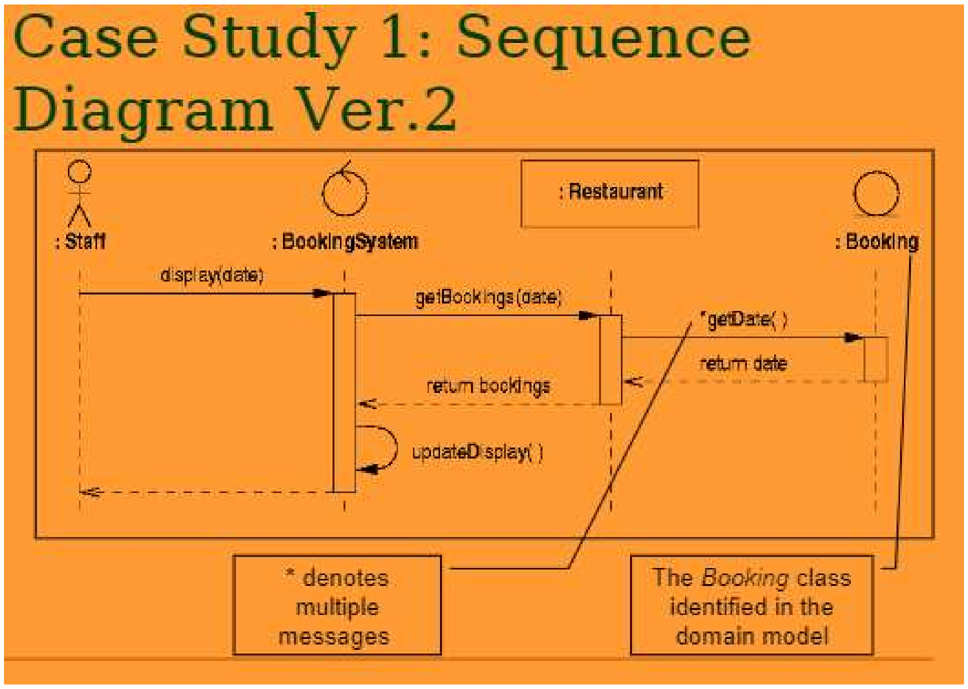
  ## Case Study 1: Refining SSD

  : Staff
  : BookingSystem
  display(date)
  ?

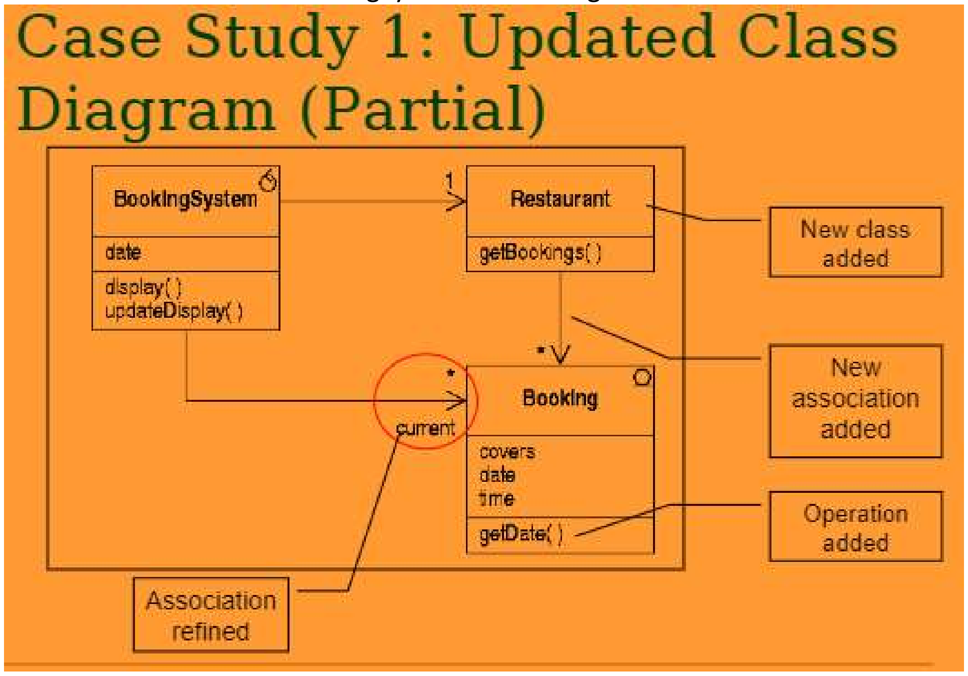
  - How does the *BookingSystem* retrieves the bookings?
  - Who should keep track of the bookings?
  - Bad choices:
    - *BookingSystem* (why?)
    - *Booking* (why?)
- Class to store a collection of Bookings needed, adding ability to bookingSystem not good (it should only handle sys messages, lowers the cohesion of the class)
- Booking class in domain model only records detail of one booking, not collection
- Another class needed, Restaurant, can take care of other collections
- 

  ## Case Study 1: Sequence Diagram Ver.1

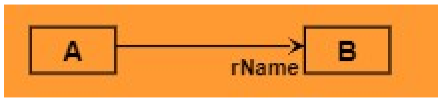
- Seq diagram ver 1 tells us:
  - Staff wants to check a booking for certain day, BookingSystem gets the request & asks the Restaurant to give all bookings on the day
- Follow Up:
  - How can Restaurant find the relevant bookings from all bookings record?
  - Can work if check the date on each booking record, and collections only those that match

- 

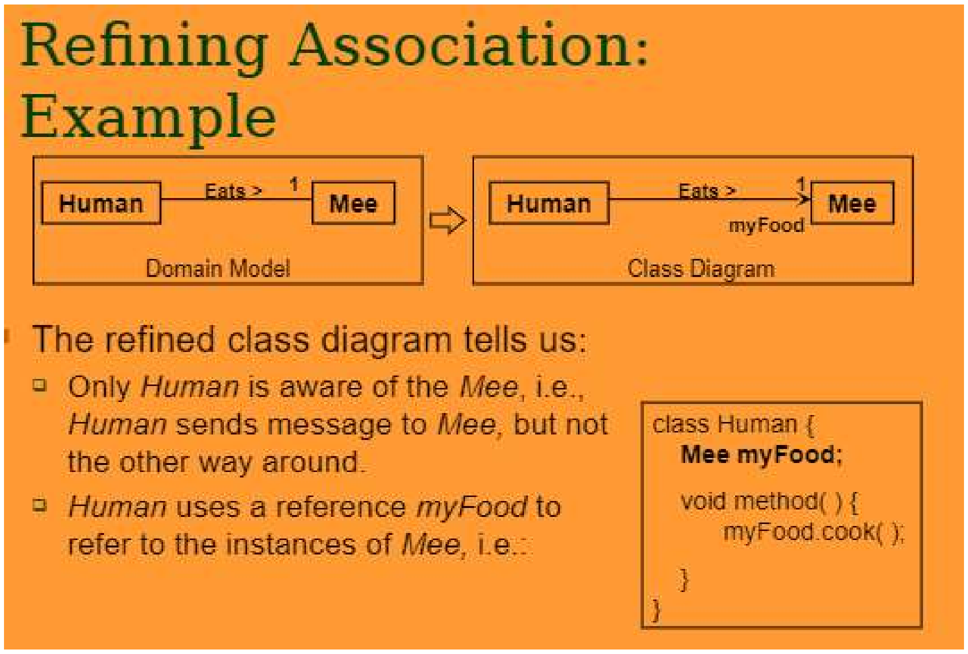
  ## Case Study 1: Sequence Diagram Ver.2

- Key changes
  - New Restaurant & BookingSystem classes w/ association btwn them
  - New association from restaurant to Booking (restaurant keeps link to all bookings, messages sent from restaurant to bookings)
  - Association from BookingSystem to Booking
- 

  ## Case Study 1: Updated Class Diagram (Partial)

* Refining Association, closer to internal rep, more technical info needed for association
  - 2 possible refinements:
    - 

    - Add navigability (arrow show only A can interact w/ B (not vice versa)
    - Add role name: A uses rName as the reference name of B
  -

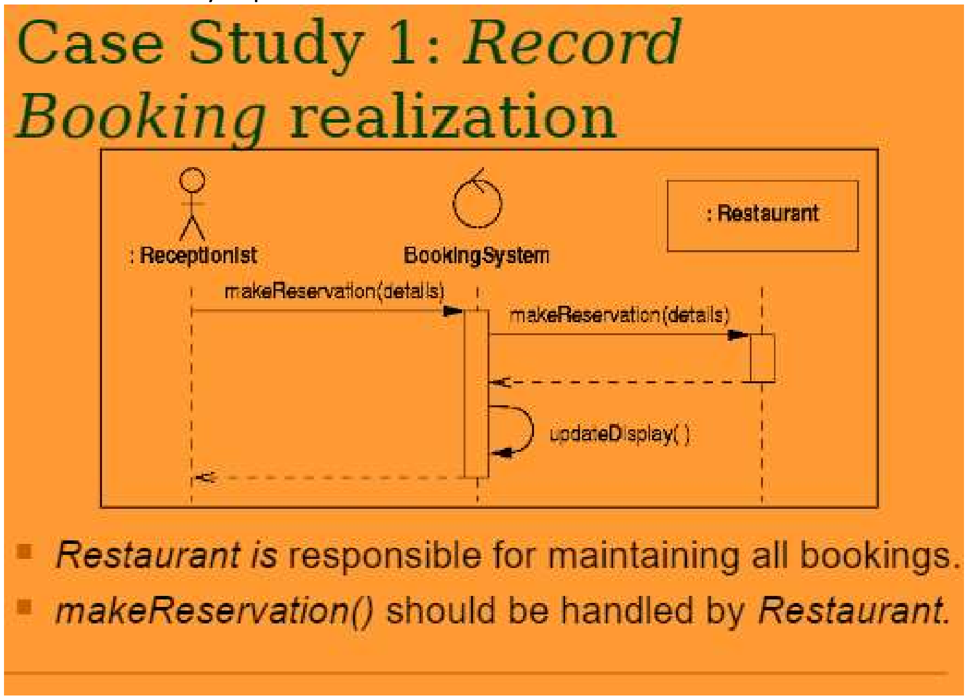
- 

  ## Refining Association: Example

  Human
Eats >
1
Mee
Domain Model
Human
Eats >
1
myFood
Mee
Class Diagram

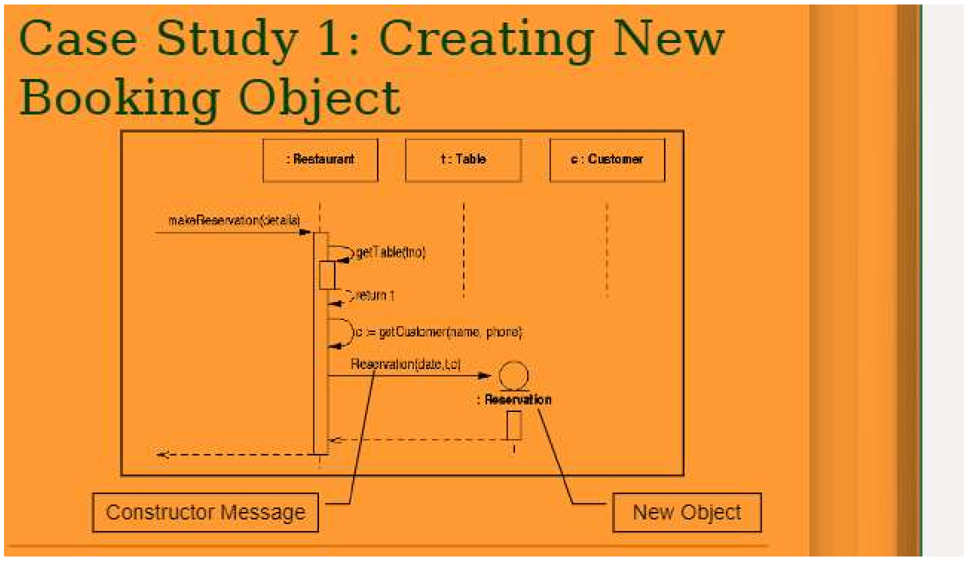
  - The refined class diagram tells us:
    - Only *Human* is aware of the *Mee*, i.e., *Human* sends message to *Mee*, but not the other way around.
    - *Human* uses a reference *myFood* to refer to the instances of *Mee*, i.e.:

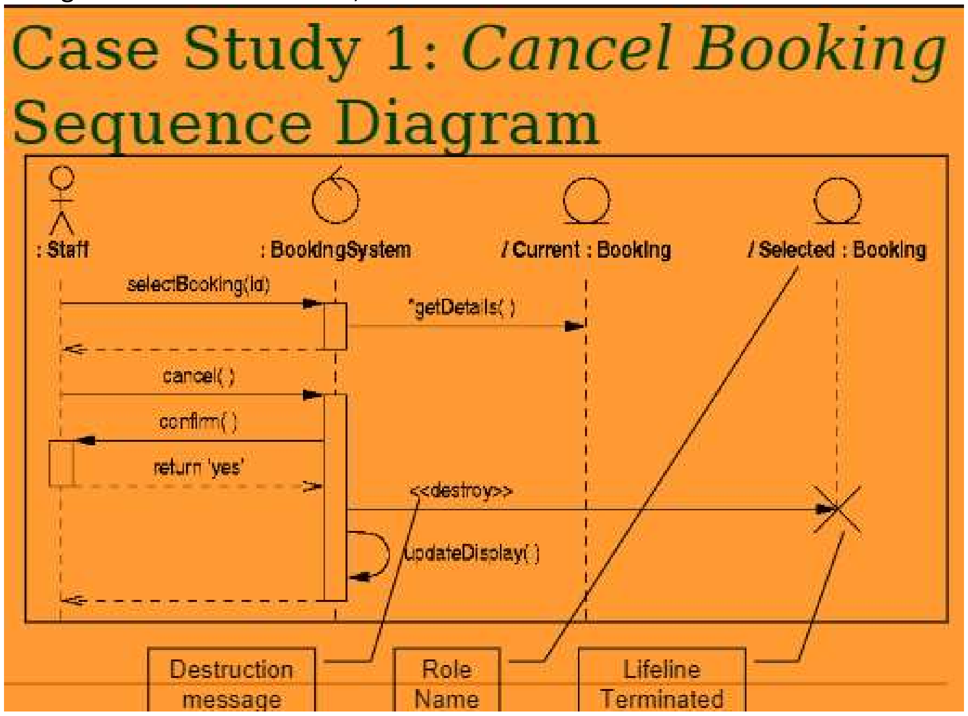
  ```
  class Human {
     Mee myFood;
     void method( ) {
         myFood.cook( );
     }
  }
  ```

- Case Study 1: Display Booking Realization
  - Complete now
  - Review:
    - Stop @ (SDS), figure out how entities in domain, can
- Case Study 1: Continue Use of Case realization
  - Basic tech already explained
  - 

  ## Case Study 1: *Record Booking* realization

  - *Restaurant is* responsible for maintaining all bookings.
  - *makeReservation()* should be handled by *Restaurant.*
- Booking realization,

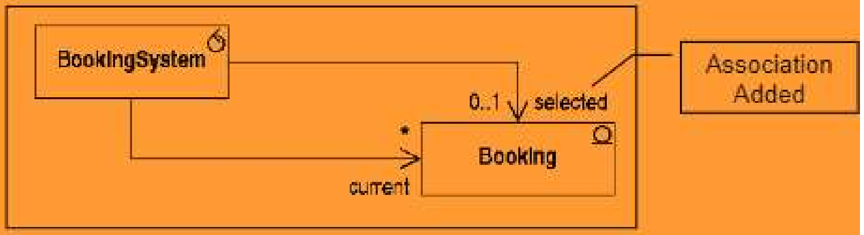
- 

- Cancel Booking realization
    - 3 stages: select book to cases,
    - 

    -

○

Case Study 1: Domain Model Refinement Continues

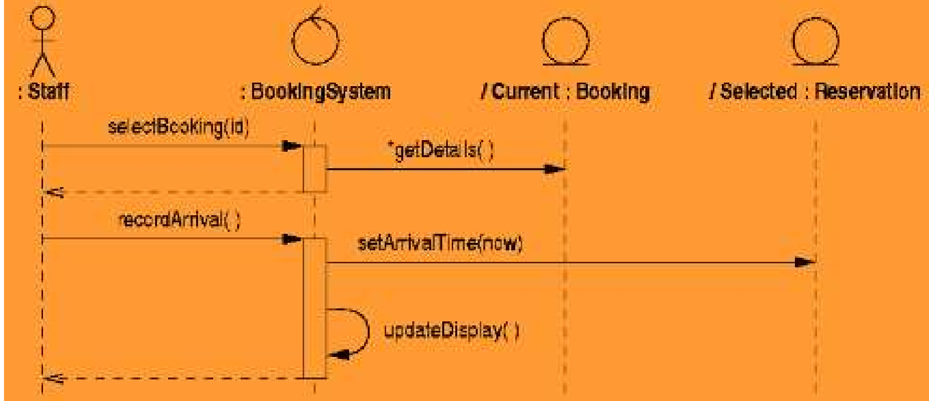


- *BookingSystem* has the responsibility to remember which booking is selected.
- Adds an association to record this.

○

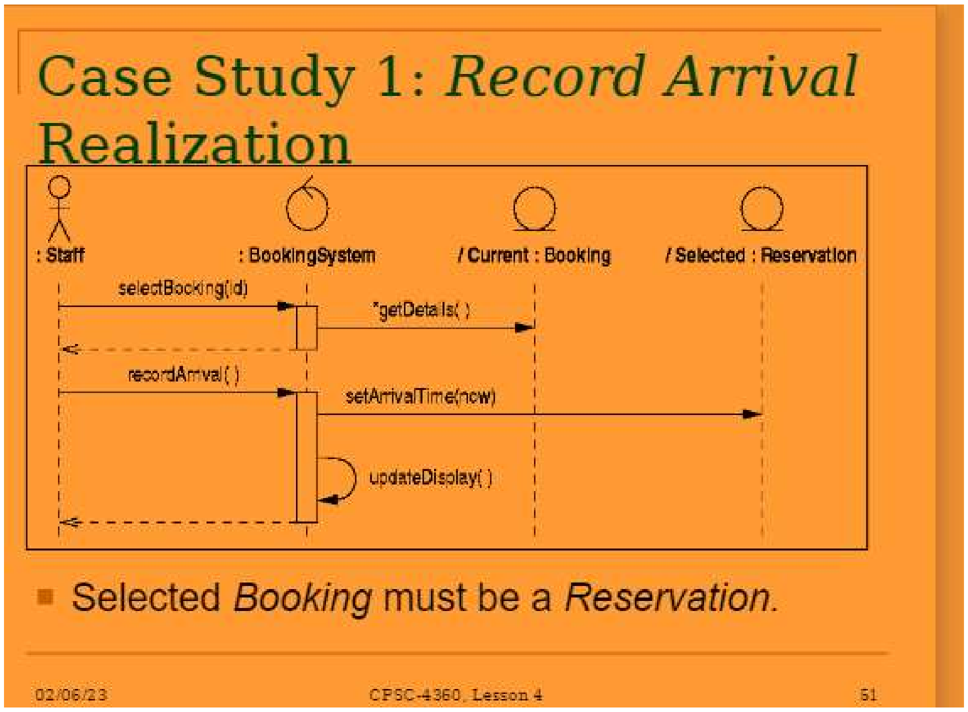
Case Study 1: *Record Arrival* Realization

○



○

○



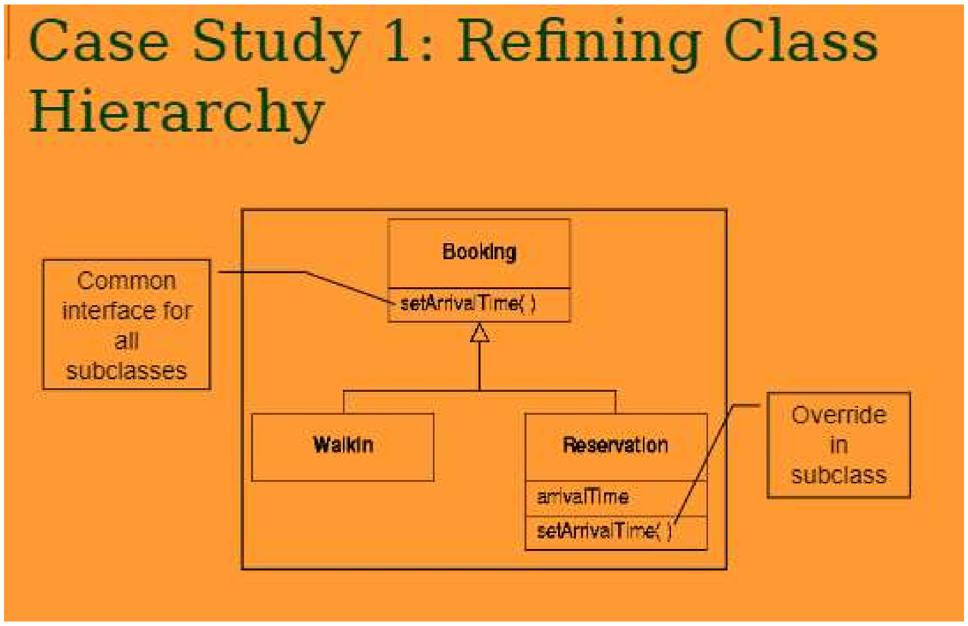
Case Study 1: *Record Arrival* Realization

- Selected *Booking* must be a *Reservation*.

02/06/23 CPSC-4360, Lesson 4 51

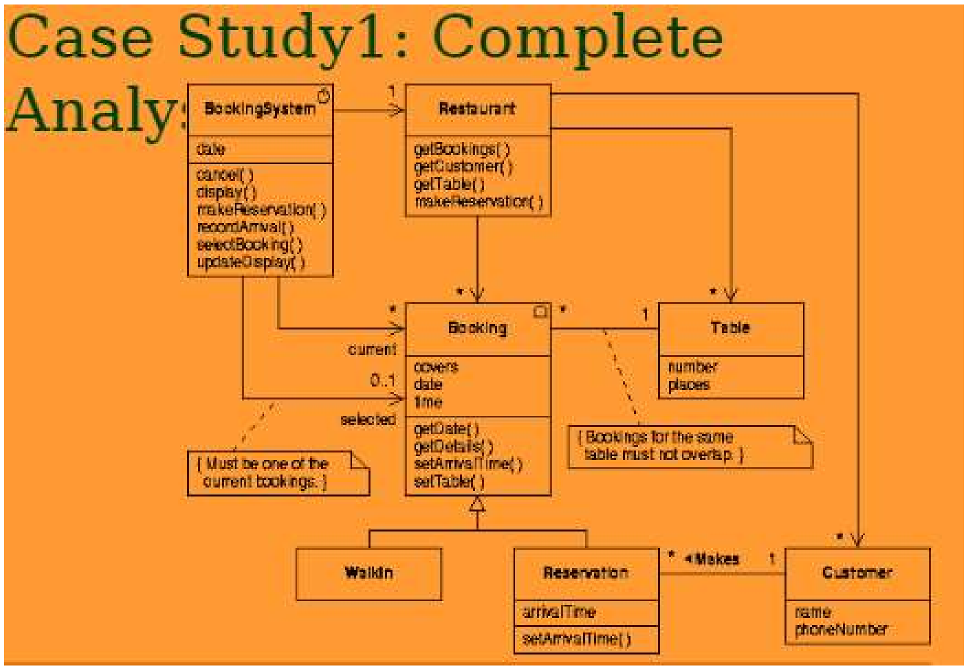
○ SA

○



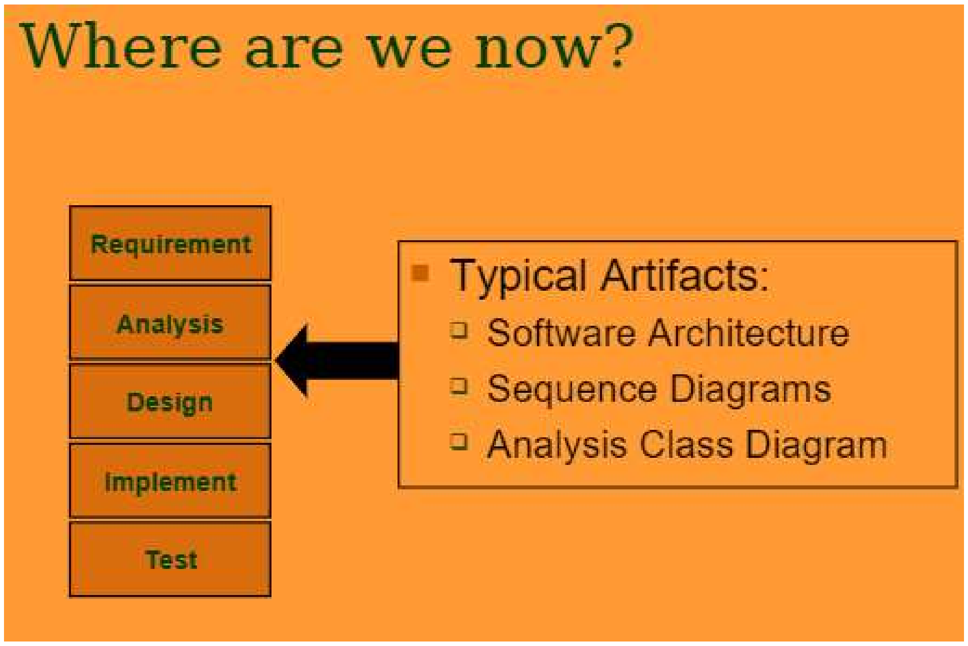
Case Study 1: Refining Class Hierarchy

○

- 

  ## Case Study1: Complete Analy

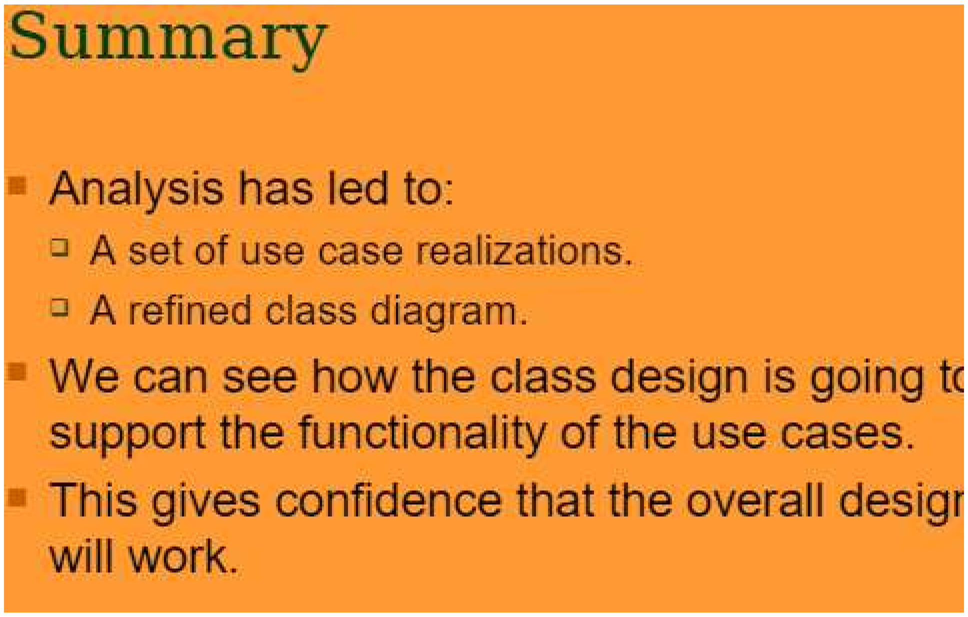
-

- 

  ## Where are we now?

  - Typical Artifacts:
    - Software Architecture
    - Sequence Diagrams
    - Analysis Class Diagram

-



Summary

- Analysis has led to:
  - A set of use case realizations.
  - A refined class diagram.
- We can see how the class design is going tc support the functionality of the use cases.
- This gives confidence that the overall desigr will work.

Assignment 4

Friday, February 10, 2023 9:45 PM

Question 1
10 Points

Perform the following activities for Case Study 2, i.e., The Monopoly Game (introduced in Lesson 3).

Draw a use case diagram and write use case description for basic course of events.

Identify any possible alternative or exceptional course of events. Construct the domain model.

Design a three-layer software architecture for the Monopoly Game.

Distinguish between cohesion and coupling.

Compare the cohesion and coupling for the Monopoly Game.

A:
Cohesion is a set of things that work well together. In monopoly, relating the tokens and money to the players makes the system easier to work with. Coupling is the inter-dependency between two entities. To lower coupling, rather than keeping location connected to the player, it should just be related to the game, even if people own a location.

Question 2
10 Points

From the following description, describe the use cases, by identifying the actors, basic course of events and possible alternatives:

“The PSL company has a website allowing online orders to buy Pizzas, Sandwiches, or Lasagna in the Lamar University campus. A customer can either walk-in at PSL to buy meals or he/she can make online orders using the PSL website. Either way, after selecting his/her choices and doing the payment, the PSL acknowledge order delivering the meals and the receipt.”

Draw a sequence diagram realized from this use case description.

A:
This use case will have one actor, the customer. The customer has communication between two possible alternative courses, the use cases of either "Walk-in at PSL", or the "PSL website". Both these use cases will then have a <<include>> use case dependency to the use case "order", which will then have an <<include>> use case dependency to the use case "payment".

Question 3
10 Points

Describe a domain model for the behavior of entering a university building using the student card. They can enter in the building without scanning the student card by the card reader between 7am to 7pm, otherwise they need to scan it. No matter the time, the lab rooms can be open by scanning the card reader only by the students that have permission to enter. Design an activity diagram modeling the considered domain model.

A:
The domain model would include a class, namely student. The student would have the attribute of have

a student card. There would be an association between the student and the building (class), with a constraint that the card would have to be used to enter if the time was not between 7 am to 7 pm. There would be another association between the building and the lab (class), with a constraint that the card would have to be used to enter.

Question 4
10 Points

What activity is NOT requested while designing the Presentation layer?

Decide and draft the appropriate type of User Interface;
Study the interaction to perform each of the use case;
Identify what data needs to be persistent;
Derive the classes needed;
None of the above.

Question 5
10 Points
Which of the following is true about cohesion?

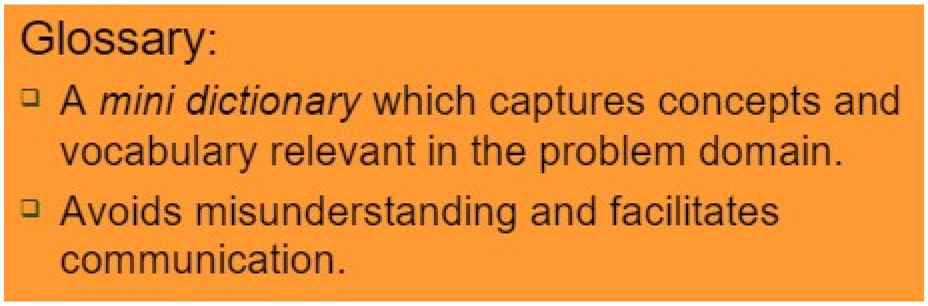
A good software architecture is the one with low cohesion and high coupling;
A higher cohesion results a good coupling;
The cohesion refers to inter-dependency among modules;
A functionally cohesive module is very hard to design and maintain;
The cohesion refers to identify the subsystems with a conscious attempt to keep things together that belong together.

Assignment 1

Thursday, February 2, 2023 6:08 PM

A) Define the business model of the software project described in Content.

Business modelling means to create the use case model, to define the use cases, to create the domain model, and to create the glossary



Glossary:

- A *mini dictionary* which captures concepts and vocabulary relevant in the problem domain.
- Avoids misunderstanding and facilitates communication.

L3-slide 39

B) Describe the use cases as well as the use cases diagram. == just describe, nm, make the use case diagram :(

C) Show how the use cases can be realized into sequence diagrams for the software project. == draw a sequence diagram w/ use cases

D) Describe a feasible domain model and ways to refine it into an analysis class diagram. == just describe

This is just class diagram, L4, domain model is L3

A:

A)

B)
The use cases for the project would include the 'Enter Phone Number', 'Enter Occupation', 'Enter Describable Characteristics', and 'Show Possible Name-Numbers'. Other possible use cases that we may or may not include would be 'Relevant Words Checker', 'System Administration Checker', and 'Word Overflow Checker' (numbers are sometimes allowed to go past 7 digits, but we should always make sure). The basic use case diagram would have an actor named 'User'. The actor would have the inherit relationships to the use cases of 'Phone Number', 'Describable Characteristics', and 'Occupation'. Since the use case diagram focuses on the user experience and interface, the model will most likely include a "Black Box" where the code and database work together to find appropriate combinations. In the use case diagram, all the inherited use cases would have communication with the 'Software' use case (our black box'. This use case would then have the include use case dependency with the use case of 'Possible Name-Numbers', so at the end of the process, the actor ('User') would be able to choose out the best combination themselves.

D)
A simple domain model would include a class named 'User', which has the attributes of 'Phone Number' and 'Relations'. This class would have an association with the class 'Database'. Finally, the database would have an association with the class 'Optional Choices', which has the attributes of the various phone number-word combinations they could choose.
This can then be refined into an analysis class diagram by the generalization of a few other classes, such as 'Relevant Words', 'Cartesian Products', 'Useful Combinations', and 'Overflow'. These would all be generalizations of the class 'Database, which was in the simple domain model. Along with this, the 'User' class would have other generalization classes, such as 'Occupation' and 'Describable Characteristics'.