



**SE-2002**

# **SOFTWARE DESIGN AND ARCHITECTURE**

**RUBAB JAFFAR**

[RUBAB.JAFFAR@NU.EDU.PK](mailto:RUBAB.JAFFAR@NU.EDU.PK)

## **Robustness diagram**

### **Lecture # 13, 14, 15**

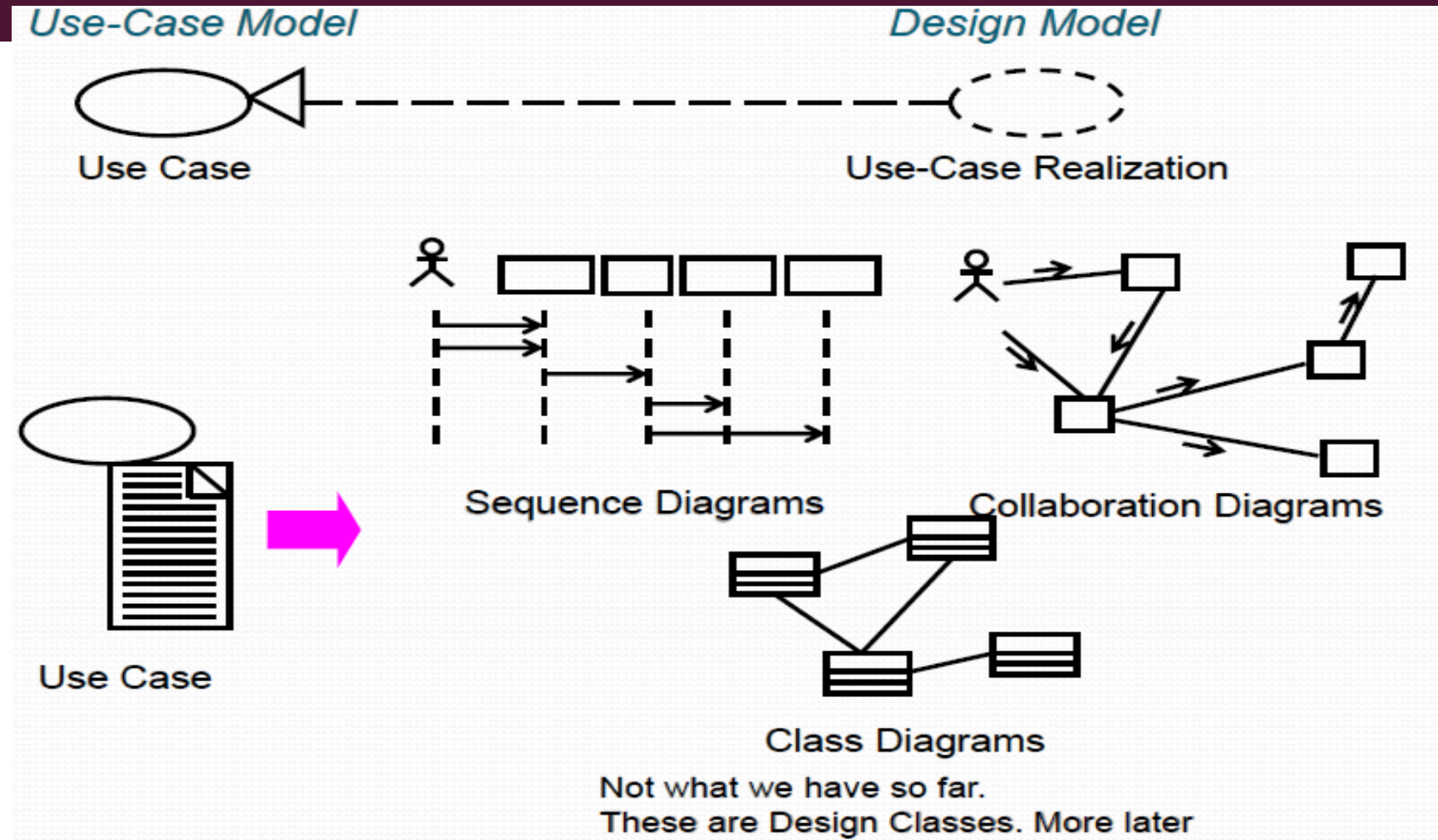
# TODAY'S OUTLINE

- Quiz
- Use-case Realization
- Analysis Classes
- ECB Pattern
- Exercises
- Revision

# USE CASE REALIZATION

- Use-case realization describes how a particular use case is realized within the Design Model, in terms of collaborating objects.
- The diagrams that may be used to realize a use case realization may include:
- Interaction Diagrams (sequence and/or collaboration diagrams) can be used to describe how the use case is realized in terms of collaborating objects.
  - These diagrams model the detailed collaborations of the use-case realization.
- Class Diagrams can be used to describe the classes that participate in the realization of the use case, as well as their supporting relationships.
  - These diagrams model the context of the use-case realization.

# USE CASE REALIZATION



# WHO DOES USE CASE REALIZATION?

- A designer: responsible for the integrity of the use-case realization.
- Must coordinate with the designers responsible for the classes and relationships employed in the use-case realization.
- The use-case realization can be used by class designers to understand the class's role in the use case and how the class interacts with other classes.
  - This implies that a team will/may distribute responsibilities for each class to developers.
- This information can be used to determine/refine the class responsibilities and interfaces.
- Let's find the classes from different behaviors the classes must provide...

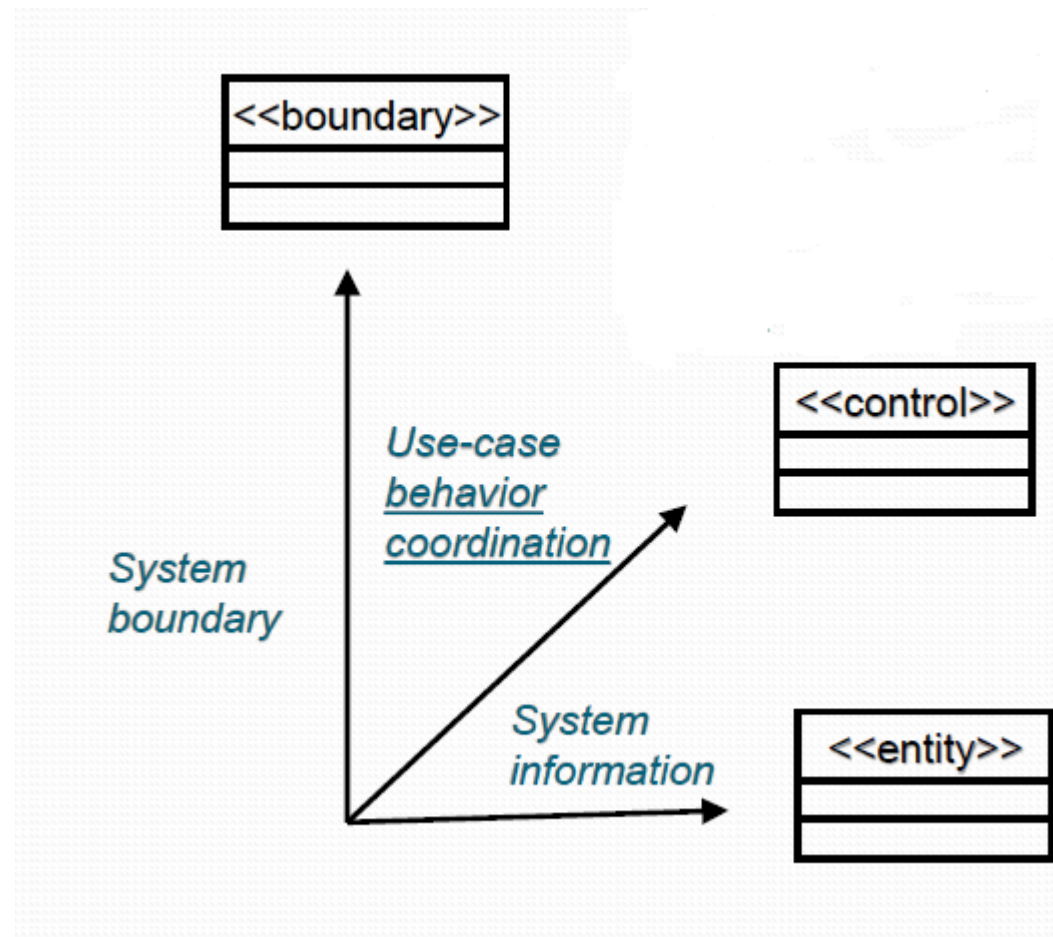
# IDENTIFYING CANDIDATE CLASSES FROM BEHAVIOR

- Will use three perspectives of the system to identify these classes.
  - The 'boundary' between the system and its actors
  - The information' the system uses
  - The 'control logic' of the system
- Will use stereotypes to represent these perspectives (boundary, control, entity)
  - These are conveniences used during analysis that will disappear or be transitioned into different design elements during the design process.
- Will result in a more robust model because these are the three things that are most likely to change in system and so we isolate them so that we can treat them separately.
  - That is, the interface/boundary, the control, and the key system entities.....

# ANALYSIS CLASSES - EARLY CONCEPTUAL MODEL

- The analysis classes, taken together, represent an early conceptual model of the system.
- This conceptual model evolves quickly and remains fluid for some time as different representations and their implications are explored.
- Analysis classes are early estimations of the composition of the system; they rarely survive intact into implementation.
- Many of the analysis classes morph into something else later on (subsystems, components, split classes, combined classes).
- They provide us with a way of capturing the required behaviors in a form that we can use to explore the behavior and composition of the system.

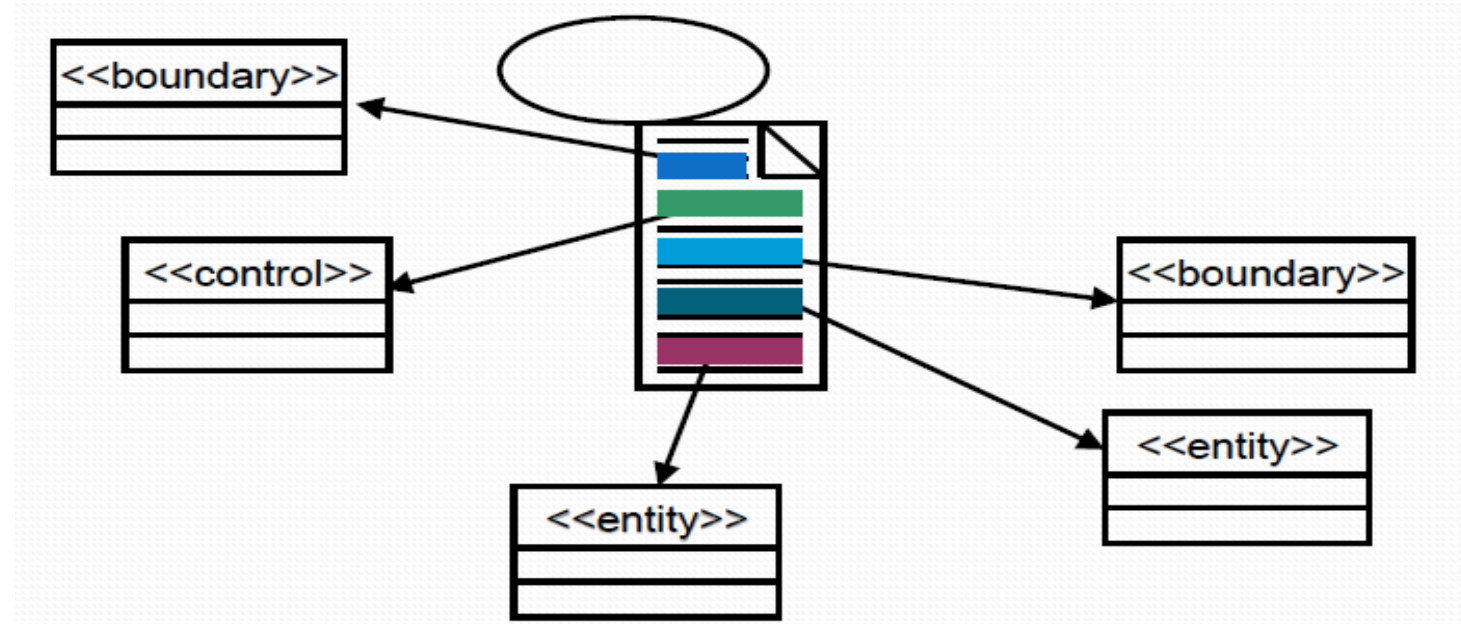
# WHAT IS AN ANALYSIS CLASS?



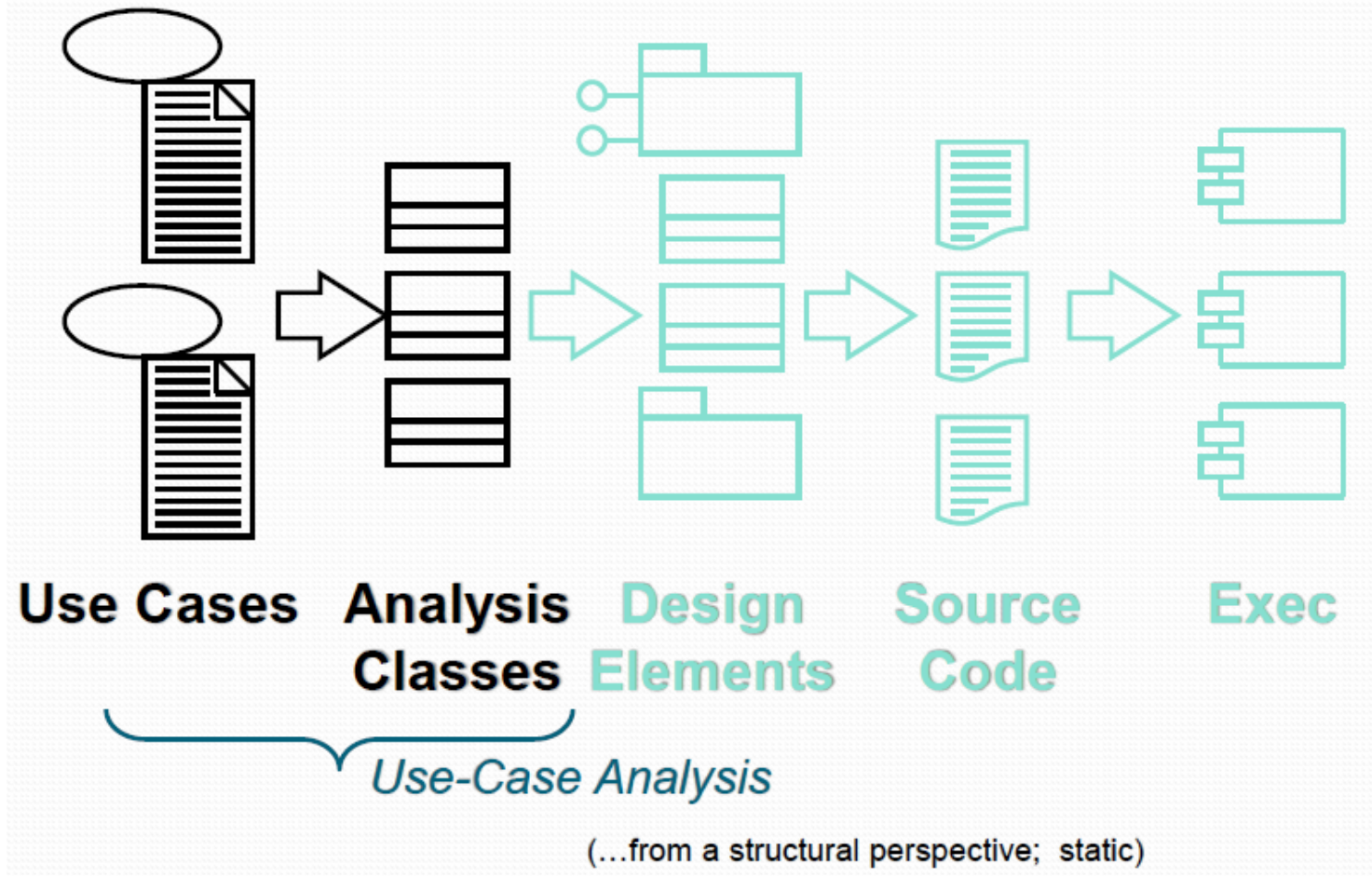


# FIND CLASSES FROM USE-CASE BEHAVIOR

- The complete behavior of a use case has to be distributed to analysis classes
- We must 'identify' these classes – identify, name, and briefly describe in a few sentences.



# ANALYSIS CLASSES: A FIRST STEP TOWARDS EXECUTABLES



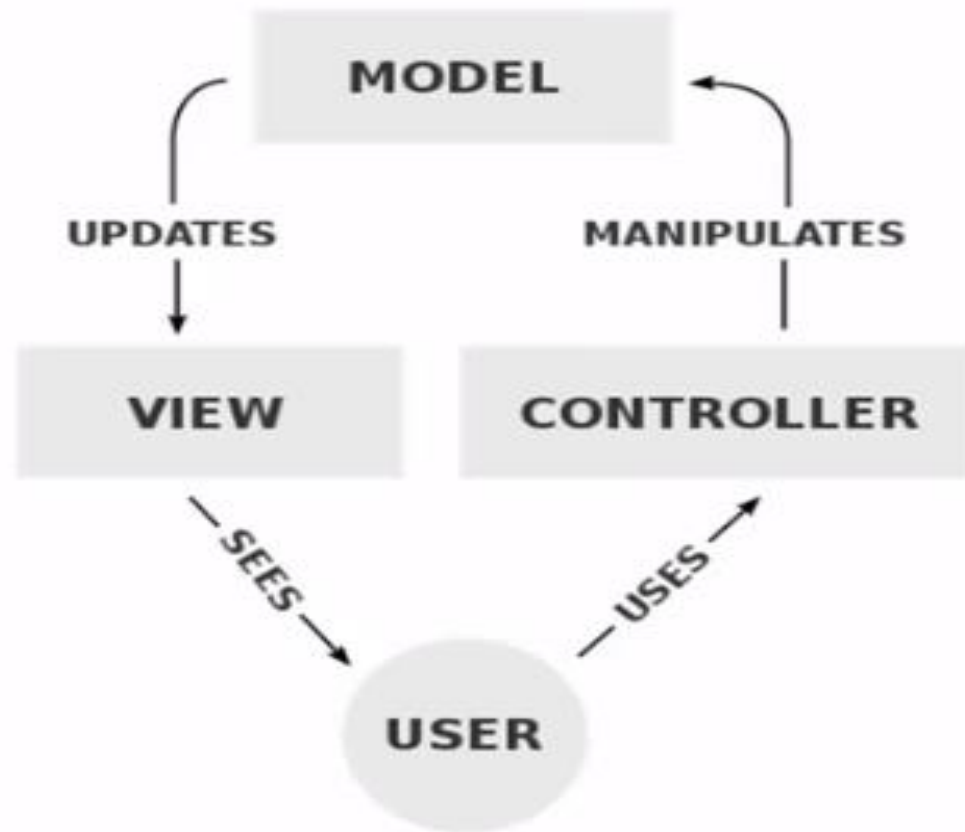
# DISCOVERING CLASSES

- **Analysis classes represent an early conceptual model for ‘things in the system which have responsibilities and behaviors’.**
- **Analysis classes are used to capture a ‘first-draft’, rough-cut of the object model of the system.**
- **Analysis classes handle primary functional requirements, interface requirements, and some control - and model these objects from the problem domain perspective.**

# ENTITY , CONTROL, AND BOUNDARY DESIGN PATTERN

- The ECB pattern represents a refinement of the model-view-controller (MVC) design pattern, a pattern that dates back to the early days of the Smalltalk-80 object-oriented language
- The goal of the MVC design pattern is to **decompose the application** into three distinct types of objects:
  - **Model Objects:** It expresses the application's behavior
  - **View Objects:** any output representation of information
  - **Controller Objects:** accepts input and converts it to commands for the model or view

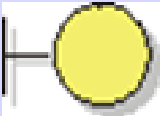


# MVC PATTERN



# ENTITY, CONTROL, AND BOUNDARY DESIGN PATTERN

- The ECB design pattern is closely related to the MVC design pattern
- As such, its goal is to decompose the application into three distinct types of objects:
  - Boundary objects
  - Control objects
  - Entity objects
- Rules govern how each of these objects can communicate with the other objects associated with the pattern
- The primary distinction between these two design patterns is the rules that govern object communication

# ENTITY, CONTROL, AND BOUNDARY DESIGN PATTERN

Stereotype	UML Element	Element in analysis class diagram	Icon in the Rational Unified Process
«boundary»	Class	Class with stereotype «boundary»	
«control»	Class	Class with stereotype «control»	
«entity»	Class	Class with stereotype «entity»	

# BOUNDARY CLASS

- Insulates the system from changes in the outside
- Several Types of Boundary Classes
- **User interface classes** – classes that facilitate communication with human users of the system
  - Menus, forms, etc. User interface classes....
- **System interface classes** – classes which facilitate communications with other systems.
  - These boundary classes are responsible for managing the dialogue with the external system, like getting data from an existing database system or flat file...
  - Provides an interface to that system for this system
- **Device Interface Classes** – provide an interface to devices which detect external events – like a sensor



## BOUNDARY CLASS- MORE

- Identify boundary classes for things mentioned in the flow of events of the use-case realization.
- Consider the source for all external events and make sure there is a way for the system to detect these events. (user inputs/responses? Connection to existing external data...)
- **One recommendation:** for the initial identification of boundary classes is one boundary class per actor/use-case pair.
  - This class can be viewed as having responsibility for coordinating the interaction with the actor.
  - This may be refined as a more detailed analysis is performed.
  - This is particularly true for window-based GUI applications, where there is typically one boundary class for each window, or one for each dialog.

# CONTROL CLASSES

- **Control objects are responsible for application specific business logic**
- **In addition, these object types also function as an intermediary between the system's various boundary and entity objects**
- **Within the context of use case realization, each boundary class will communicate with a single control class and control classes will be used to manage each use case's flow of execution**
- **To manage this flow, the control object must coordinate the activities required to support the use case realization, including interactions with other control objects and the data aware entity objects**
- **Each entity object will be tightly coupled with a control object**

# ENTITY CLASSES

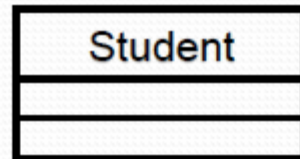
- Entity classes represent stores of information in the system
- They are typically used to represent the key items the system manages.
- Entity objects (instances of entity classes) are used to hold and update information about some phenomenon, such as an event, a person, or some real-life object.
  - (advisor, teacher, university, student etc.)
- They are usually persistent, having attributes and relationships needed for a long period, sometimes for the life of the system.
- The main responsibilities of entity classes are to store and manage information in the system.
- To display the data, the data encapsulated within the entity object will traverse a path that eventually leads to the control object that is tightly coupled to the boundary object
- At this point, the data will be passed to the boundary object for displaying in the GUI

# CANDIDATE ENTITY CLASSES

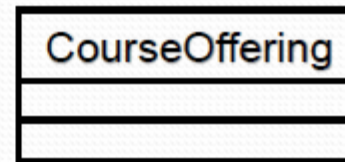
- Sometimes there is a need to model information about an actor within the system. This is not the same as modeling the actor (actors are external by definition).
- For example, a course registration system maintains information about the student which is independent of the fact that the student also plays a role as an actor of the system.
  - This information about the student that is stored in a 'Student' class is completely independent of the 'actor' role the student plays; the Student class (entity) will exist whether or not the student is an actor to the system.

# EXAMPLE: CANDIDATE ENTITY CLASSES

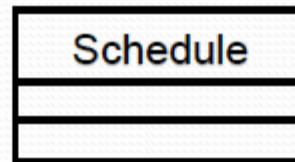
- Register for Courses (Create Schedule)



A person enrolled in classes at the university



A specific offering for a course including days of week and times



The courses a student has selected for current semester

# WHAT IS AN ANALYSIS CLASS?

- **A class that represents initial data and behavior requirements, and whose software and hardware oriented details have not been specified**
- **Analysis class diagram – a UML diagram showing analysis classes and their relationships**

# ANALYSIS CLASSES IN A NUT SHELL



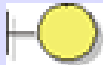









- **Entity class :**
- Persistent data
- used multiple times and in many UCs
- Still exists after the UC terminates (e.g. DB storage)
- **Boundary class:**
- (User) interface between actors and the system
- E.g. a Form, a Window (Pane)
- **Control class:**
- Encapsulates business functionality
- Proposed in RUP (Rational Unified Process)

# ECB PATTERN







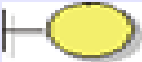








- Collaborating together, the various boundary, control, and entity objects within the BCE design pattern realize the behavior documented in the system's Use Case Model
- The rules that govern communication between the various object types within the BCE design pattern are illustrated in the tables that follow

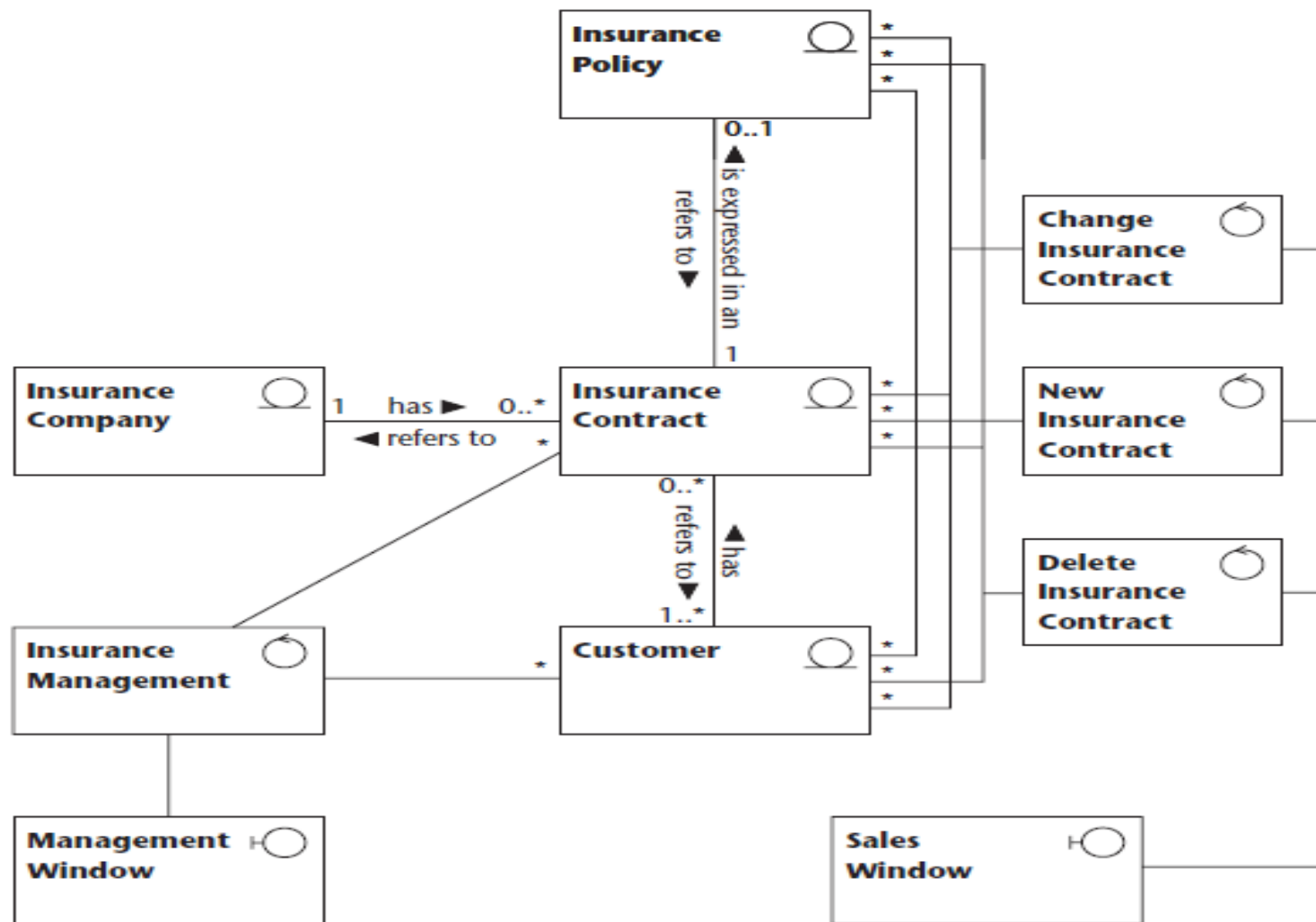


# ALLOWED COMMUNICATION WITHIN THE ECB DESIGN PATTERN

Actor or Object Initiating Communication	Flow of Communication	Target Object
		
		
		
		




# COMMUNICATION NOT ALLOWED WITHIN THE ECB

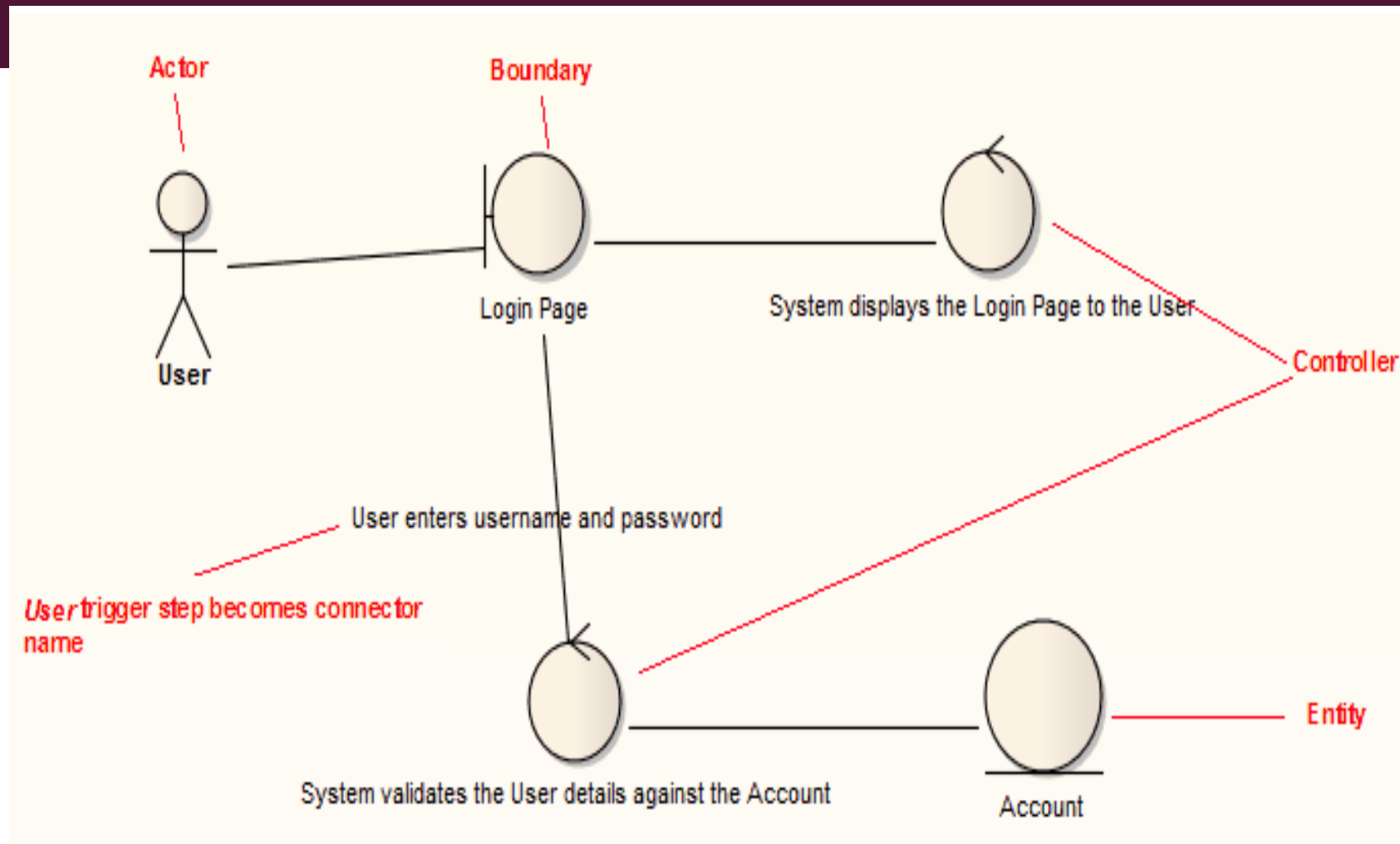
Actor or Object Initiating Communication	Flow of Communication	Target Object
		
		
		
		
		



The entity classes capture the core business, in this case, the insurance business. The entity classes are Insurance Policy, Insurance Contract, Customer, and Insurance Company. The control classes Change Insurance Contract, New Insurance Contract, and Delete Insurance Contract serve the boundary class Sales Window. The control class Insurance Management serves the Management Window with Customers and Insurance Contracts.

## ROBUSTNESS DIAGRAM ACTIVITY

Step	Action
 1	System displays the <u>Login Page</u> to the <u>User</u>
 2	<u>User</u> enters username and password
 3	System validates the <u>User</u> details against the <u>Account</u>



## YOUR TURN:

Use case ID	UC001A
Use case name	Search Products
Actors	Customer
Description	Search for products based on some criteria.
Trigger	The customer wants to browse among products or the customer would like to search for certain products.
Precondition	Customer starts a web browser.
Postcondition	Search results meeting the criteria are displayed.
Normal flow	<ol style="list-style-type: none"> <li>1. Customer visits application home page.</li> <li>2. Customer clicks Search button.</li> <li>3. Search page is displayed by the system.</li> <li>4. Customer enters search criteria.</li> <li>5. The system validates the criteria provided.</li> <li>6. The system looks up the catalog to find the products that meet the criteria.</li> <li>7. Search results page is displayed with the products fulfilling the criteria.</li> </ol>
Alternative flows	<p>Refine Search Results</p> <p>The following steps are added:</p> <ol style="list-style-type: none"> <li>8. Customer refines search results by providing additional criteria.</li> <li>9. Steps 5–7 are re-executed.</li> </ol>
Exceptions	<p>In Step 5, if search criteria is invalid or even missing then Step 3 (display search page) will be executed along with some hints on valid criteria.</p> <p>In Step 6, if no products meet the criteria then Step 3 (display search page) will be executed along with providing the error message "Product not found".</p>



That is all