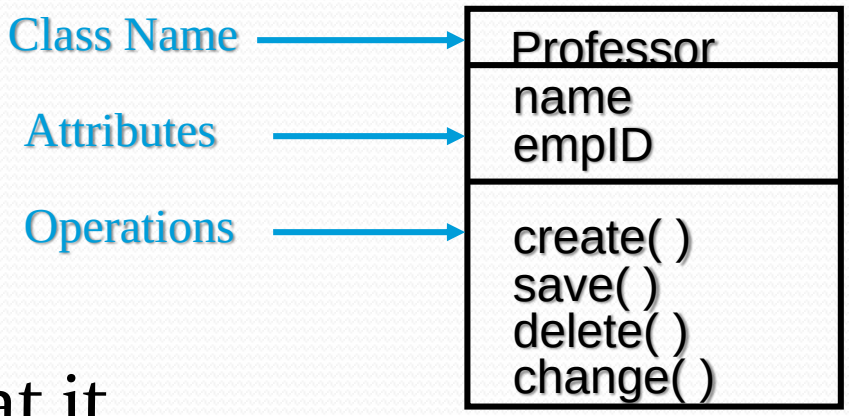
Entity Control & Boundary Classes

Review: Class

- Recall, a class is an abstraction
- Describes a group of objects with common:
  - Properties (attributes)
  - Behavior (operations)
  - Relationships
  - Semantics

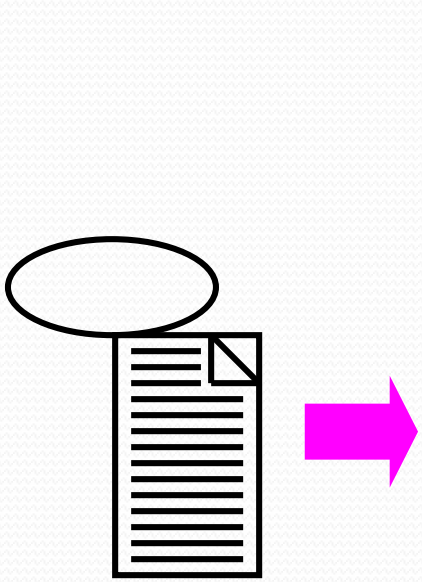


- A class is an abstraction in that it
  - Emphasizes relevant characteristics; suppresses others
  - Consists of the three sections indicated
    - First section: Class name
    - Second section: structure (attributes)
    - Third section: behavior (operations)
- For analysis classes, these entries are sufficient!

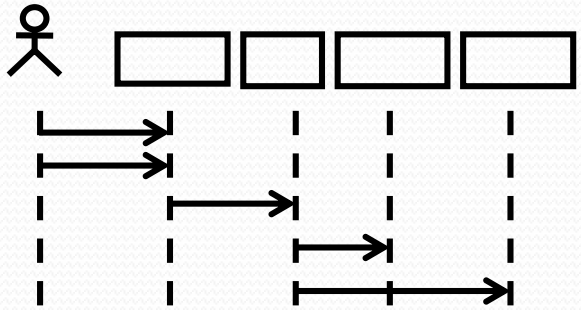
Review: Use-Case Realization

Use-Case Model *Design Model*

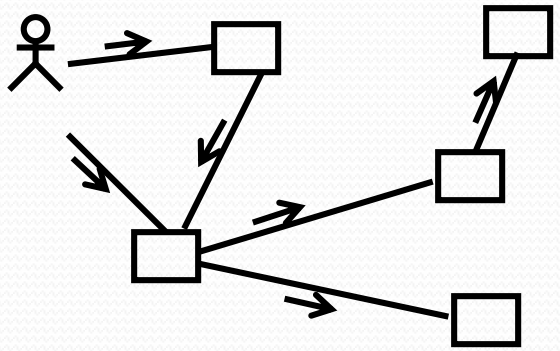
Use Case Use-Case Realization



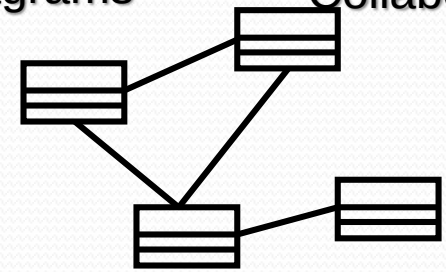
Use Case



Sequence Diagrams



Collaboration Diagrams



Class Diagrams

Not what we have so far.
These are Design Classes. More later

Review: Use-Case Realization

- Remember: a use-case realization describes how a particular use case is realized within the Design Model, in terms of collaborating objects.
  - A use-case realization is **one possible realization** of a use case.
  - A use-case realization in the Design Model can be ***traced*** to a use case in the Use-Case Model.
  - A realization relationship is drawn from the use-case realization to the use case it realizes.
- A use-case realization can be represented using set of diagrams (the number and type may vary)

Review: Use-Case Realization

- 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.

Review: Use-Case Realization

- In our Use Case Analysis step, the use-case realizations' diagrams are outlined.
  - Exactly what this entails will be discussed ahead and form part of the fourth deliverable.
- In subsequent design activities (Use-Case Design) these class diagrams will be **considerably** refined and updated according to more formal class interface definitions.

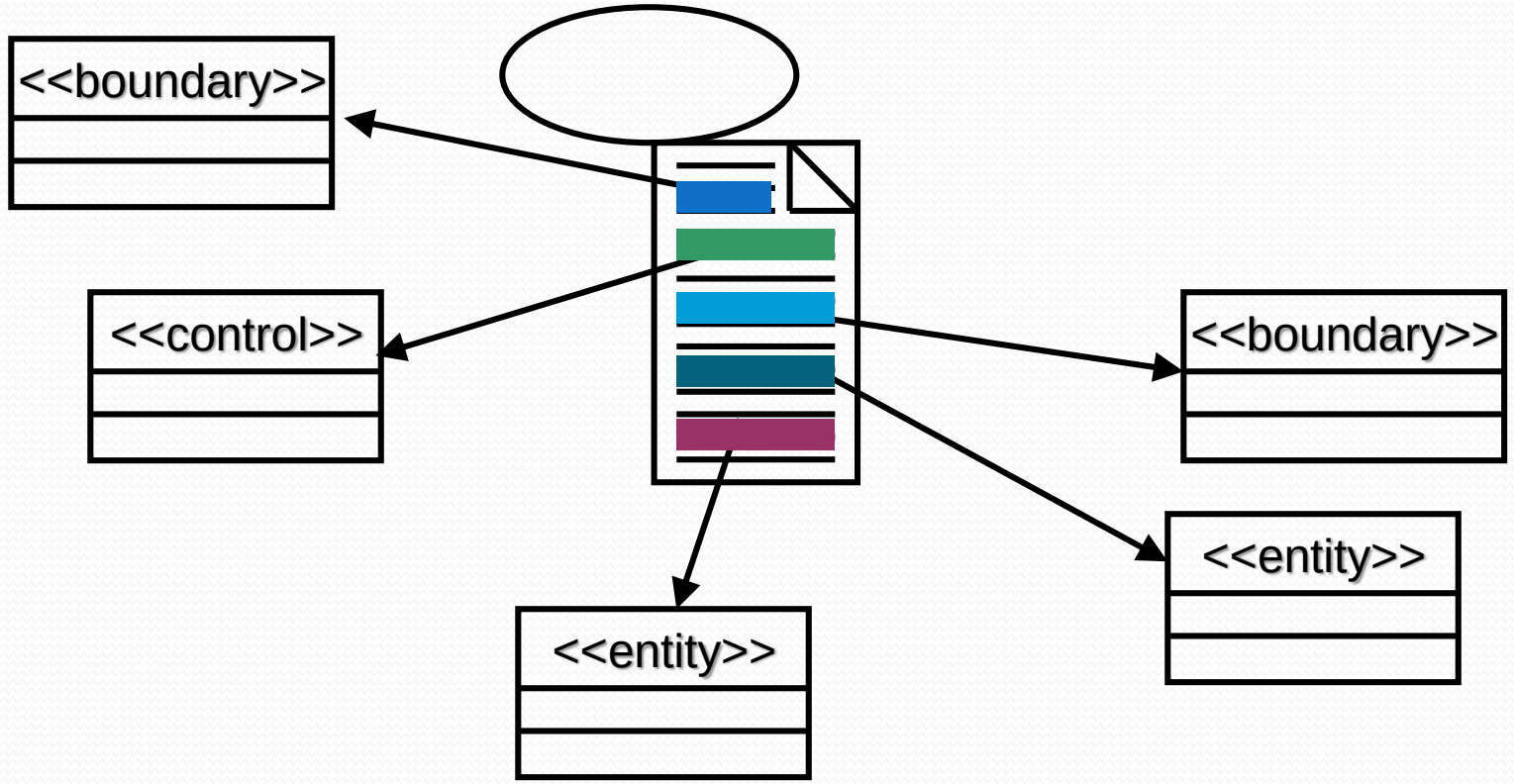
- A designer: responsible for the integrity of the use-case realization.
- Must coordinate with the designers responsible for the classes and relationships employed in the use-case realization.
- The use-case realization can be used by class designers to understand the class's role in the use case and 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…..

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.



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.

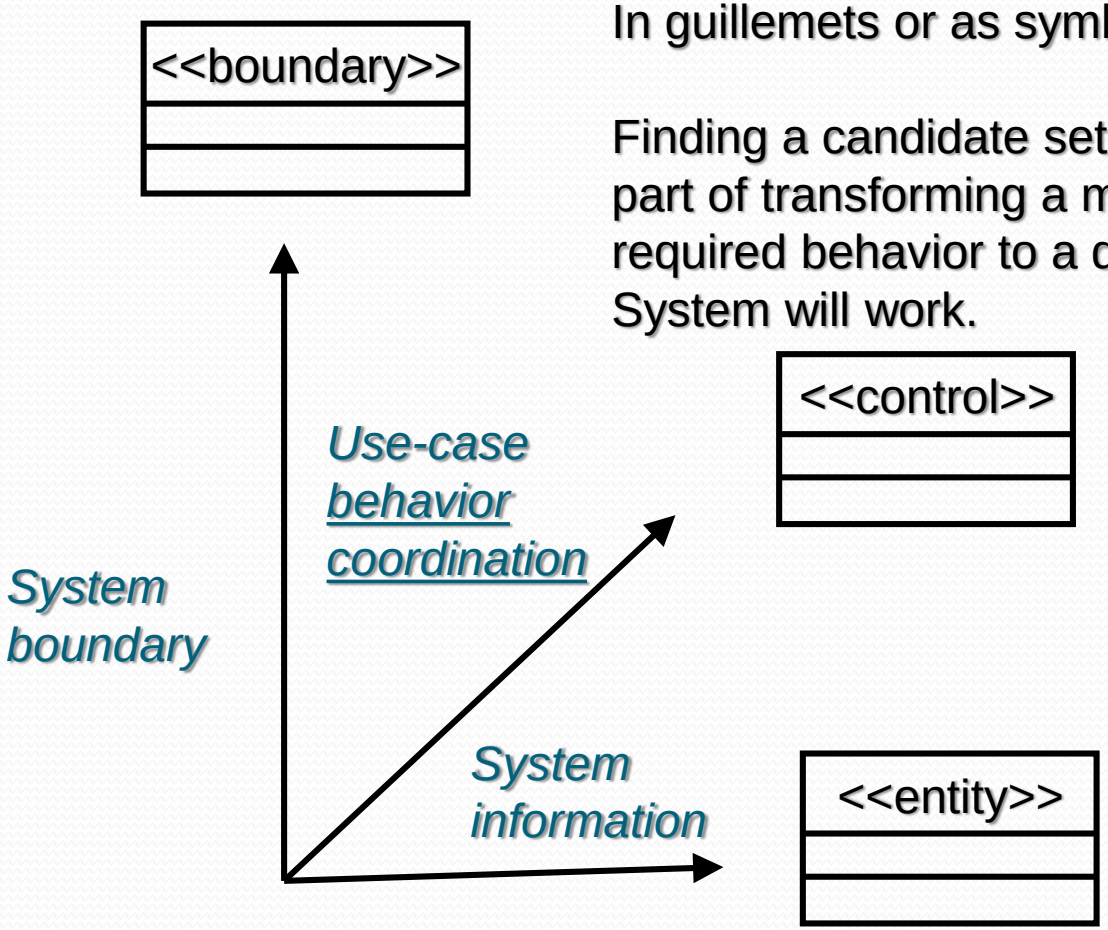
Kinds of Analysis Classes

- Three things likely to change in a system:
  - The boundary between the system and its actors (interfaces…)
  - The information a system uses (data), and
  - The control logic of the system (who does what)
- So, we isolate the different kinds of analysis classes
  - Each of these has a set of canned duties & responsibilities
  - Again, the distinction between these classes is used in analysis but goes away in design or becomes less of an issue, as we transition these analysis classes into design artifacts / design entities to accommodate the problem domain representations in a solutions space.

What is an Analysis Class?

Can use with the name of the stereotype
In guillemets or as symbols with unique icons.

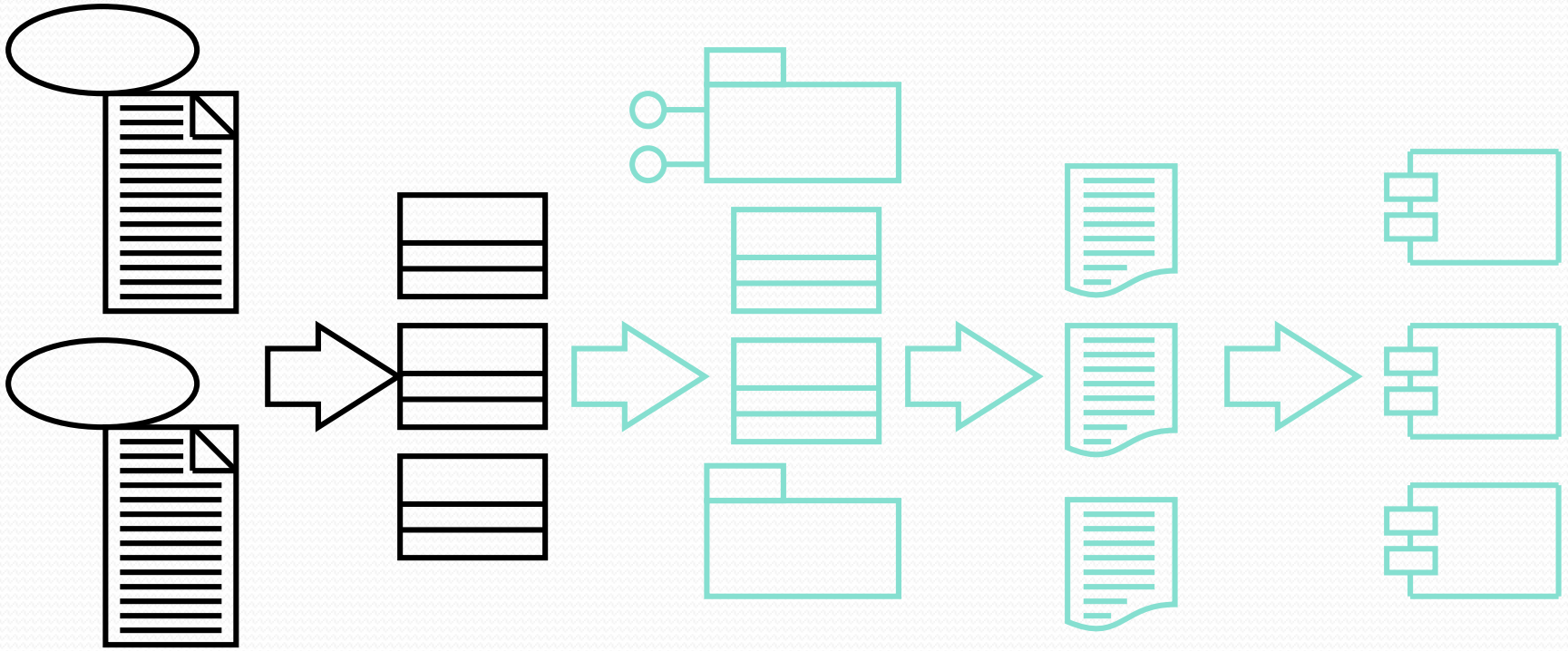
Finding a candidate set of classes is the first part of transforming a mere statement of required behavior to a description of how the System will work.



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.
- Analysis classes allow us to "play" with the distribution of responsibilities, re-allocating, as necessary.

Analysis Classes: A First Step Towards Executables



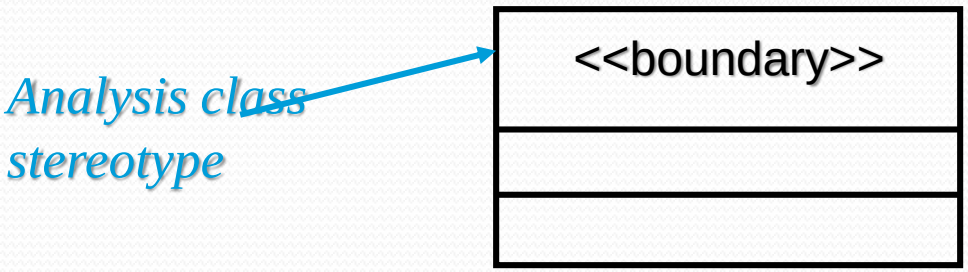
Use Cases **Analysis Classes** **Design Elements** **Source Code** **Exec**

Use-Case Analysis

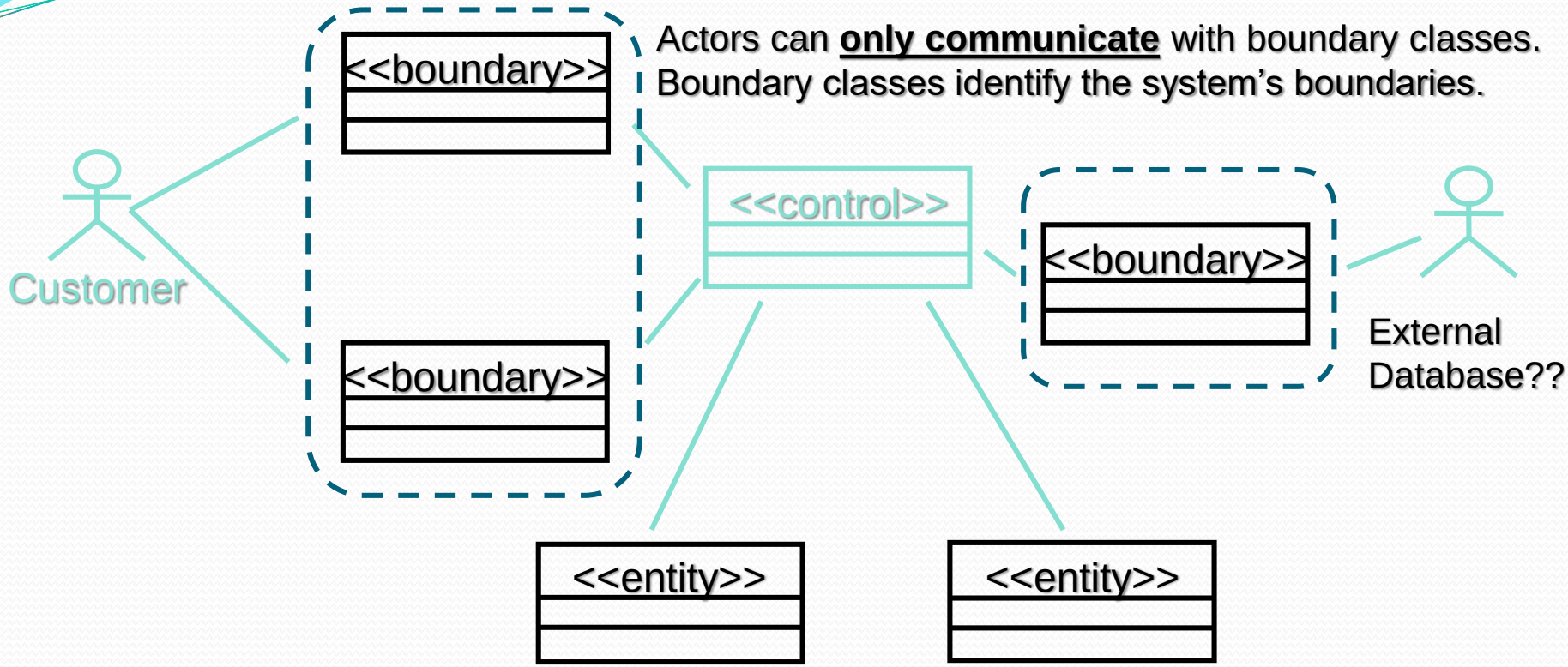
(…from a structural perspective; static)

What is a 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 or …
  - One boundary class per use case/actor pair



The Role of a Boundary Class



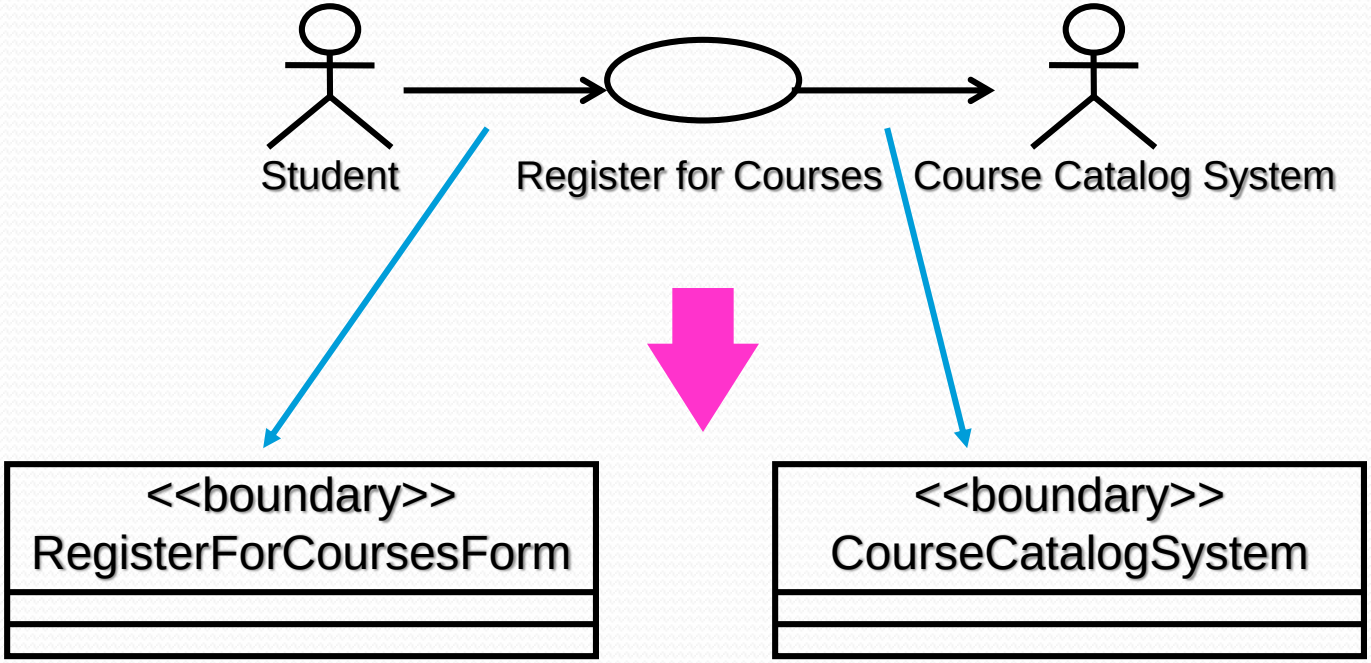
A boundary class is a class used to model interaction between the system's surroundings and its inner workings; Involves transforming and
Boundary Classes model parts of the system that depend on its surroundings. Entity and control classes model parts that are independent of the system's surroundings.
Examples of boundary classes: Classes that handle GUI or communications protocols.

Boundary Classes - 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.

Example: Finding Boundary Classes

- One boundary class per actor/use case pair:



•The RegisterForCoursesForm contains a Student's "schedule-in-progress". It displays a list of Course Offerings for the current semester from which the Student may select to be added to his/her Schedule.

•The CourseCatalogSystem interfaces with the legacy system that provides the complete catalog of all courses offered by the university.

Guidelines: Boundary Class – User Interface classes

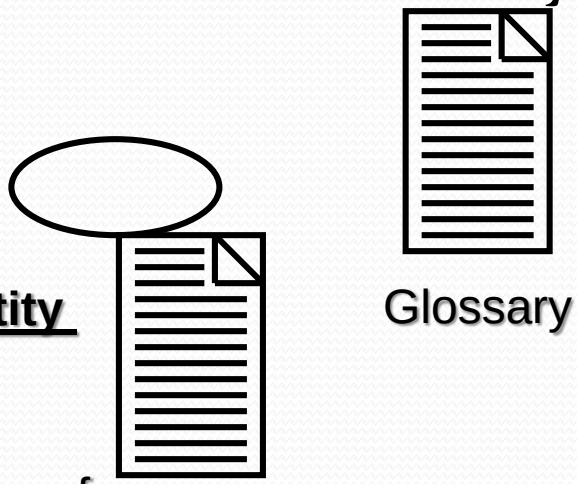
- User Interface Classes
  - Concentrate on what information is presented to the user
  - **Do NOT** concentrate on the UI details
  - Analysis Classes are meant to be a first cut at the abstraction of the system.
  - The boundary classes may be used as 'holding places' for GUI classes.
  - Do not do a GUI design in analysis, but isolate all environment-dependent behavior.
  - The expansion, refinement and replacement of these boundary classes with actual user interface classes is a very important activity of Class Design.
  - If prototyping the interface has been done, these screen dumps or sketches may be associated with a boundary class.
  - Only model the key abstractions of the system – not every button, list, etc. in the GUI.

Guidelines: Boundary Classes – System and Device

- System and Device Interface Classes
  - Concentrate on what protocols must be defined
    - Note that your application must interface with an existing 'information source.'
  - Do NOT concentrate on how the protocols will be implemented
- If the interface to an existing system or device is already well-defined, the boundary class responsibilities should be derived directly from the interface definition.
- If there is a working communication with the external system or device, make note of it for later reference during design.

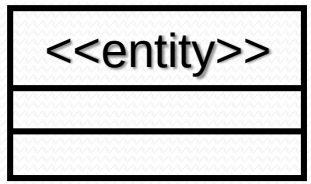
What is an Entity Class? (recall: boundary, entity, control…)

- Key abstractions of the system



Glossary

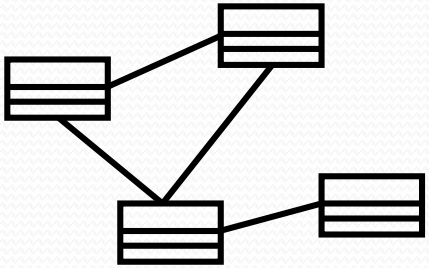
Analysis class stereotype



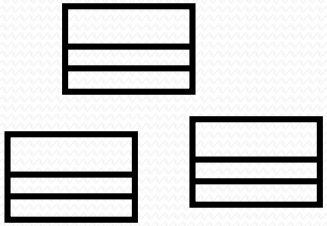
Sources for entity Classes:

Glossary
Use-Case Flow of Events
Business domain model

Use Case



Business-Domain Model



Architectural Analysis Abstractions

Entity classes show the logical data structure, which will help us understand what the system is supposed to offer to its users.

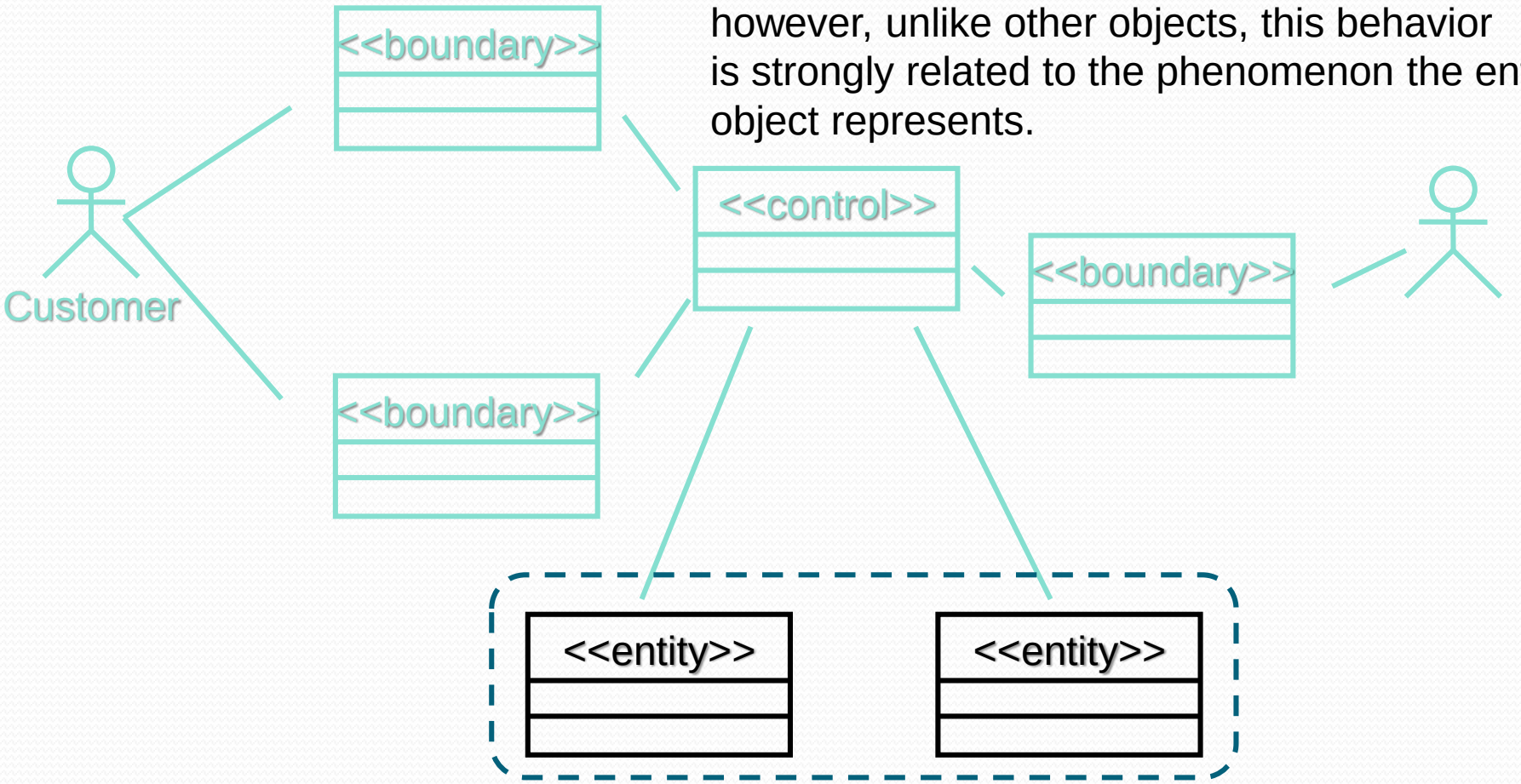
Environment Independent

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.**

The Role of an Entity Class

Entity objects can have complicated behavior; however, unlike other objects, this behavior is strongly related to the phenomenon the entity object represents.

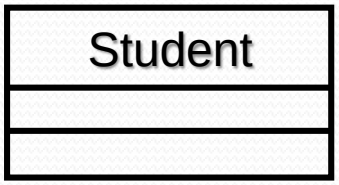


Store and manage information in the system

The values of its attributes and relationships are often given by an actor.
Entity objects are **independent** of the environment (actors)

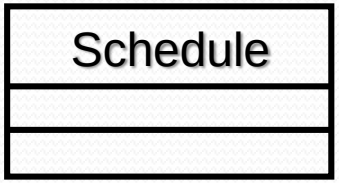
Example: Candidate Entity Classes

- 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

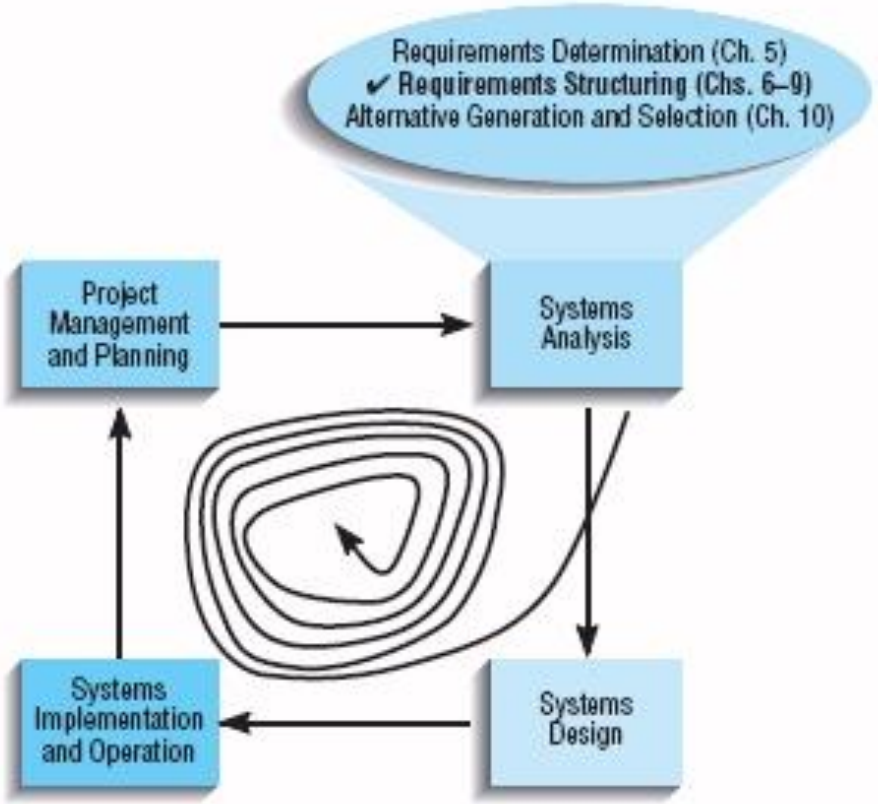
Candidate Entity Classes

- 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). These classes are sometimes called "**surrogates**".
- 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.

Analysis Classes

- • 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)

Figure 9.1 Systems Analysis Has Three Parts: Requirements Determination, Requirements Structuring, and Alternative Generation and Selection



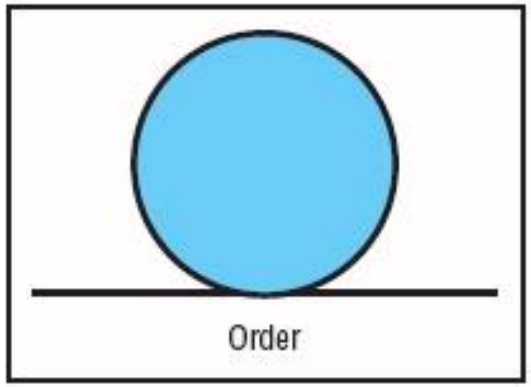
What Is an Analysis Class?

- 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

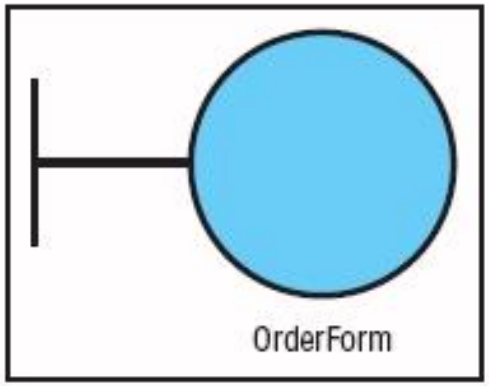
Stereotypes of Analysis Classes

Figure 9.3a
Entity Class Order



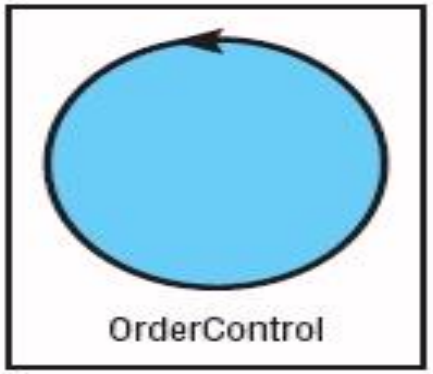
Mostly corresponds to conceptual data model classes

Figure 9.3b
Boundary Class
OrderForm



Encapsulates connections between actors and use cases

Figure 9.3c
Control Class
OrderControl



Mostly performs behaviors associated with inner workings of use cases