Object Oriented Analysis And Design
Aleenah Khan

Chapter 4
Finding Classes

Object

- An object is a representation of an entity, either real-world or conceptual.
- An object can represent something concrete, such as Joe's truck or my computer, or a concept such as a chemical process, a bank transaction, a purchase order, Mary's credit history, or an interest rate.
- An object is a concept, abstraction, or thing with well-defined boundaries and meaning for an application.

Object

- Each object in a system has three characteristics:
  - state
  - behavior
  - identity

State

- The state of an object is one of the possible conditions in which it may exist.
- The state of an object typically changes over time, and is defined by a set of properties (called attributes), with the values of the properties, plus the relationships the object may have with other objects.
- For example:
  - A **course offering** object in the registration system may be in one of two states: *open* and *closed*.
  - If the number of students registered for a course offering is less than 10, the state of the course offering is open.
  - When the tenth student registers for the course offering, the state becomes closed.

Behavior

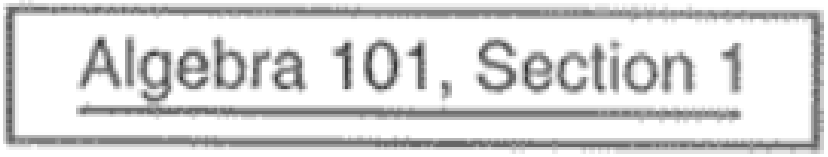
- Behavior determines how an object responds to requests from other objects and typifies everything the object can do.
- Behavior is implemented by the set of operations for the object.
- For Example:
  - In the registration system, a course offering could have the behaviors *add a student* and *delete a student*.

Identity

- Identity means that each object is unique—even if its state is identical to that of another object.
- For Example:
  - Algebra 101, Section 1, and Algebra 101, Section 2 are two objects in the Course Registration System. Although they are both course offerings, they each have a unique identity.

Object - Representation

- In the UML, objects are represented as **rectangles** and the name of the object is **underlined** as shown in Figure 4-1.



Class

- A class is a description of a group of objects with common properties (attributes), common behavior (operations), common relationships to other objects, and common semantics.
- Thus, a class is a template to create objects.
- Each object is an instance of some class and objects cannot be instances of more than one class.

Class

- For example:
  - The Course Offering class may be defined with the following characteristics:
    - Attributes—location, time offered.
    - Operations—retrieve location, retrieve time of day, add a student to the offering.
- Algebra 101, Section 1, and Algebra 101, Section 2 are objects belonging to the Course Offering class.

Class

- A good class captures one and only one abstraction—it should have one major theme.
- For Example:
  - A class that has the capability of maintaining information about a student and the information about all the course offerings that the student has taken over the years is not a good class since it does not have one major theme.
  - This class should be split into two related classes:
    - Student
    - StudentHistory.

Class - Naming Convention

- Classes should be named using the vocabulary of the domain.
- The name should be a singular noun that best characterizes the abstraction.
- If a class is named with an acronym, the full name should also be contained in the class documentation.

Object vs Class

Q: Why is Algebra 101, Section 1 an object and not a class?

Q: What makes it different from Algebra 101, Section 2?

- The answers to these questions are very subjective.
- By looking at their structure and behavior, it can be seen that both have the same structure and behavior.
- They are only different course offerings for a semester.
- In addition, it may be noted that there are many other "things" in the Course Registration System that have the same structure and behavior (e.g., Music 101, Section1; History 101, Section 1; and History 101, Section 2).
- This leads to the decision to create a CourseOffering class.

Class - Representation

- In the UML, classes are represented as compartmentalized rectangles.
- The top compartment contains the name of the class, the middle compartment contains the structure of the class (attributes), and the bottom compartment contains the behavior of the class (operations).
- A class is shown in Figure 4-2.

Figure 4-2. UML Notation for a Class

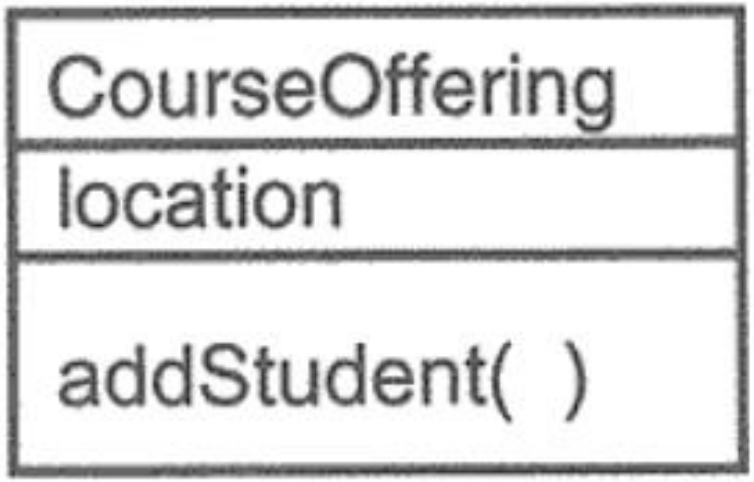
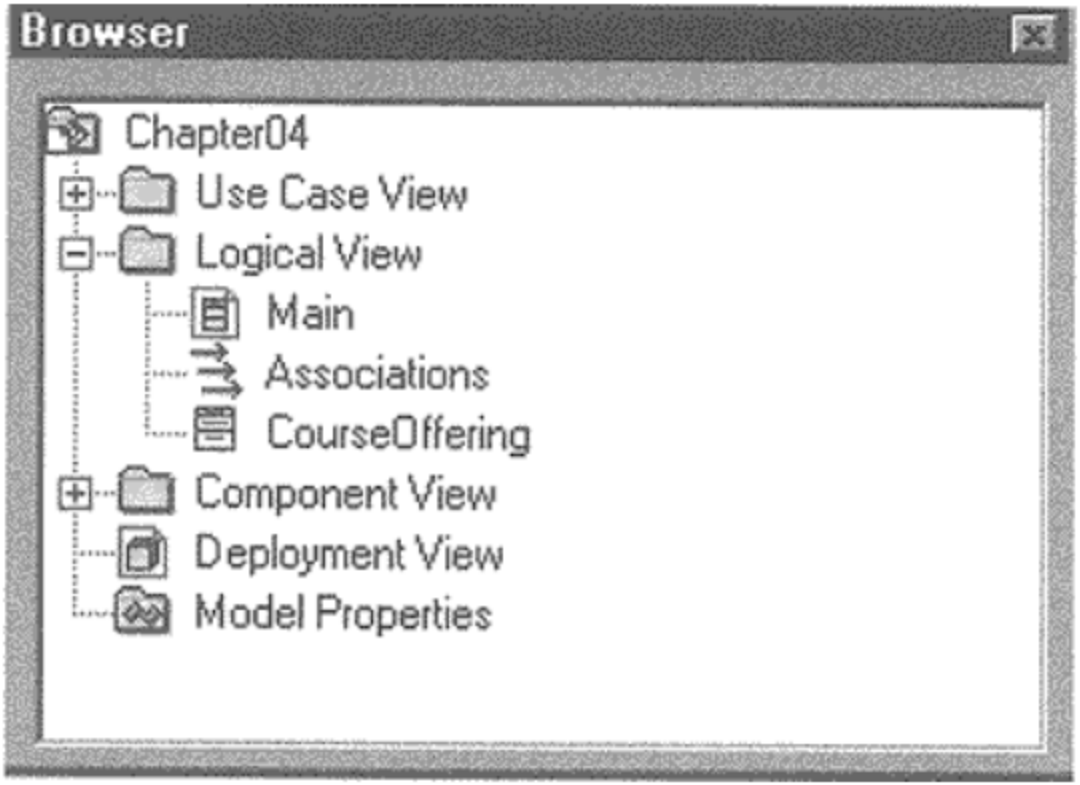


Figure 4-3. Class Created in the Browser



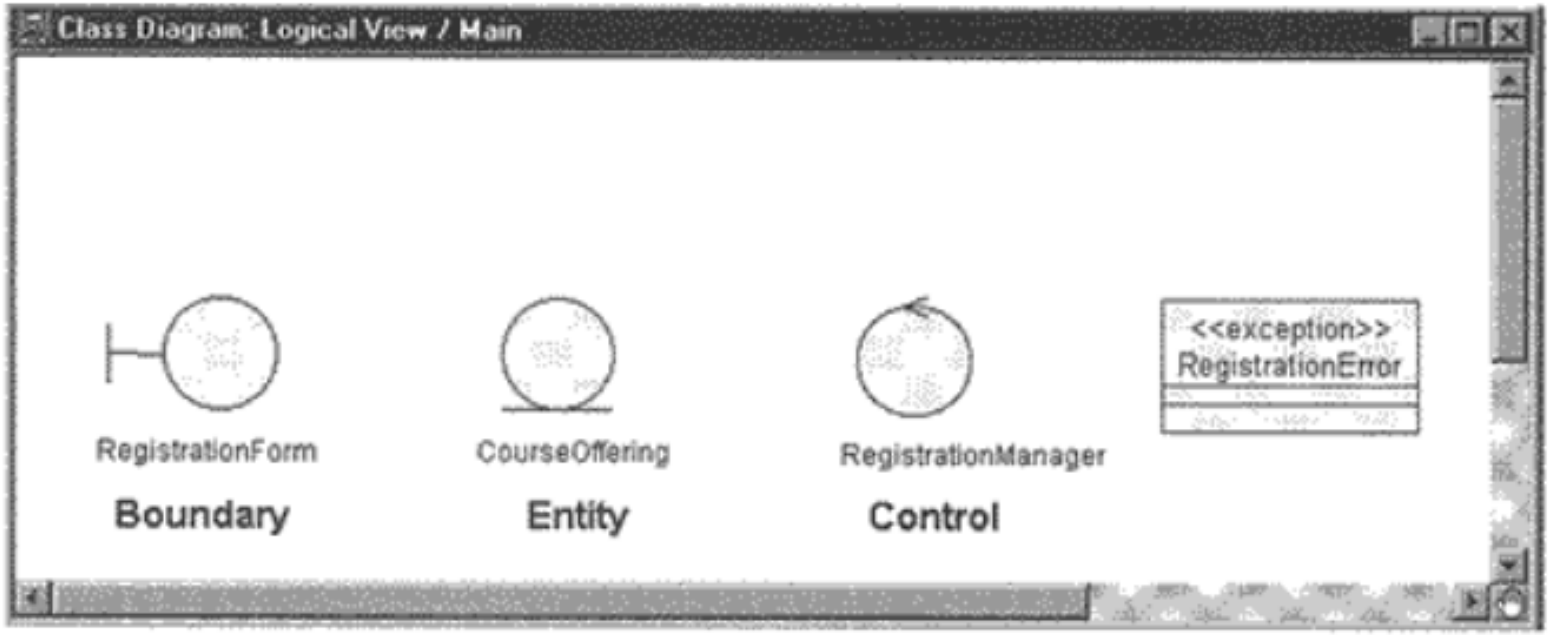
Stereotypes and Classes

- Classes can also have stereotypes like relationships.
- A stereotype provides the capability to create a new kind of modeling element (create new kinds of classes).
- Some common stereotypes for a class are **entity**, **boundary**, **control**, **utility**, and **exception**.

Stereotypes and Classes

- The stereotype for a class is shown below the class name enclosed in guillemets (<< >>).
- If desired, a graphic icon or a specific color may be associated with a stereotype.
- In Rational Rose 2000, icons for the stereotypes of control, entity, and boundary are supplied.
- These stereotypes are shown in Figure 4-4 along with an example of a class with a stereotype of exception.

Figure 4-4. Classes with Stereotypes



Discovering Classes

- The Rational Unified Process advocates finding the classes for a system under development by looking for **boundary**, **control**, and **entity** classes.
- These three stereotypes conform to a **"model-view-controller"** point of view and allow the analyst to partition the system by separating the view from the domain from the control needed by the system.

Discovering Classes

- Since the analysis and design process is iterative, the list of classes will change as time moves on.
- The initial set of classes probably will not be the set of classes that eventually gets implemented.
- Thus, the term **candidate class** is often used to describe the first set of classes found for a system.

Objects and Classes in the ESU Course Registration Problem

- Consider the ***Add a Course Offering to Teach*** scenario, which is one of the subflows of the ***Select Courses to Teach*** use case.
- The main capability provided by this scenario is the ability for the professor to select a course offering to teach for a given semester.

Entity Classes

- An entity class models information and associated behavior that is generally long lived.
- This type of class may reflect a real-world entity or it may be needed to perform tasks internal to the system.
- They are typically independent of their surroundings; that is, they are not sensitive to how the surroundings communicate with the system.
- Many times, they are application independent, meaning that they may be used in more than one application.

Entity Classes

- The first step is to examine the responsibilities documented in the flow of events for the identified use cases (i.e., what the system must do).
- Entity classes typically are classes that are needed by the system to accomplish some responsibility.
- The **nouns** and **noun phrases** used to describe the responsibility may be a good starting point.
- The initial list of nouns must be filtered because it could contain:
  - nouns that are outside the problem domain,
  - nouns that are just language expressions,
  - nouns that are redundant, and
  - nouns that are descriptions of class structures.

Entity Classes

- Entity classes typically are found early in the Elaboration Phase.
- They are often called "domain" classes since they usually deal with abstractions of real-world entities.

Identify Entity Classes

- This scenario deals with Courses, their Course Offerings, and the Professor assignment.
- We can identify three entity classes:
  - Course,
  - CourseOffering, and
  - Professor.

Boundary Classes

- Boundary classes handle the communication between the system surroundings and the inside of the system.
- They can provide the interface to a user or another system (i.e., the interface to an actor).
- They constitute the surroundings-dependent part of the system.
- Boundary classes are used to model the system interfaces.
- Each physical actor/scenario pair is examined to discover boundary classes.

Boundary Classes

- The boundary classes chosen in the Elaboration Phase of development are typically at a high level.
- For example, you may model a window but not model each of its dialogue boxes and buttons. At this point, you are documenting the user interface requirements, not implementing the interface.
- Boundary classes are also added to facilitate communication with other systems.

Identify Boundary Classes

- This use case interacts only with the Professor actor.
- The action specified in this scenario is only one capability provided by the use case (the use case also states that the Professor can modify a selection, delete a selection, review a selection, and print a selection).
- This means that something in the system must provide the ability for the Professor to select a capability.
- A class containing all the options available to the Professor as stated in the use case is created to satisfy this need. This class is called **ProfessorCourseOptions**.
- Additionally, we can identify a class that deals with the addition of a new Course Offering for the Professor. This class is called **AddACourseOffering.**

Control Classes

- Control classes model sequencing behavior specific to one or more use cases.
- Control classes coordinate the events needed to realize the behavior specified in the use case.
- You can think of a control class as "running" or "executing" the use case—they represent the dynamics of the use case.
- Control classes typically are application-dependent classes.

Control Classes

- In the early stages of the Elaboration Phase, a control class is added for each actor/use case pair.
- As analysis and design continues, control classes may be eliminated, split up, or combined.
- The control class is responsible for the flow of events in the use case.

Control Classes

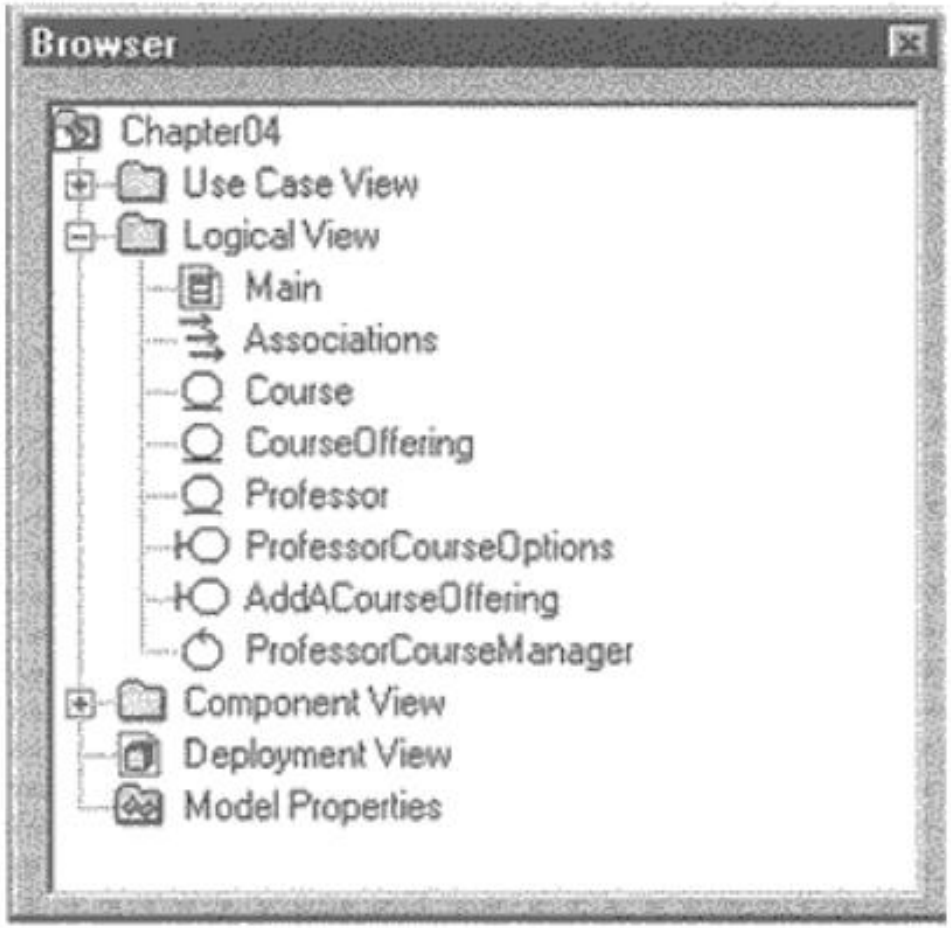
- If a control class is doing more than sequencing, then it is doing too much!
- For Example:
  - In the Course Registration System, a student selects course offerings and if the course offering is available, the student is added to it.
  - **Question: Who knows how to add the student—the control class or the course offering?**
  - **Answer: Course Offering**
  - The control class knows when the student should be added; the course offering knows how to add the student.
  - A bad control class would not only know when to add the student but how to add the student.

Identify Control Classes

- We will add one control class to handle the flow of events for the use case. This class is called **ProfessorCourseManager.**

The identified classes (with stereotypes set to entity, control, or boundary) have been added to the model as shown in Figure 4-10.

Figure 4-10. Classes for the Add a Course Offering to Teach Scenario



Documenting Classes

- As classes are created, they should also be documented.
- The documentation should state the purpose of the class and not the structure of the class.
- For example: A Student class could be documented as follows:

  Information needed to register and bill students. A student is someone currently registered to take classes at the University.

- A bad definition would be the following:

  The name, address, and phone number of a student.

Documenting Classes

The following list typifies things that can happen as classes are named and documented:

- Can identify a name and a clear concise definition—**good candidate class**.
- Can identify a name, but the definition is the same as another class—**combine the classes**.
- Can identify a name, but need a book to document the purpose—**break up the class**.
- Cannot identify a name or a definition—**more analysis is needed to determine the correct abstractions**.

Packages

- Most systems are composed of many classes, and thus you need a mechanism to group them together for ease of use, maintainability, and reusability.
- A package in the logical view of the model is a collection of related packages and/or classes.
- By grouping classes into packages, we can look at the "higher" level view of the model (i.e., the packages) or we can dig deeper into the model by looking at what is contained by the package.

Packages

- Each package contains an interface that is realized by its set of **public classes**—those classes to which classes in other packages talk.
- The rest of the classes in a package are **implementation classes**—classes do not communicate with classes in other packages.

Packages - Representation

- In the UML, packages are represented as folders. A package is shown in Figure 4-7.

Figure 4-7. UML Notation for a Package

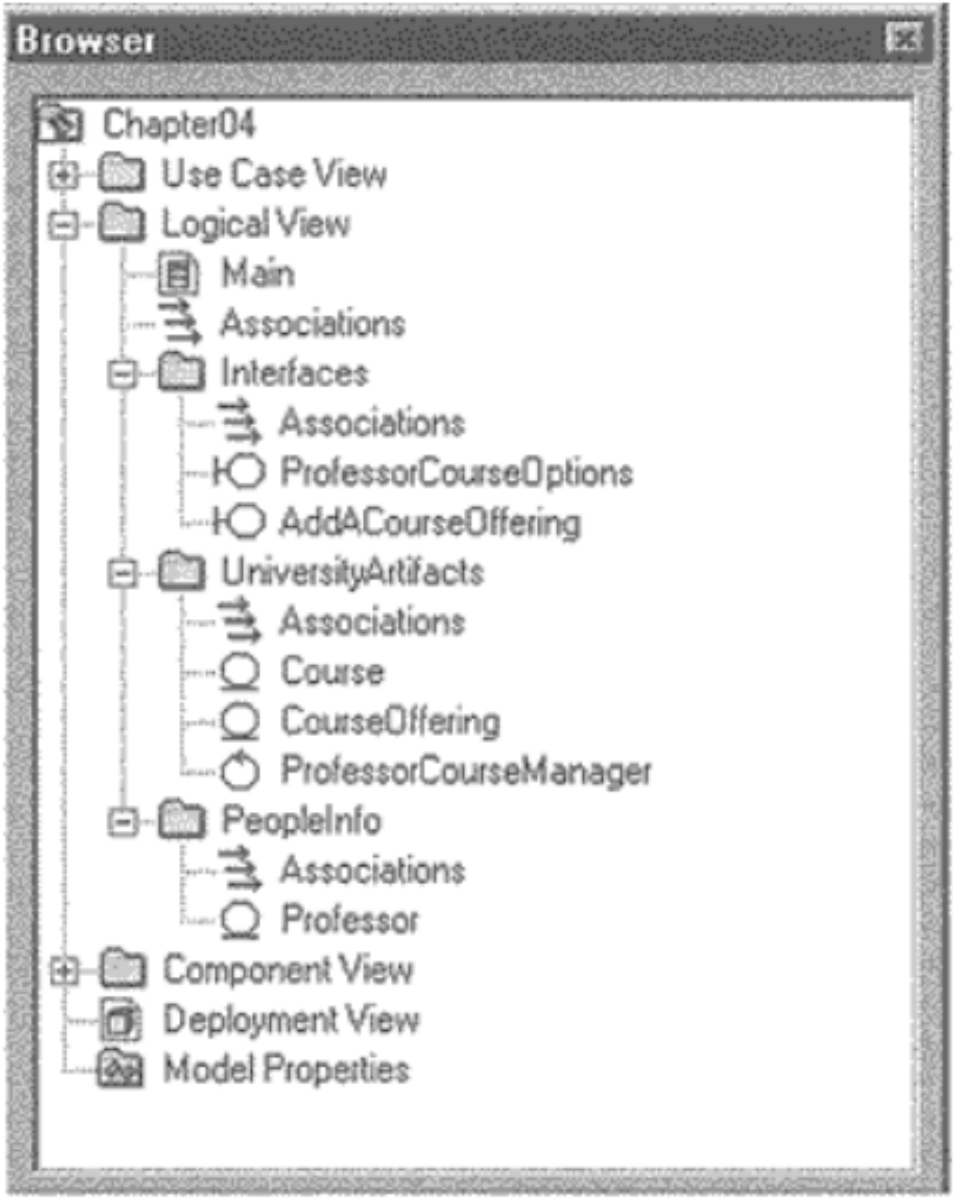


Figure 4-11. Packages in the Browser

Class Diagrams

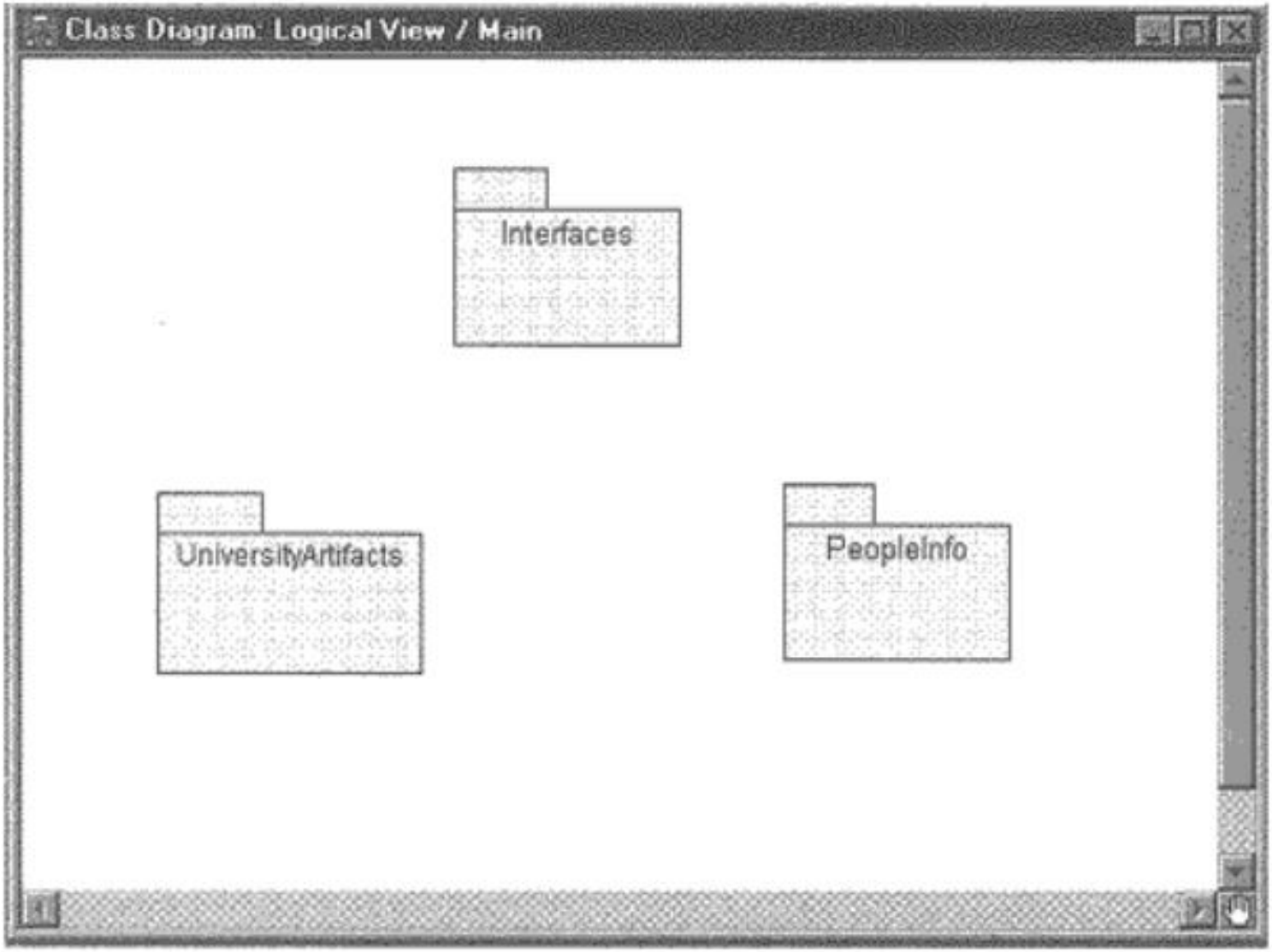
- As more and more classes are added to the model, a textual representation of the classes is not sufficient.
- Class diagrams are created to provide a picture or view of some or all of the classes in the model.

Class Diagrams

- The main class diagram in the logical view of the model is typically a picture of the packages in the system.
- Each package also has its own main class diagram, which typically displays the "public" classes of the package.
- Some typical uses of other diagrams are the following:
  - View of all the implementation classes in a package.
  - View of the structure and behavior of one or more classes.
  - View of an inheritance hierarchy.

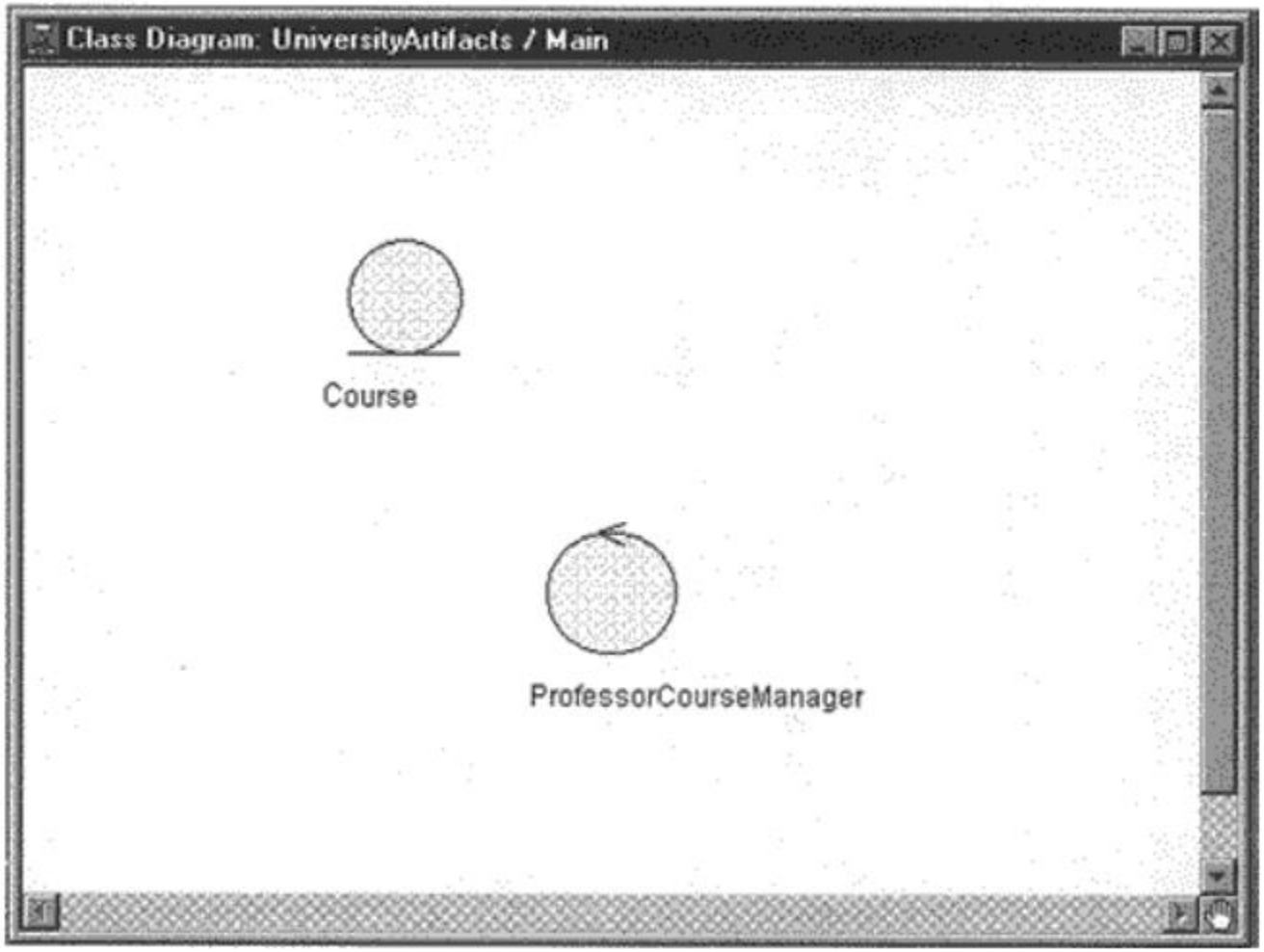
The Main class diagram for the Registration System is shown in Figure 4-12.

Figure 4-12. Main Class Diagram



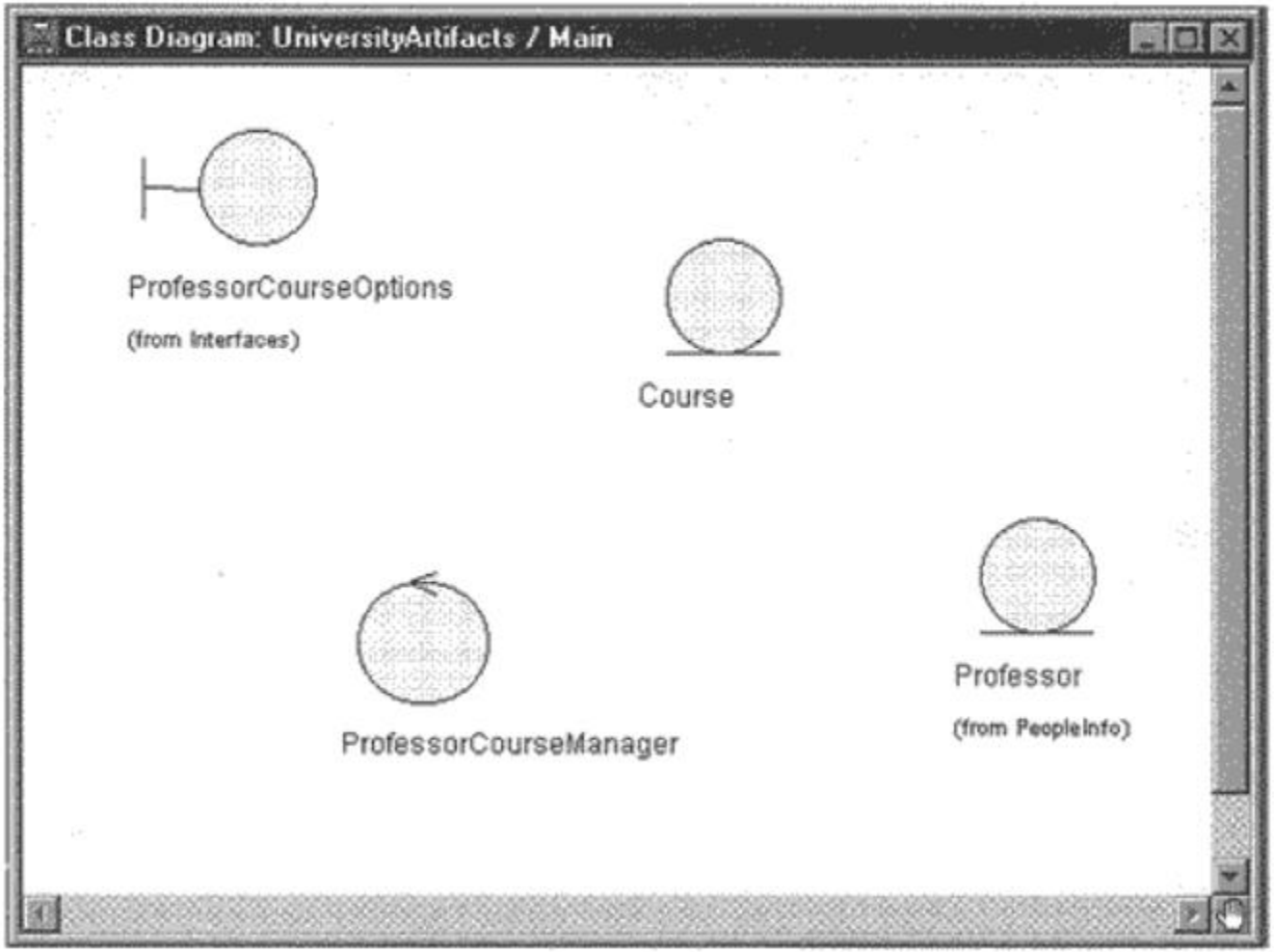
The main class diagram for the UniversityArtifacts package is shown in Figure 4-13. Notice that the CourseOffering class is not on the diagram. This is an "implementation" class in the package and we decided not to show it on the main diagram of the package. As more packages and classes are added to the model, additional diagrams are created as needed.

Figure 4-13. UniversityArtifacts Package Main Class Diagram



A class diagram showing package visibility is shown in Figure 4-14.

Figure 4-14. Class Diagram with Package Visibility



Useful Links

- UML Class Diagram Tutorial https://www.youtube.com/watch?v=UI6lqHOVHic
- StarUML https://www.youtube.com/watch?v=9ECRzQqUxdQ
- More links:

  https://www.youtube.com/watch?v=3cmzqZzwNDM

  https://www.youtube.com/watch?v=O3o9oOWBwb0