LOYOLA UNIVERSITY CHICAGO

Systems Analysis and Design with UML Class Diagrams

Dr. William L. Honig
Associate Professor
Department of Computer Science

Extended and Adapted from Robert V. Stumpf, Lavette C. Teague, Object-Oriented Systems Analysis and Design With UML, Pearson/Prentice Hall 2005

Overview

- UML Domain Models (Concepts)
  - Aka Objects and Methods or Messages
  - Abstract or Logical Components
  - Aot Instances
- UML Class Diagrams
  - Aka Class names, attributes, methods
  - With types, parameters,…
- Set the stage for programming
  - System level operation is clear
  - Internal, physical, implementation remains

Overview

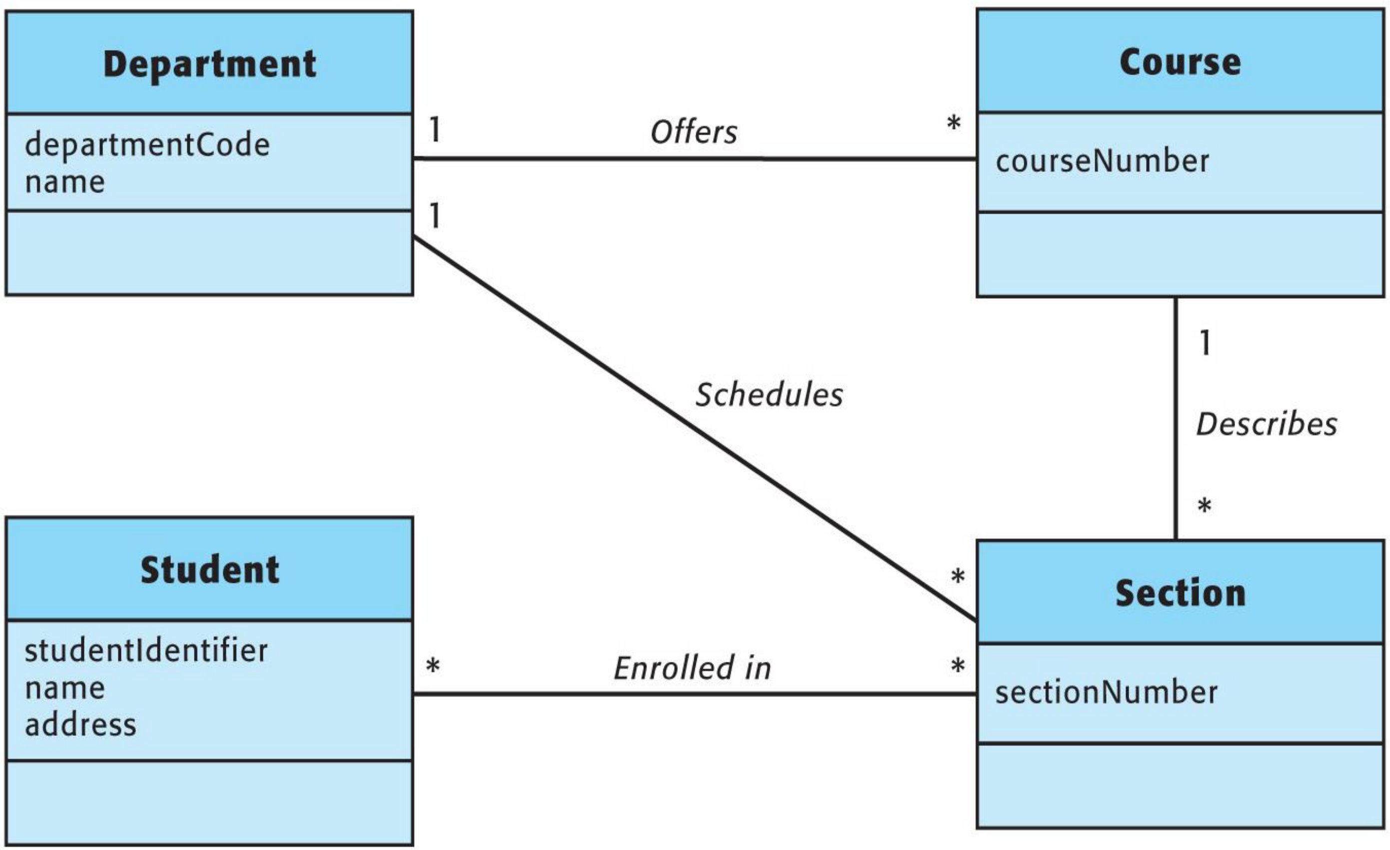
- There is one domain model for the system – a static model showing the conceptual scope of the entire system. Its components are concepts, their attributes, and associations between concepts. It also shows hierarchies of concepts.
- It is helpful to construct the domain model one use case at a time in order to understand which concepts, attributes, and associations are relevant to each use case.

Object-Oriented
Systems Analysis

Produce a domain model showing the concepts, attributes, and associations in the problem domain of the system.

Domain Model

FIGURE 5.1



Concepts, Attributes, and Associations

- A concept is an abstraction of a thing, a person, or an idea. It is represented by a rectangle.
- An attribute is a characteristic of a concept which may have a value. Attribute names appear in the lower compartment of the concept rectangle.
- An association is a significant connection between concepts. It is represented by a line connecting a pair of concepts.

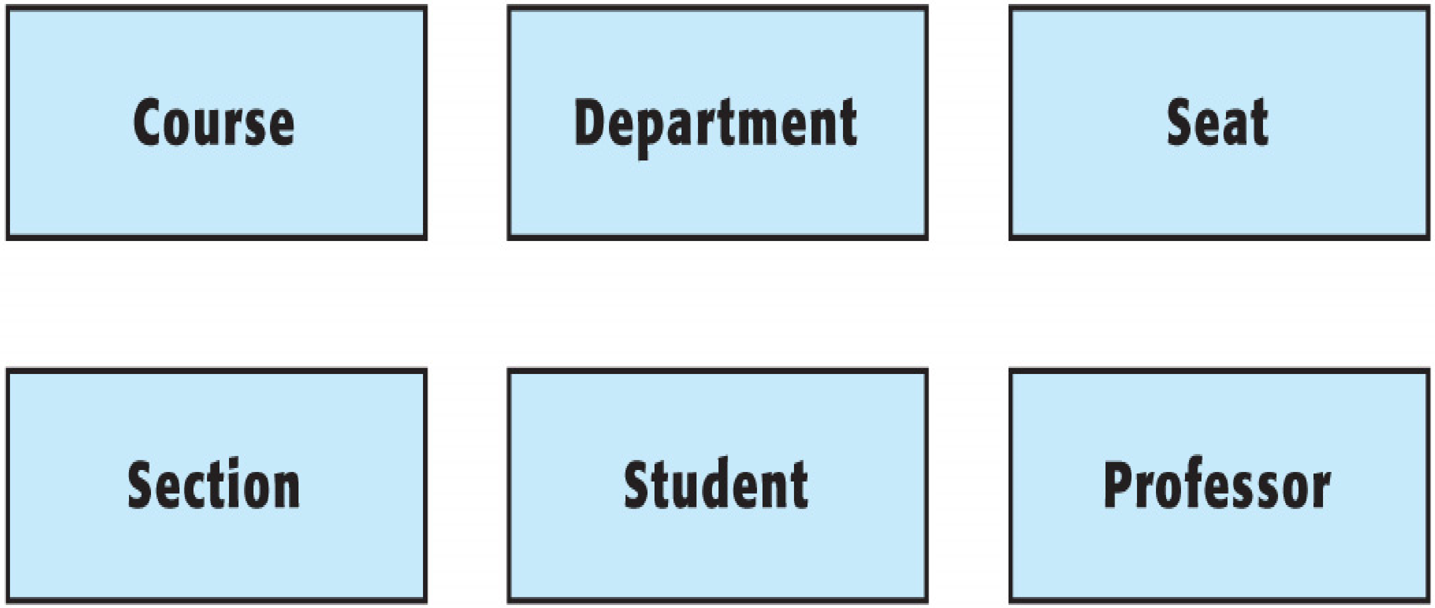
Finding Concepts

1. Look for nouns or noun phrases describing the problem domain.

Include a concept in the domain model when the system needs to store data about the concept to respond to a future event.

Concepts

FIGURE 5.4



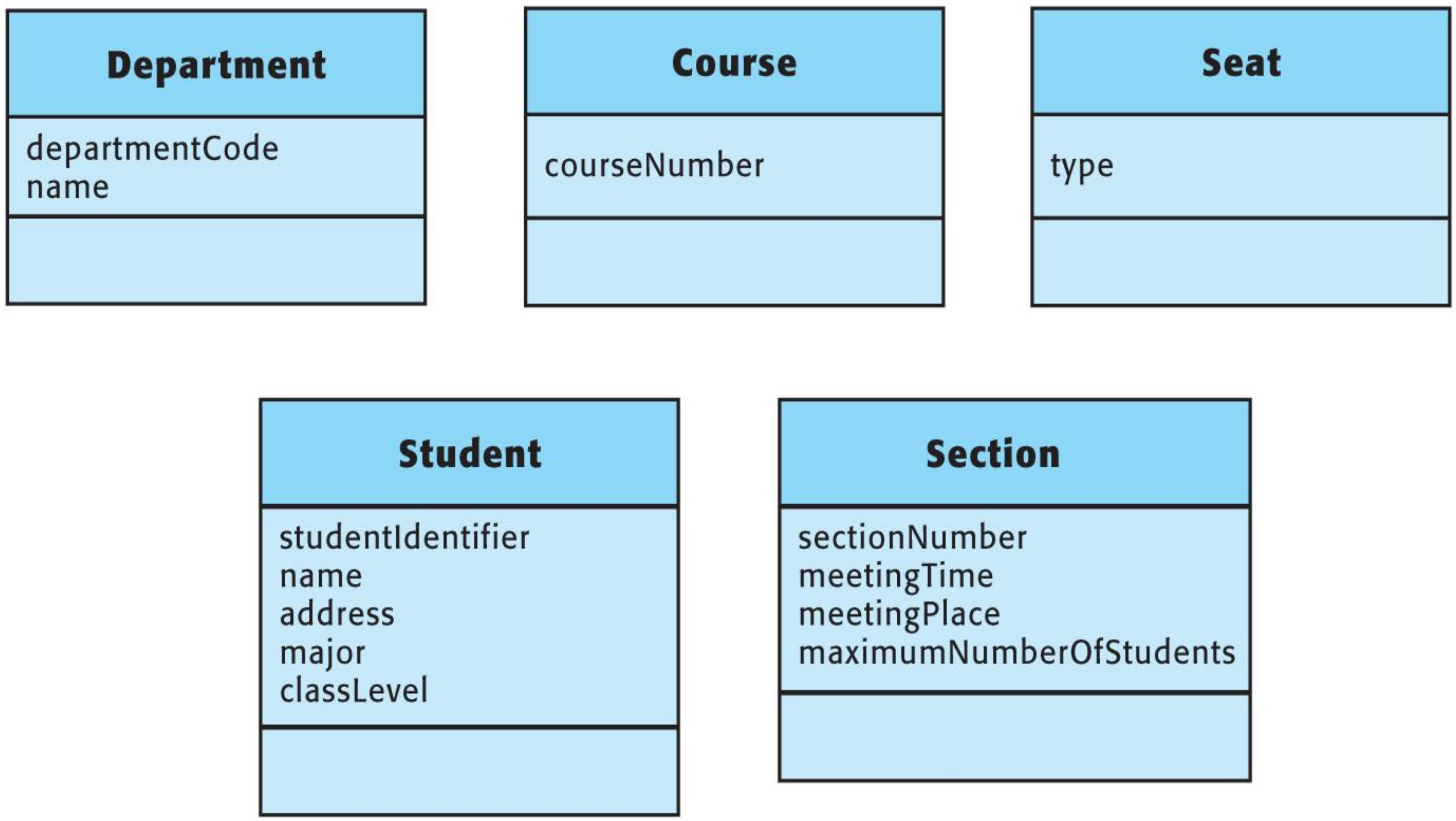
Add Attributes

Attributes describe concepts.

Concept	Student	Professor	Section
Attributes:	studentIdentifier	professorIdentifier	number
	studentName	professorName	meetingTime
	studentAddress	professorAddress	meetingPlace
	major classLevel	title	maximum NumberOf Students

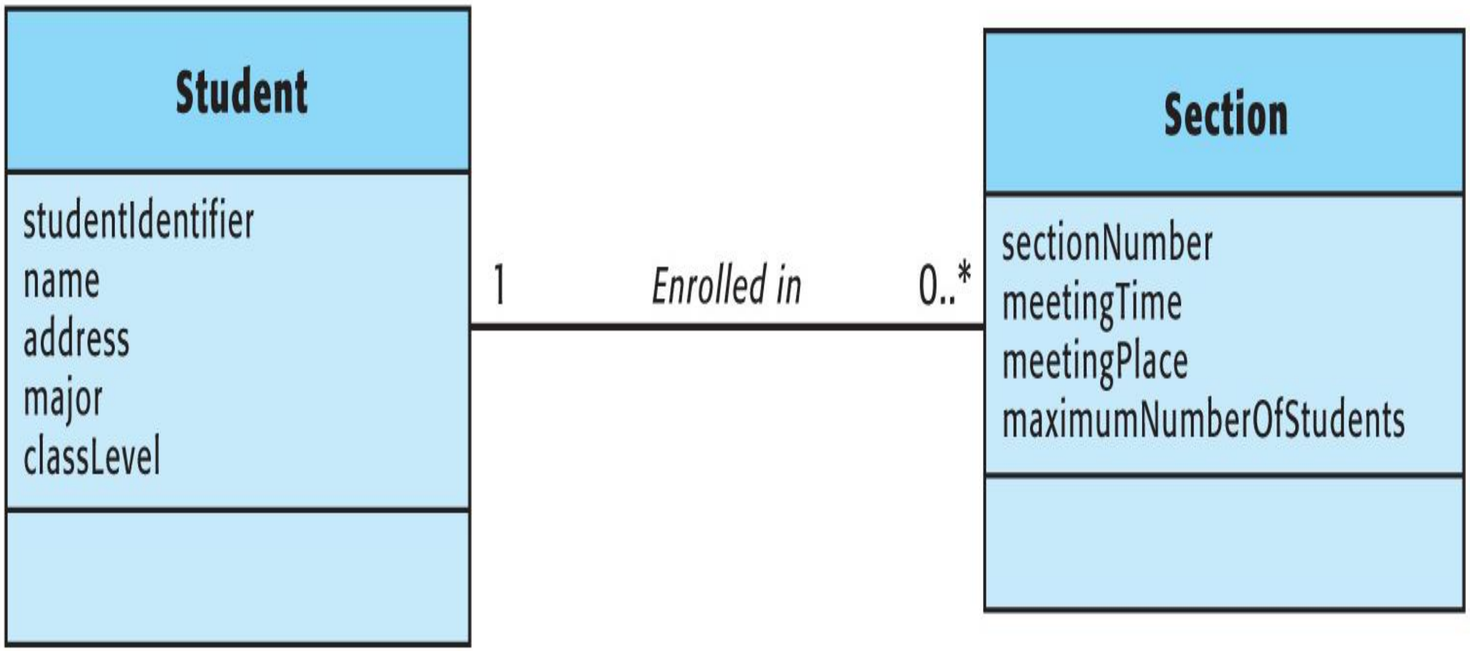
Attributes
(continued)

FIGURE 5.5



Associations
(between concepts)

FIGURE 5.7



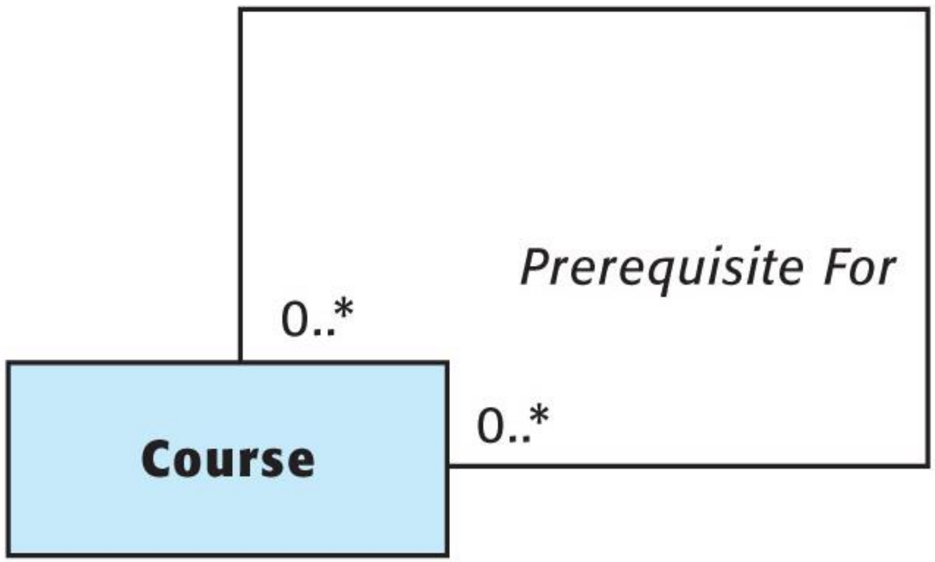
Associations
(continued)

Always model associations explicitly; never use an attribute to imply an association.

Reflexive Associations

A concept may be associated with itself.

FIGURE 5.12

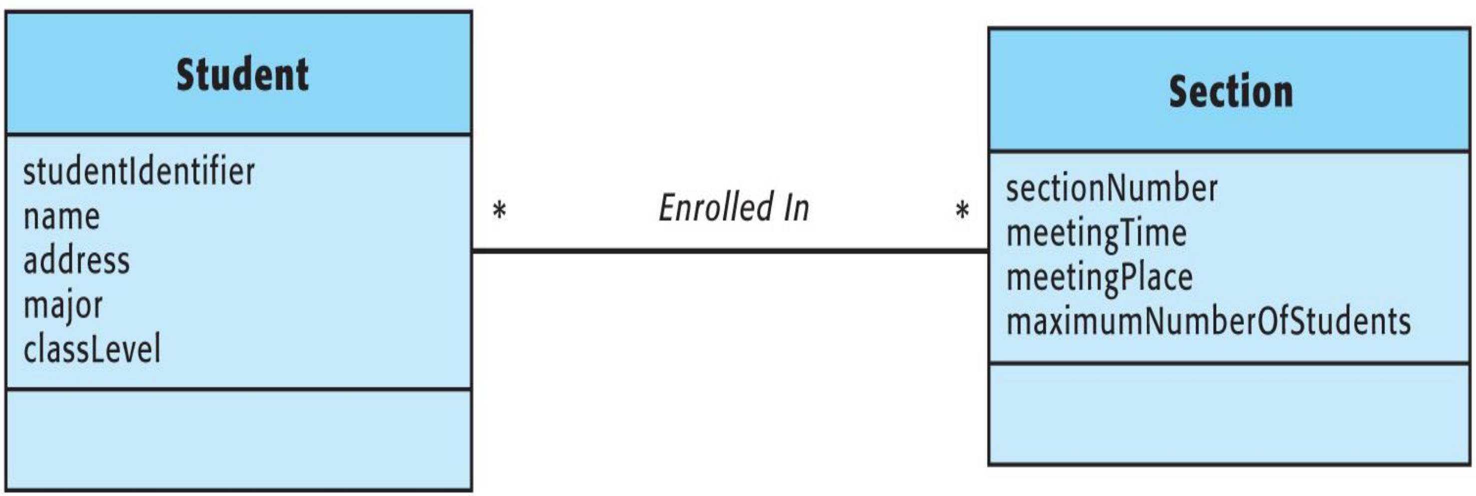


A course is a prerequisite for zero or more other courses.
A course has as prerequisites zero or more other courses.

Multiplicity of Associations

The multiplicity of an association is the number of instances of a concept which can be associated with one instance of another concept.

FIGURE 5.9



Multiplicity of Associations
(continued)

Each end of an association is labeled with the minimum and maximum values of its multiplicity.

0 .. 1

1 .. 1

.. * signifies unlimited (*more* or *many*)

* alone means *zero or more*

Associations and Generalization-Specialization Hierarchies

Identifying and adding associations and generalization-specialization hierarchies to the domain model is **Step 5c** of the process for object-oriented systems analysis.

Generalization-Specialization Hierarchies

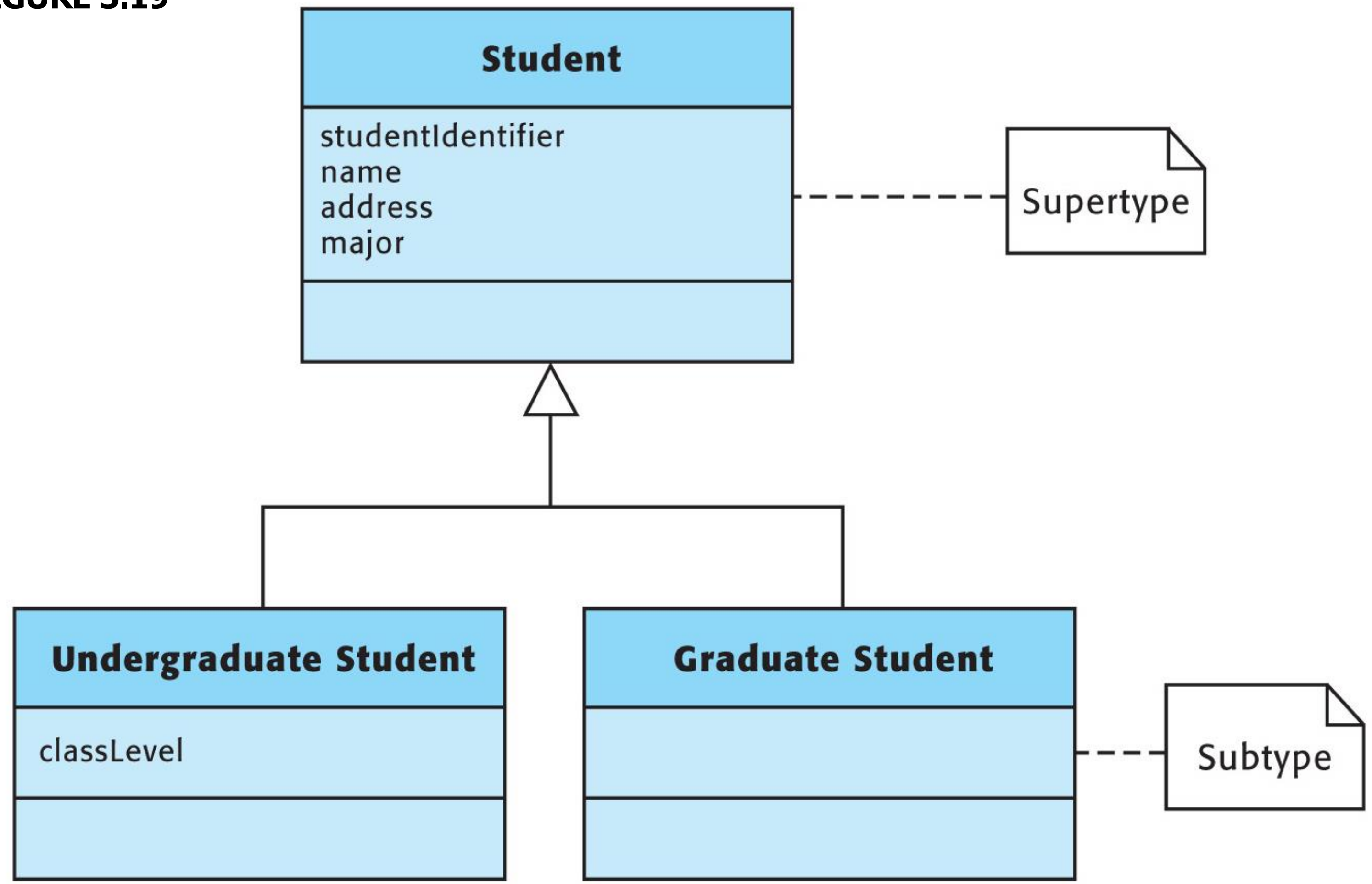
- A **generalization-specialization hierarchy** classifies a type of concept into its **subtypes**.

- Every instance of a subtype must also be an instance of its supertype.

- Subtypes have the same set of attributes as their supertype. These attributes are not duplicated in the domain model.

Generalization-Specialization Hierarchies (continued)

FIGURE 5.19



Postconditions for System Operation Contracts

- What instances of concepts must be created or deleted?
- What attributes have their values modified? To what new values?
- Which instances of associations must be added or deleted?

Use the past tense and the passive voice.

System Operation Contracts
(continued)

FIGURE 5.27

Contract Name:	**requestSection** (departmentCode, courseNumber, sectionNumber)
Responsibilities:	Enroll the Student in the Section.
Type:	System
Exceptions:	If the combination of department code, course number and section number is not valid, indicate that it was an error. If no seats are available, inform the Student.
Output:	
Preconditions:	Department and Section are known to the system.
Postconditions:	A new instance of the Enrolled In association was created, linking the Student and the Section.

Design Overview

Design is a critical intermediate step between a statement of requirements and the construction of a solution.

It produces a description of the solution – not the solution itself. This description is sufficiently complete and accurate to assure that the solution can be constructed.

Design models allow the behavior of proposed solutions to be evaluated and compared.

Responsibilities

The principal task of object-oriented program design is to **assign responsibilities to classes**.

A **responsibility** is an obligation of an object to other objects.

Responsibilities
(continued)

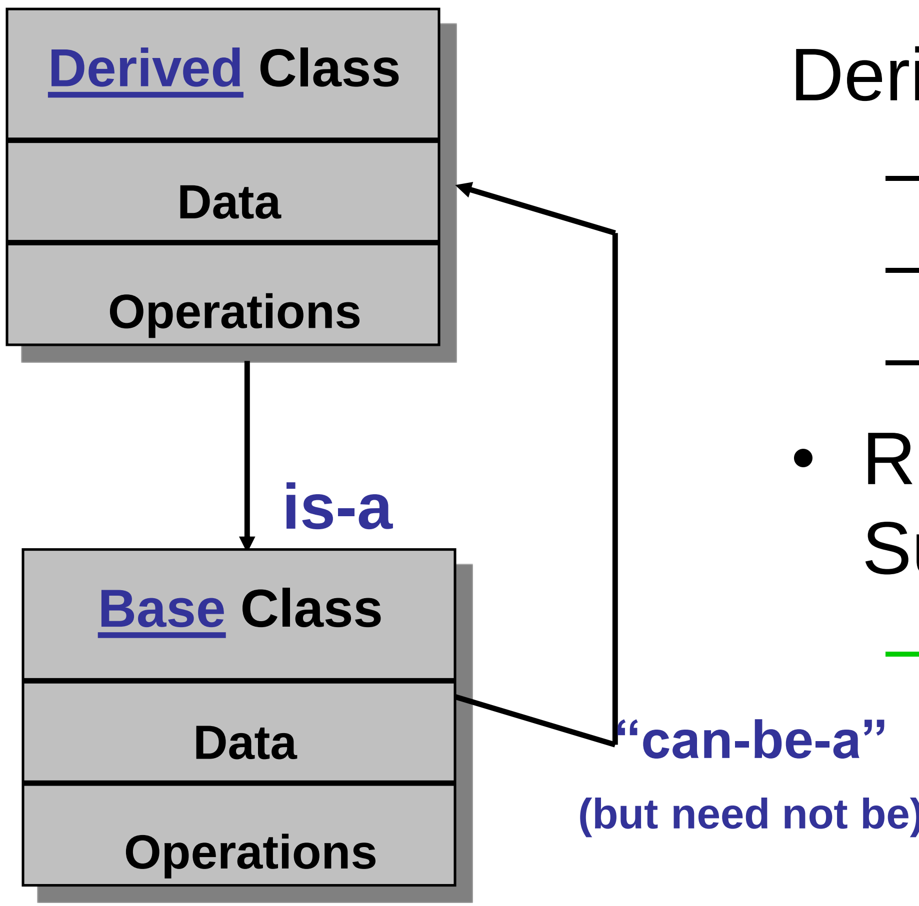
An object may be responsible for knowing:

- What it knows – its attributes
- Who it knows – the objects associated with it
- What it knows how to do – the operations it can perform

OO Review

- Class and Encapsulation (system parts)
  - Attributes (Private Information)
  - Methods (Public Behavior)
  - Inheritance
  - Polymorphism
- Interactions (doing things)
  - Messages
  - Parameters
  - “access” to or visibility of other objects
- Instances are NOT classes

Types of Relationships - Inheritance



Derived Class Object

- “is-a” Base Class Object
- “is-a-kind-of”
- “must-be-a”

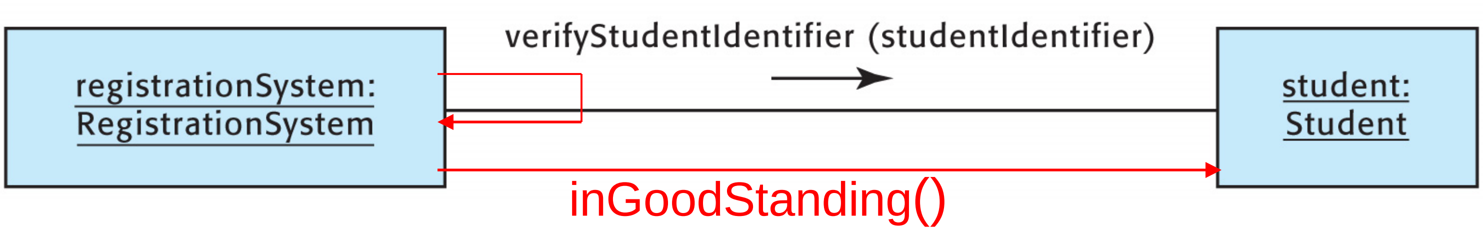
- Remember the Substitution Rule
  - **Any object of the derived class must be usable in place of a base-class object.**

Fundamental Concepts – Messages (continued)

Corrections Added
WLHonig

Visibility: For an object (the **client**) to send a message to another object (the **server**), the receiving object must be **visible** to the sending object. (That is, it must know the server's **identity**).

FIGURE 8.4



NEVER ask someone to confirm their own identity

Overview
(continued)

The interaction diagrams are developed on the basis of system operation contracts produced during analysis.

This overall approach is called "design by contract."

Design Overview

Design by contract assumes a commitment to a contract on the part of the object which receives a message.

The preconditions and postconditions of the system operation contracts drive the program design.

Design Overview
(continued)

In developing the interaction diagram for each system operation, we must assure that the operation:

- first checks whether every precondition of the contract is true, and then
- makes every postcondition of the contract come true.

Patterns for Object-Oriented Program Design

A pattern is a named statement of a design problem together with its solution and guidance for applying the pattern. Patterns include:

- Façade
- Creator
- Expert
- Singleton

The Façade Pattern

Problem: **Who should be responsible for handling a system operation message from an actor?**

Solution: **Assign this responsibility to an object representing the system as a whole.**

The Creator Pattern

Problem: **Who should be responsible for requesting the creation of a new object, i. e.,**

who sends the *create* message to the appropriate class?

Solution: **Assign this responsibility to a class which is in some way closely involved with the class.** (See Figure 8.4 in text for details.)

The Expert Pattern

Problem: What is the most basic principle for assigning responsibilities to objects?

Solution: Assign the responsibility to the class which has the information necessary to fulfill it.

Interaction Diagrams

An interaction diagram depicts the messages between objects or classes in a program. It shows collaborations between objects.

The UML includes two types of interaction diagrams – collaboration diagrams and sequence diagrams.

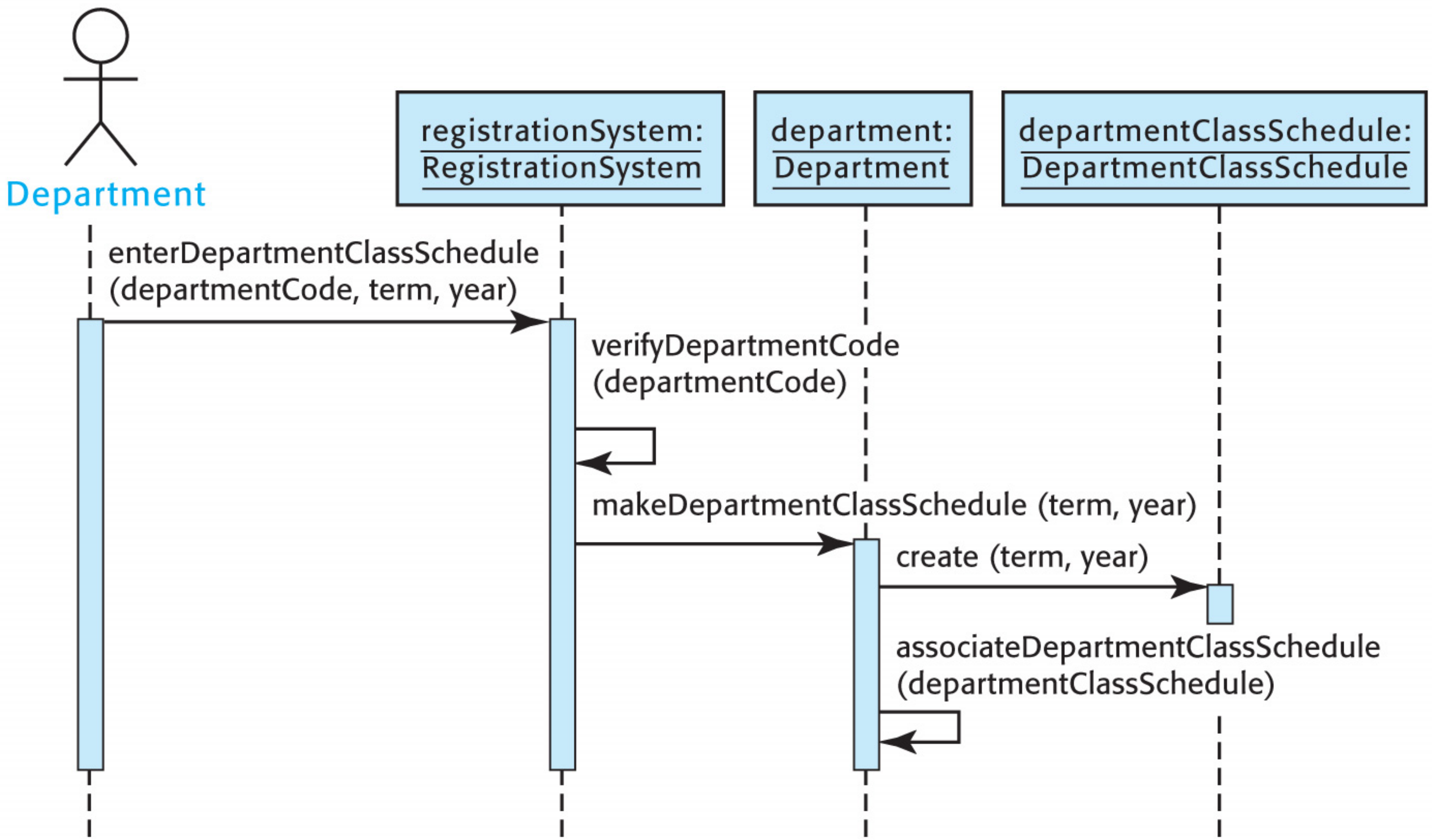
Sequence Diagrams

A **sequence diagram** shows interactions in a fence format.

The messages appear from top to bottom in the sequence in which they occur.

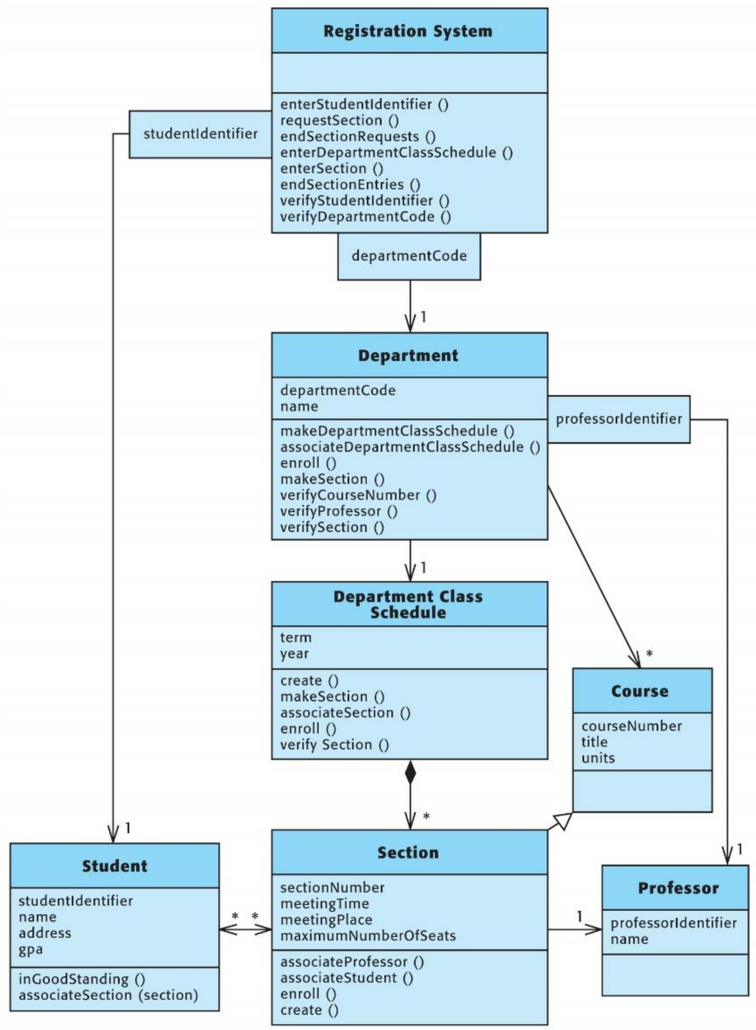
Sequence Diagrams
(continued)

FIGURE 8.6



Design Class Diagrams
(continued)

FIGURE 8.7



Design Sequence Diagrams

System sequence diagrams show only messages between the system and actors.

Design sequence diagrams show all the messages between objects inside the system.

Learning Objectives

- Explain fundamental object-oriented concepts.
- Understand what patterns are and how they are used.
- Learn how to assign responsibilities to classes using the Façade, Creator, and Expert patterns.

UML Class Diagram Checklist Footsteps for the Programmer

All needed classes defined

- Clear and accurate names

Major Associations identified

- Good names, show in one or two directions
- With clear cardinality

Full set of attributes, with good names

- With complete type definition

Full set of methods

- With complete signature