10

Object-Oriented Systems Analysis and Design Using UML

Learning Objectives

- Understand what object-oriented systems analysis and design is and appreciate its usefulness.
- Comprehend the concepts of Unified Modeling Language (UML), the standard approach for modeling a system in the object-oriented world.
- Apply the steps used in UML to break down the system into a use case model and then a class model.
- Diagram systems with the UML toolset so they can be described and properly designed.
- Document and communicate the newly modeled object-oriented system to users and other analysts.

Object-Oriented Analysis and Design

- Works well in situations where complicated systems are undergoing continuous maintenance, adaptation, and design
- Objects, classes are reusable
- The Unified Modeling Language (UML) is an industry standard for modeling object-oriented systems.

Object-Oriented Analysis and Design (continued)

- Reusability
  - Recycling of program parts should reduce the costs of development in computer-based systems
- Maintaining systems
  - Making a change in one object has a minimal impact on other objects

Major Topics

- Object-oriented concepts
- CRC cards and object think
- Unified Modeling Language
- Use case and other UML diagrams
- Packages
- Using UML

Object-Oriented Concepts

- Objects
- Classes
- Inheritance

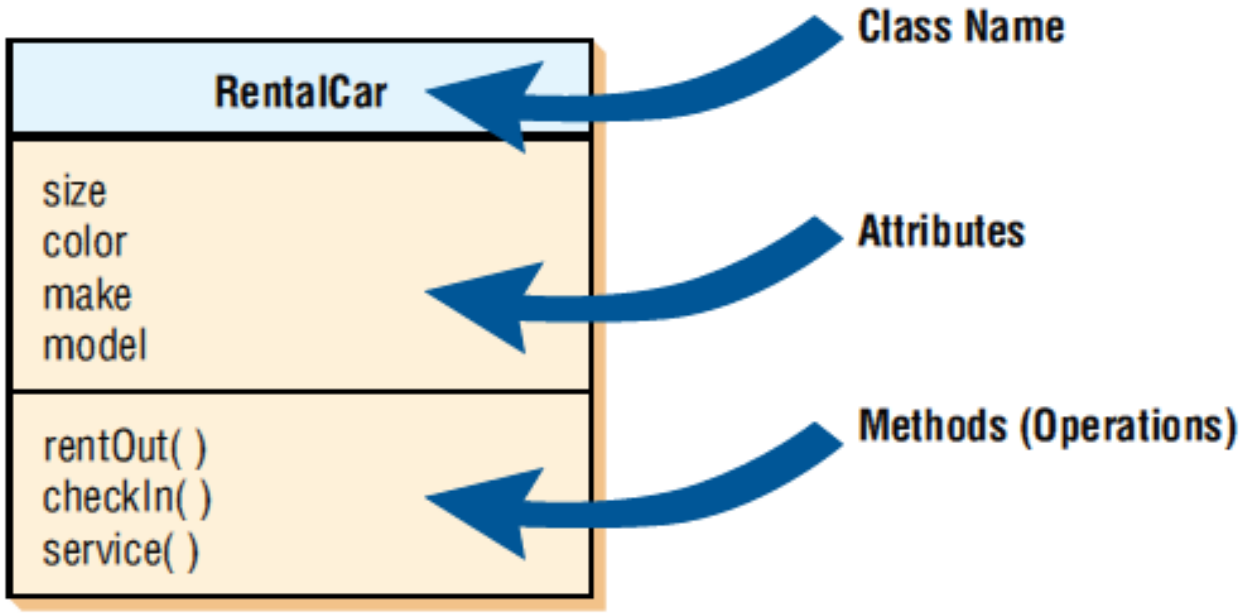
Objects

- Persons, places, or things that are relevant to the system being analyzed
- May be customers, items, orders, and so on
- May be GUI displays or text areas on a display

Classes

- Defines the set of shared attributes and behaviors found in each object in the class
- Should have a name that differentiates it from all other classes
- Instantiate is when an object is created from a class
- An attribute describes some property that is possessed by all objects of the class
- A method is an action that can be requested from any object of the class

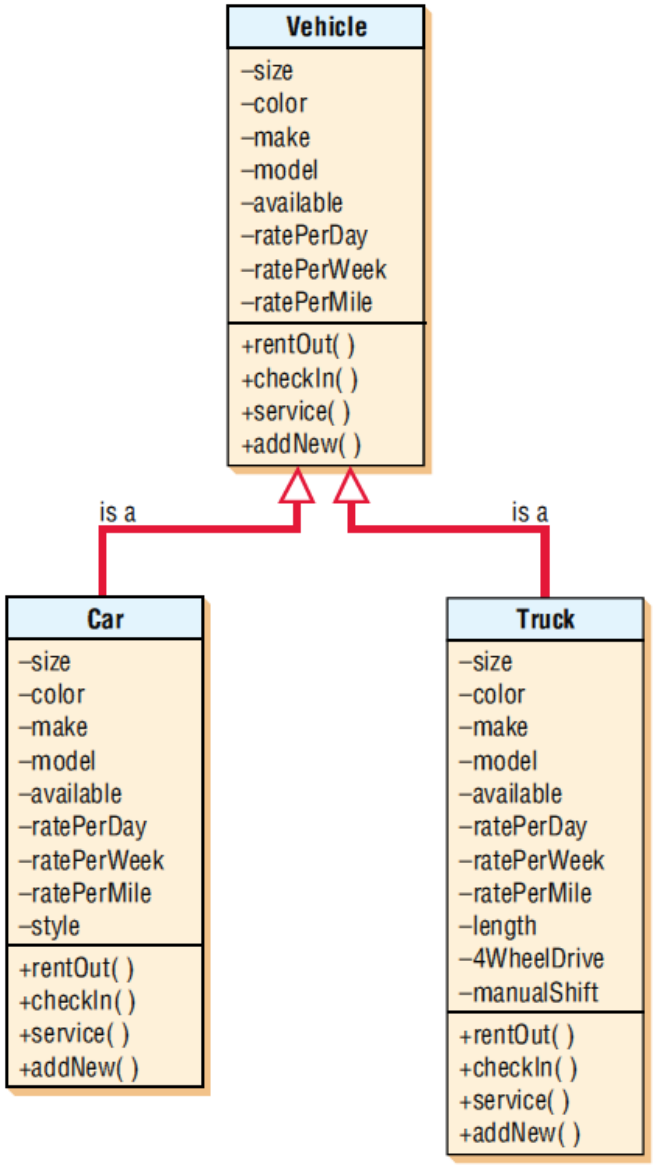
An Example of a UML Class: A Class Is Depicted as a Rectangle Consisting of the Class Name, Attributes, and Methods (Figure 10.1)



Inheritance

- When a derived class inherits all the attributes and behaviors of the base class
- Reduces programming labor by using common objects easily
- A feature only found in object-oriented systems

A Class Diagram Showing Inheritance (Figure 10.2)

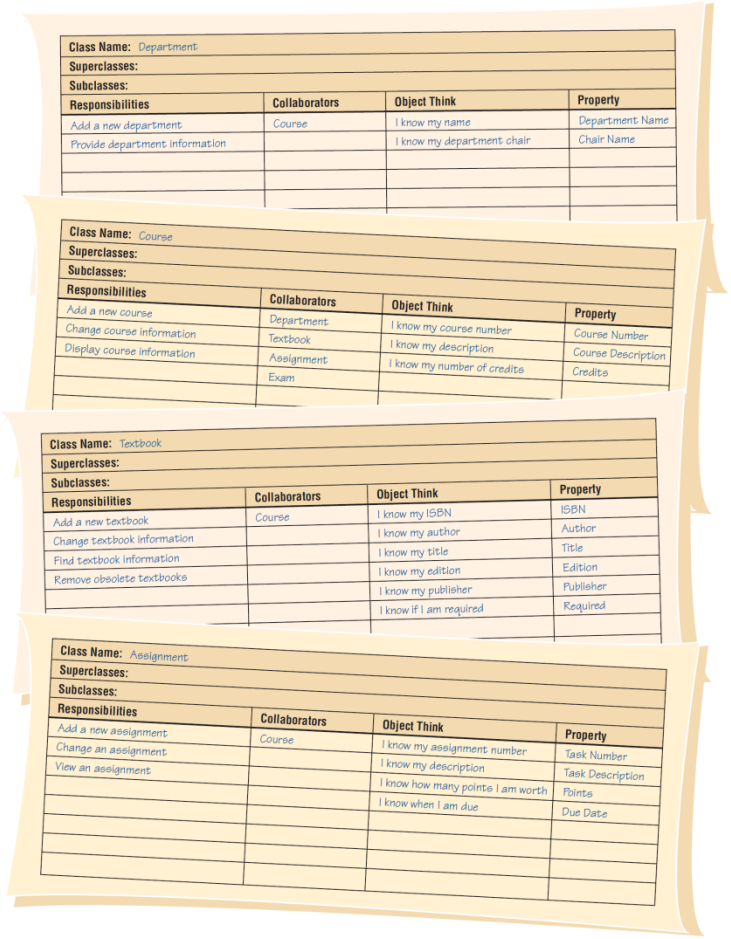


Car and truck are specific examples of vehicles and inherit the characteristics of the more general class vehicle.

CRC Cards and Object Think

- CRC
  - Class
  - Responsibilities
  - Collaborators
- CRC cards are used to represent the responsibilities of classes and the interaction between the classes

Four CRC Cards for Course Offerings Show How Analysts Fill in the Details for Classes, Responsibilities, and Collaborators, as Well as for Object Think Statements and Property Names (Figure 10.3)



Class Name: Department
Superclasses:
Subclasses:

Responsibilities	Collaborators	Object Think	Property
Add a new department	Course	I know my name	Department Name
Provide department information		I know my department chair	Chair Name

Class Name: Course
Superclasses:
Subclasses:

Responsibilities	Collaborators	Object Think	Property
Add a new course	Department	I know my course number	Course Number
Change course information	Textbook	I know my description	Course Description
Display course information	Assignment	I know my number of credits	Credits
	Exam		

Class Name: Textbook
Superclasses:
Subclasses:

Responsibilities	Collaborators	Object Think	Property
Add a new textbook	Course	I know my ISBN	ISBN
Change textbook information		I know my author	Author
Find textbook information		I know my title	Title
Remove obsolete textbooks		I know my edition	Edition
		I know my publisher	Publisher
		I know if I am required	Required

Class Name: Assignment
Superclasses:
Subclasses:

Responsibilities	Collaborators	Object Think	Property
Add a new assignment	Course	I know my assignment number	Task Number
Change an assignment		I know my description	Task Description
View an assignment		I know how many points I am worth	Points
		I know when I am due	Due Date

Interacting during a CRC Session

- Identify all the classes you can
- Create scenarios
- Identify and refine responsibilities

The Unified Modeling Language (UML) Concepts and Diagrams

- Things
- Relationships
- Diagrams

Things

- Structural things are:
  - Classes, interfaces, use cases, and other elements that provide a way to create models
  - They allow the user to describe relationships
- Behavioral things
- Describe how things work
  - Interactions and state machines
- Group things
  - Used to define boundaries
- Annotational things
  - Can add notes to the diagrams

Relationships

- Structural relationships
  - Tie things together in structural diagrams
- Behavioral relationships
  - Used in behavioral diagrams

Structural Relationships

- Dependencies
- Aggregations
- Associations
- Generalizations

Behavioral Relationships

- Communicates
- Includes
- Extends
- Generalizes

Diagrams

- Structural diagrams
  - Used to describe the relation between classes
- Behavior diagrams
  - Used to describe the interaction between people (actors) and a use case (how the actors use the system)

Structural Diagrams

- Class diagrams
- Object diagrams
- Component diagrams
- Deployment diagrams

Behavioral Diagrams

- Use case diagrams
- Sequence diagrams
- Collaboration diagrams
- Statechart diagrams
- Activity diagrams

An Overall View of UML and Its Components: Things, Relationships, and Diagrams (Figure 10.4)

UML Category	UML Elements	Specific UML Details
Things	Structural Things	Classes Interfaces Collaborations Use Cases Active Classes Components Nodes
	Behavioral Things	Interactions State Machines
	Grouping Things	Packages
	Annotational Things	Notes
Relationships	Structural Relationships	Dependencies Aggregations Associations Generalizations
	Behavioral Relationships	Communicates Includes Extends Generalizes
Diagrams	Structural Diagrams	Class Diagrams Component Diagrams Deployment Diagrams
	Behavioral Diagrams	Use Case Diagrams Sequence Diagrams Communication Diagrams Statechart Diagrams Activity Diagrams

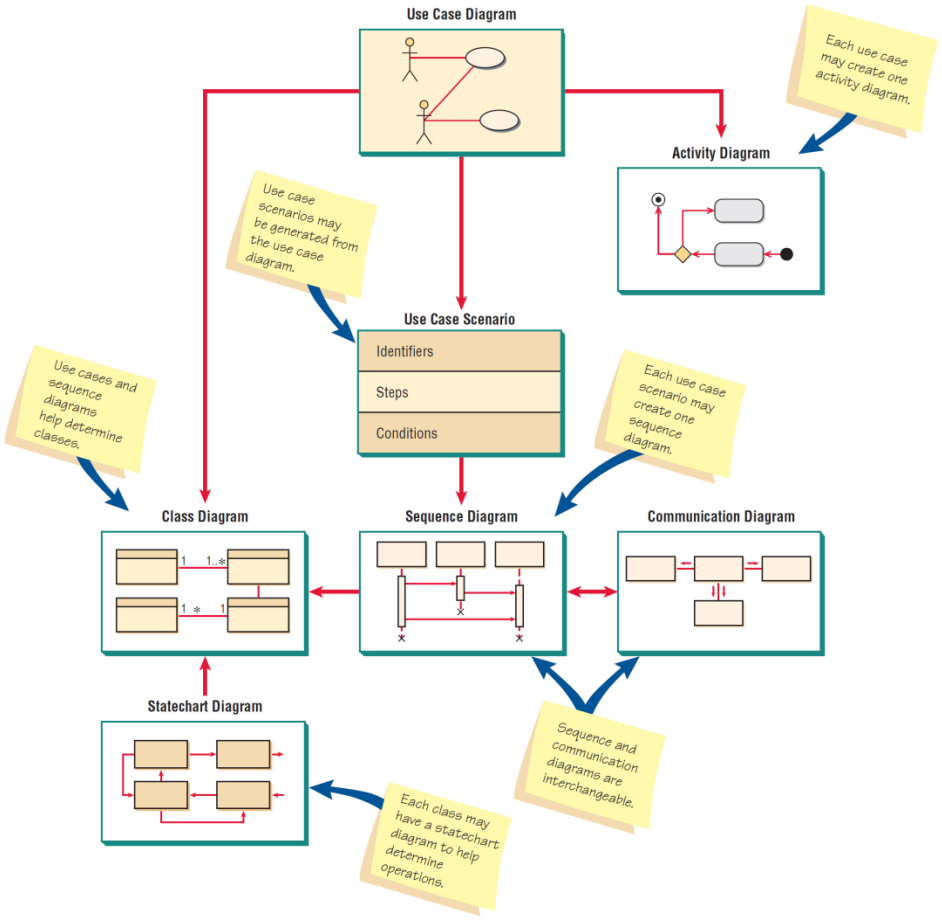
Commonly Used UML Diagrams

- Use case diagram
  - Describing how the system is used
  - The starting point for UML modeling
- Use case scenario
  - A verbal articulation of exceptions to the main behavior described by the primary use case
- Activity diagram
  - Illustrates the overall flow of activities

Commonly Used UML Diagrams (continued)

- Sequence diagrams
  - Show the sequence of activities and class relationships
- Class diagrams
  - Show classes and relationships
- Statechart diagrams
  - Show the state transitions

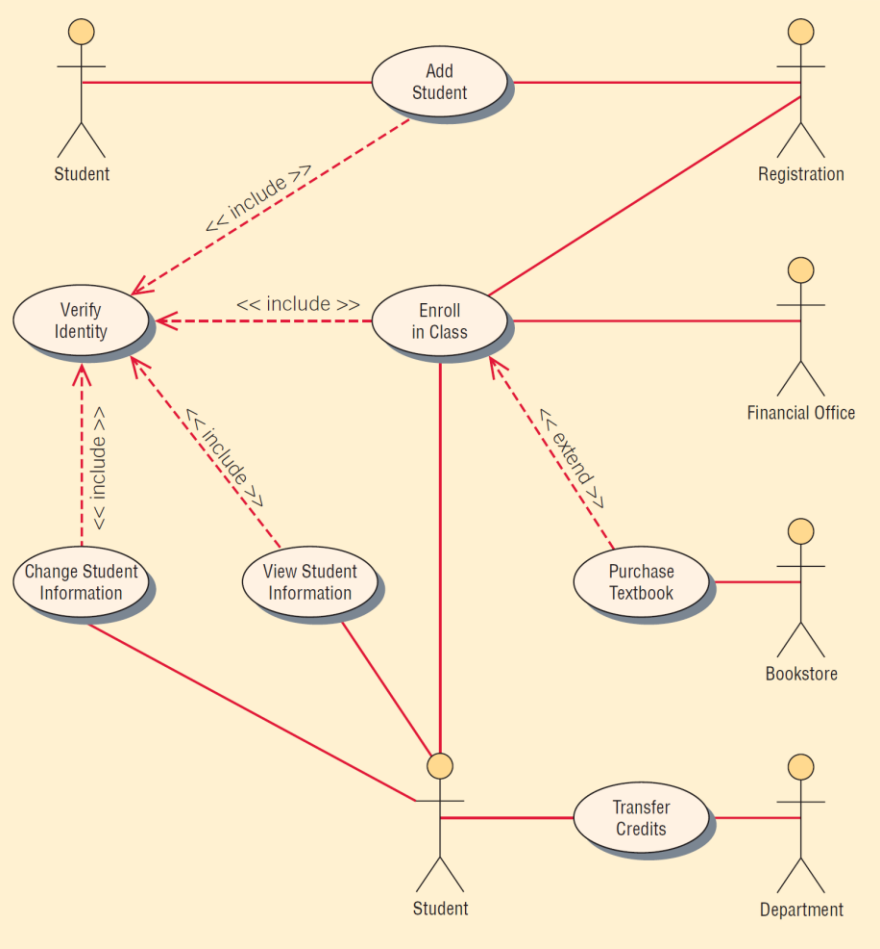
An Overview of UML Diagrams Showing How Each Diagram Leads to the Development of Other UML Diagrams (Figure 10.5)



Use Case Modeling

- Describes what the system does, without describing how the system does it
- Based on the interactions and relationships of individual use cases
- Use case describes
  - Actor
  - Event
  - Use case

A Use Case Example of Student Enrollment (Figure 10.6)



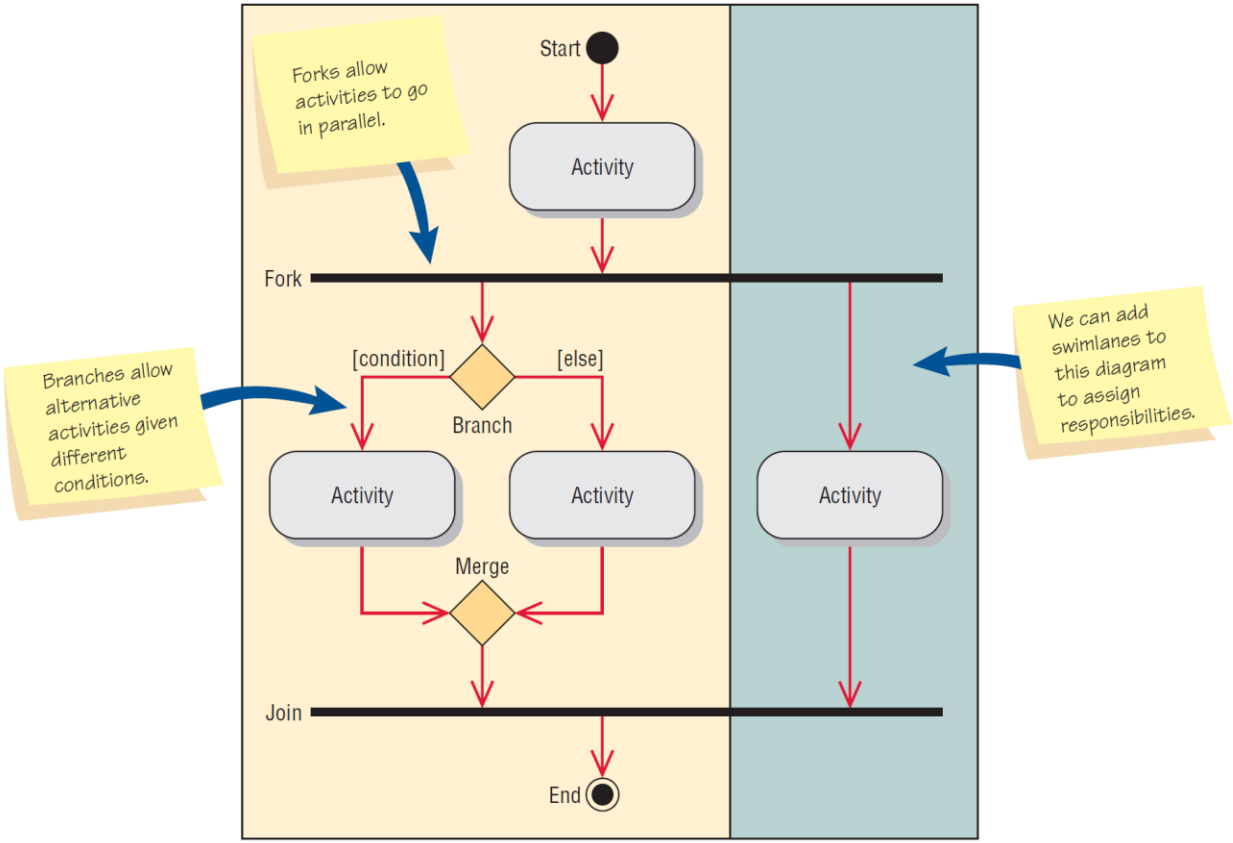
A Use Case Scenario Is Divided into Three Sections: Identification and Initiation, Steps Performed, and Conditions, Assumptions, and Questions (Figure 10.7)

Use case name:	Change Student Information	**UniqueID:** Student UC 005
Area:	Student System	
Actor(s):	Student	
Description:	Allow student to change his or her own information, such as name, home address, home telephone, campus address, campus telephone, cell phone, and other information using a secure Web site.	
Triggering Event:	Student uses Change Student Information Web site, enters student ID and password, and clicks the **Submit** button.	
Trigger type:	☒ External ☐ Temporal	
Steps Performed (Main Path)		**Information for Steps**
1. Student Logons on to the secure Web server.		Student ID, Password
2. Student record is read and password is verified.		Student Record, StudentID, Password
3. Current student personal information is displayed on the Change Student Web page.		Student Record
4. Student enters changes on the Change Student Web form and clicks **Submit** button.		Change Student Web Form
5. Changes are validated on the Web server.		Change Student Web Form
6. Change Student Journal record is written.		Change Student Web Form
7. Student record is updated on the Student Master.		Change Student Web Form, Student Record
8. Confirmation Web page is sent to the student.		Confirmation Page
Preconditions:	Student is on the Change Student Information Web page.	
Postconditions:	Student has successfully changed personal information.	
Assumptions:	Student has a browser and a valid user ID and password.	
Requirements Met:	Allow students to be able to change personal information using a secure Web site.	
Outstanding Issues:	Should the number of times a student is allowed to logon be controlled?	
Priority:	Medium	
Risk:	Medium	

Activity Diagrams

- Show the sequence of activities in a process, including sequential and parallel activities, and decisions that are made
- Symbols
  - Rectangle with rounded ends
  - Arrow
  - Diamond
  - Long, flat rectangle
  - Filled-in circle
  - Black circle surrounded by a white circle
  - Swimlanes

Specialized Symbols Are Used to Draw an Activity Diagram (Figure 10.8)



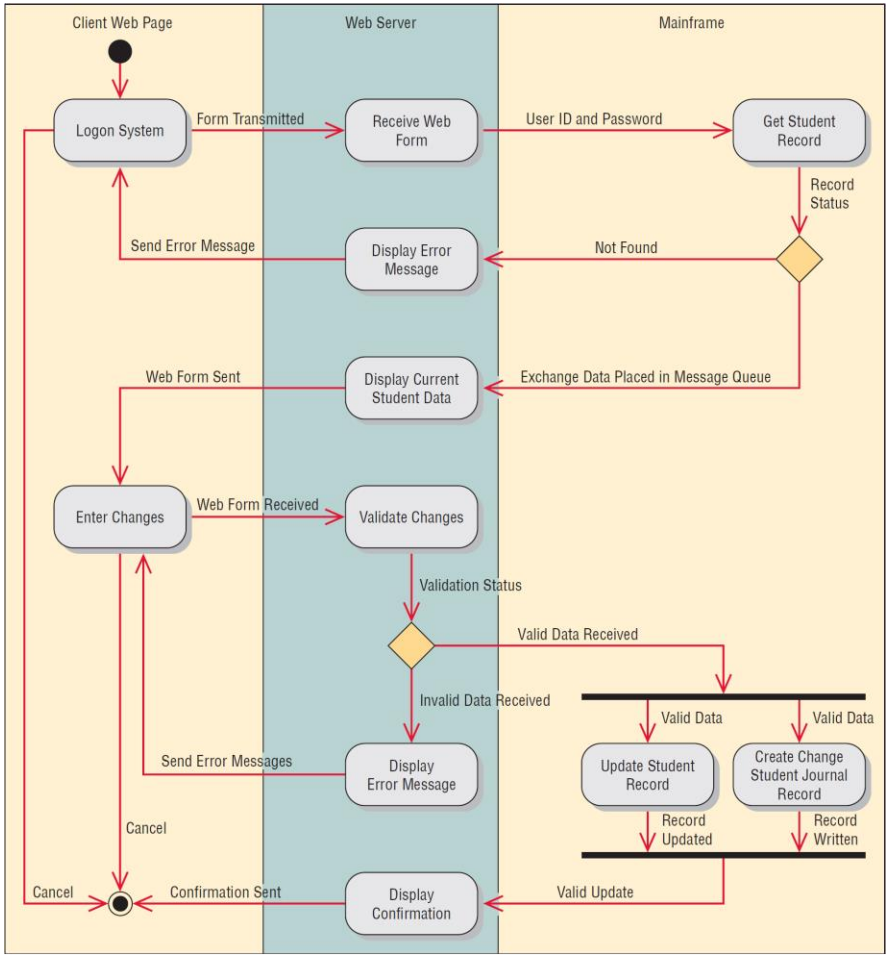
Creating Activity Diagrams

- Created by asking what happens first, what happens second, and so on
- Must determine what activities are done in sequence or in parallel
- The sequence of activities can be determined from physical data flow diagrams
- Can be created by examining all the scenarios for a use case

Swimlanes

- Useful to show how the data must be transmitted or converted
- Help to divide up the tasks in a team
- Makes the activity diagram one that people want to use to communicate with others

This Activity Diagram Shows Three Swimlanes: Client Web Page, Web Server, and Mainframe (Figure 10.9)



Activity Diagrams and Test Plans

- Activity diagrams may be used to construct test plans
- Each event must be tested to see if the system goes to the next state
- Each decision must be tested

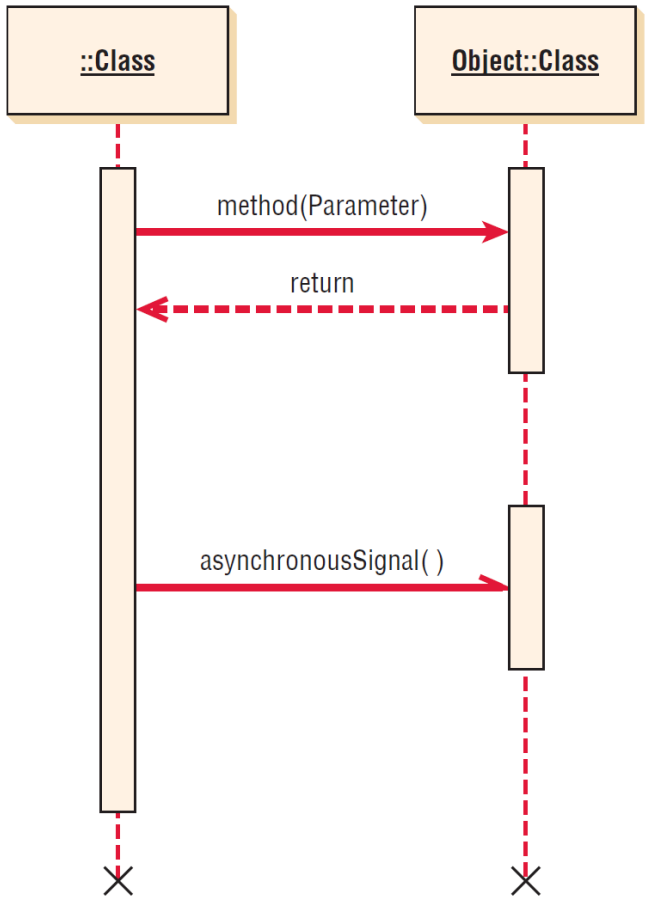
Activity Diagrams Not Created for All Use Cases

- Use an activity diagram when:
  - It helps to understand the activities of a use case
  - The flow of control is complex
  - There is a need to model workflow
  - When all scenarios for a use case need to be shown

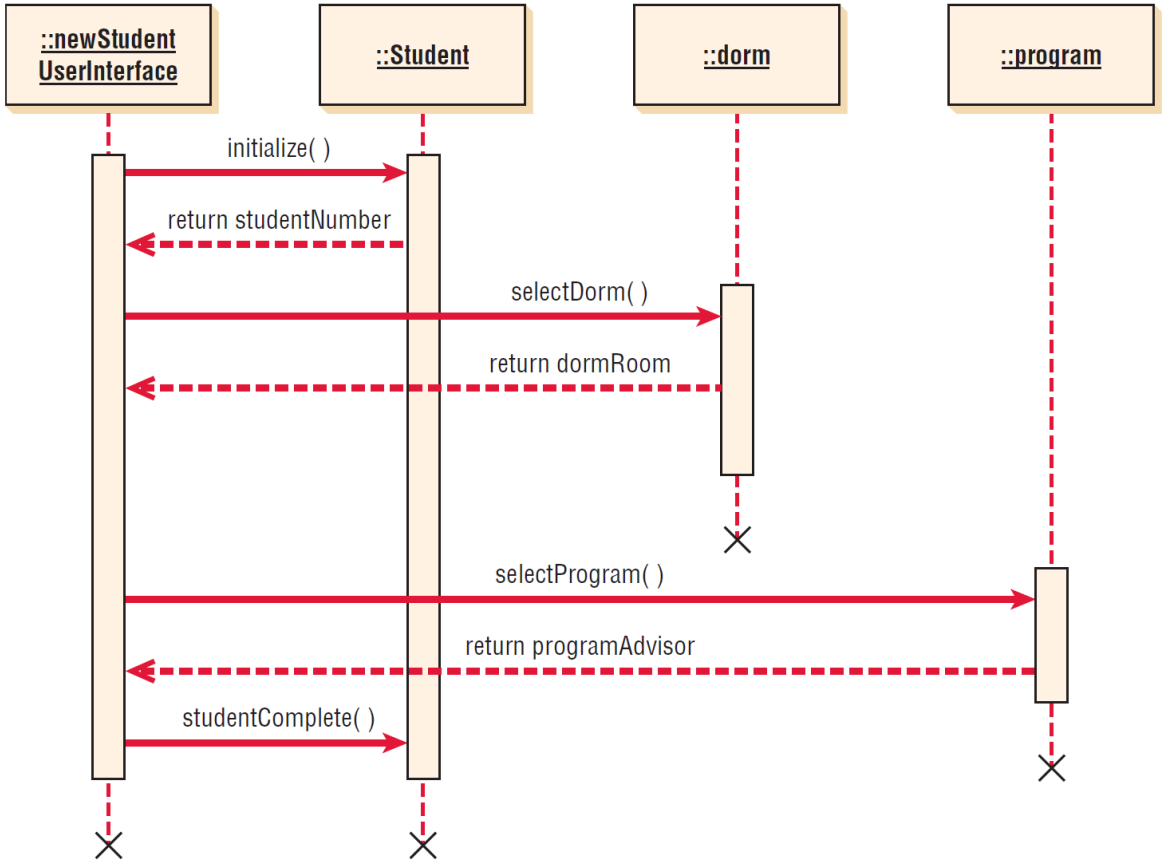
Sequence Diagrams

- Illustrate a succession of interactions between classes or object instances over time
- Often used to show the processing described in use case scenarios
- Used to show the overall pattern of the activities or interactions in a use case

Specialized Symbols Used to Draw a Sequence Diagram (Figure 10.10)



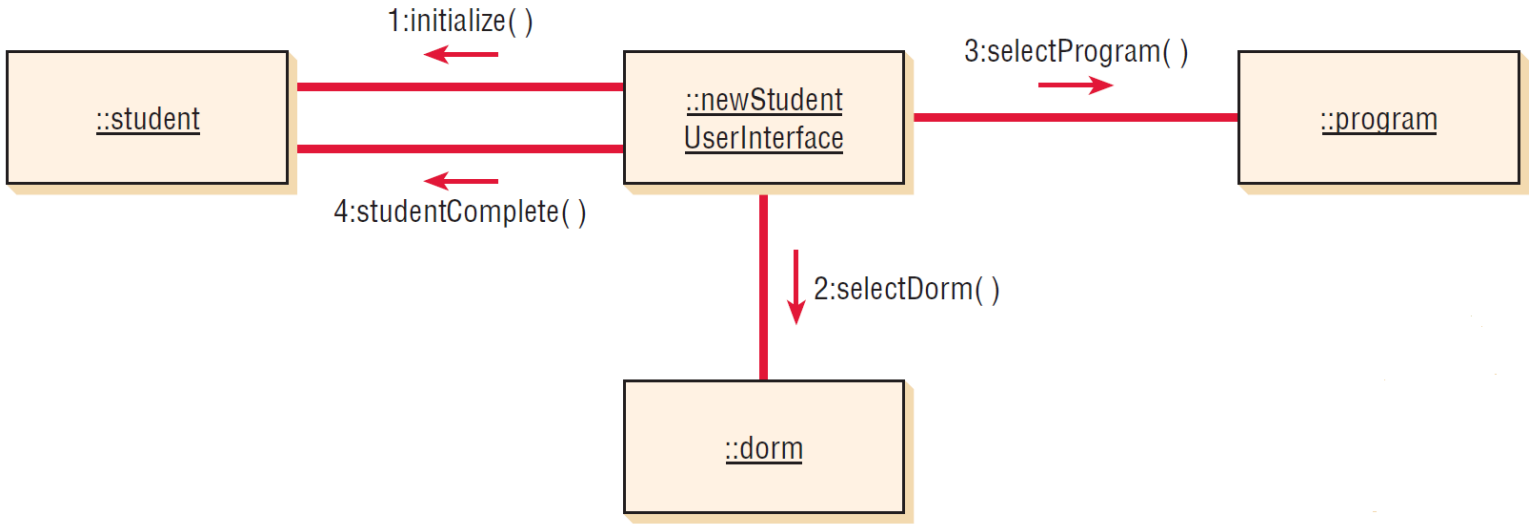
A Sequence Diagram for Student Admission: Sequence Diagrams Emphasize the Time Ordering of Messages (Figure 10.11)



Communication Diagrams

- Describes the interactions of two or more things in the system that perform a behavior that is more than any one of the things can do alone
- Shows the same information as a sequence diagram, but may be more difficult to read
- Emphasizes the organization of objects
- Made up of objects, communication links, and the messages that can be passed along those links

A Communication Diagram for Student Admission (Figure 10.12)



Communication diagrams show the same information that is depicted in a sequence diagram but emphasize the organization of objects rather than the time ordering.

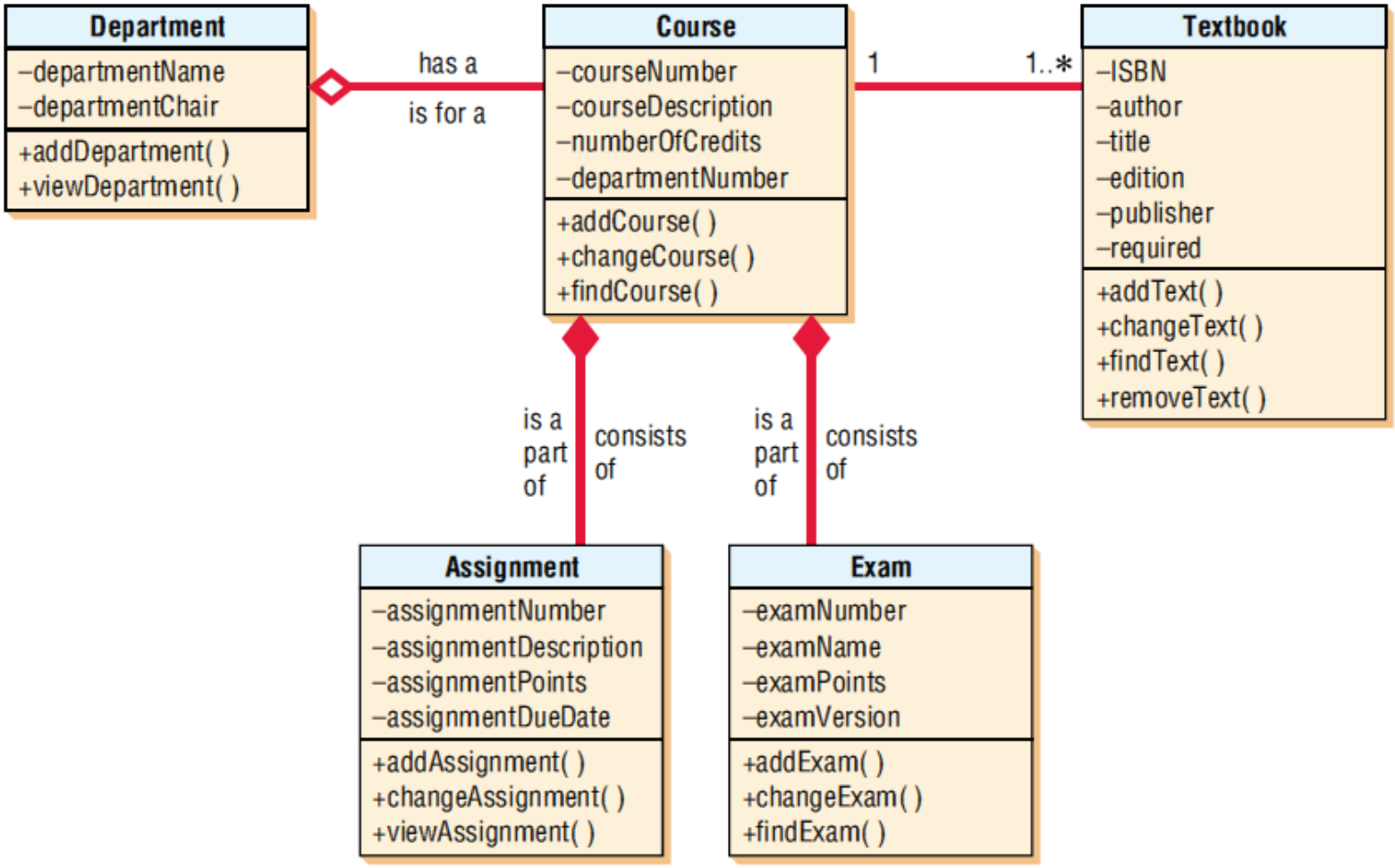
Class Diagrams

- Show the static features of the system and do not represent any particular processing
- Show the nature of the relationships between classes
- Show data storage requirements as well as processing requirements

Class Diagrams (continued)

- Classes
- Attributes
  - Private
  - Public
  - Protected
- Methods
  - Standard
  - Custom

A Class Diagram for Course Offerings: The Filled-In Diamonds Show Aggregation and the Empty Diamond Shows a Whole-Part Relationship (Figure 10.13)



Method Overloading

- Including the same method (or operation) several times in a class
- The same method may be defined more than once in a given class, as long as the parameters sent as part of the message are different

Types of Classes

- Entity classes
- Interface classes
- Abstract classes
- Control classes

Entity Classes

- Represent real-world items
- The entities represented on an entity-relationship diagram

Interface or Boundary Classes

- Provide a means for users to work with the system
- Human interfaces may be a display, window, Web form, dialogue box, touch-tone telephone, or other way for users to interact with the system
- System interfaces involve sending data to or receiving data from others

Abstract Classes

- Linked to concrete classes in a generalization/specialization relationship
- Cannot be directly instantiated

Control Classes

- Used to control the flow of activities
- Many small control classes can be used to achieve classes that are reusable

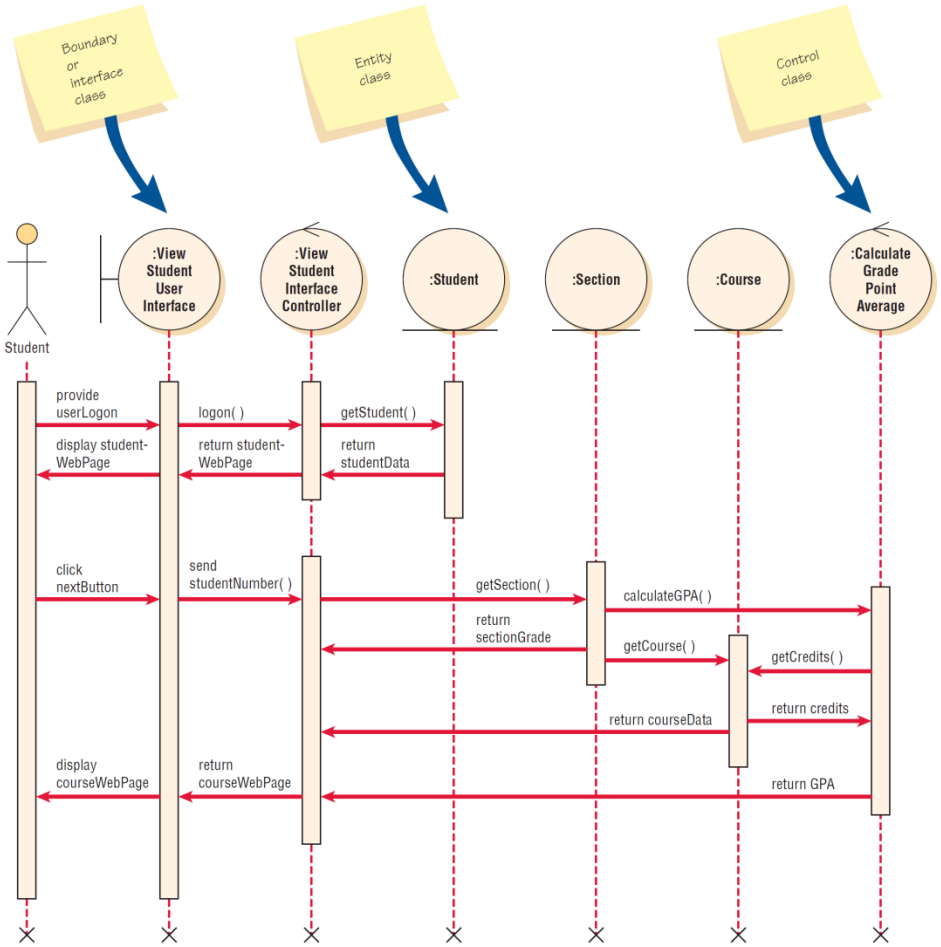
Presentation, Business, and Persistence Layers

- Sequence diagrams may be discussed using three layers:
  - Presentation layer, what the user sees, corresponding to the interface or boundary classes
  - Business layer, containing the unique rules for this application, corresponding roughly to control classes
  - Persistence or data access layer, for obtaining and storing data, corresponding to the entity classes

Defining Messages and Methods

- Each message may be defined using a notation similar to that described for the data dictionary
- The methods may have logic defined using structured English, a decision table, or a decision tree

A Sequence Diagram for Using Two Web Pages: One for Student Information, One for Course Information (Figure 10.15)



Create Sequence Diagrams

- Include the actor from the use case diagram
- Define one or more interface classes for each actor
- Each use case should have one control class
- Examine the use case to see what entity classes are required
- The sequence diagram may be modified when doing detailed design

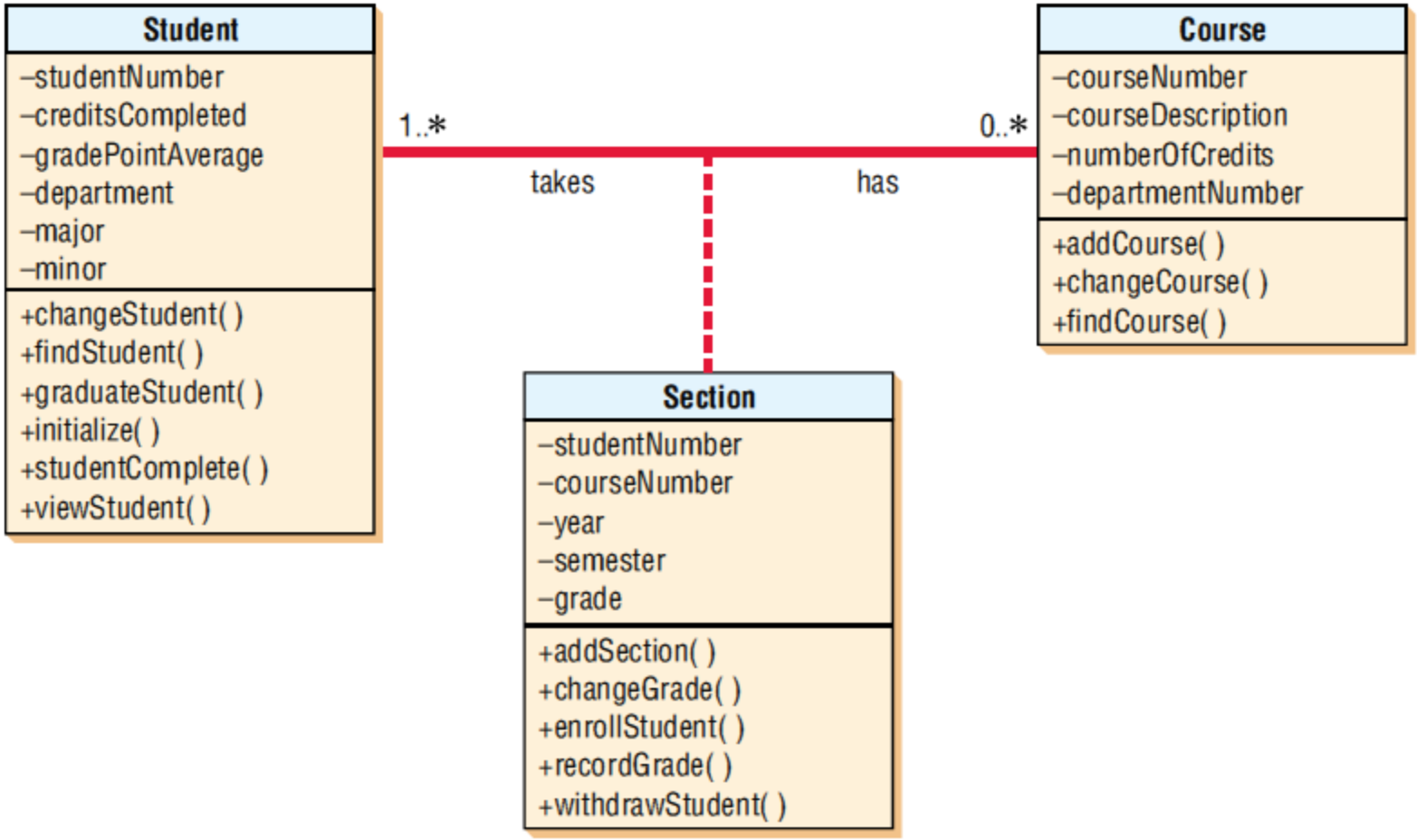
Creating a Test Plan from a Sequence Diagram

- Does each method return correct results?
- Ensure that entity classes store or obtain the correct attribute values
- Verify that all JavaScript paths work correctly
- Ensure that the server control classes work correctly
- Ask, “What may fail?”
- Determine what to do if something can fail

Relationships

- The connections between classes
  - Associations
  - Whole/part

An Example of an Associative Class in Which a Particular Section Defines the Relationship between a Student and a Course (Figure 10.18)



Associations

- The simplest type of relationship
- Association classes are those that are used to break up a many-to-many association between classes
- An object in a class may have a relationship to other objects in the same class, called a reflexive association

Whole/Part Relationships

- When one class represents the whole object, and other classes represent parts
- Categories
  - Aggregation
  - Collection
  - Composition

Aggregation

- A “has a” relationship
- Provides a means of showing that the whole object is composed of the sum of its parts

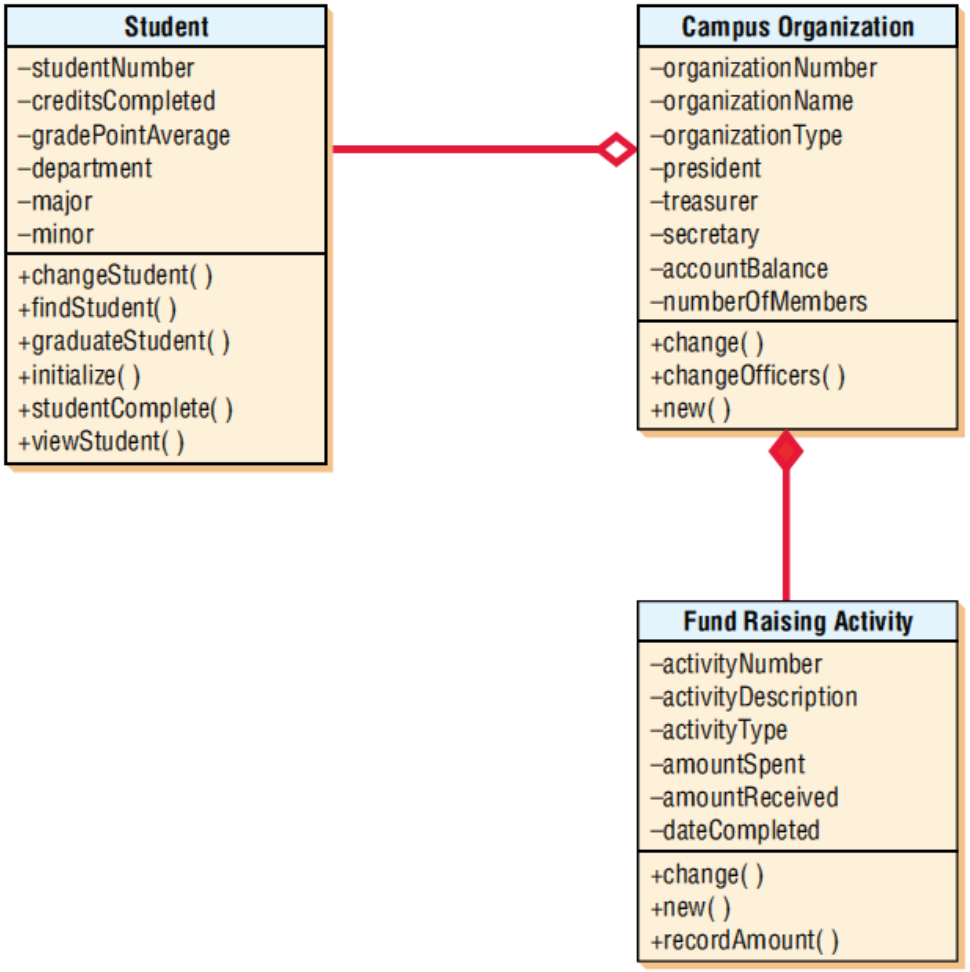
Collection

- Consists of a whole and its members
- Members may change, but the whole retains its identity
- A weak association

Composition

- The whole has a responsibility for the parts, and is a stronger relationship
- If the whole is deleted, all parts are deleted

An Example of Whole-Part and Aggregation Relationships (Figure 10.19)



Generalization/Specialization Diagrams

- Generalization
- Inheritance
- Polymorphism
- Abstract classes
- Messages

Generalization

- Describes a relationship between a general kind of thing and a more specific kind of thing
- Described as an "is a" relationship
- Used for modeling class inheritance and specialization
- General class is a parent, base, or superclass
- Specialized class is a child, derived, or subclass

Inheritance

- Helps to foster reuse
- Helps to maintain existing program code

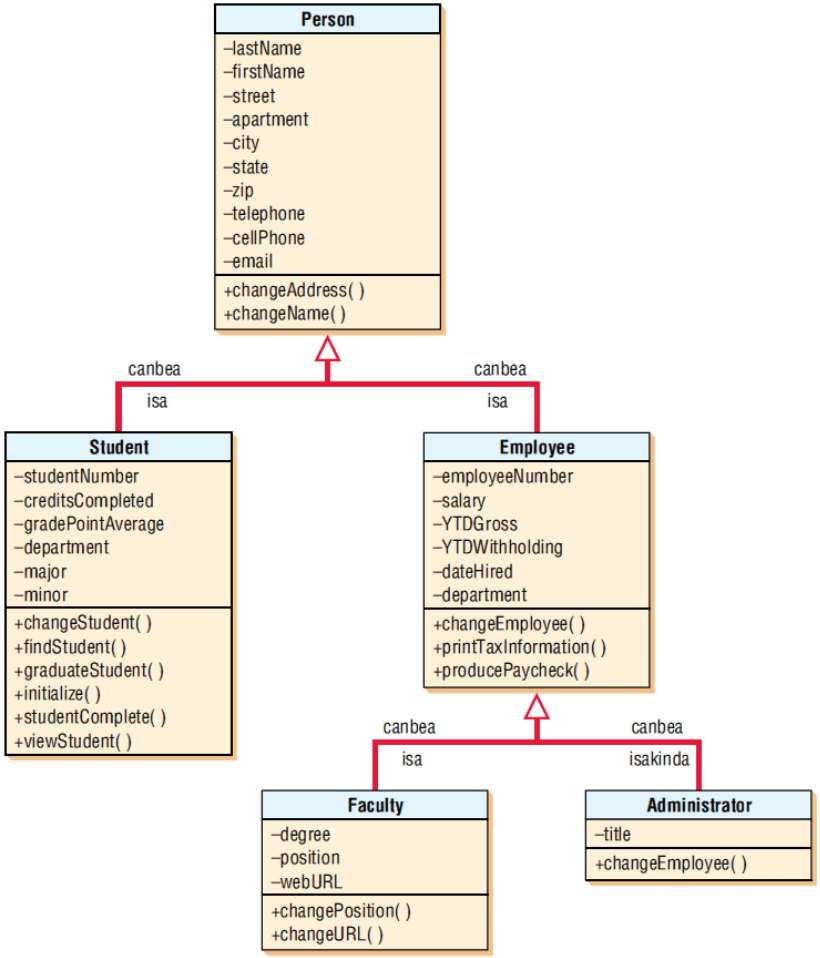
Polymorphism

- The capability of an object-oriented program to have several versions of the same method with the same name within a superclass/subclass relationship
- The subclass method overrides the superclass method
- When attributes or methods are defined more than once, the most specific one is used

Abstract Classes

- Abstract classes are general classes
- No direct objects or class instances, and is only used in conjunction with specialized classes
- Usually have attributes and may have a few methods

A Generalization/Specification Diagram Is a Refined Form of a Class Diagram (Figure 10.20)



Finding Classes

- During interviewing or JAD sessions
- During facilitated team sessions
- During brainstorming sessions
- Analyzing documents and memos
- Examining use cases, looking for nouns

Determining Class Methods

- Standard methods
- Examine a CRUD matrix

Messages

- Used to send information by an object in one class to an object in another class
- Acts as a command, telling the receiving class to do something
- Consists of the name of the method in the receiving class, as well as the attributes that are passed with the method name
- May be thought of as an output or an input

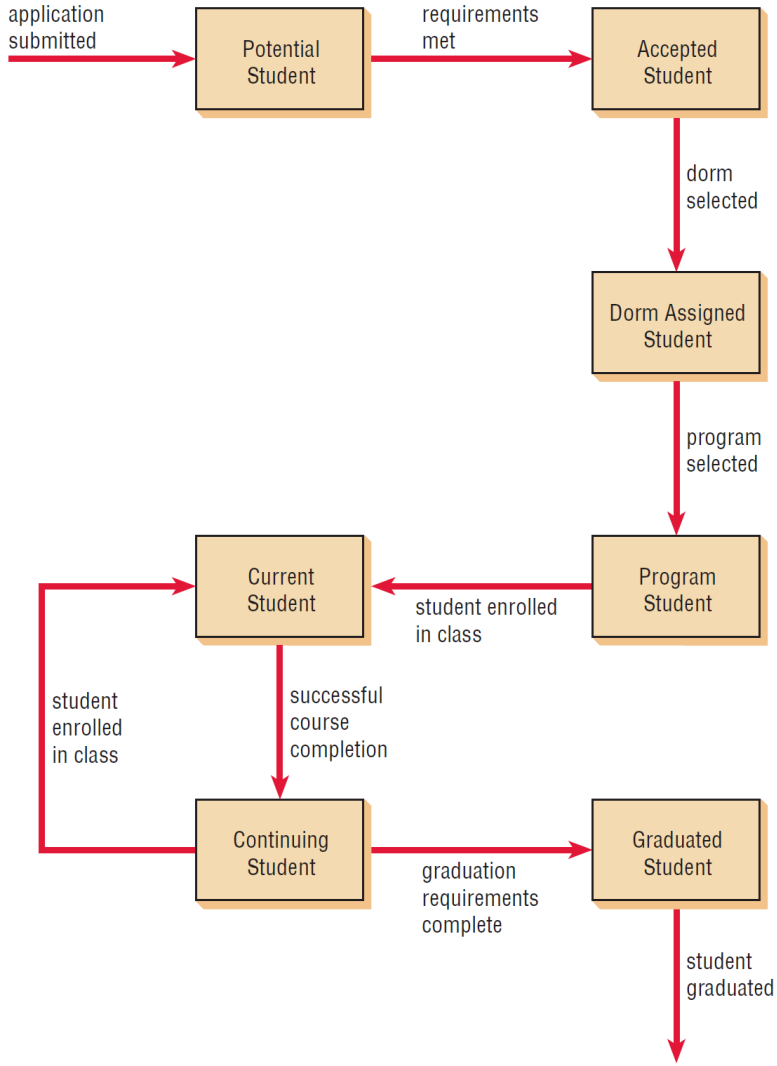
Statechart Diagrams

- Used to examine the different states that an object may have
- Created for a single class
  - Objects are created, go through changes, and are deleted or removed
- Objects
- States
- Events
  - Signals or asynchronous messages
  - Synchronous
  - Temporal events

Statechart Diagrams (continued)

- Created when:
  - A class has a complex life cycle
  - An instance of a class may update its attributes in a number of ways through the life cycle
  - A class has an operational life cycle
  - Two classes depend on each other
  - The object's current behavior depends on what happened previously

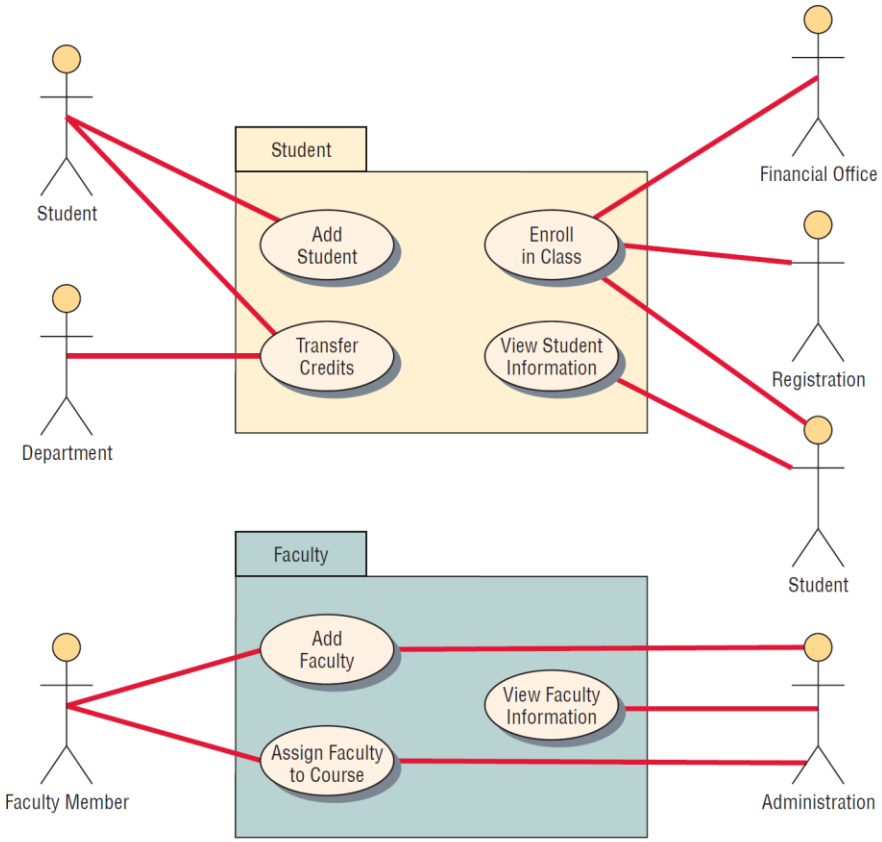
A Statechart Diagram Showing How a Student Progresses from a Potential Student to a Graduated Student (Figure 10.22)



Packages

- Containers for other UML things
- Show system partitioning
- Can be component packages
- Can be physical subsystems
- Use a folder symbol
- May have relationships

Use Cases Can Be Grouped into Packages (Figure 10.23)



Putting UML to Work

The steps used in UML are:

- Define the use case model
- Continue UML diagramming to model the system during the systems analysis phase
- Develop the class diagrams
- Draw statechart diagrams
- Begin systems design by refining the UML diagrams
- Document your system design in detail

Summary

- Object-oriented systems
  - Objects
  - Classes
  - Inheritance
- CRC cards
- UML and use case modeling
- Components of UML
  - Things
  - Relationships
  - Diagrams

Summary (continued)

- UML diagrams
  - Use case diagrams
  - Activity diagrams
  - Sequence diagrams
  - Communication diagrams
  - Class diagrams
  - Statechart diagrams
- Using UML