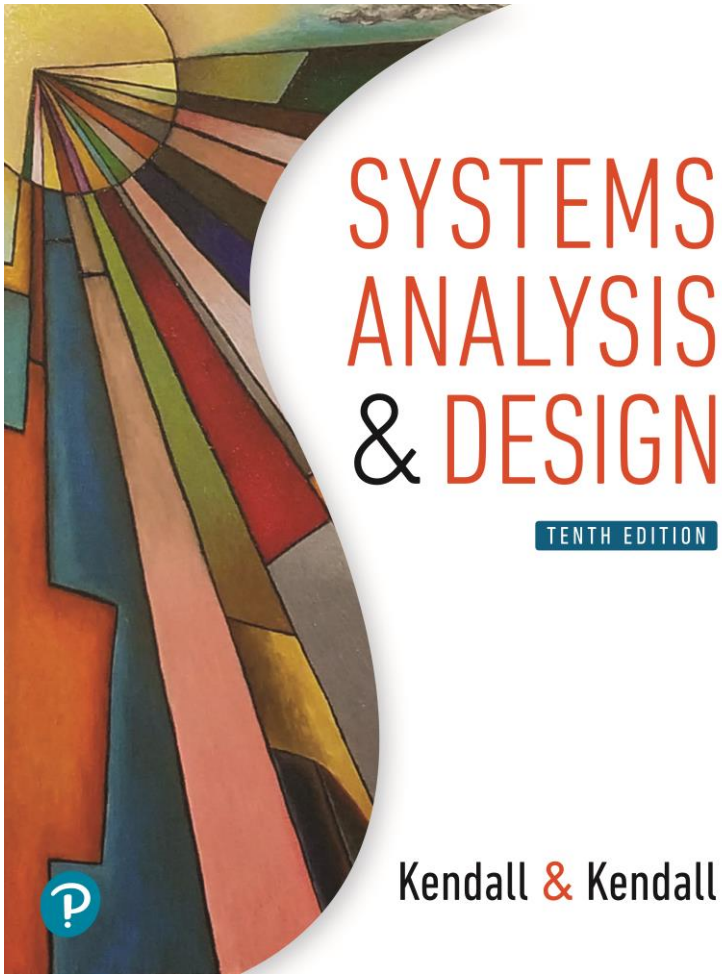
Systems Analysis & Design

Tenth Edition



Chapter 10 Object-Oriented Systems Analysis and Design Using UML

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

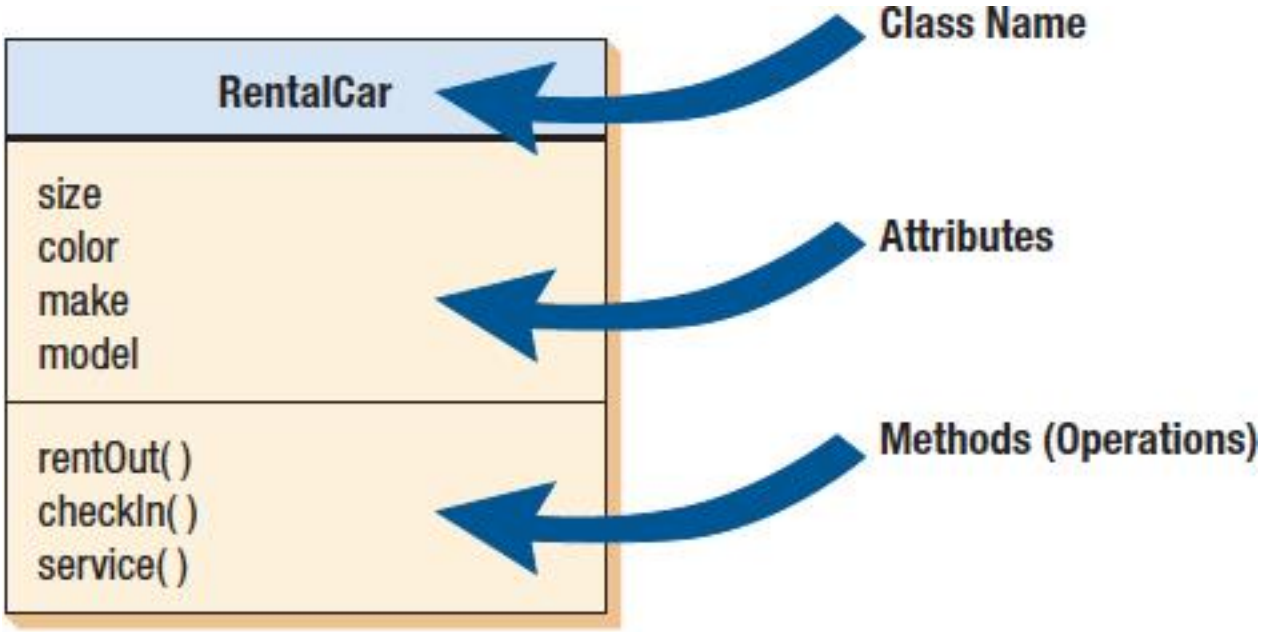
Objects

- 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

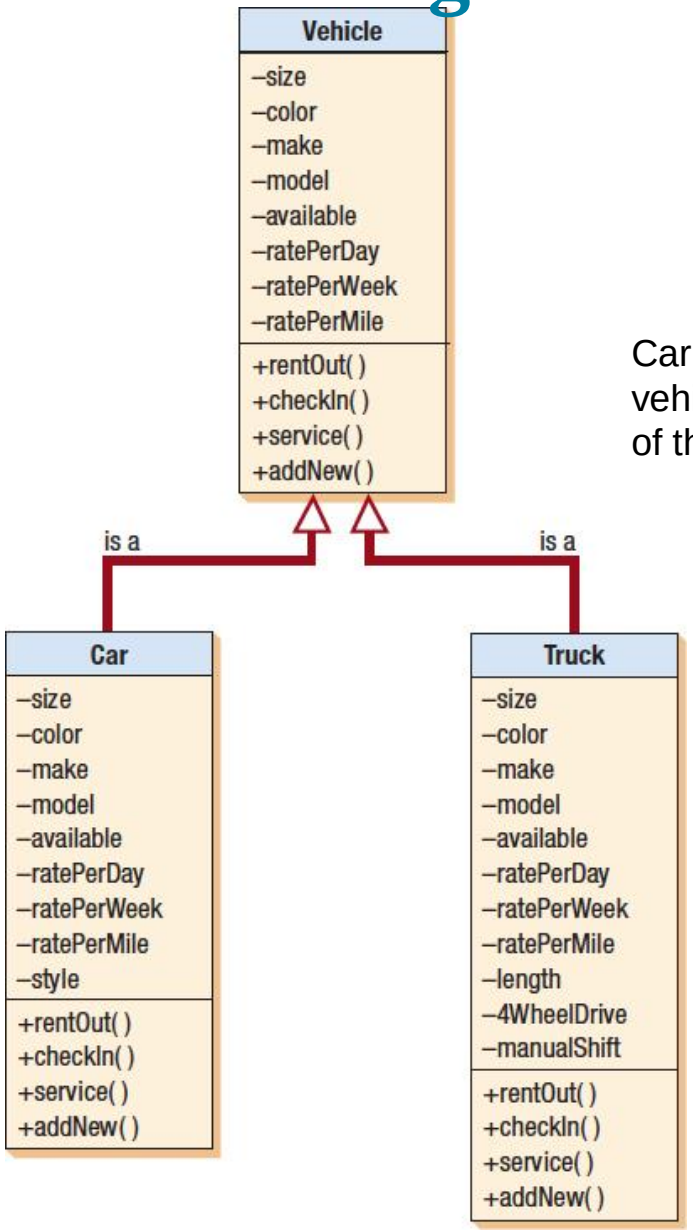
Figure 10.1 An Example of a UML Class:



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

Figure 10.2 A Class Diagram Showing Inheritance



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

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
- Behavioral diagrams
  - Used to describe the interaction between people (actors) and a use case (how the actors use the system)

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

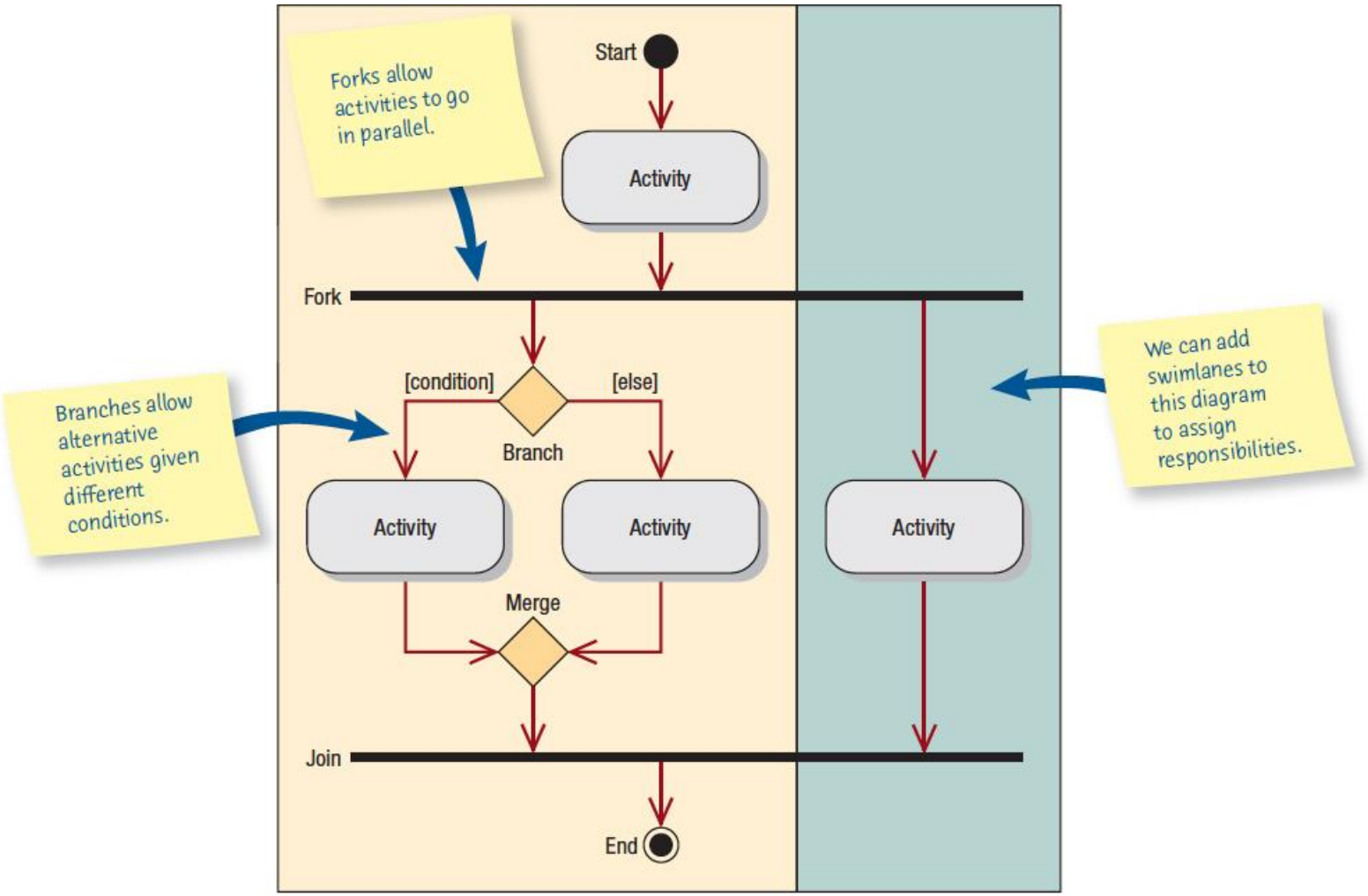
Commonly Used UML Diagrams (continued)

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

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

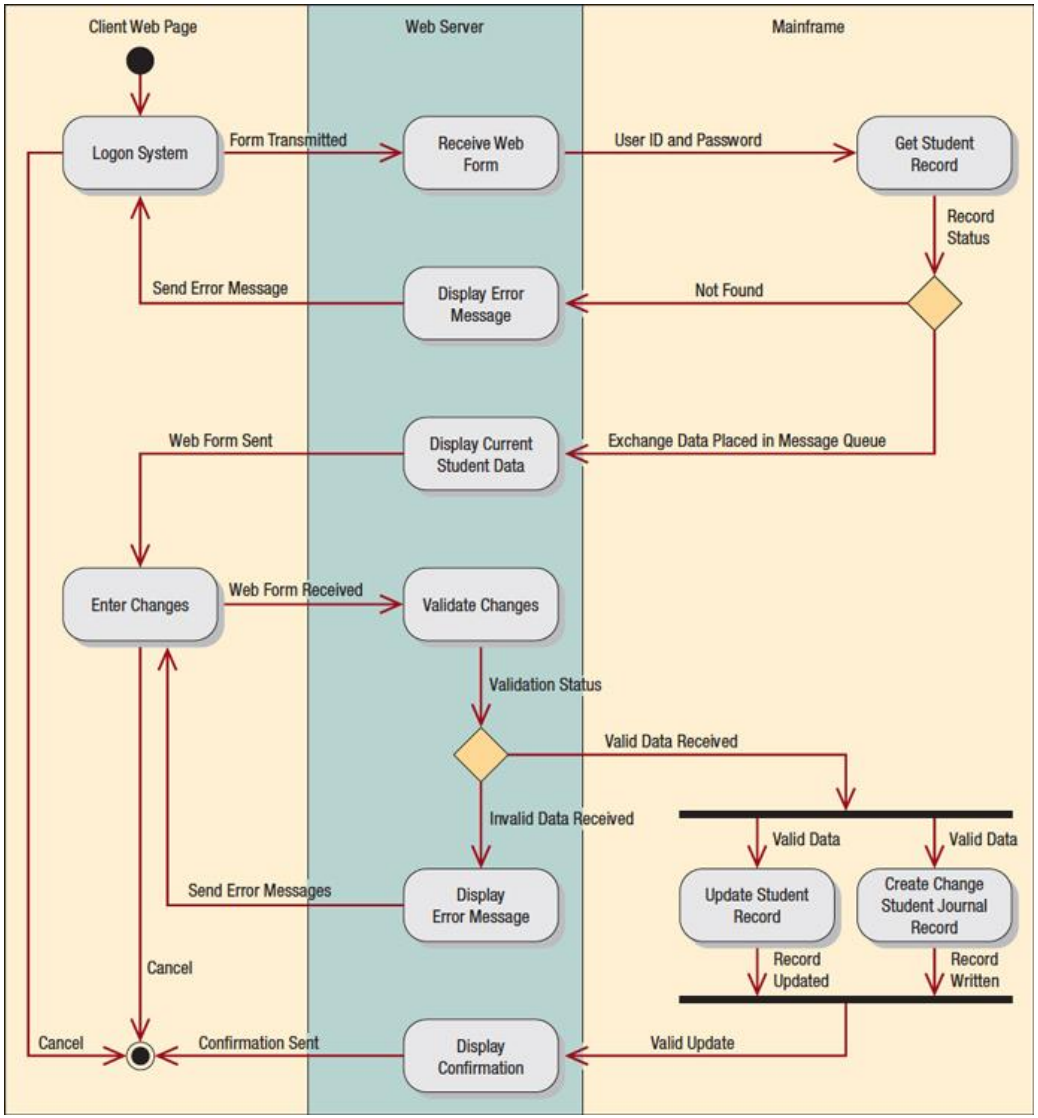
Figure 10.8 Specialized Symbols Are Used to Draw an Activity Diagram



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

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



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

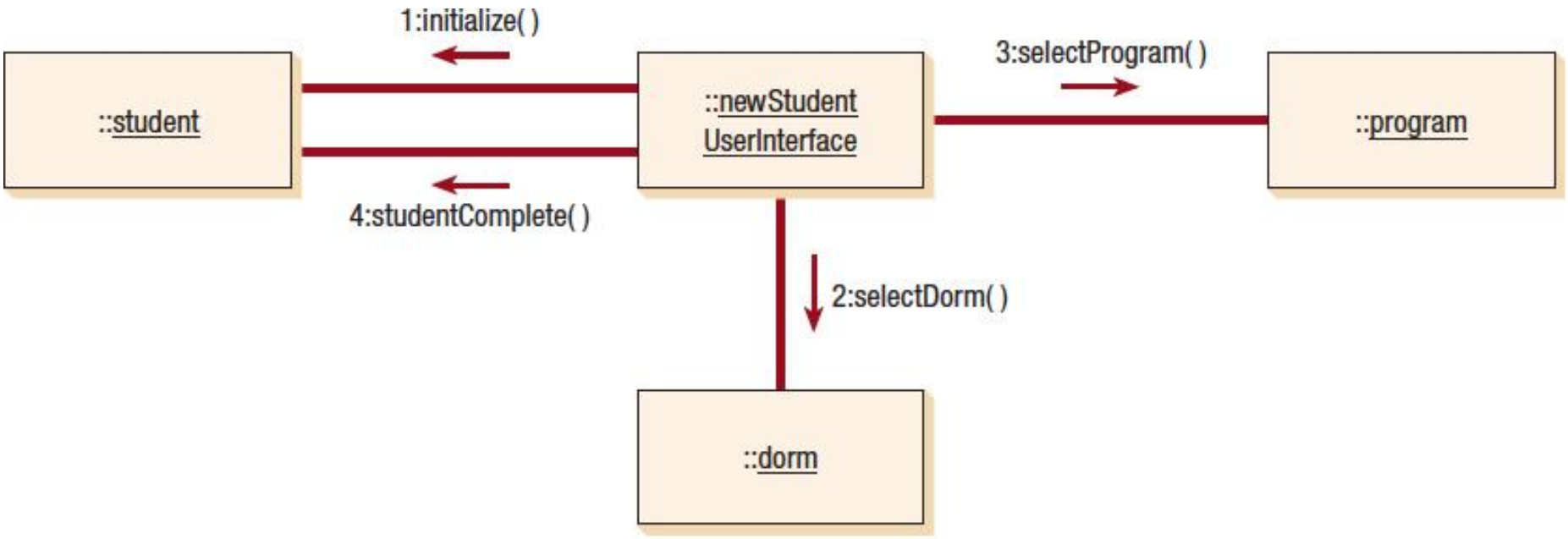
Sequence Diagrams

- 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

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

Figure 10.12 A Communication Diagram for Student Admission



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

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

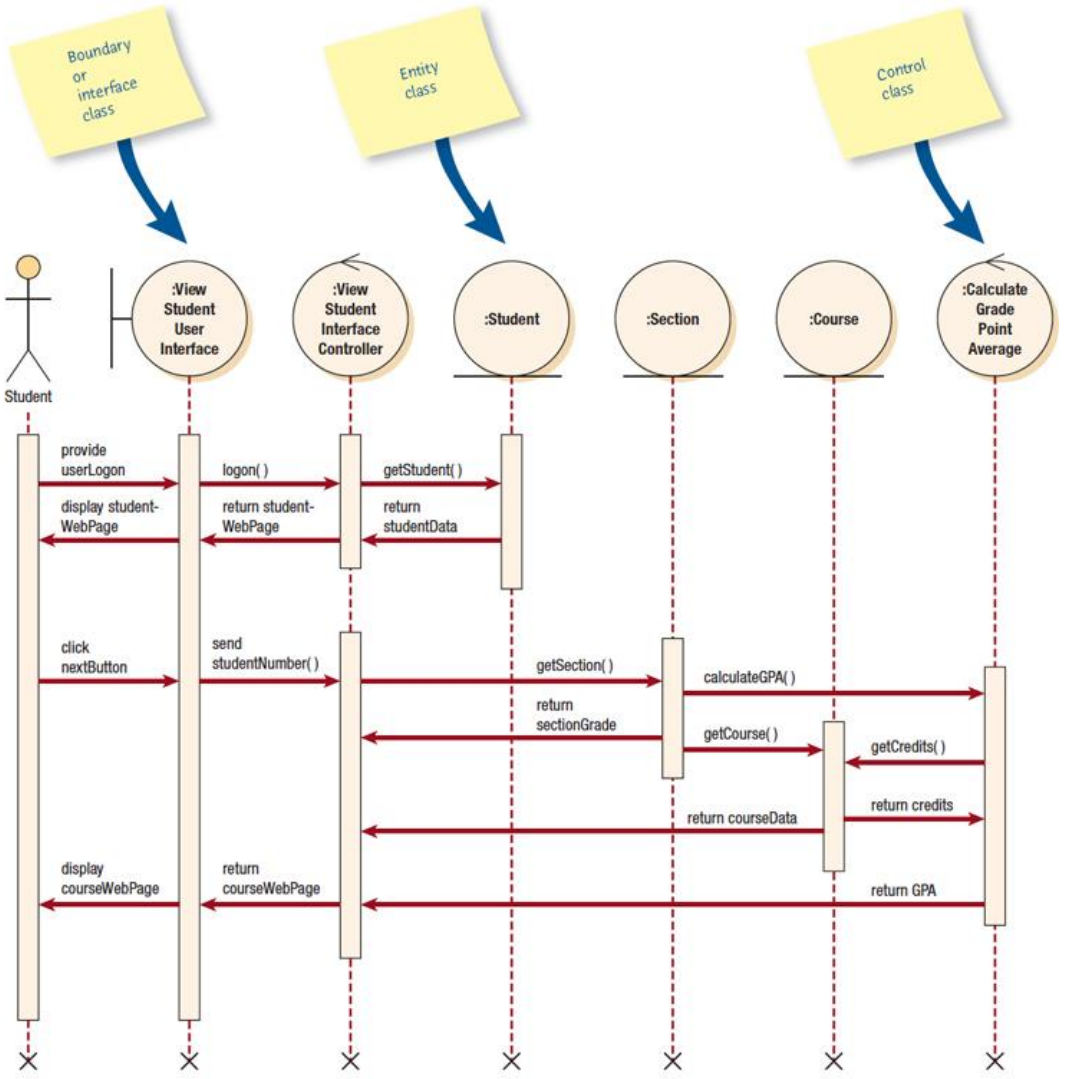
Presentation, Business, and Persistence Layers

- 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

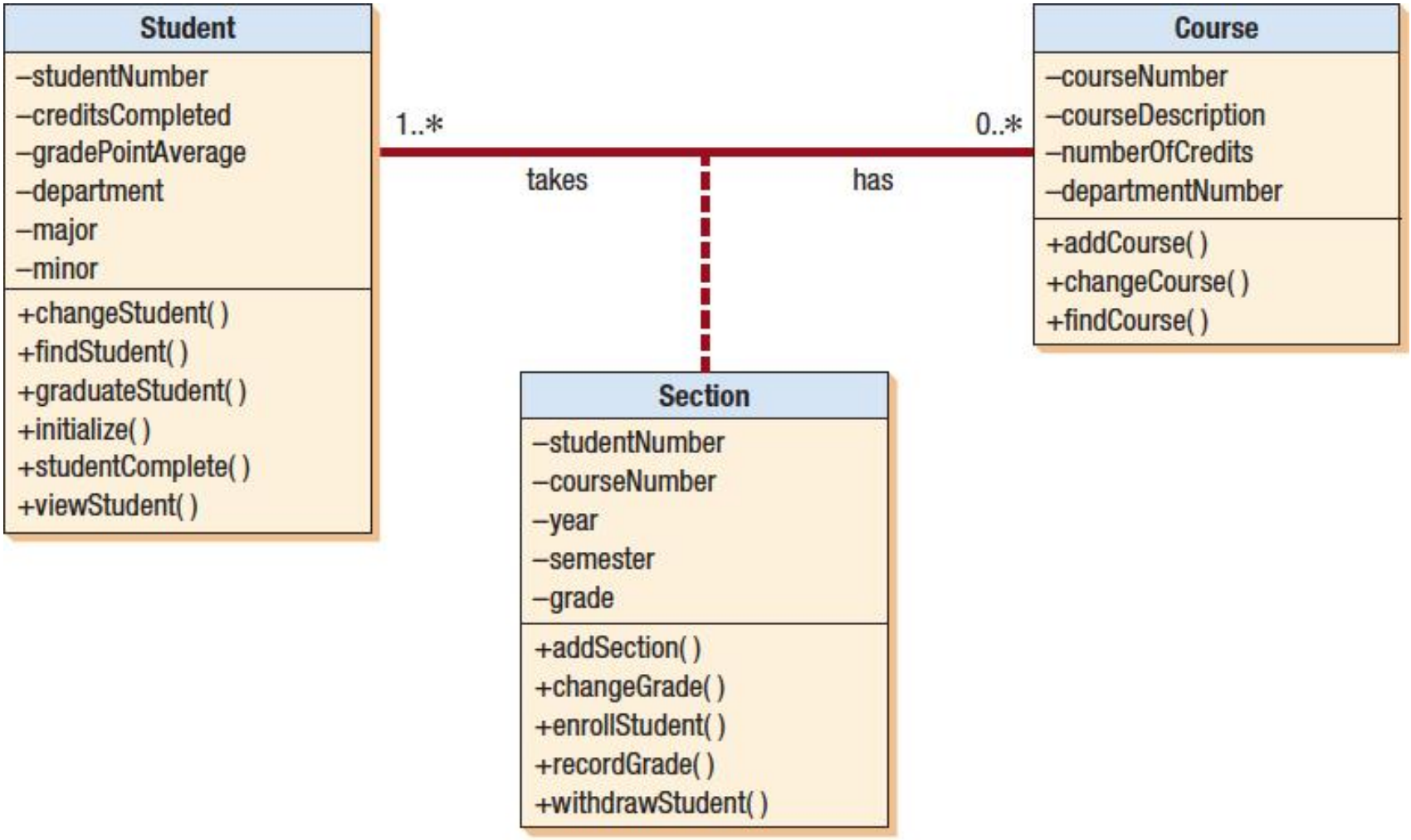
Figure 10.15 A Sequence Diagram for Using Two Web Pages: One for Student Information, One for Course Information



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

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



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

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

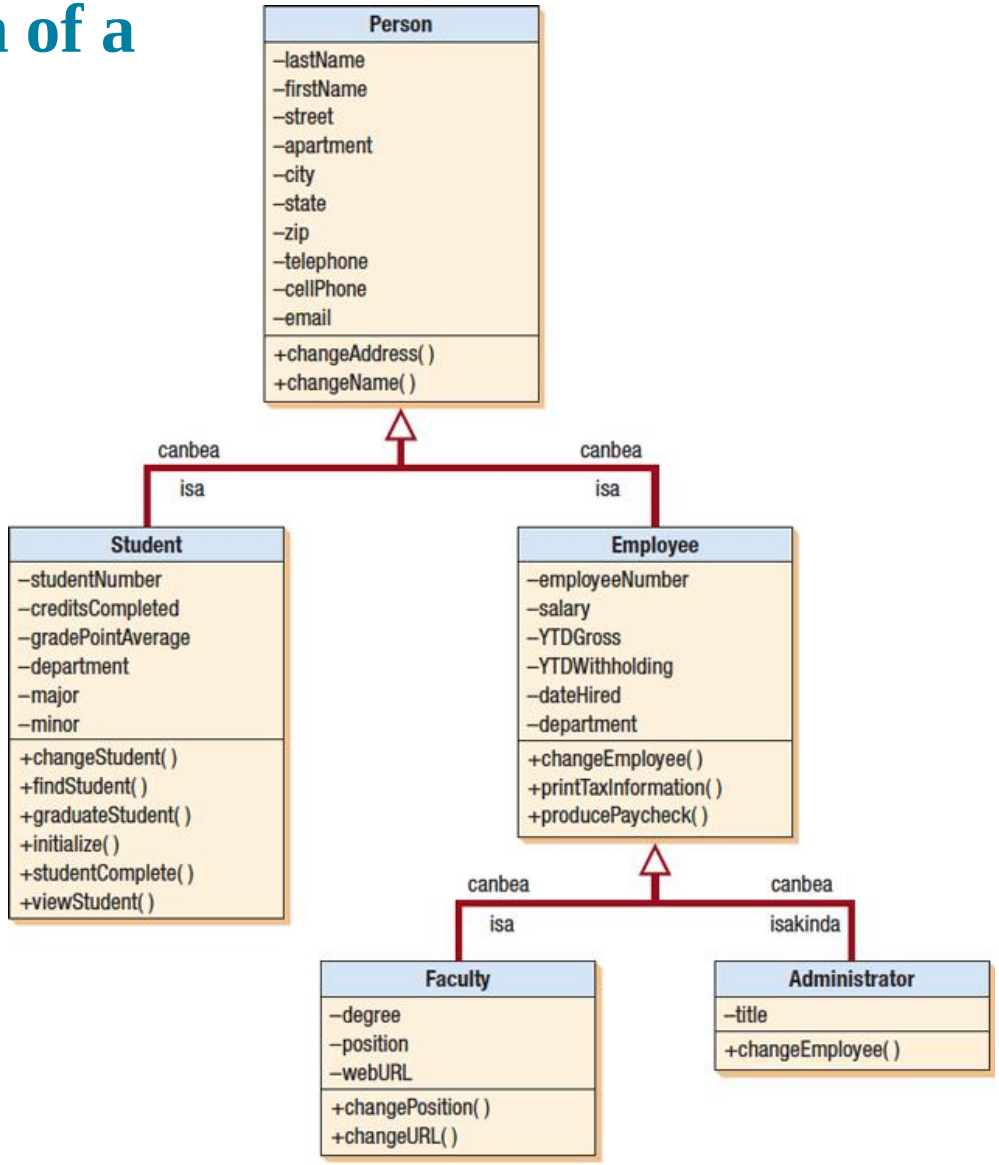
Polymorphism

- 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

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



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

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