Object-Oriented Systems Analysis and Design Using UML

10

Systems Analysis and Design, Kendall & Kendall

Object-Oriented Analysis and Design

- Works well in situations where complicated systems are undergoing continuous maintenance, adaptation, and design
- Objects, classes and reusability
- 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
- Unified Modeling Language
- Use case and other UML diagrams
- Packages
- Using UML

Object-Oriented Concepts

- Objects
- Classes
- Inheritance

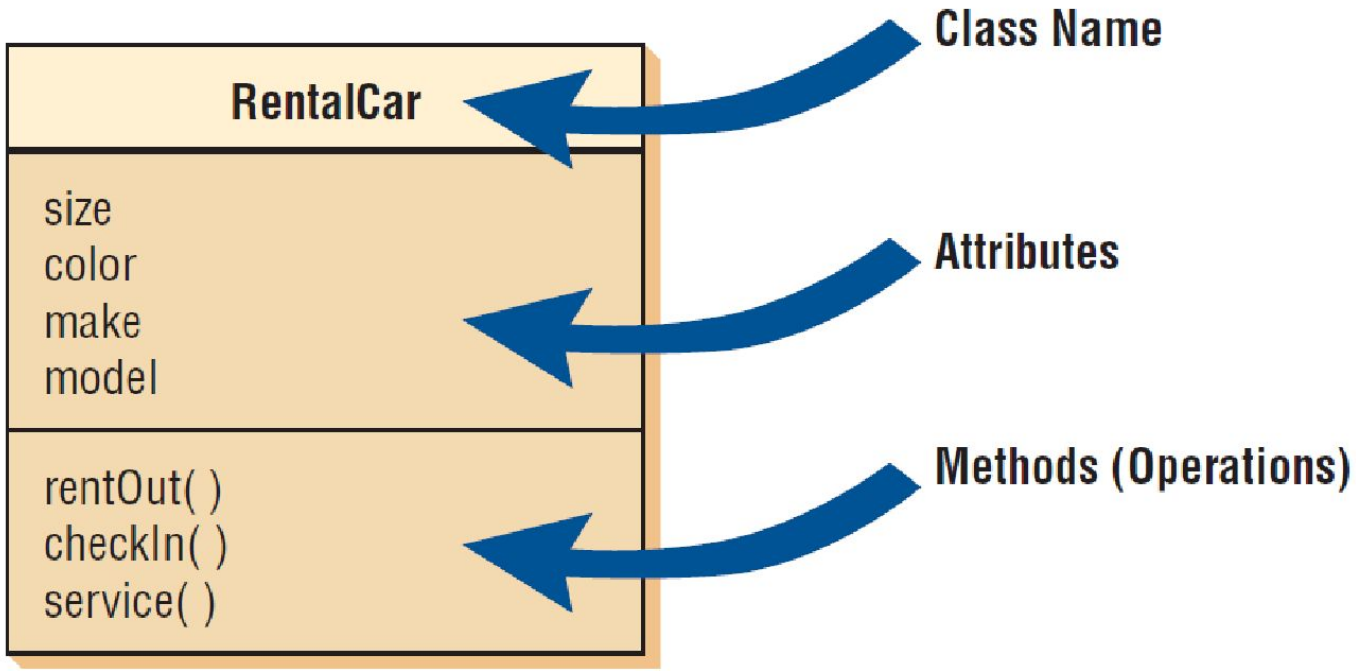
Objects

- 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 attributes 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

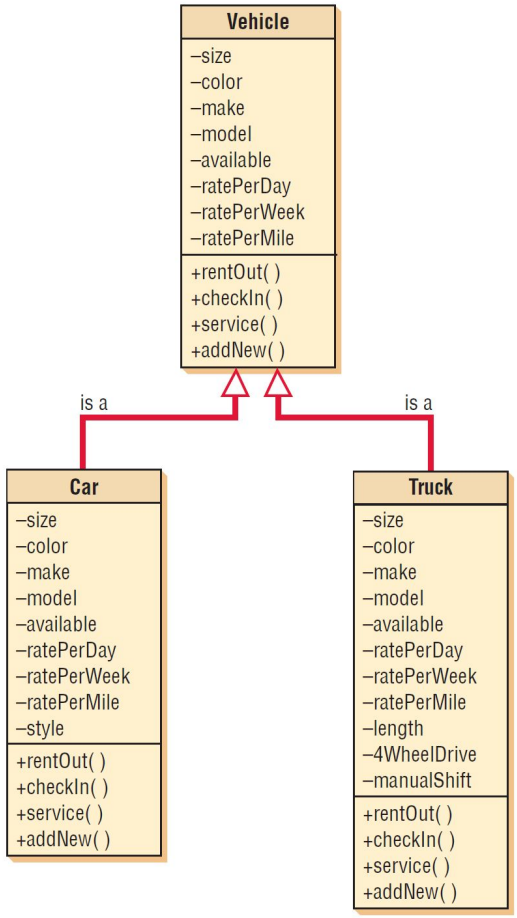
Figure 18.1 An example of a UML class. A class is depicted as a rectangle consisting of the class name, attributes, and methods



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 18.2 A class diagram showing inheritance. 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

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

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		

Interacting during a CRC Session

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

The Unified Modeling Language (UML) Concepts and Diagrams

- Things
- Relationships
- Diagrams

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

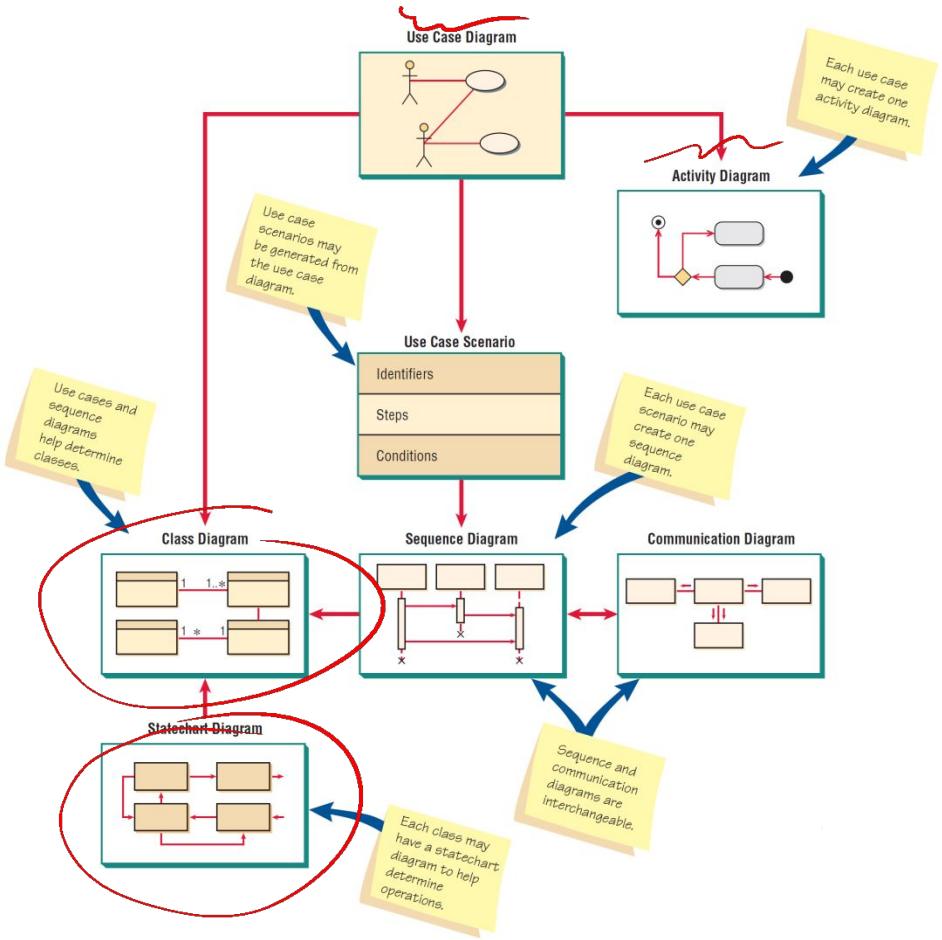
flowchart

- 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

Figure 18.5 An overview of UML diagrams showing how each diagram leads to the development of other UML diagrams



The Unified Modeling Language (UML) Concepts and Diagrams

- widely used method of visualizing and documenting software systems design
- means of communication between the development team and the business team on a project.

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
- A requirements analysis concept
- Describes the system's actions from a the point of view of a user

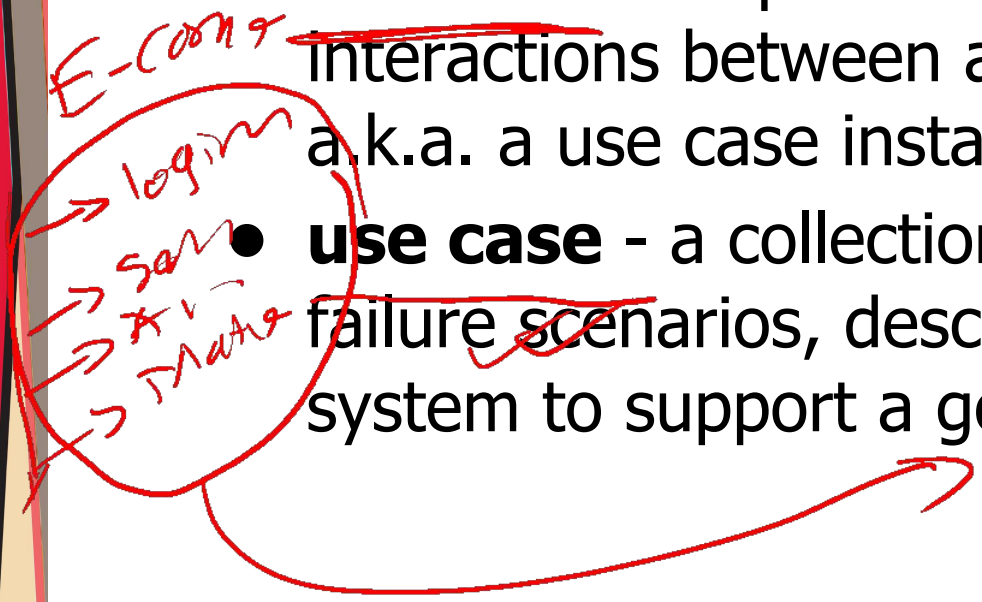
* complex details

Types of Use Case Modeling

- Textual or tabular descriptions
- Informal User Case
- Diagrams

Use Case Descriptions

- **actors** - something with a behavior or role, e.g., a person, another system
- **scenario** - a specific sequence of actions and interactions between actors and the system, a.k.a. a use case instance
- **use case** - a collection of related success and failure scenarios, describing actors using the system to support a goal.



Pay for job posting example

Use Case Title: Pay for a job posting
Primary Actor: Recruiter
Level: Actor goal
Precondition: The job information has been entered but is not viewable.
Minimal Guarantees: None
Success Guarantees: Job is posted; recruiter's credit card is charged.
Main Success Scenario:

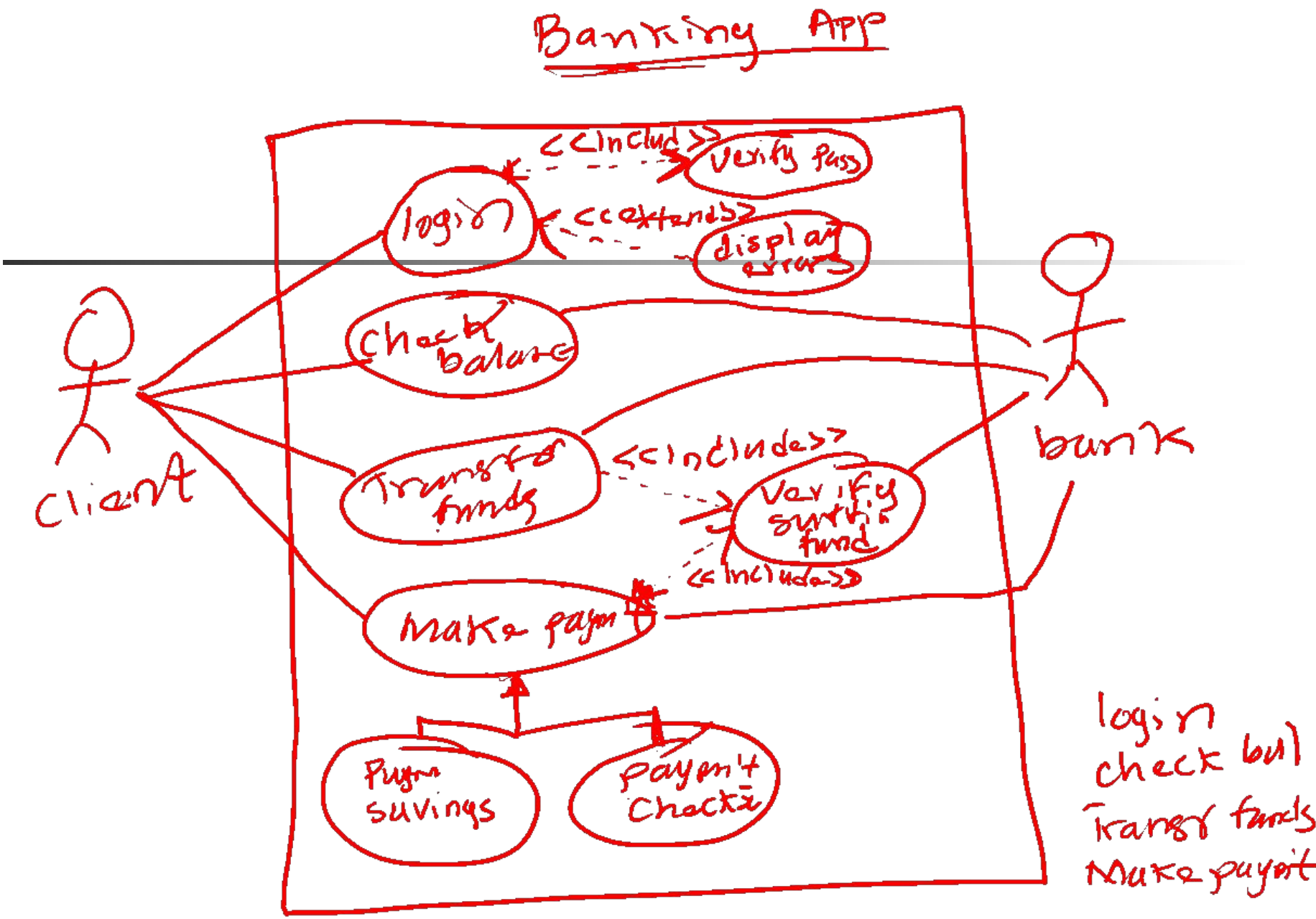
1. Recruiter submits credit card number, date, and authentication information.
2. System validates credit card.
3. System charges credit card full amount.
4. Job posting is made viewable to Job Seekers.
5. Recruiter is given a unique confirmation number.

Extensions:

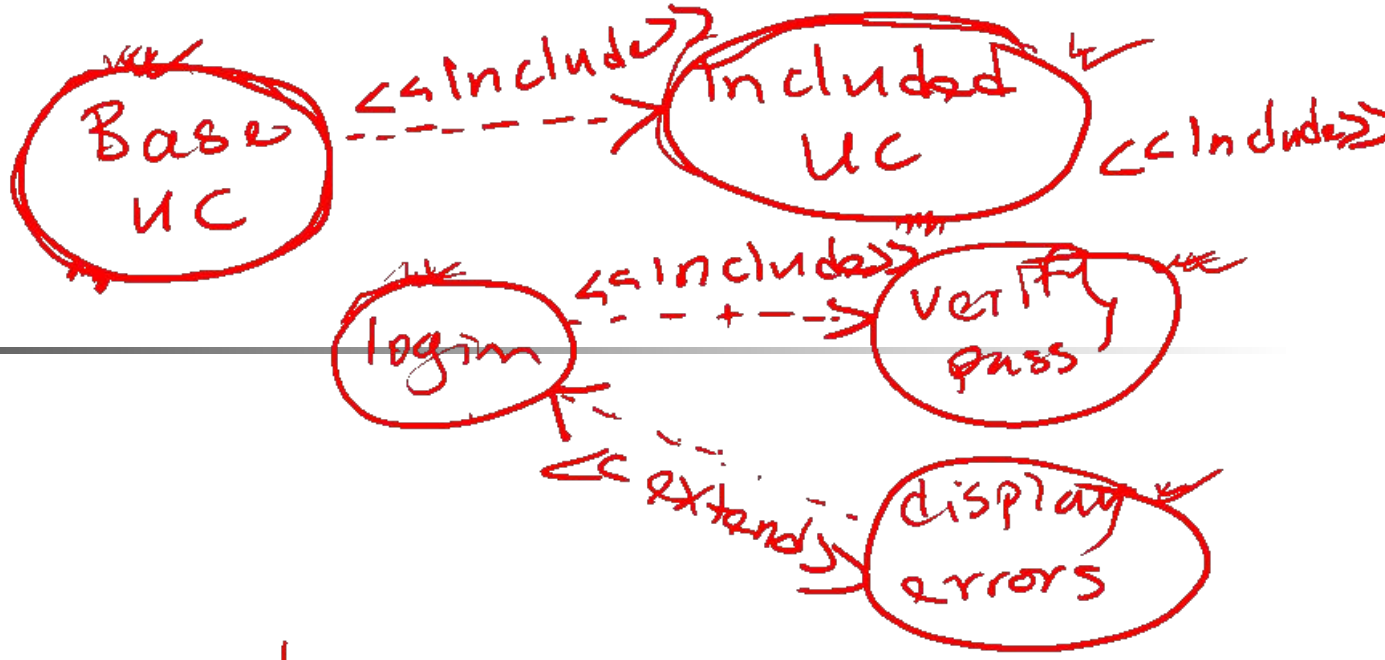
2a: The card is not of a type accepted by the system:
    2a1: The system notifies the user to use a different card.
2b: The card is expired:
    2b1: The system notifies the user to use a different card.
2c: The card is expired:
    2c1: The system notifies the user to use a different card.
3a: The card has insufficient available credit to post the ad.
    3a1: The system charges as much as it can to the current credit card.
    3a2: The user is told about the problem and asked to enter a second credit card for the remaining charge. The use case continues at Step 2.

Use Case Diagrams

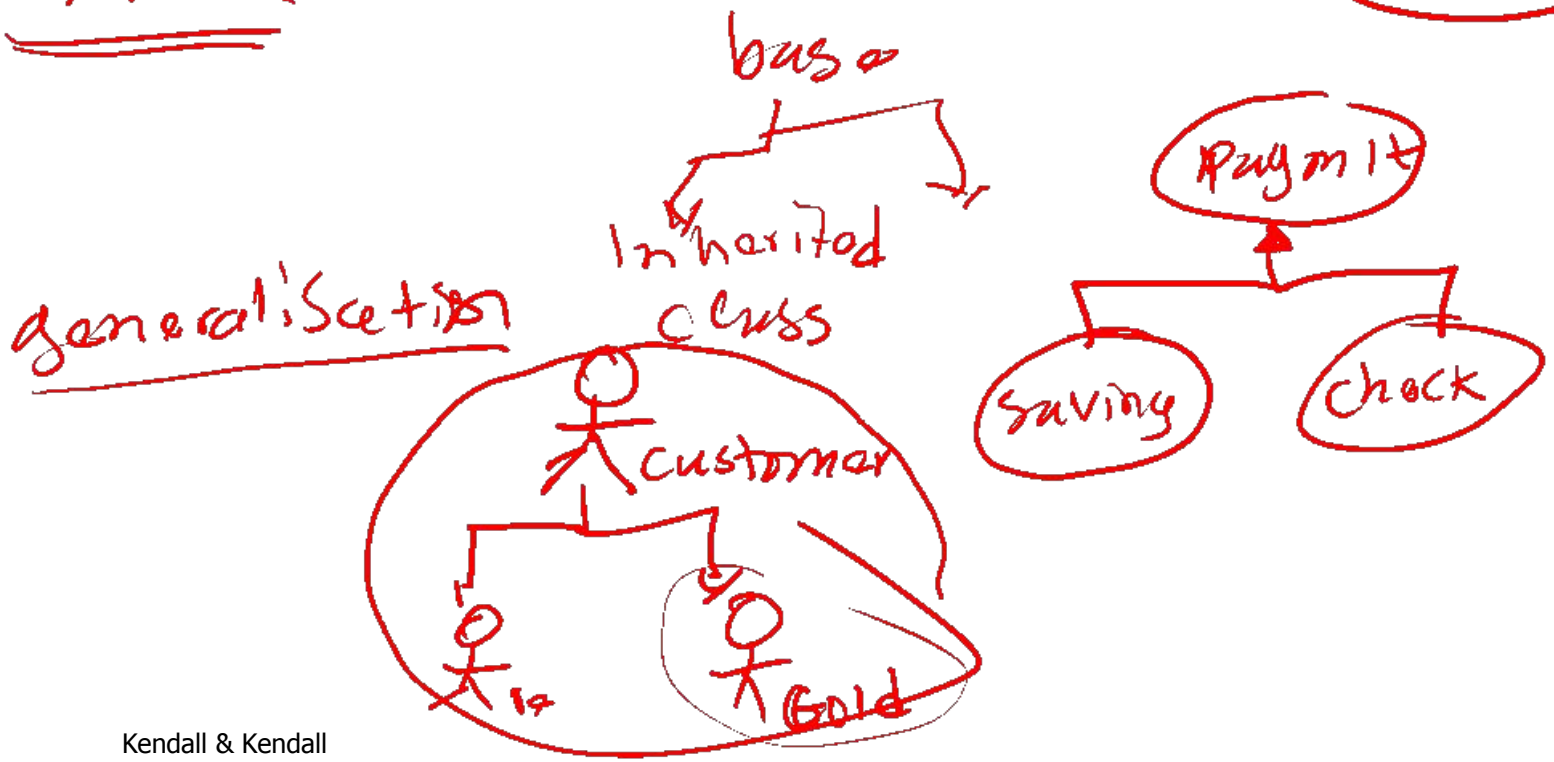
- Actor
- Use Case
- Boundary of system
- Relationship

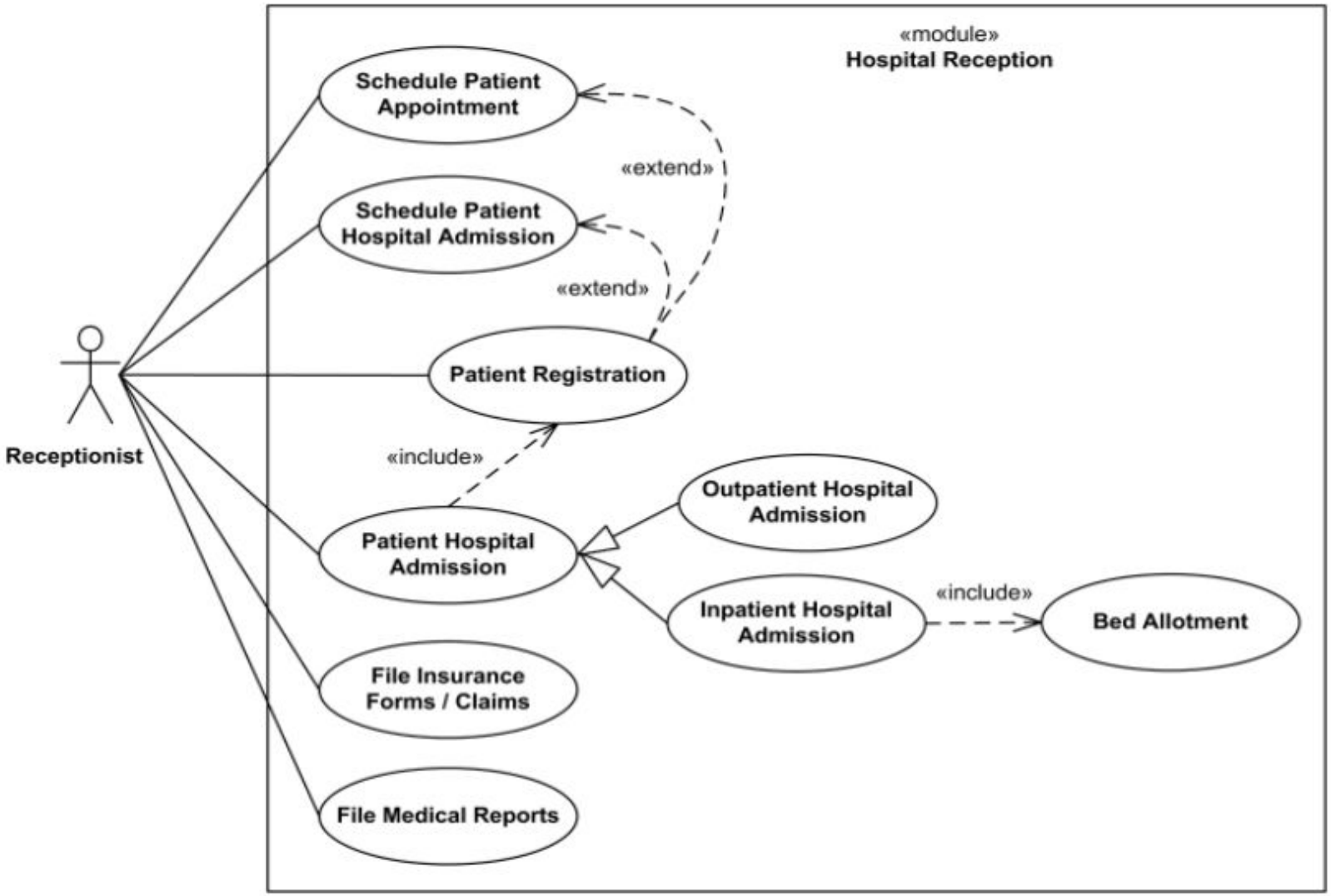

Banking App
<<Include>>
Verify pass
login
<<extends>>
display errors
Check balance
client
bank
Transfer funds
<<include>>
Verify sufficient fund
<<Include>>
Make payment
Pay in savings
payment Checking
login
check bal
Transfer funds
Make payment

Include



extend




«module»
Hospital Reception
Schedule Patient Appointment
«extend»
Schedule Patient Hospital Admission
«extend»
Patient Registration
Receptionist
«include»
Outpatient Hospital Admission
Patient Hospital Admission
Inpatient Hospital Admission
«include»
Bed Allotment
File Insurance Forms / Claims
File Medical Reports

A use case diagram for a university registration system

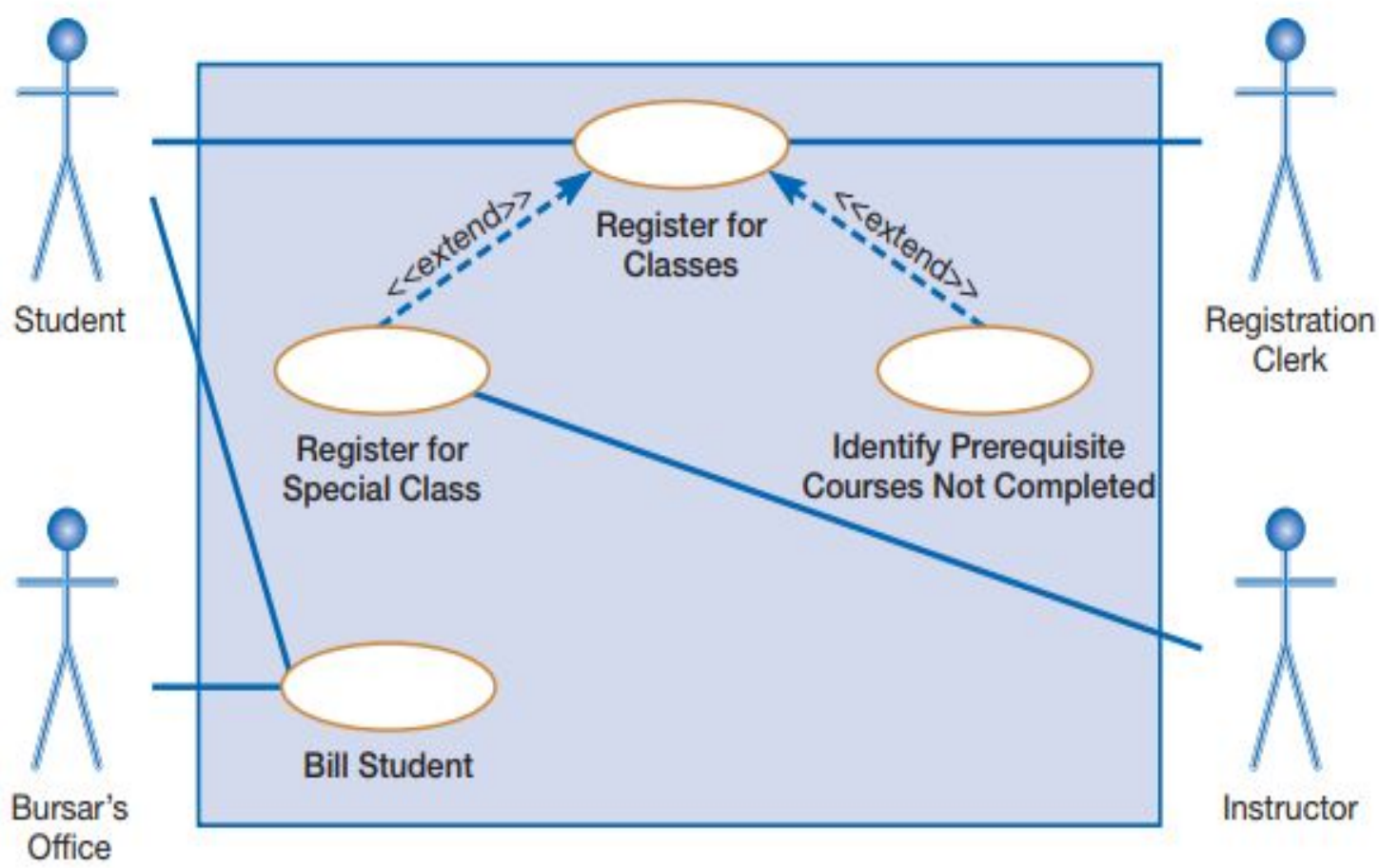
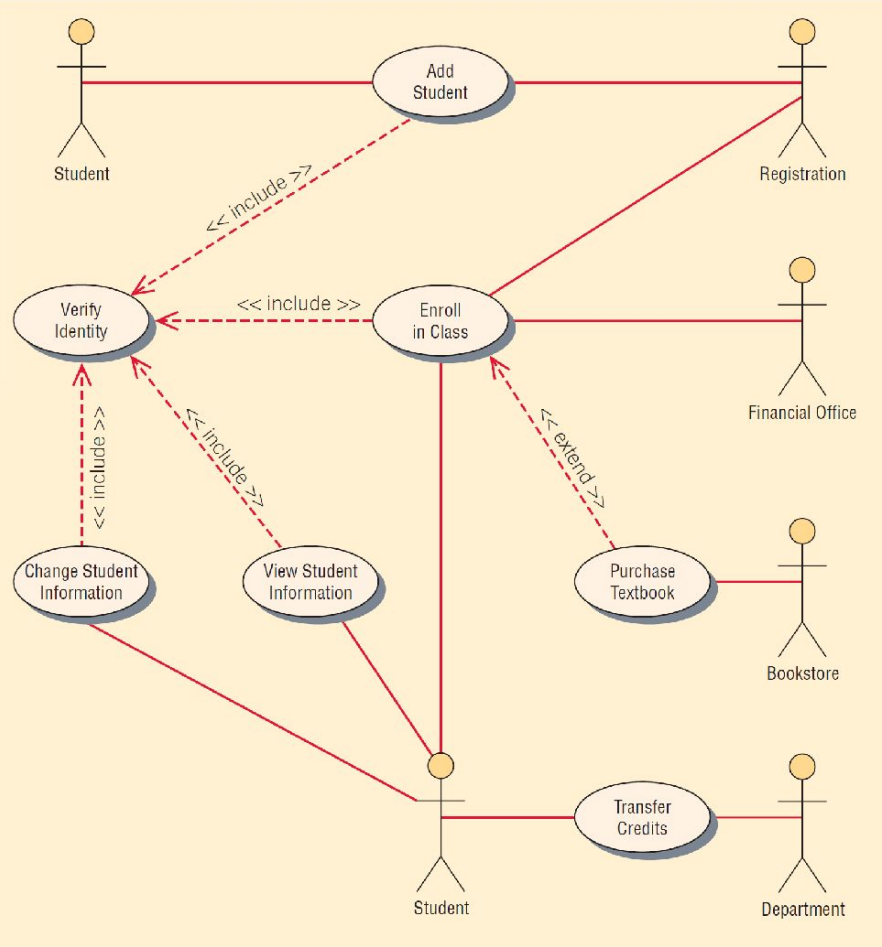
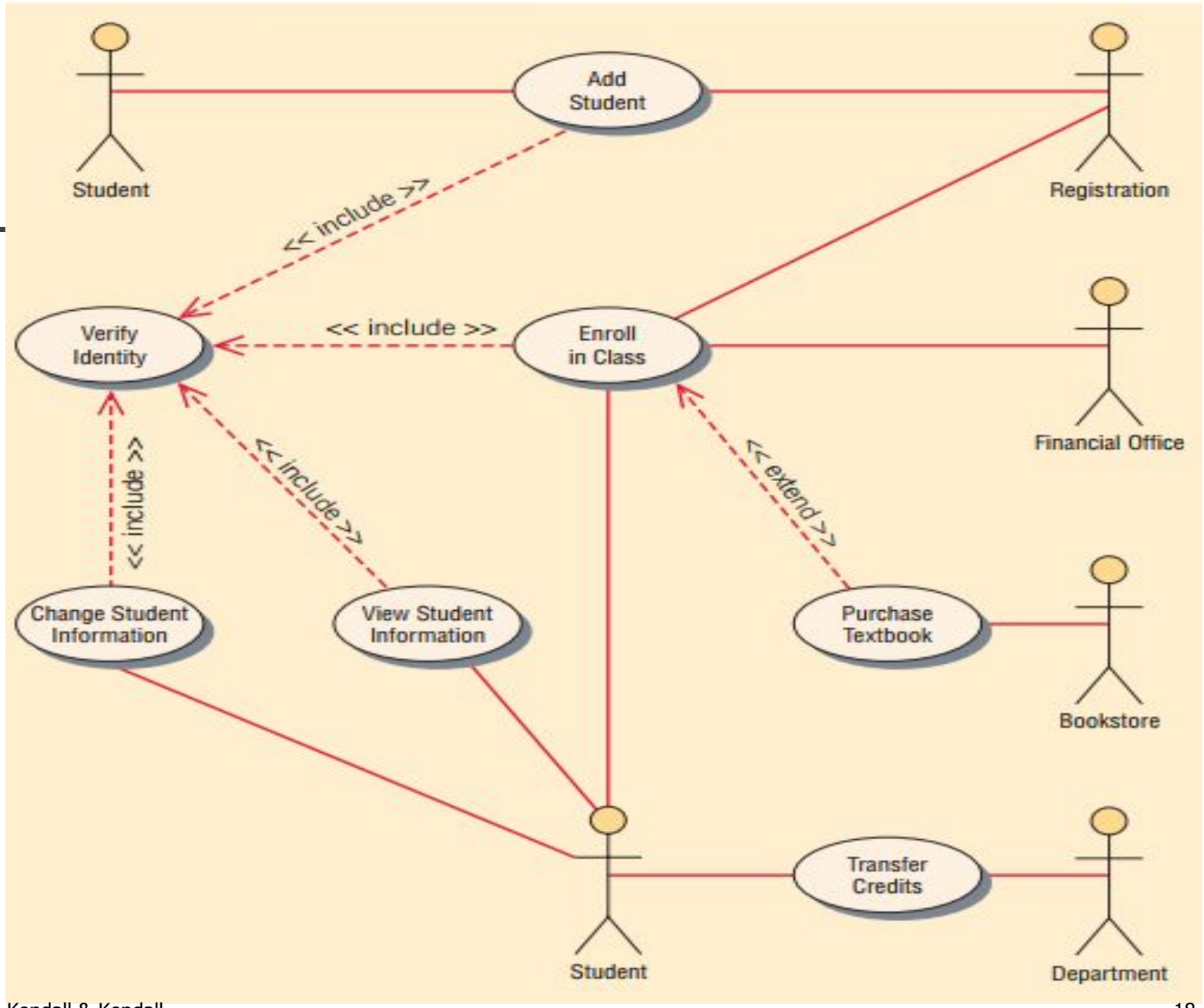


Figure 18.6 A use case example of student enrollment

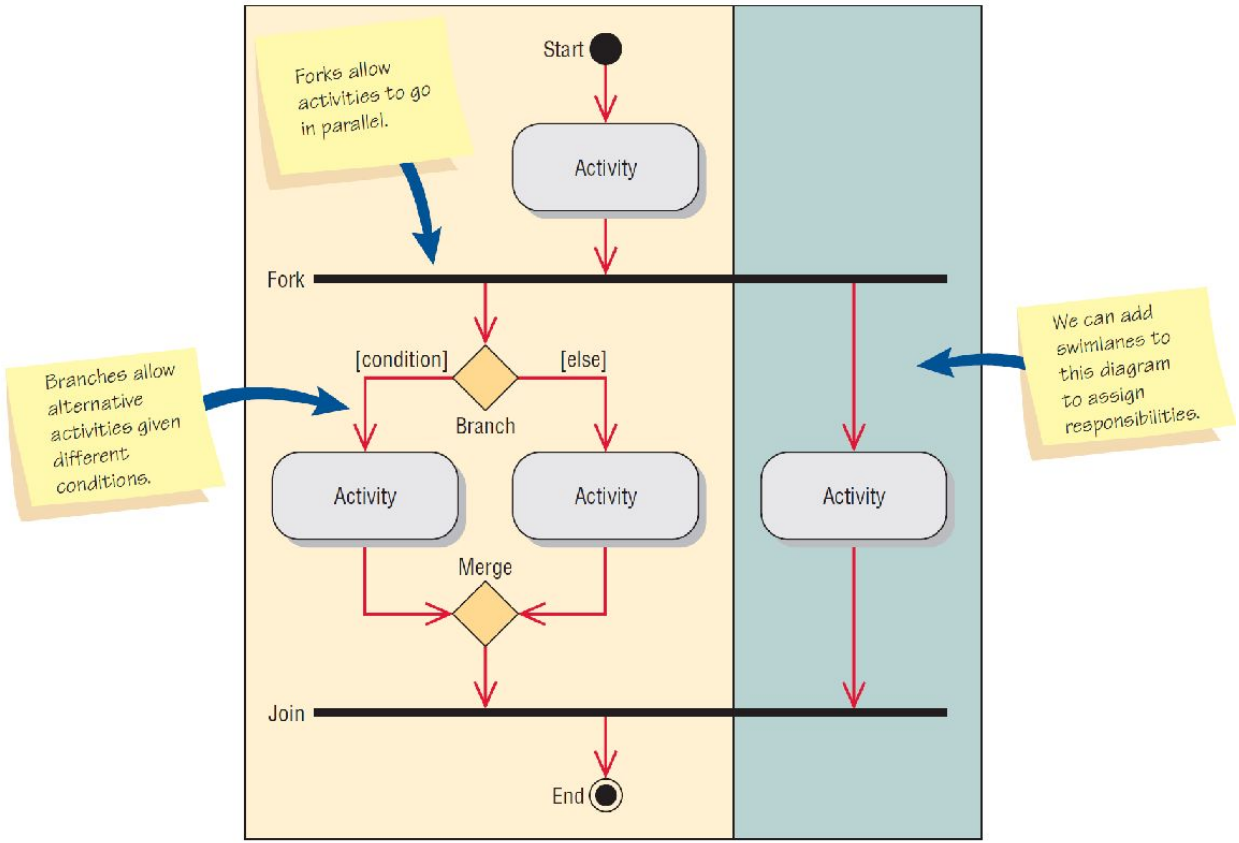



Add Student
Student
Registration
<< include >>
Verify Identity
<< include >>
Enroll in Class
Financial Office
<< include >>
<< include >>
<< extend >>
Change Student Information
View Student Information
Purchase Textbook
Bookstore
Transfer Credits
Student
Department

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 18.8 Specialized symbols are used to draw an activity diagram



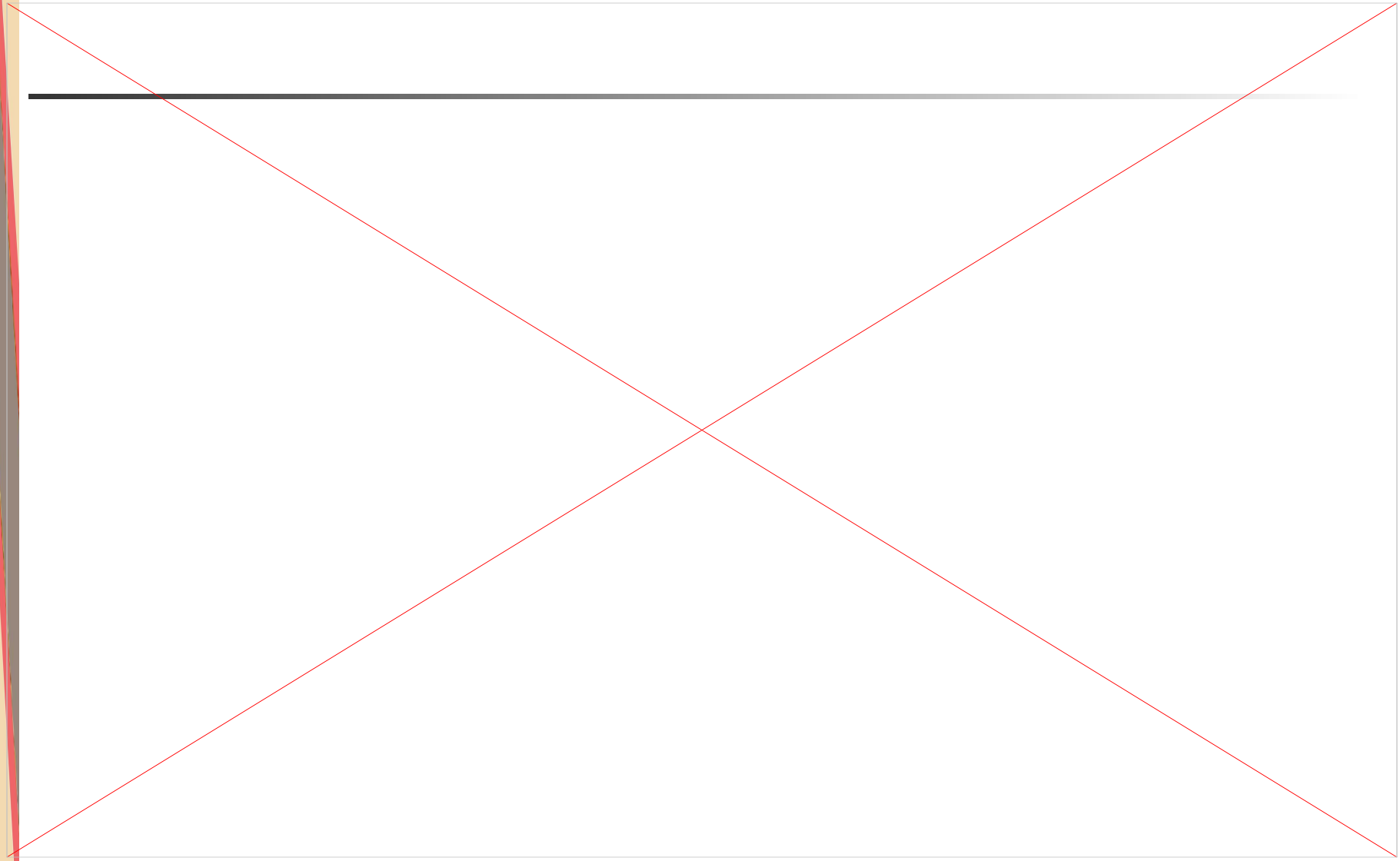
Creating Activity Diagrams

- 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



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

Sequence Diagrams

- presented either in generic form or in an instance form.
- Alternatives

Figure 18.10 Specialized symbols used to draw a Sequence Diagram

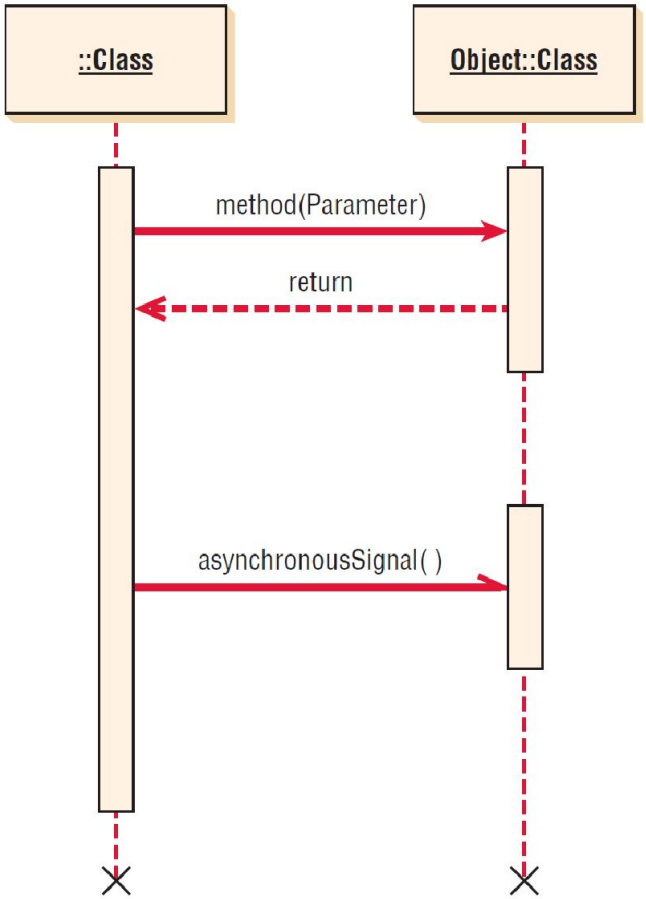
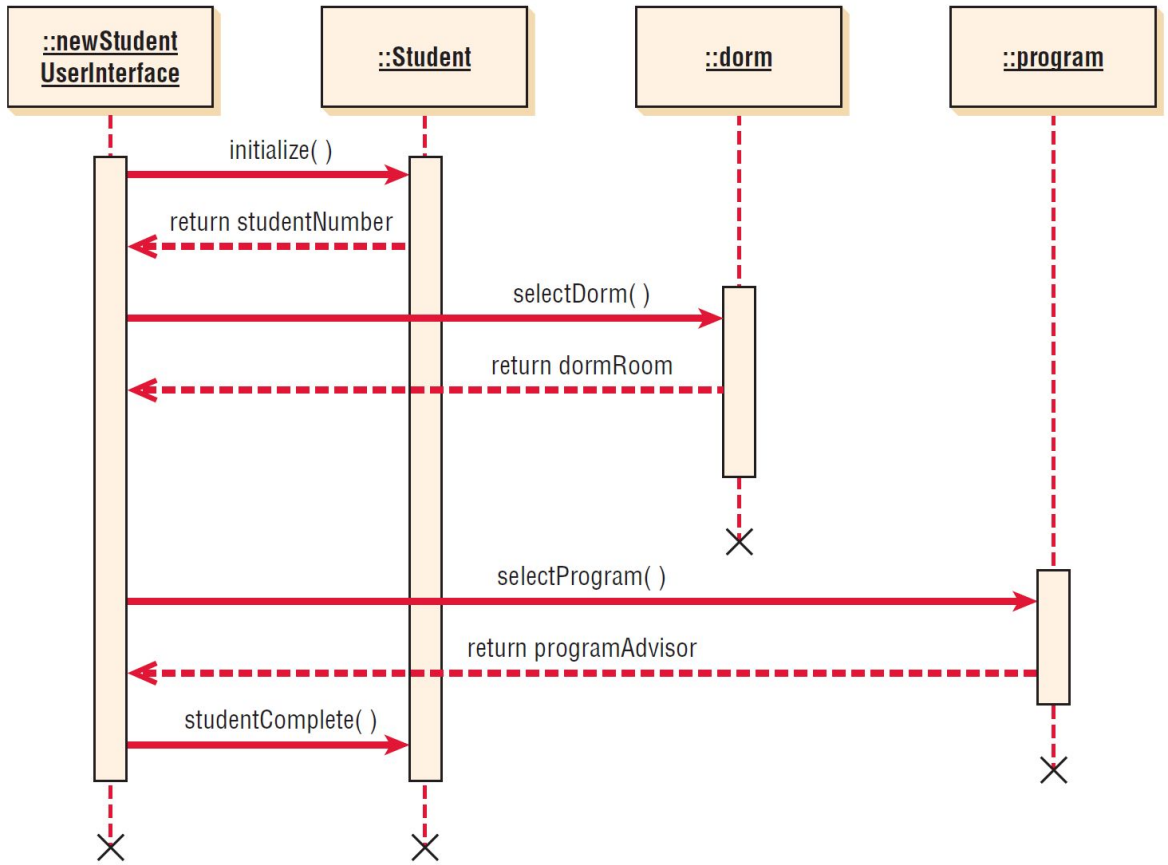
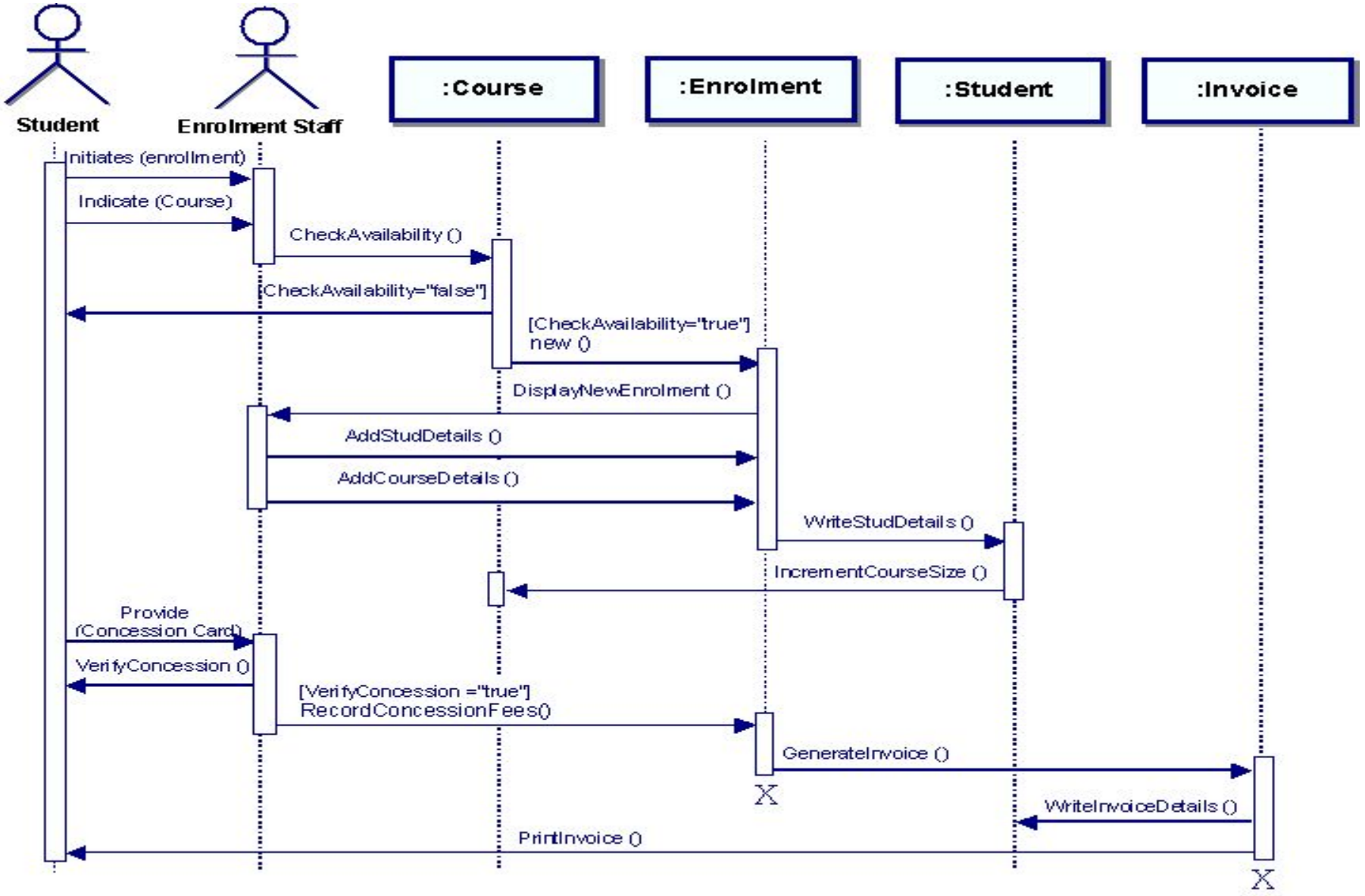


Figure 18.11 A sequence diagram for student admission. Sequence diagrams emphasize the time ordering of messages




Student
Enrolment Staff
:Course
:Enrolment
:Student
:Invoice
Initiates (enrolment)
Indicate (Course)
CheckAvailability ()
[CheckAvailability="false"]
[CheckAvailability='true']
new ()
DisplayNewEnrolment ()
AddStudDetails ()
AddCourseDetails ()
WriteStudDetails ()
IncrementCourseSize ()
Provide
(Concession Card)
VerifyConcession ()
[VerifyConcession ="true"]
RecordConcessionFees()
GenerateInvoice ()
WriteInvoiceDetails ()
PrintInvoice ()

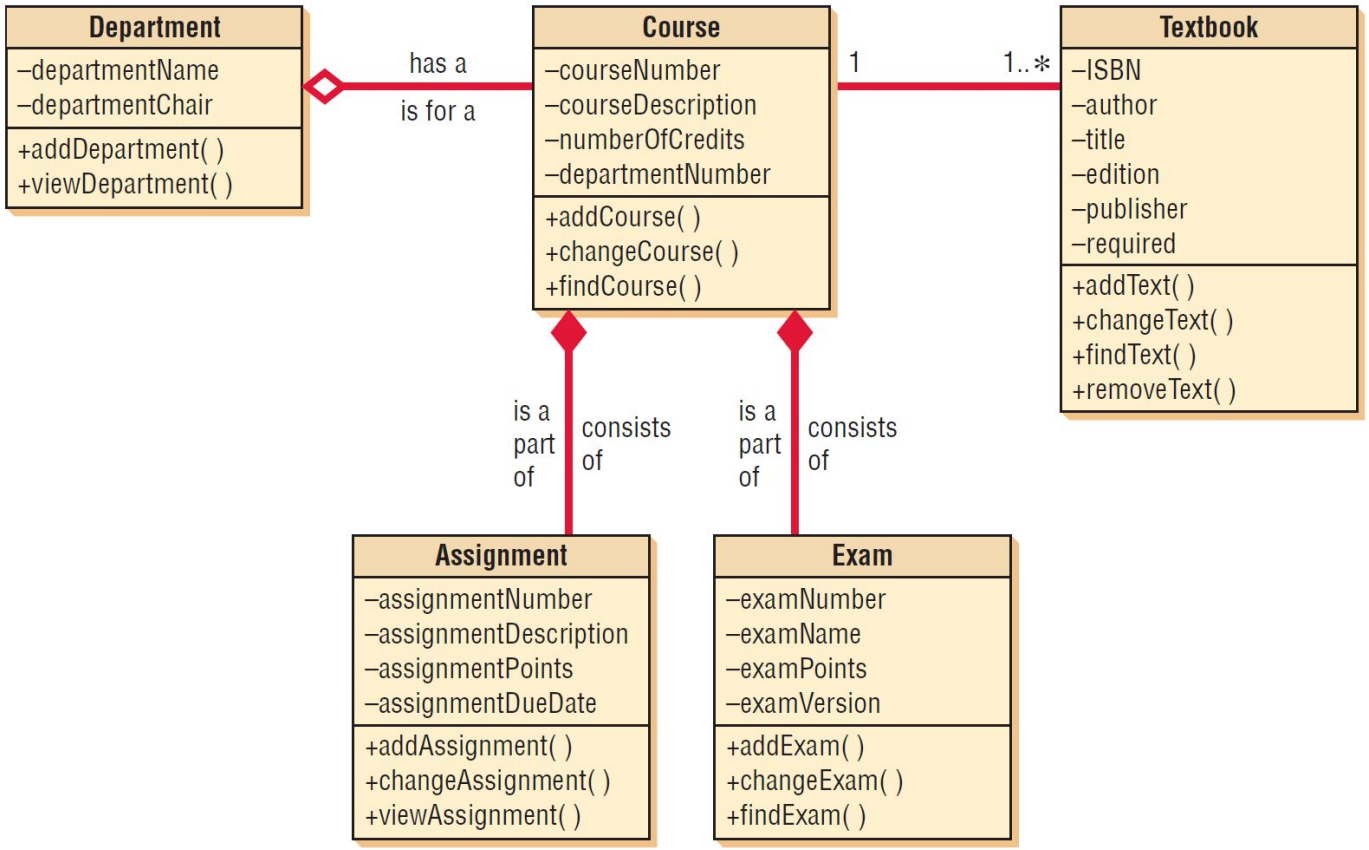
Class Diagrams

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

Class Diagrams (Continued)

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

Figure 18.13 A class diagram for course offerings. The filled-in diamonds show aggregation and the empty diamond shows a whole-part relationship



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

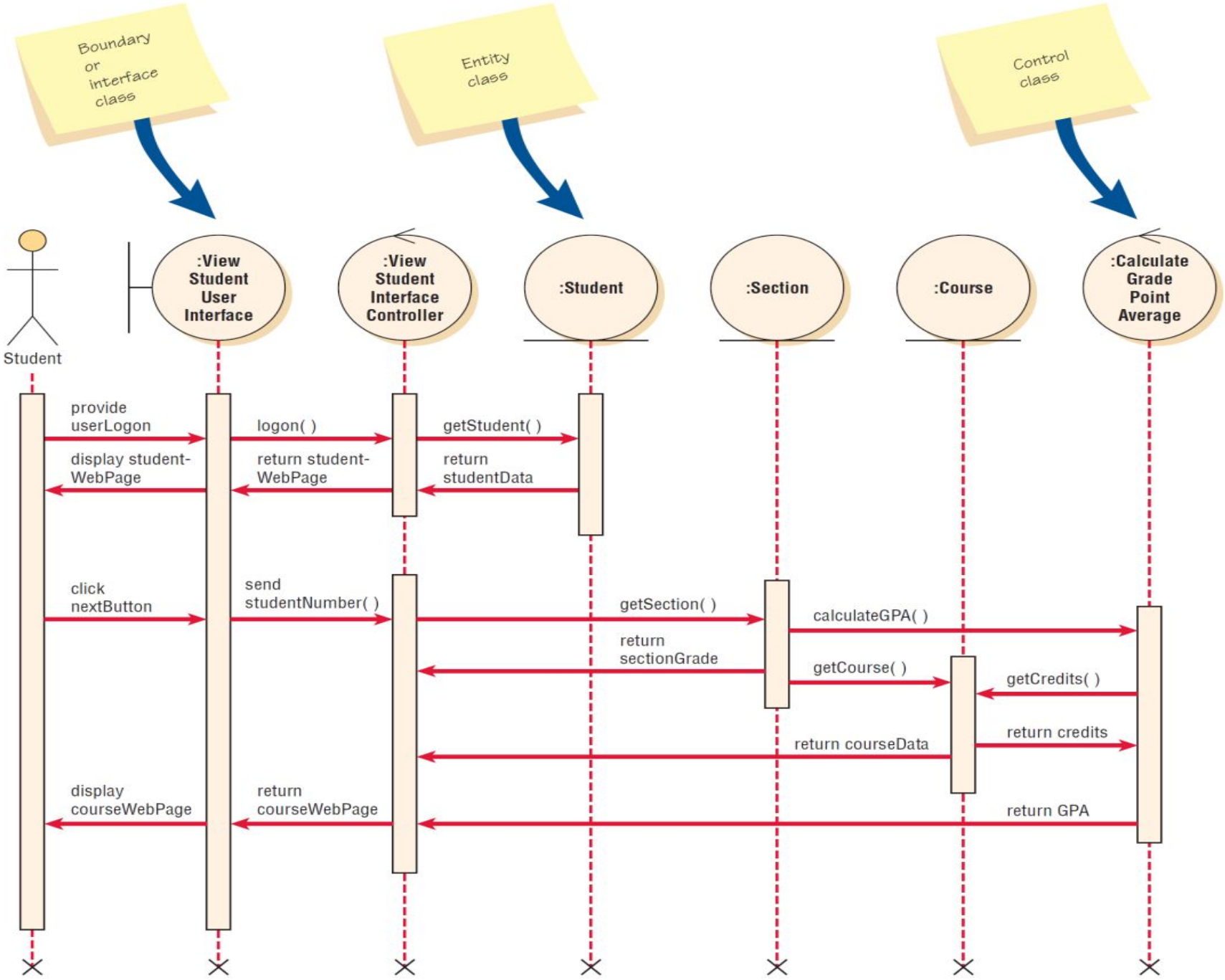
- Provide a means for users to work with the system
- Human interfaces may be a display, window, Web form, dialogue box, touch-tone telephone, or other way for users to interact with the system
- System interfaces involve sending data to or receiving data from other

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


Boundary or interface class
Entity class
Control class
Student
:View Student User Interface
:View Student Interface Controller
:Student
:Section
:Course
:Calculate Grade Point Average
provide userLogon
logon( )
getStudent( )
display student-WebPage
return student-WebPage
return studentData
click nextButton
send studentNumber( )
getSection( )
calculateGPA( )
return sectionGrade
getCourse( )
getCredits( )
return credits
return courseData
display courseWebPage
return courseWebPage
return GPA

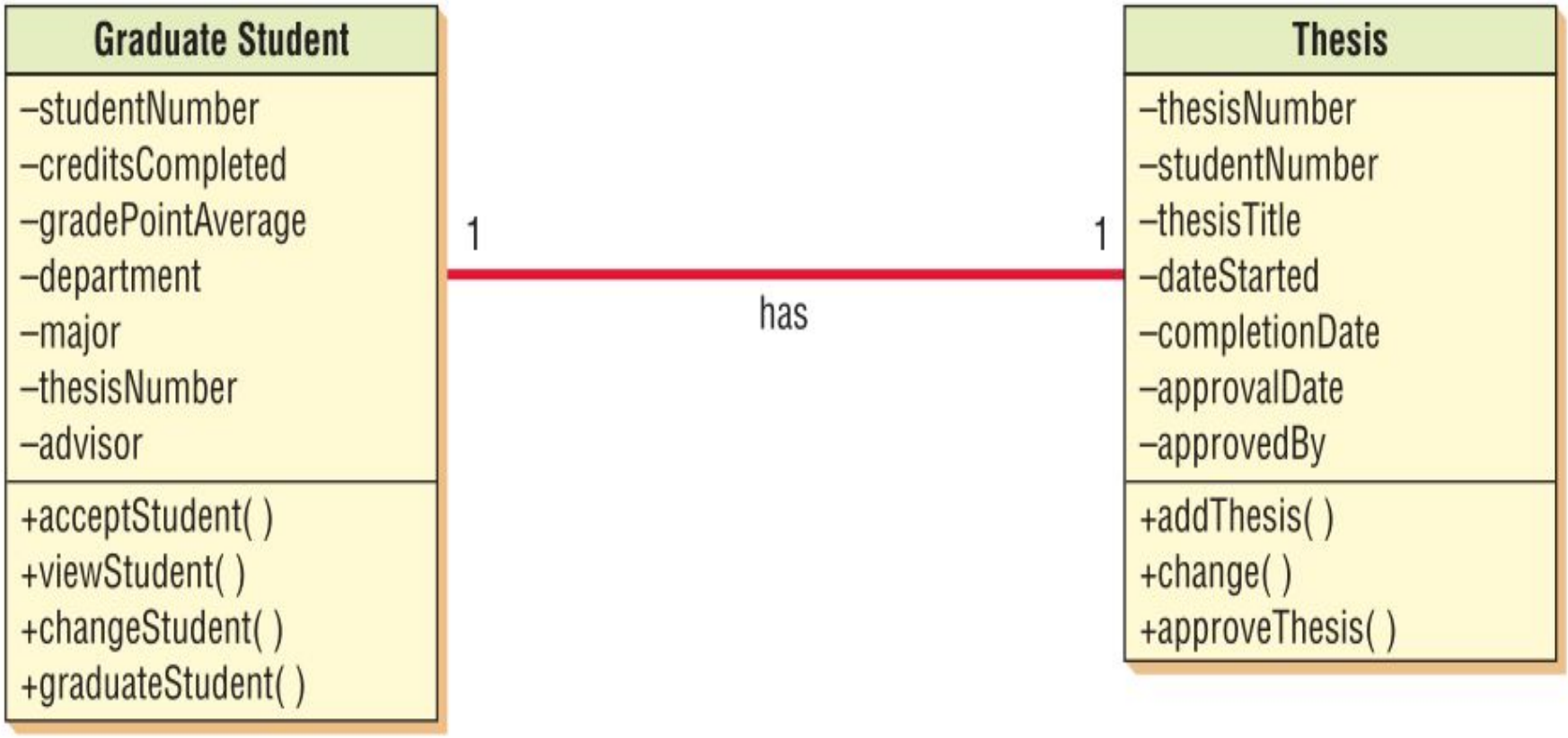
Relationships

- The connections between classes
  - Associations
  - Whole/part

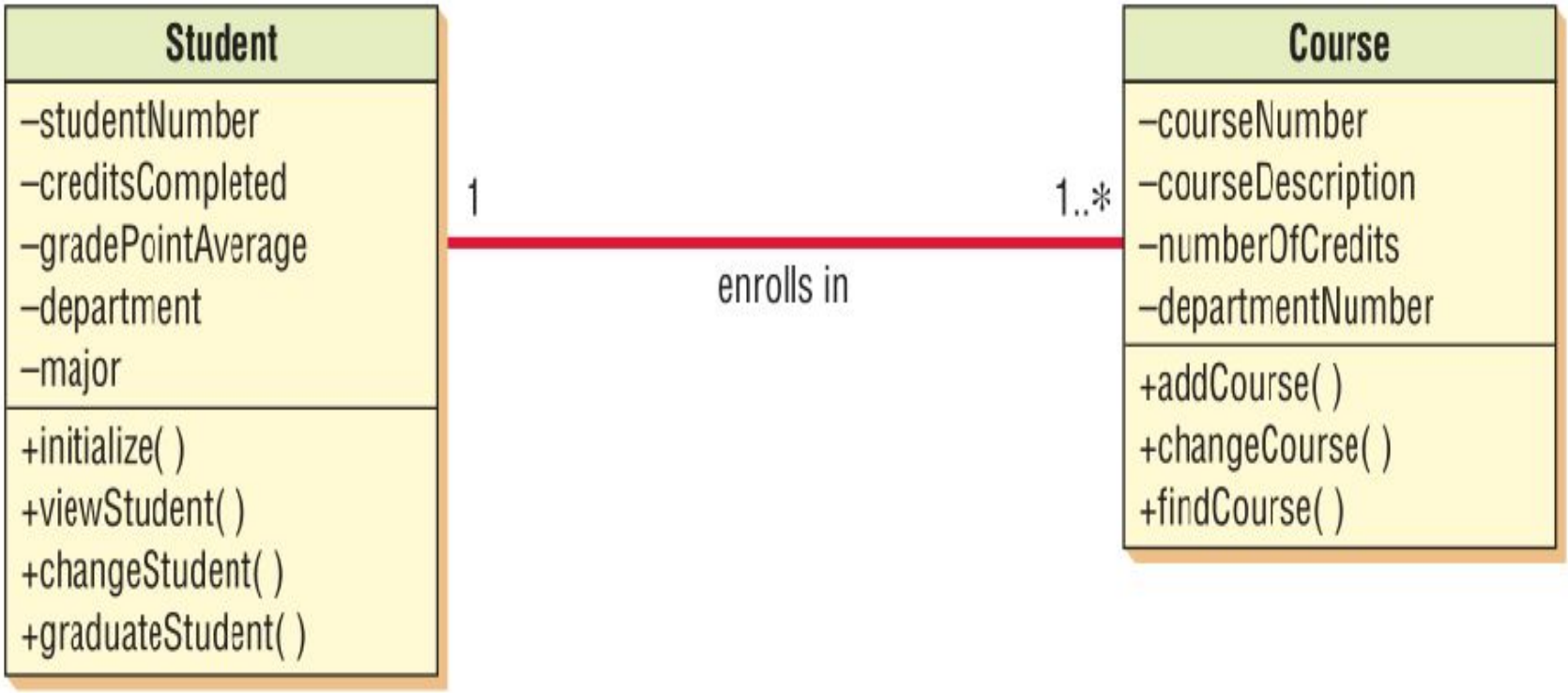
Associations

- 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

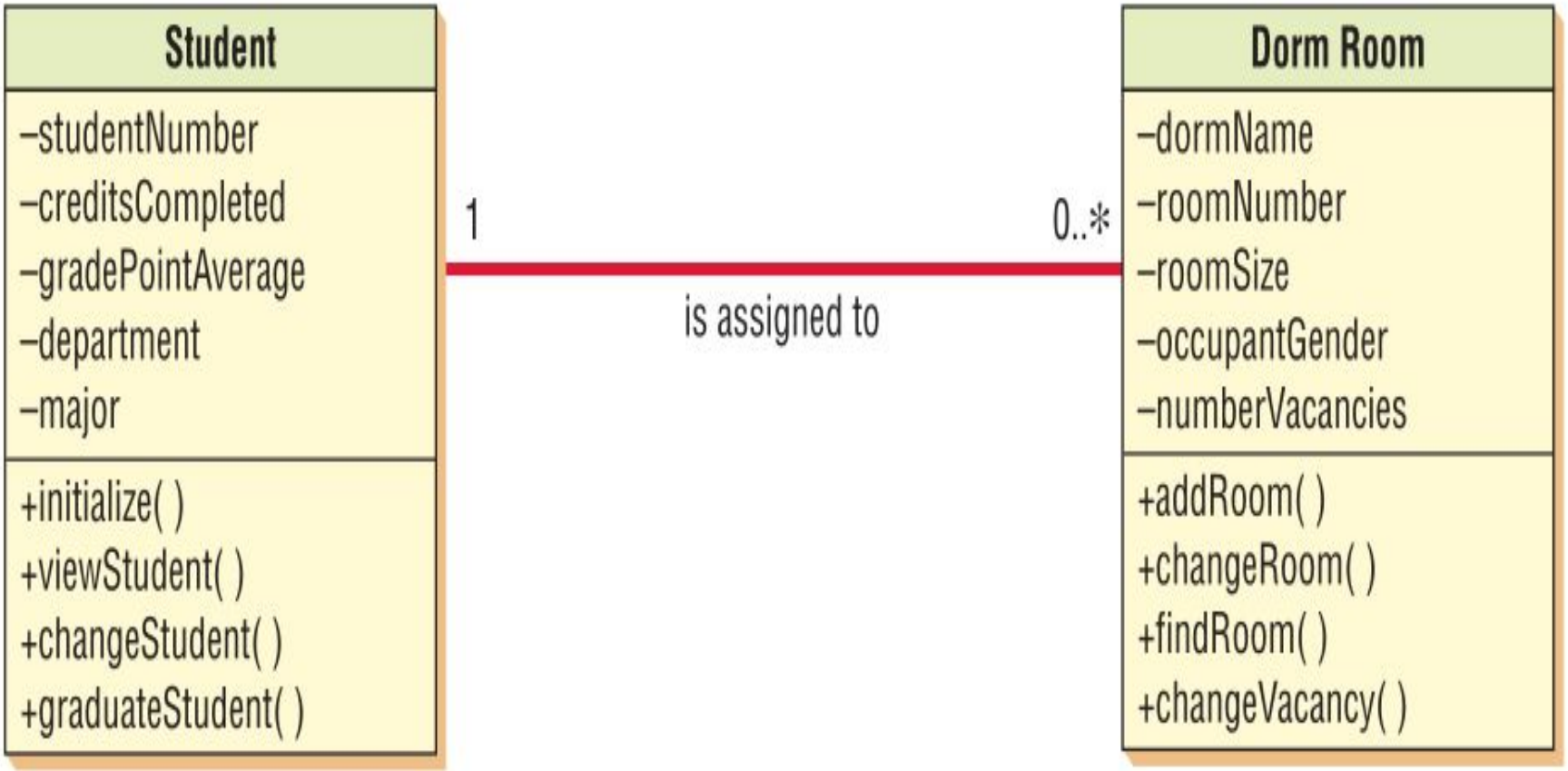
Association



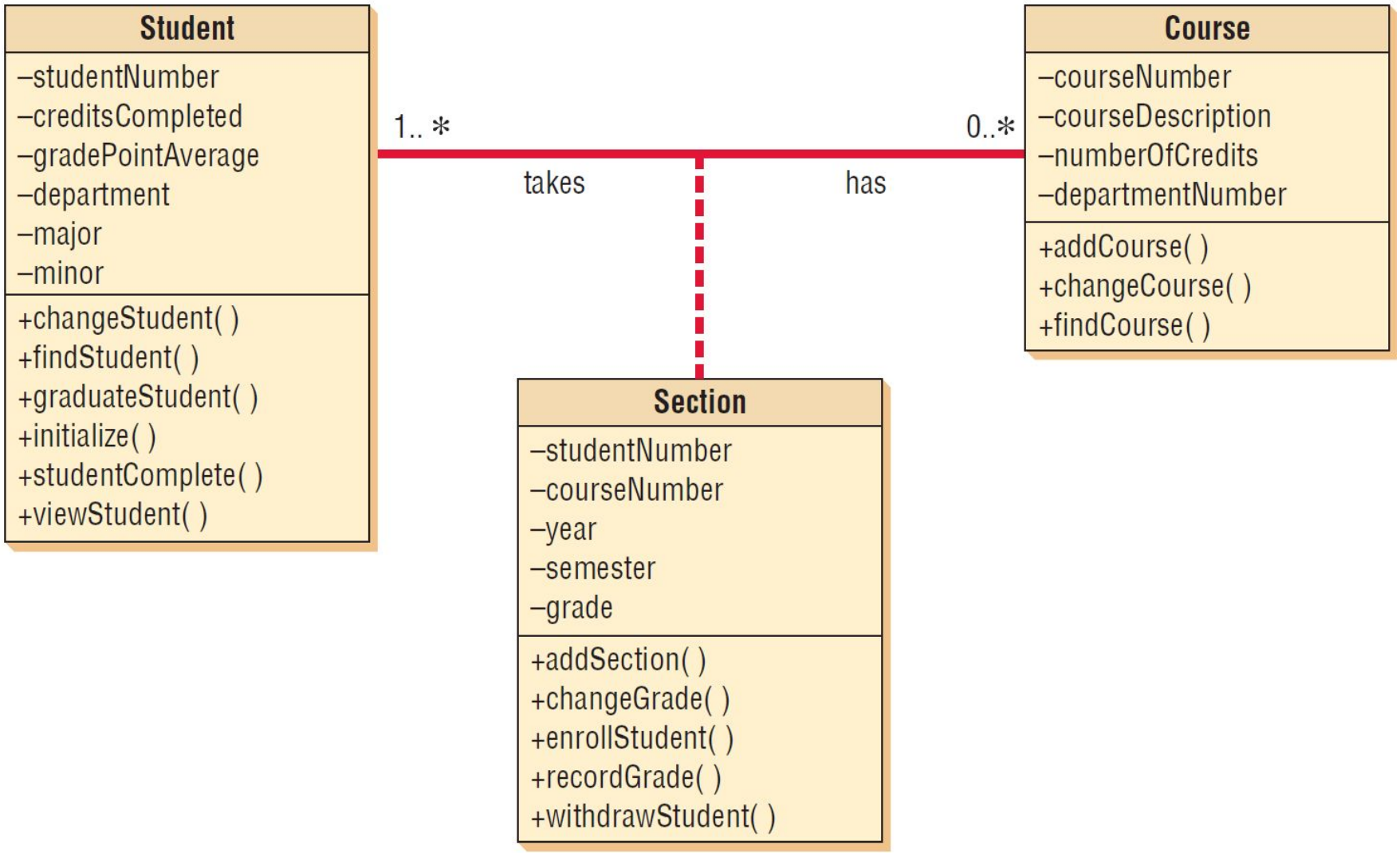
Association



Association



Association Class



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

Figure 18.13 A class diagram for course offerings. The filled-in diamonds show aggregation and the empty diamond shows a whole-part relationship

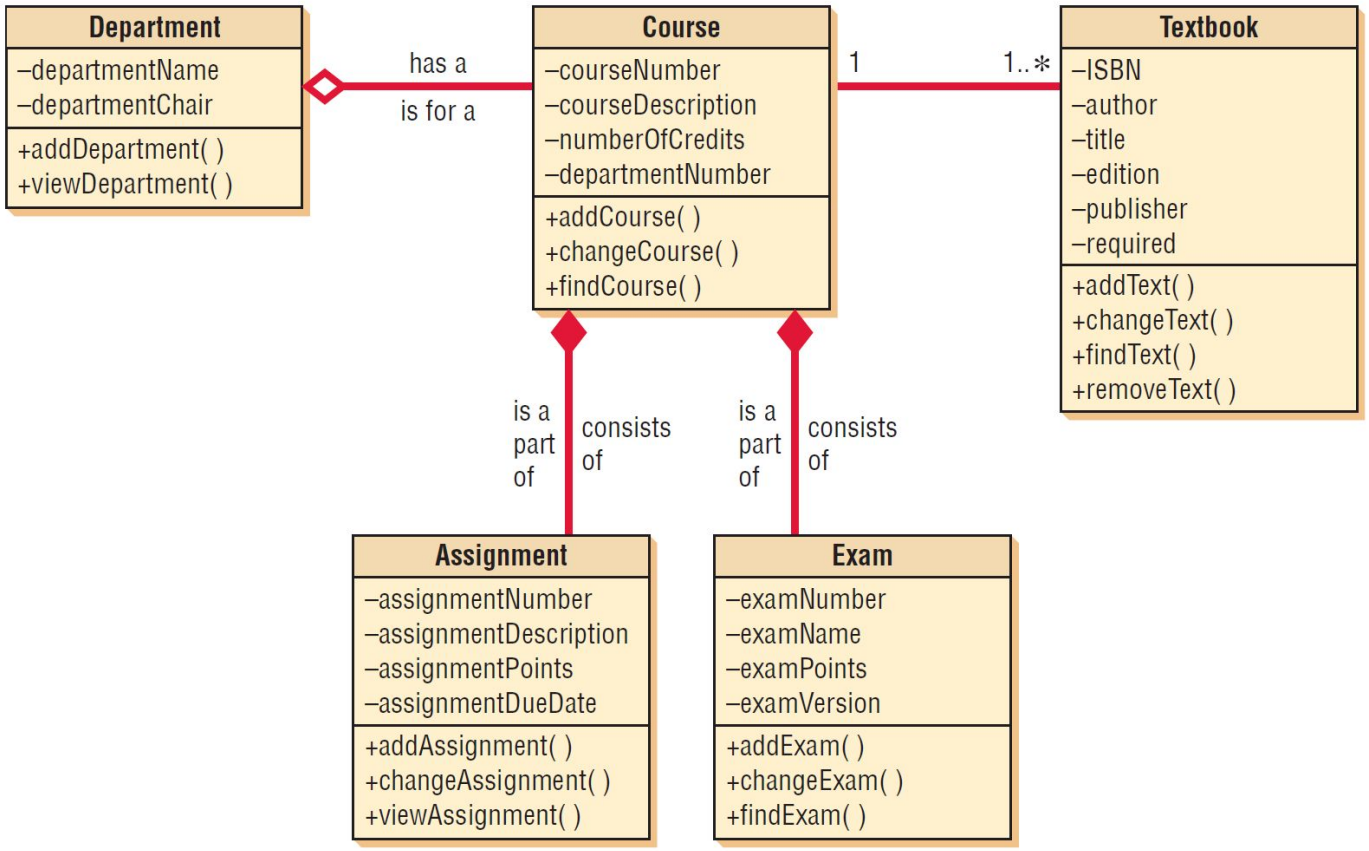
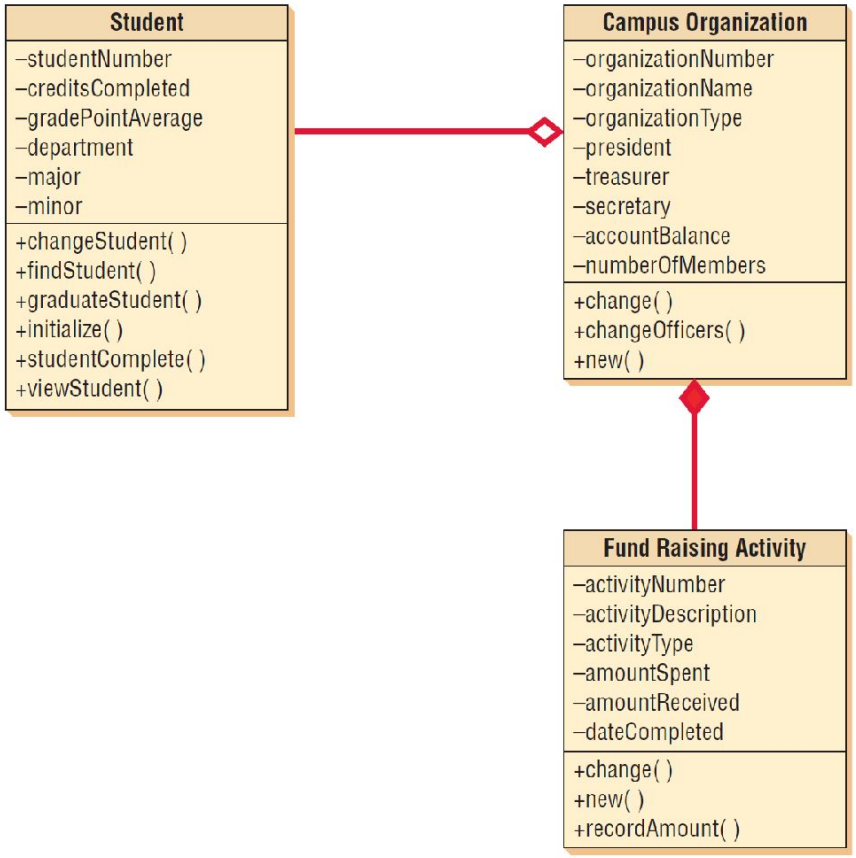


Figure 18.19 An example of whole-part and aggregation relationships



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

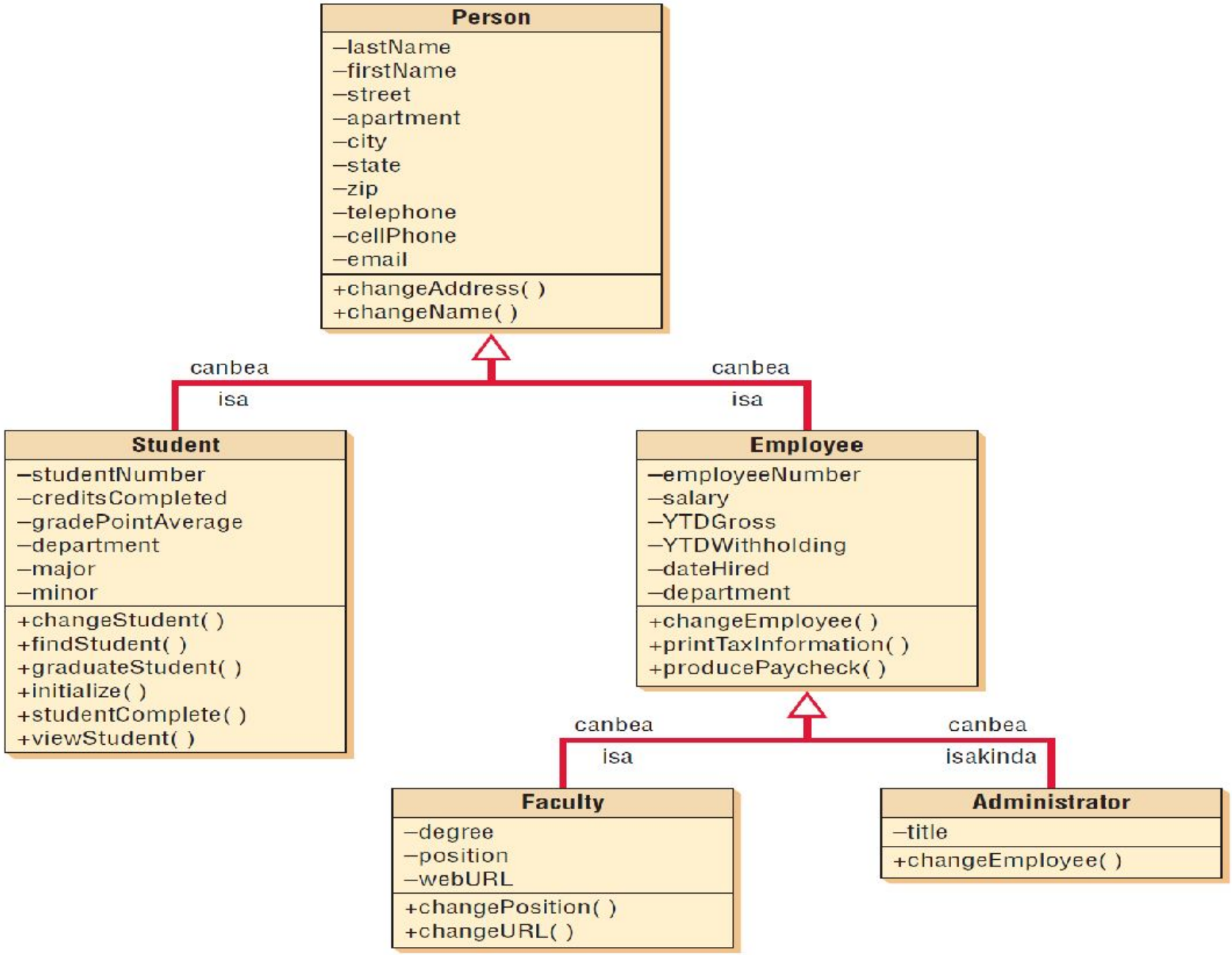
- Helps to foster reuse
- Helps to maintain existing program code

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


Person
–lastName
–firstName
–street
–apartment
–city
–state
–zip
–telephone
–cellPhone
–email
+changeAddress( )
+changeName( )
canbea
isa
canbea
isa
Student
–studentNumber
–creditsCompleted
–gradePointAverage
–department
–major
–minor
+changeStudent( )
+findStudent( )
+graduateStudent( )
+initialize( )
+studentComplete( )
+viewStudent( )
Employee
–employeeNumber
–salary
–YTDGross
–YTDWithholding
–dateHired
–department
+changeEmployee( )
+printTaxInformation( )
+producePaycheck( )
canbea
isa
canbea
isakinda
Faculty
–degree
–position
–webURL
+changePosition( )
+changeURL( )
Administrator
–title
+changeEmployee( )

Finding Classes

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

Packages

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

Figure 18.23 Use cases can be grouped into packages