

# **Modern Systems Analysis and Design**

**Third Edition**

**Jeffrey A. Hoffer  
Joey F. George  
Joseph S. Valacich**

## **Chapter 20**

# **Object-Oriented Analysis and Design**

# Learning Objectives

## Key terms

Use Case

Object

Object class

State

Behavior

Operation

Encapsulation

Constructor Operation

Query Operation

Update Operation

Association

Multiplicity

Abstract Class

Concrete Class

Class-Scope attribute

Abstract operation

Method

Polymorphism

Overriding

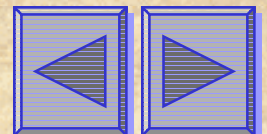
Aggregation

Composition

Event

State transition

Sequence diagram



# Learning Objectives

Discuss the concepts and principles underlying the object-oriented approach

Describe the activities in the different phases of the object-oriented development life cycle

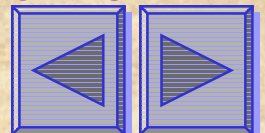
State the advantages of object-oriented modeling versus traditional systems development approaches

Learn to develop requirements models using use-case diagrams

Learn to use class diagrams to develop object models of the problem domain

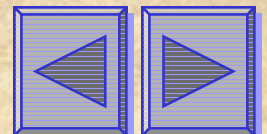
Learn to develop dynamic models using state, interaction and activity diagrams

Model real-world applications using UML diagrams



# Introduction

- ◆ Object-Oriented systems development life cycle
  - Process of progressively developing representation of a system component (or object) through the phases of analysis, design and implementation
  - The model is abstract in the early stages
  - As the model evolves, it becomes more and more detailed



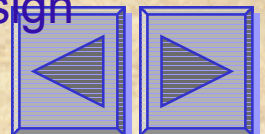
# The Object-Oriented Systems Development Life Cycle

## ◆ Analysis Phase

- Model of the real-world application is developed showing its important properties
- Model specifies the functional behavior of the system independent of implementation details

## ◆ Design Phase

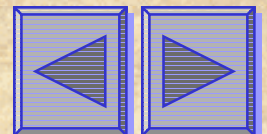
- Analysis model is refined and adapted to the environment
- Can be separated into two stages
  - ◆ System design
    - Concerned with overall system architecture
  - ◆ Object design
    - Implementation details are added to system design



# The Object-Oriented Systems Development Life Cycle

## ◆ Implementation Phase

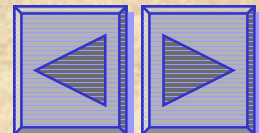
- Design is implemented using a programming language or database management system



# The Object-Oriented Systems Development Life Cycle

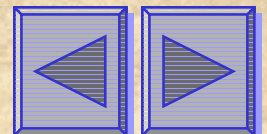
## ◆ Deliverables and Outcomes

1. The ability to tackle more challenging problem domains
2. Improved communication among users, analysts, designers and programmers
3. Increased consistency among analysis, design and programming activities
4. Explicit representation of commonality among system components
5. Reusability of analysis, design and programming results
6. Increased consistency among the models developed during object-oriented analysis, design, and programming



# The Unified Modeling Language (UML)

- ◆ A notation that allows the modeler to specify, visualize and construct the artifacts of software systems, as well as business models
- ◆ Techniques and notations
  - Use cases
  - Class diagrams
  - State diagrams
  - Sequence diagrams
  - Activity diagrams



# Use-Case Modeling

- ◆ Applied to analyze functional requirements of the system
- ◆ Performed during the analysis phase to help developers understand functional requirements of the system without regard for implementation details
- ◆ Use Case
  - A complete sequence of related actions initiated by an actor
- ◆ Actor
  - An external entity that interacts with the system

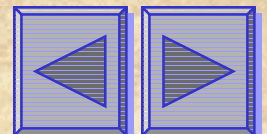
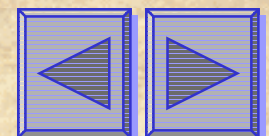
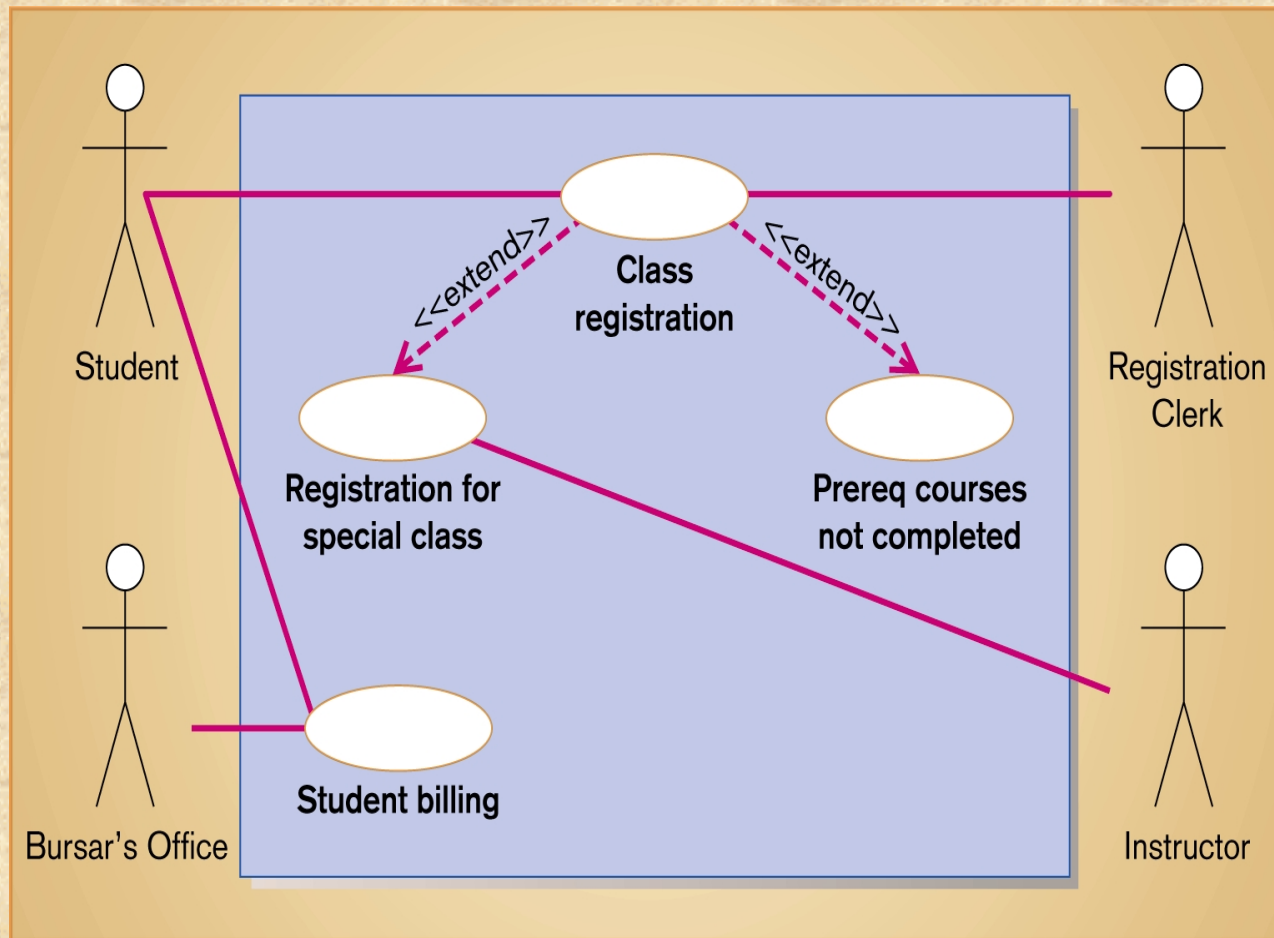


Figure 20-2  
Use-case diagram for a university registration system



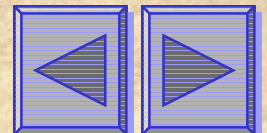
# Use-Case Modeling

## ◆ Developing Use-Case Diagrams

- Use cases are always initiated by an actor
- Use cases represent complete functionality of the system

## ◆ Relationships Between Use Cases

- Use cases may participate in relationships with other use-cases
- Two types
  - ◆ Extends
    - Adds new behaviors or actions to a use case
  - ◆ Include
    - One use case references another use case



# Object Modeling

## Class Diagrams

### ◆ Object

- An entity that has a well-defined role in the application domain, and has state, behavior, and identity

### ◆ State

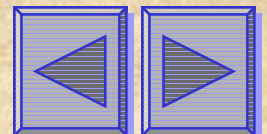
- A condition that encompasses an object's properties and the values those properties have

### ◆ Behavior

- A manner that represents how an object acts and reacts

### ◆ Object Class

- A set of objects that share a common structure and a common behavior

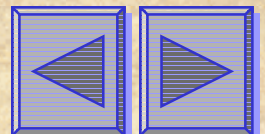


# Object Modeling

## Class Diagrams

### ◆ Class Diagram

- Class is represented as a rectangle with three compartments
- Objects can participate in relationships with objects of the same class



# Object Modeling

## Object Diagrams

### ◆ Object Diagram

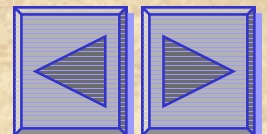
- A graph of instances that are compatible with a given class diagram; also called an instance diagram
- Object is represented as a rectangle with two compartments

### ◆ Operation

- A function or service that is provided by all the instances of a class

### ◆ Encapsulation

- The technique of hiding the internal implementation details of an object from its external view

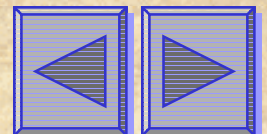


# Object Modeling

## Object Diagrams

### ◆ Types of Operations

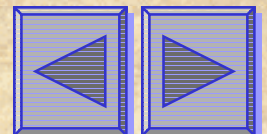
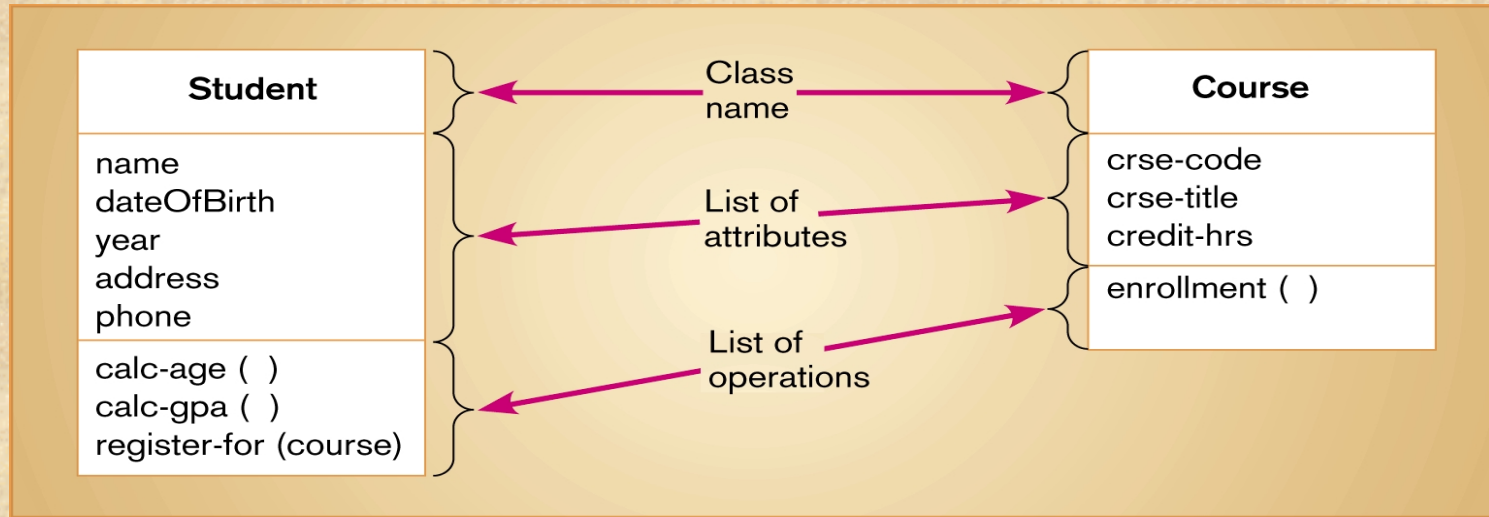
- Query
  - ◆ An operation that accesses the state of an object but does not alter the state
- Update
  - ◆ An operation that alters the state of an object
- Scope
  - ◆ An operation that applies to a class rather than an object instance
- Constructor
  - ◆ An operation that creates a new instance of a class



# Figure 20-5

## UML class and object diagrams

(a) Class diagram showing two classes  
 (b) Object diagram with two instances



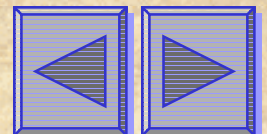
# Representing Associations

## ◆ Association

- A relationship between object classes
- Degree may be unary, binary, ternary or higher
- Depicted as a solid line between participating classes

## ◆ Association Role

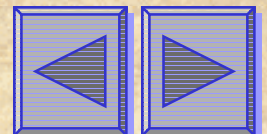
- The end of an association where it connects to a class
- Each role has multiplicity, which indicates how many objects participate in a given association relationship



# Representing Association Classes

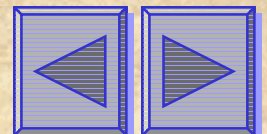
## ◆ Association Class

- An association that has attributes or operations of its own, or that participates in relationships with other classes
- ◆ Similar to an associative entity in ER modeling
- ◆ Figure 20-9 shows one example



# Representing Derived Attributes, Derived Associations and Derived Roles

- ◆ Derived attributes, associations and roles can be computed from other attributes, associations or roles
- ◆ Shown by placing a slash (/) before the name of the element



# Representing Generalization

## ◆ Generalization

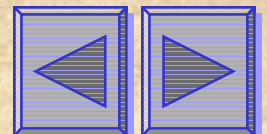
- Abstraction of common features among multiple classes, as well as their relationships, into a more general class

## ◆ Subclass

- A class that has been generalized

## ◆ Superclass

- A class that is composed of several generalized subclasses



# Representing Generalization

## ◆ Discriminator

- Shows which property of an object class is being abstracted by a generalization relationship

## ◆ Inheritance

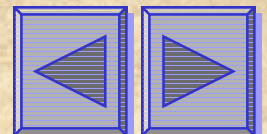
- A property that a subclass inherits the features from its superclass

## ◆ Abstract Class

- A class that has no direct instances, but whose descendents may have direct instances

## ◆ Concrete Class

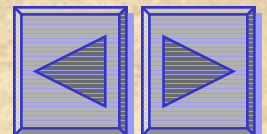
- A class that can have direct instances



# Representing Generalization

## ◆ Semantic Constraints among Subclasses

- Overlapping
  - ◆ A descendant may be descended from more than one of the subclasses
- Disjoint
  - ◆ A descendant may not be descended from more than one of the subclasses
- Complete
  - ◆ All subclasses have been specified. No additional subclasses are expected
- Incomplete
  - ◆ Some subclasses have been specified, but the list is known to be incomplete



# Representing Generalization

## ◆ Class-scope Attribute

- An attribute of a class which specifies a value common to an entire class, rather than a specific value for an instance.

## ◆ Abstract Operation

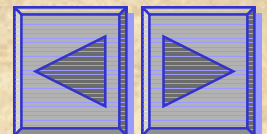
- Defines the form or protocol of an operation but not its implementation

## ◆ Method

- The implementation of an operation

## ◆ Polymorphism

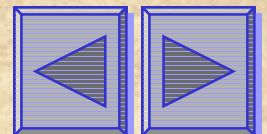
- The same operation may apply to two or more classes in different ways



# Interpreting Inheritance and Overriding

## ◆ Overriding

- Process of replacing a method inherited from a superclass by a more specific implementation of that method in a subclass
- Three Types
  - ◆ Overriding for Extension
    - Operation inherited by a subclass from its superclass is extended by adding some behavior
  - ◆ Overriding for restriction
    - The protocol of the new operation in the subclass is restricted
  - ◆ Overriding for optimization
    - The new operation is implemented with improved code by exploiting the restrictions imposed by a subclass



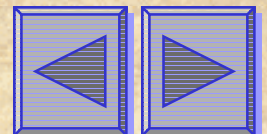
# Representing Multiple Inheritance

## ◆ Multiple Classification

- An object is an instance of more than one class
- Use is discouraged and not supported by UML semantics

## ◆ Multiple Inheritance

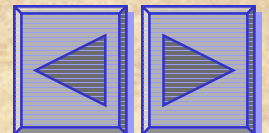
- Allows a class to inherit features from more than one superclass



# Representing Aggregation

## ◆ Aggregation

- A part-of relationship between a component object and an aggregate object
- Example: Personal computer
  - ◆ Composed of CPU, Monitor, Keyboard, etc



# Dynamic Modeling: State Diagrams

## ◆ State

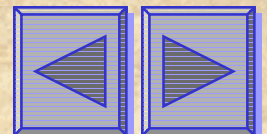
- A condition during the life of an object during which it satisfies some conditions, performs some actions or waits for some events
- Shown as a rectangle with rounded corners

## ◆ State Transition

- The changes in the attribute of an object or in the links an object has with other objects
- Shown as a solid arrow
- Diagrammed with a guard condition and action

## ◆ Event

- Something that takes place at a certain point in time



# Dynamic Modeling: Sequence Diagrams

## ◆ Sequence Diagram

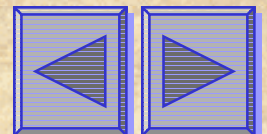
- A depiction of the interaction among objects during certain periods of time

## ◆ Activation

- The time period during which an object performs an operation

## ◆ Messages

- Means by which objects communicate with each other



# Dynamic Modeling

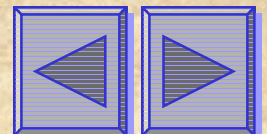
## Sequence Diagrams

### ◆ Synchronous Message

- A type of message in which the caller has to wait for the receiving object to finish executing the called operation before it can resume execution itself

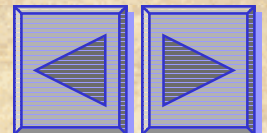
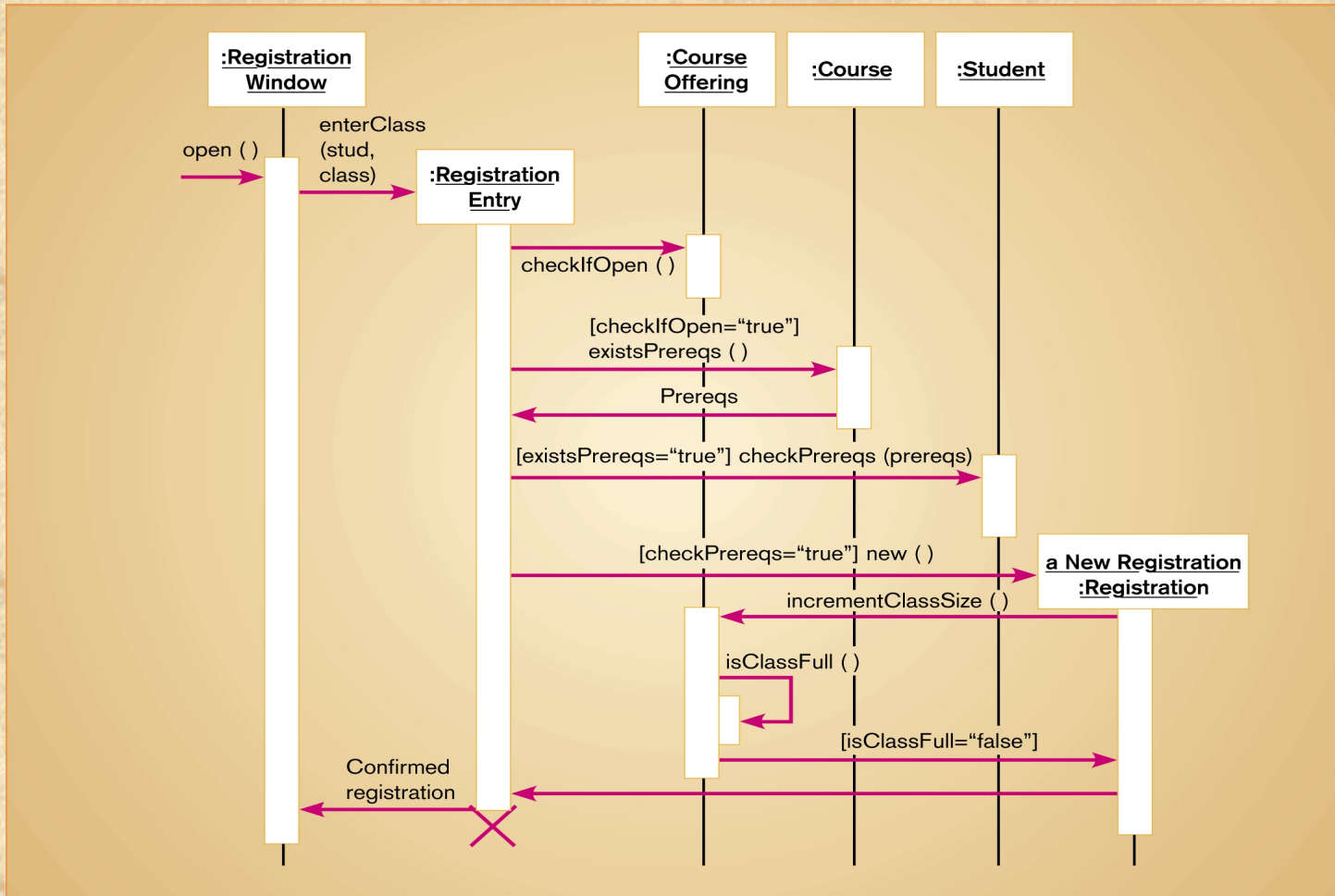
### ◆ Simple Message

- A message that transfers control from the sender to the recipient without describing the details of the communication



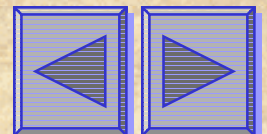
# Figure 20-24

## Sequence diagram for a class registration scenario with prerequisites



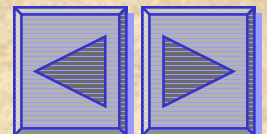
# Process Modeling: Activity Diagrams

- ◆ Shows the conditional logic for the sequence of system activities needed to accomplish a business process
- ◆ Clearly shows parallel and alternative behaviors
- ◆ Can be used to show the logic of a use case



# Analysis Versus Design

- ◆ Start with existing set of analysis model
- ◆ Progressively add technical details
- ◆ Design model must be more detailed than analysis model
- ◆ Component Diagram
  - A diagram that shows the software components or modules and their dependencies
- ◆ Deployment Diagram
  - A diagram that shows how the software components, process and objects are deployed into the physical architecture of the system

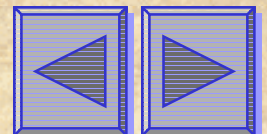


# Summary

## ◆ Object-Oriented modeling approach

- Benefits
- Unified Modeling Language
  - ◆ Use cases
  - ◆ Class diagrams
  - ◆ State diagrams
  - ◆ Sequence diagrams
  - ◆ Activity Diagrams

## ◆ Use-case modeling



# Summary

- ◆ Object Modeling: Class Diagrams
  - Associations
  - Generalizations
  - Inheritance and Overriding
  - Aggregation
- ◆ Dynamic Modeling: State Diagrams
- ◆ Dynamic Modeling: Sequence Diagrams
- ◆ Analysis Versus Design

