OBJECT ORIENTED DESIGN
with the Unified Process

Use Case Realization
... (Zoom-Into Design)

**Requirement**
- Requirement Specification (Functional & Non-Functional)

**analysis**
- Requirement Models (Use-case diagram, class diagram, activity diagram, system sequence diagram)

**Design**
- Architectural design model (Package diagram)
- Detailed design models (Class diagram, sequence diagram, state diagram)
Objectives

• Explain the purpose and objectives of object-oriented design

• Develop design class diagrams

• Develop detailed sequence diagrams as the core process in systems design
Objectives (continued)

• Develop communication diagrams as part of systems design

• Document the architecture design using package diagrams
Overview

• Develop detailed object-oriented design models
• Develop models for each layer of a three-layer design
• Design class diagrams
  – Extend domain model
• Interaction diagrams
  – Extend system sequence diagrams
• Package diagrams
  – Show relationships and dependencies among classes
Object-oriented event-driven program flow
Object-Oriented Design Models

• Identify all objects that must work together to carry out a use case
• Divide objects into groups for a multilayer design
• Interaction diagrams describe the messages that are sent between objects
  – Includes sequence and communication diagrams
• Design class diagrams document and describe the programming classes
Design class for Student class

```
«Design Class»
Student

- studentID: integer {key}
- name: string
- address: string
- dateAdmitted: date
- lastSemesterCredits: number
- lastSemesterGPA: number
- totalCreditHours: number
- totalGPA: number
- major: string

+ createStudent (name, address, major): Student
+ createStudent(studentID): Student
+ changeName (name)
+ changeAddress (address)
+ changeMajor (major)
+ getName ( ): string
+ getAddress ( ): string
+ getMajor ( ): string
+ getCreditHours ( ): number
+ updateCreditHours ( )
```
public class Student
{
    // attributes
    private int studentID;
    private String firstName;
    private String lastName;
    private String street;
    private String city;
    private String state;
    private String zipcode;
    private Date dateAdmitted;
    private float numberCredits;
    private String lastActiveSemester;
    private float lastActiveSemesterGPA;
    private float gradePointAverage;
    private String major;

    // constructors
    public Student (String inFirstName, String inLastName, String inStreet,
        String inCity, String inState, String inZip, Date inDate)
    {
        firstName = inFirstName;
        lastName = inLastName;
        ...
    }
    public Student (int inStudentID)
    {
        // read database to get values
    }

    // get and set methods
    public String getFullName ()
    {
        return firstName + " " + lastName;
    }
    public void setFirstName (String inFirstName)
    {
        firstName = inFirstName;
    }
    public float getGPA ()
    {
        return gradePointAverage;
    }
    // and so on

    // processing methods
    public void updateGPA ()
    {
        // access course records and update lastActiveSemester and
        // to-date credits and GPA
    }
}
Object-Oriented Design Models (continued)

- Statecharts capture information about the valid states and transitions of an object
- Package diagrams denote which classes work together as a subsystem
- Design information is primarily derived from
  - Domain model class diagrams
  - Interaction diagrams
Design models with their respective input models
Object-Oriented Design Process

• Create a first-cut model of the design class diagrams

• Develop interaction diagrams for each use case or scenario

• Update the design class diagrams
  – Method names, attributes, and class relationships

• Partition the design class diagrams into related functions using package diagrams
Object-Oriented Design Process

DEVELOP DESIGN CLASS DIAGRAMS
Design Class Symbols

• Stereotypes
  – UML notation to categorize a model element as a certain type

• Two types of notation
  – Full notation with guillemets (« »)
  – Shorthand notation with circular icons

• Standard stereotypes
  – Entity, control, boundary, data access
Figure 8-5
Standard stereotypes found in design models
Boundary class

- Intermediates the interface to something outside the system
- Several types:
  - User interface classes
    - Display windows/screens
  - System interface classes
    - Interface to external systems, legacy systems
  - Device interface classes
    - Interface to devices which detect external events
Example – Boundary class

```
Student

Register for Courses

Course Catalog System

<<boundary>>
RegisterForCoursesForm

<<boundary>>
CourseCatalogSystem
```
Entity class

- Entity objects represent the key concepts of the system being developed
- Store and manage information in the system (usually persistent)
Example – Entity class

- Register for Courses (Create Schedule)
  - CourseOffering
  - Student
  - Schedule
Control class

- Use case behavior coordinator
- One control class per use case
  - “Orchestrate and Delegate Behavior”
Example – Control class

- One controller class per use case
  - As analysis continues, a complex use case’s control class may evolve into more than one class

```
   \[\text{Student} \rightarrow \text{Register for Courses} \rightarrow \text{Course Catalog System}\]
```

```
\text{<<controller>>}
\text{RegistrationController}
```
Example 1

There is another stereotype <<data access>>

Sequence Diagram:
Login / E-1:Invalid User Info

AnyUsers

1: //get user info()

2: getUserInfo(uid, pwd)

LoginForm

LoginCtrl

User

MainAppForm

Boundary

Control

Entity

Boundary
Object-oriented event-driven program flow and possibility for design class diagram
Design Class Notation

- Class name and stereotype information
- Attribute information
  - Visibility, type-expression, name, initial value, and properties
- Method signature
  - Visibility, name, type-expression, and parameter list
  - Use the entire signature to identify a method to distinguish between overloaded methods
Internal symbols used to define a design class

- **Stereotype Name**
  - Class Name::Parent Class

- Attribute list
  - visibility name:type-expression = initial-value {property}

- Method list
  - visibility name:type-expression (parameter list)
Internal Symbols and Example

```
«Stereotype Name»
Class Name::Parent Class

Attribute list
visibility name:type-expression = initial-value {property}

Method list
visibility name:type-expression (parameter list)

«Design Class»
Student

- studentID: integer (key)
- name: string
- address: string
- dateAdmitted: date
- lastSemesterCredits: number
- lastSemesterGPA: number
- totalCreditHours: number
- totalGPA: number
- major: string

+ createStudent (name, address, major): Student
+ createStudent(studentID): Student
+ changeName (name)
+ changeAddress (address)
+ changeMajor (major)
+ getName (): string
+ getAddress (): string
+ getMajor (): string
+ getCredithours (): number
+ updateCreditHours ()
```
Student class examples for the domain diagram and the design class diagram
Developing the First-Cut Design Class Diagram

- Elaborate the attributes with type and initial value information
  - Most attributes should be private

- Add relationships to the classes (aggregation, generalization and association)
  - Based on which classes need access to which other classes
  - Can be bidirectional
  - Will need to be updated as design progresses
First-cut RMO design class diagram

- **Catalog**
  - catalogID: string
  - season: string
  - year: string
  - description: string
  - effectiveDate: date
  - endDate: date

- **CatalogProduct**
  - catalogID: string
  - productID: string
  - price: float
  - specialPrice: float

- **ProductItem**
  - productID: string
  - vendor: string
  - gender: string
  - description: string

- **InventoryItem**
  - inventoryID: string
  - size: string
  - color: string
  - options: string
  - quantityOnHand: integer
  - averageCost: float
  - reorderQuantity: integer

- **ReturnItem**
  - inventoryID: string
  - reason: string
  - condition: string
  - disposal: string

- **Customer**
  - accountNo: string
  - name: string
  - billingAddress: string
  - shippingAddress: string
  - dayPhone: string
  - nightPhone: string

- **OrderItem**
  - productID: integer
  - inventoryID: integer
  - description: string
  - price: float
  - quantity: integer
  - backorderStatus: string

- **Order**
  - orderID: integer
  - orderDate: date
  - priorityCode: string
  - shippingHandling: float
  - tax: float
  - grandTotal: float

- **OrderTransaction**
  - transactionID: integer
  - date: date
  - transactionType: string
  - amount: float
  - paymentMethod: string
  - creditCardInformation: string

- **Shipper**
  - shipperID: integer
  - name: string
  - address: string
  - contactName: string
  - telephone: string

- **Shipment**
  - trackingNo: string
  - dateSent: date
  - timeSent: time
  - shippingCost: float
  - dateArrived: date
  - timeArrived: time

- **WebOrder**
  - emailAddress: string
  - replyMethod: string

- **TelephoneOrder**
  - phoneClerk: string
  - callStartTime: time
  - lengthOfCall: float

- **MailOrder**
  - dateReceived: date
  - processorClerk: string
Interaction Diagrams—Realizing Use Cases and Defining Methods

- Interaction diagrams are at the heart of object-oriented design
- Realization of a use case
  - Determine what objects collaborate by sending messages to each other
- Two types
  - Sequence
  - Communication
Object-Oriented Design Process

DEVELOP DETAILED SEQUENCE DIAGRAMS AS THE CORE PROCESS IN SYSTEMS DESIGN
Designing with Sequence Diagrams

• An SSD captures the interactions between the system and the external world represented by actors
  – The system is treated like a black box

• A detailed sequence diagram uses all of the same elements as an SSD
  – The :System object is replaced by all of the internal objects and messages within the system
System object

Internal object

Order Clerk

System

startOrder (accountNo)

Loop for all items

addItem (catalogID, prodID, size, quantity)

description, price, extendedPrice

completeOrder ( )

totalDue

completeOrder (paymentAmt, ccInformation)
SSD for the Look up item availability use case
First-Cut Sequence Diagram

• Determine which other objects may need to be involved to carry out the use case
• Replace the \texttt{:System} object with a use case controller object
• Determine which other messages will be sent
  – Define the source and destination object for each message
• Use activation lifelines to indicate when an object is executing a method
First-cut sequence diagram for the *Look up item availability* use case
Sequence Diagram vs Class Diagram
Developing a Multilayer Design

- **View layer**
  - Design the user interface for each use case
  - Develop dialog designs for forms
  - Add the window classes to the sequence diagram
- **Data access layer**
  - Initialize domain objects with data from the database
  - Query the database and send a reference object
  - Return information in the reference object
Developing Multilayer

Business Layer – entity classes/ objects
DAO – data access classes/ object

- The purpose of DAO is for:
  - Easier to change database without changing other classes
  - Security where only certain classes (DAO) can access the database information
Use Case Controller

- An artifact invented by the designer to handle a system function
  - Serves as a collection point for incoming messages
  - Intermediary between the outside world and the internal system
  - For Look Up Item Availability - the use case controller is called AvailabilityHandler
Completed three-layer design for Look up item availability
A First-Cut Sequence Diagram for an RMO Telephone Order

• Define a user controller object – `OrderHandler`
• Define a “create” message for new `Order` objects
  – Customer object creates the `Order` object
• Define other messages
  – `addItem`, `createOrdItem`, `getDescription`, `getPrice`, `updateQty`
• Identify source, destination, and navigation visibility for each message
SSD for the telephone order scenario of the *Create new order* use case
Sequence diagram for the telephone order scenario of the **Create new order** use case
Developing a Multilayer Design for the Telephone Order Scenario

- Extend one message at a time
- View layer
  - Open Order window and return a Customer object
- Data layer
  - Customer object initializes itself
  - Add items to an order with a repeating message
  - Save Order and OrderItem to the database
  - Update database inventory
  - Complete transaction
Telephone order sequence diagram for the `startOrder` message
Telephone order sequence diagram for the **addItem** message
Telephone order sequence diagram for the final messages
Object-Oriented Design Process

DEVELOP COMMUNICATION DIAGRAMS AS PART OF SYSTEMS DESIGN
Designing with Communication Diagrams

• Shows a view of the use case that emphasizes coupling
• Uses the same symbols as a sequence diagram for actors, objects, and messages
• Lifeline symbols are not used
• Link symbols indicate that two items share a message
• Numbers indicate the sequence in which messages are sent
The symbols of a communication diagram
A communication diagram for Create new order
Updating the Design Class Diagram

• Add classes for the view and data access layers
• Update classes with method signatures
  – Constructor and get and set methods are optional
  – Use case specific methods are required
• Every message in a sequence diagram requires a method in the destination object
• Include the new user controller classes and add class relationships.
Updated design class diagram for the domain layer
Object-Oriented Design Process

DOCUMENT THE ARCHITECTURE DESIGN USING PACKAGE DIAGRAMS
Package Diagrams-Structuring the Major Components

- Associates classes of related groups
- One option is to separate the view, domain, and data access layers into separate packages
- Indicate dependency relationships
  - Shows which elements affect other elements in a system
  - May exist between packages, or between classes within packages
- Packages can be nested
Three layer architecture design and package diagram for RMO

***The arrows show where the package should be placed in the architecture***
Implementation Issues for Three-Layer Design

- IDE tools can help programmers construct systems
- IDE tools can also make a system difficult to maintain
  - Creates window classes that generate class definitions
  - Inserts business logic code into the user interface
- Use good design principles when developing a system
  - Define object responsibility for each layer
Summary

• Design is driven by use cases
• Two primary models developed during design
  – Design class diagrams
  – Sequence class diagrams
• Multilayer designs partition classes into groups
  – View, domain, and data access layers
• Communication diagrams are a viable alternative to sequence diagrams