Introduction to Object Orientation

Fall 2003 CSC 532

Objective: Introduction to Object Orientation

• Understand the basic principle of object orientation
• Understand the basic concepts and terms of object orientation and the associated UML notation
• Appreciate the strengths of object orientation
• Understand some basic UML modeling mechanisms

Introduction to Object Orientation Topics

• Basic Principles of Object Orientation
• Basic Concepts of Object Orientation
• Strengths of Object Orientation
• General UML Modeling Mechanisms
Basic Principles of Object Orientation

Object Orientation

• Abstraction
• Encapsulation
• Modularity
• Hierarchy

What is Abstraction?

What is Encapsulation

• Hide implementation from clients
  • Clients depend on interface
What is Modularity?

• The breaking up of something complex into manageable pieces

Order Processing System → Order Entry
Order Fulfillment
Billing

What is Hierarchy?

• Level of abstraction

Increasing abstraction
Asset
Bank Account
Security
Real Estate
Savings
Checking
Stock
Bond

Decreasing Abstraction

Basic Concepts Of Object Orientation

• Object
• Class
• Attributes
• Operation
• Interface (Polymorphism)
• Components
• Package
• Subsystem
• Relationships
What is an Object?

- Informally, an object represents an entity, either physical, conceptual, or software
  - Physical entity
  - Conceptual entity
  - Software entity

A More Formal Definition

- An object is a concept, abstraction, or thing with sharp boundaries and meaning for an application
- An object is something that has
  - State
  - Behavior
  - Identity

An Object Has State (stop here)

- The state of an object is one of the possible conditions in which an object may exist
- The state of an object normally changes over time
- Represented by: Attribute values + Links (relationship instances)
An Object Has Behavior

• Behavior determines how an object acts and reacts to requests from other objects
• Behavior is represented by the set of messages it can respond to (the operations the object can perform)

An Object Has Identity

• Each object has a unique identity, even if its state is identical to that of another object

Representing Objects

• An object is represented as rectangles with underlined names
**Example: Objects**

<table>
<thead>
<tr>
<th>Intro to OO 180</th>
<th>English 101</th>
</tr>
</thead>
<tbody>
<tr>
<td>World History 200</td>
<td>Geology 110</td>
</tr>
<tr>
<td>Algebra 110</td>
<td>Music History 200</td>
</tr>
</tbody>
</table>

**What is a Class?**

- A class is a description of a group of objects with common properties (attributes), behavior (operations), relationships, and semantics
- An object is an instance of a class
- A class is an abstraction in that it:
  - Emphasizes relevant characteristics
  - Suppresses other characteristics

**Sample Class**

<table>
<thead>
<tr>
<th>Properties</th>
<th>Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Add a student</td>
</tr>
<tr>
<td>Location</td>
<td>Delete a student</td>
</tr>
<tr>
<td>Days offered</td>
<td>Get course roster</td>
</tr>
<tr>
<td>Credit hours</td>
<td>Determine if it is full</td>
</tr>
<tr>
<td>Start time</td>
<td>End time</td>
</tr>
</tbody>
</table>
Representing Classes

• A class is represented using a compartmented rectangle

Class Compartments

• A class is comprised of three sections
  • The first section contains the class name
  • The second section shows the structure (attributes)
  • The third section shows the behavior (operations)

Class Compartments (cont.)

• The second and third sections may be suppressed if they need not be visible on the diagram
Example: Class

```
CourseOffering
- Intro to OO 180
- English 101
- World History 200
- Geography 110
- Algebra 110
- Music History 200
```

Classes of Objects

- How many class can you see?

The Relationship Between Classes and Objects

- A class is an abstract definition of an object
  - It defines the structure and behavior of each object in the class
  - It serves as a template for creating objects
- Objects may be grouped into classes

```
Professor Smith  Professor Jones  Professor Mellon
```


What is Attribute?

Class
- Attribute
  - number
  - startTime
  - endTime

Object
- Attribute
  - number
  - startTime
  - endTime

CourseOffering
- number
- startTime=900
- endTime=1100

ODCourseOffering
- number
- startTime=100
- endTime=1100

What is Operation?

Class
- Operation
  - addStudent
  - deleteStudent
  - getStartTime
  - getEndTime

What is Polymorphism?

- The ability to hide many different implementations behind a single interface

Manufacture A
Manufacture B
Manufacture C

OO Principle:
Encapsulation
What is an Interface?

- Interface formalize polymorphism
- Interface support “plug-and-play” architecture

```
<<interface>>

Shape

Draw
Move
Scale
Rotate

Tube
Pyramid
Cube
```

What is a Component?

- A non-trivial, nearly independent, and replaceable part of a system that fulfills a clear function in the context of a well-defined architecture
What is a Component? (Cont.)

- A component may be:
  - A source code component
  - A run time component or
  - An executable component

Source Code Components

- Visualize compilation dependencies between source code files

Account.h
Account.cpp
Report.cpp

Executable and Run-time Components

- Visualizing all of the pieces of an executable release, their interfaces, and their relationships

ATM
Account
Bank
Account
Bank
Interfaces and Components

- Interfaces can be realized by components

**Implementation Model**

```
Component --> Interface
          ^          |
         Source File Name
```

What is a Package?

- A package is a general purpose mechanism for organizing elements into groups
- A model element which can contain other model elements

**Uses**

- Organize the model under development
- A unit of configuration management

What is a Subsystem?

- A “cross between” a package (can contain other model elements) and a class (has behavior)
- Realizes one or more interfaces which define its behavior

```
Subsystem
          |  Realization
          |      Interface
          |     Subsystem
          |    Subsystem Name
```
Subsystems and Components

- Components are the physical realization of an abstraction in the design
- Subsystems can be used to represent the component in the design

<table>
<thead>
<tr>
<th>Design Model</th>
<th>Implementation Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;&lt;subsystem&gt;&gt;</td>
<td>Component Name</td>
</tr>
<tr>
<td>Component Name</td>
<td>Component Interface</td>
</tr>
</tbody>
</table>

Relationships

- Association
- Aggregation
- Composition
- Dependency
- Generalization
- Realization

Relationships: Association

- Models a semantic connection among classes

<table>
<thead>
<tr>
<th>Association name</th>
<th>Professor</th>
<th>University</th>
</tr>
</thead>
<tbody>
<tr>
<td>Works for</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Class Association</th>
<th>Role Names</th>
<th>Professor</th>
<th>Employee</th>
<th>University</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Employee</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Relationship: Aggregation

- A special form of association that models a whole-part relationship between an aggregate (the whole) and its parts.

```
<table>
<thead>
<tr>
<th>Whole</th>
<th>Part</th>
</tr>
</thead>
<tbody>
<tr>
<td>Student</td>
<td>Schedule</td>
</tr>
</tbody>
</table>
```

Relationship: Composition

- A form of aggregation with strong ownership and coincident lifetimes.
  - The parts cannot survive the whole/aggregate.

```
<table>
<thead>
<tr>
<th>Whole</th>
<th>Part</th>
</tr>
</thead>
<tbody>
<tr>
<td>Human</td>
<td>Heart</td>
</tr>
</tbody>
</table>
```

Association: Multiplicity and Navigation

- Multiplicity defines how many objects participate in a relationship.
  - The number of instances of one class related to ONE instance of the other class.
  - Specified for each end of the association.
- Associations and aggregations are bi-directional by default, but it is often desirable to restrict navigation to one direction.
  - If navigation is restricted, an arrowhead is added to indicate the direction of the navigation.
**Association: Multiplicity**

- Unspecified
- Exactly one
- Zero or more (many, unlimited)
- One or more
- Zero or one
- Specified range
- Multiple, disjoint ranges

**Example: Multiplicity and Navigation**

![Diagram of Student and Schedule multiplicity](image)

**Relationship: Dependency**

- A relationship between two model elements where a change in one may cause a change in the other
- None-structural, “using” relationship
Relationship: Generalization

• A relationship among classes where one class shares the structure and/or behavior of one or more classes
• Defines a hierarchy of abstractions in which a subclass inherits from one or more super classes
  – Single inheritance
  – Multiple inheritance
• Generalization is an “is-a-kind of” relationship

Example: Single Inheritance

• One class inherits from another

```
Superclass (parent)  Ancestor
  Account
    balance
    name
    number
    Withdraw()
    CreateStatement()
  Checking
    Withdraw()
  Savings
    (savings)
    Withdraw()
```

Subclasses  Generalization Relationship  Descendents

Multiple Inheritance

• A class can inherit from several other classes
What Gets Inherited?

- A subclass inherits its parent’s attributes, operations, and relationships
- A subclass may:
  - Add additional attributes, operations, relationships
  - Redefine inherited operations (use caution)
- Common attributes, operations, and/or relationships are shown at the highest applicable level in the hierarchy

Example: What Gets Inherited

<table>
<thead>
<tr>
<th>Superclass (parent)</th>
<th>GroundVehicle</th>
<th>Person</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>weight</td>
<td>owner</td>
</tr>
<tr>
<td></td>
<td>licenseNumber</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Subclasses</th>
<th>Car</th>
<th>Truck</th>
<th>Trailer</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>size</td>
<td>tonnage</td>
<td>length</td>
</tr>
</tbody>
</table>

Relationship: Realization

- One classifier serves as the contract that the other classifier agrees to carry out
- Found between
  - Interfaces and the classifiers that realize them
  - Use cases and the collaborations that realize them

Canonical form
Strengths of Object Orientation

- A single paradigm
- Facilitates architectural and code reuse
- Models more closely reflect the real world
  - More accurately describe corporate data and processes
  - Decomposed based on natural partitioning
  - Easier to understand and maintain
- Stability
  - A small change in requirements does not mean massive changes in the system under development

A simple Sales Order Example

Class Diagram for the Sales Example
Effect of Requirements Change

Suppose you need a new type of shipping vehicle

Stereotypes

- Classify and extend the UML notational elements
- Define a new model element in terms of another model element
- May be applied to all modeling elements
- Represented with name in guillemets or as a different icon

Example: Stereotypes
Notes

• A note can be added to any UML element
• Notes may be added to add more information to the diagram
• It is a ‘dog eared’ rectangle
• The note may be anchored to an element with a dashed line

Properties

• A property, or specific attribute, of a UML element
• Also called tagged values
• Some properties are defined by UML
  – Persistence
  – Location (e.g., client, server)
• Properties can be created by UML modelers for any purpose

Constraints

• Supports the addition of new rules or modification of existing rules