Introduction to Object Orientation: in UML flavors

Fall 2004, CSC532
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?

- A model that includes most important aspects of a given problem while ignoring less important details

- An example of an order processing abstraction
What is Encapsulation

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

• The breaking up of something complex into manageable pieces
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

• 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)

Professor Clark
Name: Joy Clark
Employee ID: 567138
Date hired: March 21, 1987
Status: Tenured
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).

Registration System

Assign Prof. Clark
(Returns: confirmation)

Algebra 101 Course
An Object Has Identity

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

Prof. “J. Clark”
Teaches algebra

Prof. “J. Clark”
Teaches algebra

Prof. “J. Clark”
Teaches algebra
Representing Objects

- An object is represented as rectangles with underlined names

- **ProfessorClark**
  - Class Name Only
  - Class and Object Name

- **Professor**
  - Object Name Only
Example: Objects

- Intro to OO 180
- World History 200
- Algebra 110
- English 101
- Geology 110
- Music History 200
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 emphasizes relevant characteristics
Sample Class

• Class

Properties
- Name
- Location
- Days offered
- Credit hours
- Start time
- End time

Course

Behavior
- Add a student
- Delete a student
- Get course roster
- Determine if it is full
Representing Classes

• A class is represented using a compartmented rectangle
• 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)
• The second and third sections may be suppressed if they need not be visible on the diagram

Class Compartments (cont.)
Example: Class

CourseOffering

- Intro to OO 180
- World History 200
- Algebra 110
- English 101
- Geology 110
- Geology 110
- Music History 200

Correct this.
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

Object

Attribute

CourseOffering

Attribute Value

:CourseOffering

Number=110
startTime=900
endTime=1100

:CourseOffering

Number=104
startTime=1300
endTime=1500
What is Operation?

Class

:CourseOffering

- 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>>

<table>
<thead>
<tr>
<th>Shape</th>
<th>Tube</th>
</tr>
</thead>
<tbody>
<tr>
<td>Draw</td>
<td>Pyramid</td>
</tr>
<tr>
<td>Move</td>
<td>Cube</td>
</tr>
<tr>
<td>Scale</td>
<td></td>
</tr>
<tr>
<td>Rotate</td>
<td></td>
</tr>
</tbody>
</table>
Interface Representations

Elided/Iconic Representation
(“lollipop”)

Canonical (Class/Stereotype) presentation

<<interface>>
Shape

- Tube
- Pyramid
- Cube

- 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
A component may be

- A source code component
- A run time component or
- An executable component

Source File Name

<<DLL>> Component Name

<<EXE>> Executable Name
• Visualize compilation dependencies between source code files

Source Code Components

Account.h

Account.cpp

Report.cpp
• Visualizing all of the pieces of an executable release, their interfaces, and their relationships
• Interfaces can be realized by components
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
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.

```
Current Interface

Design Model

<<subsystem>>
Component Name

Component Interface

Implementation Model

Component Name
```
Relationships

• Association
  • Aggregation
  • Composition
• Dependency
• Generalization
• Realization
Relationships: Association

- Models a semantic connection among classes
  - Association name and/or Role name

```
Class  Association  Role Names
Professor  Works for  Employee, Employer
University

Professor  Works for  University
```

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

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

1
0..*
 *
1..*
0..1
2..4
2,4..6
Example: Multiplicity and Navigation

```
Student      | 1 | 0..* |
            v
Schedule
```

Multiplicity

Navigation
A relationship between two model elements where a change in one may cause a change in the other.

None-structural, "using" relationship.
• 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

<table>
<thead>
<tr>
<th>Ancestor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Account</td>
</tr>
<tr>
<td>balance</td>
</tr>
<tr>
<td>name</td>
</tr>
<tr>
<td>number</td>
</tr>
<tr>
<td>Withdraw()</td>
</tr>
<tr>
<td>CreateStatement()</td>
</tr>
</tbody>
</table>

**Superclass** (parent)

<table>
<thead>
<tr>
<th>Checking</th>
</tr>
</thead>
<tbody>
<tr>
<td>Withdraw()</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Savings</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetInterest()</td>
</tr>
<tr>
<td>Withdraw()</td>
</tr>
</tbody>
</table>

**Subclasses**

**Descendents**

**Generalization Relationship**
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

Superclass (parent)

GroundVehicle
- weight
- licenseNumber
- register()

Subclasses

Car
- size

Truck
- tonnage
- getTax()

Generalization

Person
- owner
  - 0..*
  - 1

Trailer

0..*
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

- Use cases and the collaborations that realize them
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

Order → Product → Ship via

Order

Product

Ship via
Class Diagram for the Sales Example

- Sale
  - Seller
    - Salesperson
      - Corporate
  - Buyer
    - Customer
      - Individual
    - Product
      - Truck
      - Train
  - Item sold
  - Shipping mechanism
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

\[
\text{MyBoundaryClass}
\]
Example: Stereotypes

<<boundary>>

<<trace>>

DesignClass

<<Processor>>
Processor #1

<<Processor>>
Processor #1
• 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

PersistentClass
{persistence}

anObject: ClassA
{location=server}
• Supports the addition of new rules or modification of existing rules
Group exercise

Design class diagram for paid e-mail services

• must include relationships
• Ask questions if you need more input

– 5-10 min for customer interviews
– 10-15 min for team to work on the solution