Software Engineering
G22.2440-001

Session 7 – Sub-Topic Presentation 6
Introduction to OOAD Modeling and UML

Dr. Jean-Claude Franchitti

New York University
Computer Science Department
Courant Institute of Mathematical Sciences
Agenda

- Concepts of OO
- Review of Object Technology
- Review of SDLC
- OOAD Modeling and UML
Part I

Concepts of OO
Objectives: Concepts of Object Orientation

• Review the basic principles of object orientation
• Review the basic concepts and terms of object orientation and the associated UML notation
• Review the strengths of object orientation
• Review some basic UML modeling notation
Best Practices Implementation

• Object Technology helps implement these Best Practices.
  – Develop iteratively: tolerates changing requirements, integrates elements progressively, facilitates reuse.
  – Use component-based Architectures: architectural emphasis, component-based development.
  – Model visually: easy understanding, easy modification.
What Is Object Technology?

- **Object Technology**
  - A set of principles guiding software construction together with languages, databases, and other tools that support those principles. (*Object Technology - A Manager’s Guide*, Taylor, 1997)
Strengths of Object Technology

• A single paradigm
  – A single language used by users, analysts, designers, and implementers

• Facilitates architectural and code reuse

• Models more closely reflect the real world
  – More accurately describes corporate entities
  – 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

• Adaptive to change
What Is a Model?

- A model is a simplification of reality.
Why Do We Model?

• We build models to better understand the system we are developing.

• Modeling achieves four aims. Modeling:
  – Helps us to visualize a system as we want it to be.
  – Permits us to specify the structure or behavior of a system.
  – Gives us a template that guides us in constructing a system.
  – Documents the decisions we have made.

• We build models of complex systems because we cannot comprehend such a system in its entirety.
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 an entity with a well-defined boundary and identity that encapsulates **state** and **behavior**.
  – State is represented by attributes and relationships.
  – Behavior is represented by operations, methods, and state machines.
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.

Name: Jane Doe
Employee ID: 567138
Date Hired: July 25, 1991
Status: Tenured
Discipline: Finance
Maximum Course Load: 3 classes

Professor Doe
An Object Has Behavior

- Behavior determines how an object acts and reacts.
- The visible behavior of an object is modeled by the set of messages it can respond to (operations the object can perform).

Professor Doe's behavior:
- Submit Final Grades
- Accept Course Offering
- Take Sabbatical
- Maximum Course Load: 3 classes
An Object Has Identity

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

Professor “Jane Doe” teaches Biology

Professor “Jane Doe” teaches Biology
Representing Objects in the UML

• An object is represented as a rectangle with an underlined name.

Professor J Clark

J Clark : Professor

Named Object

: Professor

Unnamed Object
Basic Principles of Object Orientation

Object Orientation

Abstraction
Encapsulation
Modularity
Hierarchy
What Is Abstraction?

• The essential characteristics of an entity that distinguish it from all other kinds of entities
• Defines a boundary relative to the perspective of the viewer
• Is not a concrete manifestation, denotes the ideal essence of something
Example: Abstraction

Student

Professor

Course Offering (9:00 AM, Monday-Wednesday-Friday)

Course (e.g. Algebra)
What Is Encapsulation?

- Hide implementation from clients.
  - Clients depend on interface.

*Improves Resiliency*
Encapsulaton Illustrated

- needs to be able to teach four classes in the next semester.

SetMaxLoad(4)
What Is Modularity?

- Modularity is the breaking up of something complex into manageable pieces.
- Modularity helps people to understand complex systems.
Example: Modularity

Course Registration System

Billing System

Course Catalog System

Student Management System
What Is Hierarchy?

Elements at the same level of the hierarchy should be at the same level of abstraction.
What Is a Class?

• A class is a description of a set of objects that share the same attributes, 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.
Representing Classes in the UML

• A class is represented using a rectangle with compartments.

Professor
- name
- employeeID : UniqueId
- hireDate
- status
- discipline
- maxLoad

+ submitFinalGrade()
+ acceptCourseOffering()
+ setMaxLoad()
+ takeSabbatical()
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 are grouped into classes.

Professor
What Is an Attribute?

• An attribute is a named property of a class that describes a range of values that instances of the property may hold.
  – A class may have any number of attributes or no attributes at all.

<table>
<thead>
<tr>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Student</td>
</tr>
<tr>
<td>- name</td>
</tr>
<tr>
<td>- address</td>
</tr>
<tr>
<td>- studentID</td>
</tr>
<tr>
<td>- dateOfBirth</td>
</tr>
</tbody>
</table>
What Is an Operation?

• An operation is the implementation of a service that can be requested from any object of the class to affect behavior.

• A class may have any number of operations or none at all.

<table>
<thead>
<tr>
<th>Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Student</td>
</tr>
<tr>
<td>+ get tuition()</td>
</tr>
<tr>
<td>+ add schedule()</td>
</tr>
<tr>
<td>+ get schedule()</td>
</tr>
<tr>
<td>+ delete schedule()</td>
</tr>
<tr>
<td>+ has pre-requisites()</td>
</tr>
</tbody>
</table>
What Is Polymorphism?

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

**OO Principle:** Encapsulation
Example: Polymorphism

Get Current Value

Stock
Bond
Mutual Fund
What is an Interface?

- Interfaces formalize polymorphism
- Interfaces support “plug-and-play” architectures

```
<<Interface>>
Shape

draw()
move()
scale()
rotate()
```

- Tube
- Pyramid
- Cube

Realization relationship
How Do You Represent An Interface?

Elided/Iconic Representation ("lollipop")

 Canonical Representation (Class/Stereotype)

(stay tuned for realization relationships)
What Is a Package?

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

- A package can be used
  - To organize the model under development.
  - As a unit of configuration management.
What is a Subsystem?

• A combination of a package (can contain other model elements) and a class (has behavior)
• Realizes one or more interfaces which define its behavior

OO Principles: Encapsulation and Modularity

(Stay tuned for realization relationship)
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 components or
  – An executable component

*OO Principle: Encapsulation*
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

**OO Principles: Encapsulation and Modularity**
What Is an Association?

- The semantic relationship between two or more classifiers that specifies connections among their instances
  - A structural relationship, specifying that objects of one thing are connected to objects of another

```
<<entity>>
Student

<<entity>>
Schedule

<<entity>>
Course
```
What Is Multiplicity?

- Multiplicity is the number of instances of one class relates to ONE instance of another class.
- For each association, there are two multiplicity decisions to make, one for each end of the association.
  - For each instance of Professor, many Course Offerings may be taught.
  - For each instance of Course Offering, there may be either one or zero Professor as the instructor.

![Multiplicities Diagram]

- **Professor**
  - 0..1 instances
  - Instructor

- **CourseOffering**
  - 0..* instances
  - Instructor
Multiplicity Indicators

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

<table>
<thead>
<tr>
<th>Indicator</th>
<th>Multiplicity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unspecified</td>
<td></td>
</tr>
<tr>
<td>Exactly one</td>
<td>1</td>
</tr>
<tr>
<td>Zero or more (many, unlimited)</td>
<td>0..*</td>
</tr>
<tr>
<td>Zero or one (optional scalar role)</td>
<td>*</td>
</tr>
<tr>
<td>One or more</td>
<td>1..*</td>
</tr>
<tr>
<td>Zero or one (optional scalar role)</td>
<td>0..1</td>
</tr>
<tr>
<td>Specified range</td>
<td>2..4</td>
</tr>
<tr>
<td>Multiple, disjoint ranges</td>
<td>2, 4..6</td>
</tr>
</tbody>
</table>
What Is Aggregation?

- An aggregation is a special form of association that models a whole-part relationship between an aggregate (the whole) and its parts.
  - An aggregation “Is a part-of” relationship.
- Multiplicity is represented like other associations.
What Is Navigability?

- Indicates that it is possible to navigate from a associating class to the target class using the association

```plaintext
<<control>>
RegistrationController

<<entity>>
Schedule

<<entity>>
CourseOffering
```
Relationships: Dependency

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

• Non-structural, “using” relationship
What Is 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 superclasses
  – Single inheritance
  – Multiple inheritance
• Is an “is a kind of” relationship
Example: Single Inheritance

• One class inherits from another

Superclass (parent)

Ancestor

Account
- balance
- name
- number

+ withdraw()
+ createStatement()

Generalization Relationship

Subclasses

Savings

Descendents

Checking
Example: Multiple Inheritance

- A class can inherit from several other classes.

Use multiple inheritance only when needed and always with caution!
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

*Inheritance leverages the similarities among classes*
Example: What Gets Inherited

**Superclass (parent)**

- GroundVehicle
  - weight
  - licenseNumber
  - register()

**Subclass**

- Car
  - size

- Truck
  - tonnage
  - getTax()

- Trailer

**generalization**

**owner**

0..* 1
What Is 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-case Realization

Canonical form

Component

Subsystem

Class

Interface

Elided form
What Are Stereotypes?

• Stereotypes define a new model element in terms of another model element.

• Sometimes, you need to introduce new things that speak the language of your domain and look like primitive building blocks.
What Are Notes?

- A comment that can be added to include more information on the diagram
- May be added to any UML element
- A ‘dog eared’ rectangle
- May be anchored to an element with a dashed line

MaintainScheduleForm

There can be up to one MaintainScheduleForm per user session.
Tagged Values

• Extensions of the properties, or specific attributes, of a UML element

• 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}
Review: Concepts of Object Orientation

- What are the four basic principles of object orientation? Provide a brief description for each.
- What is an object and what is a class? What is the difference between the two?
- What is an attribute?
- What is an operation?
- What is an interface? What is polymorphism?
Review: Concepts of Object Orientation (continued)

• What is a package?
• What is a subsystem? How does it relate to a package? How does it relate to a class?
• Name the four basic UML relationships and describe each.
• Describe the strengths of object orientation
• What are stereotypes?
Part II

Review of Object Technology: Basics of Objects, Definitions, Effects on SDLC
Basics of Objects

- The newer object world concentrates on: Encapsulation, Message Passing & Polymorphism, and Classes & Inheritance
- The older structured world concentrated on sequence, selection, and iteration
Basics of Objects

• First what is an object?
• A software packet that abstracts the salient behavior and characteristics of a real object into a software package that simulates the real object
Basics of Objects

Examples of objects:

• Number
• Drop Down List Box
• Window
• ATM (Automated Teller Machine)
• Customer
# Basics of Objects

Types of objects:

<table>
<thead>
<tr>
<th>Example</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number</td>
<td>primitive</td>
</tr>
<tr>
<td>List Box</td>
<td>control</td>
</tr>
<tr>
<td>Window</td>
<td>GUI</td>
</tr>
<tr>
<td>ATM</td>
<td>system</td>
</tr>
<tr>
<td>Customer</td>
<td>business</td>
</tr>
</tbody>
</table>
## Basics of Objects

Characteristic of objects:

<table>
<thead>
<tr>
<th>Example</th>
<th>Characteristic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number</td>
<td>5</td>
</tr>
<tr>
<td>List Box</td>
<td>location</td>
</tr>
<tr>
<td>Window</td>
<td>size</td>
</tr>
<tr>
<td>ATM</td>
<td>amount on hand</td>
</tr>
<tr>
<td>Customer</td>
<td>balance</td>
</tr>
</tbody>
</table>
Basics of Objects

Behavior of objects:

<table>
<thead>
<tr>
<th>Example</th>
<th>Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number</td>
<td>add</td>
</tr>
<tr>
<td>Drop Down List Box</td>
<td>drop down</td>
</tr>
<tr>
<td>Window</td>
<td>open</td>
</tr>
<tr>
<td>ATM</td>
<td>give cash</td>
</tr>
<tr>
<td>Customer</td>
<td>pay bill</td>
</tr>
</tbody>
</table>
# Basics of Objects

Encapsulation of objects:

<table>
<thead>
<tr>
<th>(nouns)</th>
<th>(adjectives)</th>
<th>(verbs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example</td>
<td>Characteristic</td>
<td>Behavior</td>
</tr>
<tr>
<td>• Number</td>
<td>5</td>
<td>add</td>
</tr>
<tr>
<td>• List Box</td>
<td>location</td>
<td>drop down</td>
</tr>
<tr>
<td>• Window</td>
<td>size</td>
<td>open</td>
</tr>
<tr>
<td>• ATM</td>
<td>amount on hand</td>
<td>give cash</td>
</tr>
<tr>
<td>• Customer</td>
<td>balance</td>
<td>pay bill</td>
</tr>
</tbody>
</table>
Basics of Objects

The Three Keys to Object Technology

• A software object has both characteristics and behavior **encapsulated** in it
• Software objects communicate by **messages**
• Software objects can **inherit** characteristics and behavior just like many real objects
Basics of Objects

Encapsulation:

• The containment of the data behind a software membrane consisting of methods. The data can only be accessed through the encapsulated behavior.
Basics of Objects

Message:

• A signal from a client object requesting services from a server object
• The message may contain arguments
• The server object may return a response
Basics of Objects

Polymorphism:

- Messages mean different things to different objects:
  - Print Word document
  - Print Excel document

- This means different implementations can be hidden behind a common interface
Basics of Objects

Class:

- A Collection of like objects
- A template for defining new object instances
- The behaviors reside in the class
- Behavior is implemented by methods
Classes & Inheritance

Instance:
- A term used to refer to a software object
- It is a common synonym of object
Classes & Inheritance

Inheritance:

• A technique to allow classes to use a parent classes methods and data

• Inheritance can have many levels
The Ten Big Definitions

• Object - A software package
• Method - An objects procedure
• Message - A signal from one object to another
• Class - A template to create objects
• Subclass - A special case of a class
The Ten Big Definitions

• Instance - An objects other name
• Inheritance - A mechanism to use another objects innards
• Encapsulation - Data & methods Together
• Abstraction - Capturing behaviors & characteristics
• Polymorphism - Hiding alternative methods behind a common interface
Effect on SDLC

- SDLC - System Development Life Cycle
- A system must be developed that encompasses the theory of objects
- The program code must be divided into methods that are placed in objects with business meanings
Effect on SDLC

- These objects are called domain objects
- Together they make up the business or domain model
- If model is created correctly the system is very easy to maintain
Effect on SDLC

Many languages now support this model
• Smalltalk
• C++
• Java
Effect on SDLC

Many CASE tool vendors also support this model

• Rational Corporation * [http://www.rational.com/](http://www.rational.com/)
• Peter Coad [http://www.powerj.com/](http://www.powerj.com/)

* We will be using Rational Rose (or a similar tool for business/application modeling)
Effect on SDLC

• “The most single important ability in object- oriented analysis and design is to skillfully assign responsibilities to software components”

• “… a close second in terms of importance is finding suitable objects or abstractions”

• Craig Larman - author of Applying UML & Patterns
Summary

• If you have not guessed it by now, the points are:
  – Encapsulation
  – Message Passing & Polymorphism
  – Classes & Inheritance

• The ten big definitions must be understood

• The outcome of the SDLC is a business object model
Part III

Review of SDLC
What is a SDLC

System Development Life Cycle:
• It is developing a computer system
• It concerns a process which takes from two months to two years
• This is called a system development life cycle
What is a SDLC

There are two forms:
• Rapid (Prototype)
  – Plan and Elaborate
  – Developmental Cycle 1
  – Developmental Cycle 2
• And Waterfall (classical)
What is a SDLC

• Waterfall (classical)
  – Requirements
  – Analysis
  – Design
  – Construction
  – Implementation
What is a SDLC

Both forms are followed by a maintenance cycle:

- Maintenance is the most expensive part
- If all the steps are done carefully maintenance is reduced
- For maintenance to be effective, documentation must exist
What is a SDLC

The system really consists of two parts:

• Model
  – Prototypes
  – Diagrams and supporting Documents

• System
  – Hardware
  – Software
Definitions

Prototype:

• A first system usually done with a rapid development tool
• Usually has limited functionality
• Users can see results very quickly
Definitions

• Planning
• The process of gathering what is needed to solve a business problem
• Includes a feasibility study
• Includes project steps
Definitions

- Analysis
- The process of determining detail requirements in the form of a model
Definitions

• Design
• The process of drawing blueprints for a new system
Definitions

• Construction
• The actual coding of the model into a software package
• Uses one of three languages:
  – Java
  – Smalltalk
  – C++
Definitions

- Implementation
- Doing whatever is necessary to startup a system
- Includes:
  - Database
  - Networks
  - Hardware configuration
Definitions

• Maintenance
• Doing whatever is necessary to keep a system running
• Includes:
  – repairs to correct errors
  – enhancements to accommodate changes in requirements
Deliverables

• Deliverables consist mainly of diagrams and their supporting documentation
• For example:
  – Models that emphasize dynamics
  – Models that emphasize structure
  – Models can be used for specifying the outcome of analysis
  – Models can be used for specifying the outcome of design
Deliverables

Planning:

• System Functions
• A simple list of each requirement a system must do
• For example:
  – record video rental
  – calculate fine
Deliverables

Planning:
- System Attributes
- A simple property describing each requirement of a system
- For example:
  - record video rental under 15 seconds
  - calculate fine and return response in 5 seconds
Deliverables

• Planning:
  Environmental Diagram

Rent Video

Pay Employees

Video Store
Information System

Clerk
Deliverables

Planning:

• Prototype

• Recall it is a first system usually done with a rapid development tool

• Since users can see results very quickly they will pay attention

• Final product is seldom created in same tool as the prototype
Deliverables

Analysis:
• Use case
• Shows the dynamics between the users (actors) of the system and the system itself
• This is a narrative representation
Deliverables

Analysis:

• Conceptual Diagram
• Shows the structure of the objects and their relationships
• This is a graphical representation
Deliverables

Analysis:
• System Sequence Diagram
• Shows the dynamics between the users (actors) of the system and the system itself
• This is a graphical representation
Deliverables

Analysis:

• Contracts

• Shows the state of each object before each action

• This is a narrative representation
Deliverables

Design:

• Interaction Diagram
• Shows the interaction between objects
• This is a graphic representation
• It is a dynamic blueprint
Deliverables

Design:
• Class Diagram
• Shows the structure between objects
• Shows the structure inside objects
• This is a graphic representation
• It is a static blueprint
UML provides a standard for the following artifacts:

• Use Case (Dynamic Analysis Output)
• Conceptual Model (Static Analysis Output)
• Interaction Diagram (Dynamic Design Blueprint)
• Class Diagram (Static Design Blueprint)
Part IV

*OO Analysis and Design and UML*
What it is

• Environmental Diagram

Customer ➔ Rent Video ➔ Video Store Information System ➔ Clerk

Pay Employees ➔ Payroll Clerk
What it is

- A picture containing all the important players (Actors)
- Includes players both inside and outside of the system
- Actors are a critical component
- External events are a second critical component
Creating the Diagram

• To create an environmental diagram
• 1. Identify all the initiating actors
• 2. Identify all the related external events associated with each actor
Why it is used

- A diagram is needed to show the context or scope of the proposed system
- At this time actors and external events are the critical components
- It is helpful to include all the participants as well
Creating the Diagram

3. Identify all the participating Actors

These actors may be inside (internal) or outside (external) to the system
Creating the Diagram

• Examples of an internal actor
  – Clerk who enters the purchase into a Point of Sale terminal
  – Clerk who places paper in the printer
  – Accountant who audits report
Creating the Diagram

• Examples of an external actor
  – Accountant who audits report
  – A credit authorizing service
  – A DMV check for renting a car
Creating the Diagram

• 4. Draw a cloud
• 5. Then draw initiating actors on the left of the cloud
• 6. Then draw participating external actors outside the cloud
• 7. Then draw participating internal actors inside the cloud
• Recall actors are stick figures
Creating the Diagram

• 8. Lastly connect the initiation actors to the cloud
• 9. Label each connection with an external event name
• 10. It is not necessary to label connections to the participating external actors; just connect them
Creating the Diagram

• An example from the textbook
• First draw a cloud
Creating the Diagram

- An example from the textbook
- Label the system

POST Application
Creating the Diagram

• An example from the textbook
• Insert and label the initiating actor

[Diagram showing a Customer initiating a POST Application]
Creating the Diagram

- An example from the textbook
- Connect the actor with an external event

Customer ➔ Buy Items ➔ POST Application
Creating the Diagram

• An example from the textbook
• Insert and label any internal participating actors

Customer ➔ Buy Items ➔ POST Application ➔ Cashier
Creating the Diagram

• An example from the textbook
• Insert and label any external participating actors

Customer → Buy Items → POST Application → Cashier → Manager
Summary

• The environmental diagram is a useful to depict a lot of useful information
• At a glance it shows all the critical entities (actors) that interact with the system