Introduction to Unified Modeling Language (UML)

By Rick Mercer with help from

The Unified Modeling Language (UML)

- UML or Unified Modeling Language comes from Rumbaugh, Booch, and Jacobson (the three amigos) who combined efforts to standardize on one modeling language.
- This is primarily a graphical communication mechanism for developers and customers.
- We will learn some, but not all, of the UML.
  - It is very complex, few understand all of it.
The main purpose of UML is to
- support communication about the analysis and design of the system being developed
- support the movement from the problem domain in the "world" to the solution domain in the machine

Two views of the same system
- one view has diagrams
- source code is another view

Sometimes it's nice to look at the overview
- Reverse engineer your code with a UML tool to see how your code looks in UML
UML is a Modeling Language

♦ UML
  - graphical notation to describe software design
  - has rules on how to draw models of
    ♦ classes
    ♦ associations between classes
    ♦ message sends between objects
  - has become the de facto industry standard
    ♦ Not official, but everyone uses it
  - like a blueprint to show what is going on during analysis, design and implementation
    ♦ Some Projects require UML documentation
The Unified Modeling Language User Guide, Booch, Rumbaugh, Jacobson states:

The UML is a language for

- visualizing
- specifying
- constructing
- documenting

the artifacts of a software intensive system
First up: Class Diagrams

◊ A class diagram
  - expresses class definitions to be implemented
  - lists name, attributes, and methods for each class
  - shows relationships between classes

◊ UML allows different levels of detail on both the attributes and the methods of one class
  - could be just the class name in a rectangle
  - or like the general form shown on the next slide
<table>
<thead>
<tr>
<th>Software Specification (Class Name)</th>
</tr>
</thead>
<tbody>
<tr>
<td>attribute</td>
</tr>
<tr>
<td>attribute : type</td>
</tr>
<tr>
<td>attribute : type = initial value</td>
</tr>
<tr>
<td>classAttribute</td>
</tr>
<tr>
<td>derivedAttribute</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>method1()</td>
</tr>
<tr>
<td>method2(parameter : Type) : return type</td>
</tr>
<tr>
<td>abstractMethod()</td>
</tr>
<tr>
<td>+publicMethod()</td>
</tr>
<tr>
<td>-privateMethod()</td>
</tr>
<tr>
<td>#protectedMethod()</td>
</tr>
<tr>
<td>classMethod()</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>
AccountCollection

- allAccounts : HashMap

+AccountCollection ()
+getAccountWithID (ID: String) : Account
+add(accountToAdd: Account) : boolean
+iterator() : Iterator

Note: iterator is needed by the bank manager
Sterotypes

- Stereotype is a UML element that allows designers to extend the UML vocabulary
  - Often used to distinguish an abstract class name from an interface, both of which are written in *italic*

<table>
<thead>
<tr>
<th>&lt;&lt;interface&gt;&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Iterator</em></td>
</tr>
<tr>
<td>+hasNext(): boolean</td>
</tr>
<tr>
<td>+next(): Object</td>
</tr>
<tr>
<td>+remove(): void</td>
</tr>
</tbody>
</table>
Different levels of detail

♦ Tips for modeling
  - Express as much or as little detail as needed
  - Often, a rectangle with a name is enough
    ♦ perhaps a method or an attribute clarifies
  - Simple is good
  - Sketches on paper or white board are effective
Relationships

♦ Three Relationships in UML
  1) Dependency
  2) Association
  3) Generalization

♦ Understanding these relationships is more important than the lines that UML uses
1) Dependency: A Uses Relationship

Dependencies
- occurs when one object depends on another
- if you change one object's interface, you need to change the dependent object
- arrow points from dependent to needed objects
2) Association: Structural Relationship

- **Association**
  - a relationship between classes indicates some meaningful and interesting connection
  - Can label associations with a hyphen connected verb phrase which reads well between concepts

![Diagram showing association between Jukebox and JukeboxAccountCollection with a method getAccountWithID]
Associations

♦ Associations imply
  - our knowledge that a relationship must be preserved for some time (0.01 ms to forever)
  ♦ Between what objects do we need to remember a relationship?
    • Does a Transaction need to remember Account?
    • Would AccountCollection need to remember Accounts?

```
AccountCollection 1 Stores 0..* Account
```
Notation and Multiplicity Adornments

♦ UML Association:
- a line between two concepts and a name
- they are bi-directional
- can have a multiplicity
- exist in class diagrams

Multiplicity adornments

T

1..*

T

1..52

T

* 

zero or more; "many"

one or more

one to fifty two

exactly five
Association Names

♦ Read this Type-VerbPhrase-Type (POST is a Point of Sale Terminal)

♦ Not shown here: Attributes and Methods

♦ This just shows associations between objects
Aggregation: A Special Association

- Aggregation: whole/part relationships
  - An association that models HAS-A relationships
  - The objects can exist independently of each other
  - No one object is more important than the other
  - Place an open diamond on the whole
  - School contains a collection of Student objects

  ![Diagram](image)

- In Java, this is the same as an association, an instance variable, no special syntax
Composition: A Special Association

❖ Composition: Stronger relationship
  - One can not exist without the other
  - If the school folds, students live on
    ❖ but the departments go away with the school
  - If a department closes, the school can go on \( AIC^* \) e.g.

❖ Model aggregation or composition? When in doubt, use association (just a simple line) \( don't \ sweat \ the \ diff \ in \ 335 \)
Example UML Class Diagrams

One view of the **Composite** design pattern
Part of internet shopping
Active Learning

♦ In teams of two or three, using examples in these slides and the names of the objects we discovered in the Five Card Draw, complete a class diagram that shows a design of a software system to model the game as it would exist on a gambling website.
Assignment #5, due 3-Sep 4:45 pm

♦ Draw rectangles for classes
♦ Include the class name
  - In at least five classes, write one
♦ Draw associations between objects
  - missing diamonds and arrows are okay
    ♦ one solid line will suffice
  - an association implies there will be some relationship between the objects as some point
♦ There is a separate handout for this
  - You must be in class to get credit
Sequence Diagrams

- Interaction diagrams describe how groups of objects collaborate in some behavior
- The UML defines several forms of interaction diagram, the most common is the sequence diagram
- A class diagram shows a fixed view of a system
- A sequence diagram represents a dynamic view of a system by capturing message sends over time
  - Can document a scenario such as
    - Dealer deals cards to all players
    - Withdraw Money when there is enough balance
    - Withdraw Money when there is not enough balance
Sequence Diagrams

♦ Not good at showing details of algorithms such as loops and conditional
♦ Good at showing the calls between participants
♦ Gives a good picture about which participants are doing which processing
More Distributed Control Shown here
Syntax

♦ Objects are lined up on top in rectangles
♦ Object names: CardReader
♦ Dashed lines represent lifetime of objects
♦ Rectangles are activation lines
  - When the object is "alive"
  - Activation bar of the receivers of the message is smaller than the sender's activation bar
♦ Not much detail written
Another Example

Scenario: The user tries to use an ATM, but the account is not known
Scenario: The user successfully withdraws money from an ATM