INF 111 / CSE 121: Software Tools and Methods

Lecture Notes for Fall Quarter, 2007
Michele Rousseau
Set 16

(Some notes adapted from Susan E. Sim & UML Distilled)

Announcements

- **Quiz #3- Friday 11/9**
  - All readings assigned since the last quiz
  - Plus the readings not covered on the last quiz:
    - Ch 2 from "The Mythical Man-Month"
    - Van Vliet Ch. 4
  - Lectures from 10/31 – 11/7
  - What will it cover?
    - Configuration Management
    - Intro to UML

Announcements (2)

- **Assignment #2 and Quiz #2 are in the Distribution Center**
  - Please submit re-grades to me in class by next Wednesday
- **Re-grades for Quiz #1 are completed and in the Distribution Center**
  - If you are not satisfied with the re-grades please check them and return them to me after class by next Wednesday.
- **Readings on UML**
  - Read: Van Vliet Ch. 12
    - Other info on UML that might be useful:
      - http://atlas.kennesaw.edu/~dbraun/csis4650/A&D/UML_tutorial/
    - Argo UML Info:
      - http://argouml.tigris.org/
  - Some books on UML:
    - Fowler (2004). UML Distilled: Third Edition: A Brief Guide to the Standard Object Modeling Language, Addison-Wesley, 2004
    - Larman (2005) .Applying UML and Patterns, Third Edition. Prentice Hall PTR, 2005
  - Another book on UML:
    - McLaughlin, Pollice & West (2006). Head First Object-Oriented Analysis & Design. O'Reilly, 2006.

Previously in INF 111…

- **OOAD & UML**

Today's Lecture

- **UML**
  - Class Diagrams
  - Use Case Diagrams

Types of UML Diagrams

Structure	Behavior
(6 types)	(4 types)
Class diagrams	Activity diagram
Object diagram	Use Case diagram
Package diagram	State machine diagram
Composite structure diagram	Interaction diagrams
Component diagram	**Sequence diagram**
Deployment Diagram	**Communication diagram**
	Interaction overview diagram
	Timing diagram

If the appropriate diagram is not part of UML
use it anyways

UML & the Software Process (Requirements)

- Use Cases
  - Describe how people interact with the system
- Class Diagram
  - Drawn from a conceptual perspective
  - Can build up a rigorous vocab of the domain
- Activity Diagram
  - Shows the workflow of the org.
    - Shows how s/w and human activities interact
  - Context for Use Cases
  - Details of complex Use Cases
- State Diagram
  - Shows states and events that change the state
    - Can be useful with interesting life cycles

Communication is key
Customers may not be familiar with S/W techniques

Break the rules is it enhances Communication

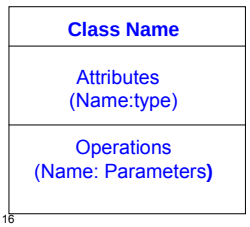
UML & the S/W Process (Design)

- **Class Diagrams**
  - From a software perspective
    - Show classes & how they interrelate
- **Sequence Diagrams**
  - For Common Scenarios
    - Pick most significant scenarios from Use Cases
    - Use CRC cards or sequence diagrams to determine how the software should behave
      - Class, Responsibilities, Collaborators (CRC) cards are index cards used to represent
        - the responsibilities of classes
        - interaction between the classes
- **Package Diagrams**
  - Show large-scale organization of the system
- **State Diagrams**
  - Used for classes with complex lifecycles
- **Deployment Diagrams**
  - Show the physical layout of the software

All of these can be used for design

Class Diagrams

"A ***Class Diagram*** describes the types of objects in the system and the various kinds of *static relationships* that exist among them"



Makes it easier to see the big picture

- Know what a class does at a glance

Attributes and Operations

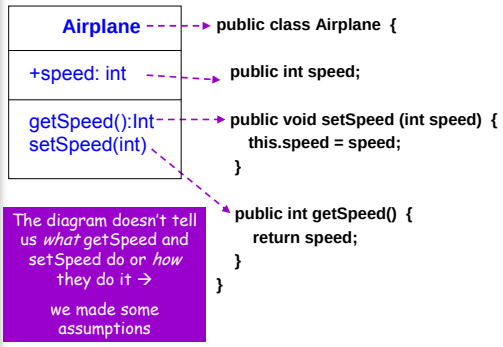
- **Attributes →**
  - Describes a property as a line of text within the class box
  - Attribute name corresponds to the name of a field in a programming language
  - ***Visibility Marker →***
    - **Denotes whether an attribute is…**
      - **Public (+) or Private (-)**
- **Operations →**
  - Actions that a class knows to carry out
  - *Corresponds* to methods on a class

Operation & Method are *not* the same thing

An Operation is the procedure declaration
A Method is the body of a procedure

- **Associations →**
  - Describe the relationship between two classes

Example of a Class

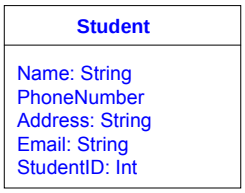


The diagram doesn't tell us *what* getSpeed and setSpeed do or *how* they do it →

we made some assumptions

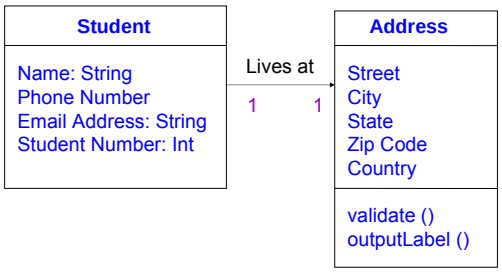
Example 2 of a Class

Different level of abstraction



Associations -- Notation

Associations



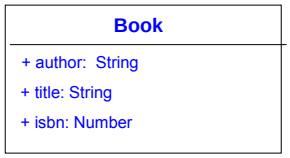
Properties → Attributes & Associations

- **Properties →**
  - A structural feature of a class (fields in a class)
  - Can be represented 2 ways: *Attributes* or *Associations*

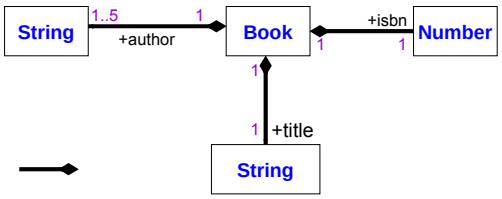
Attributes and Associations →
Different notations for the same thing

Example: Properties as Attributes

Simple Example



Properties as Associations



Attributes & Associations

- **Same properties → different notations**
- **When do you use which?**
  - Attributes for more simple properties (such as Booleans or Dates)
  - Associations for more significant properties (such as Orders or Customers)
- **Associations show more – such as multiplicities (covered in discussion)**

Some Basic Concepts

- **Generalization (AKA Inheritance)**
  - For all instances of a superclass…
    - The subclass *inherits…*
      - Attributes
      - Operations
      - Associations (we'll talk about these later)
    - Whatever is true for the superclass is true for the subclass

Inheritance lets you build classes based on other classes without having to duplicate or repeat code.

Example of Generalization

Airplane is the superclass for Jet.

Jet is the subclass

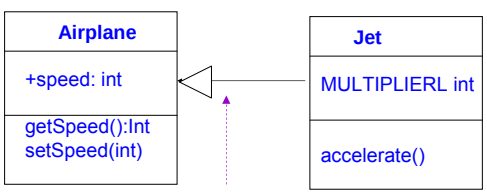
```
Public class Jet extends Airplane {
    private static final int MULTIPLIER =2;

   public Jet () {
        super();
    }
   public void setSpeed (int speed) {
      super.setSpeed(speed *
     MULTIPLIER)
    }
   public void () {
        super.setSpeed (getSpeed() *2);
   }
}
```

Jet extends from the Airplane class – that means it inherits all of Airplane's behavior

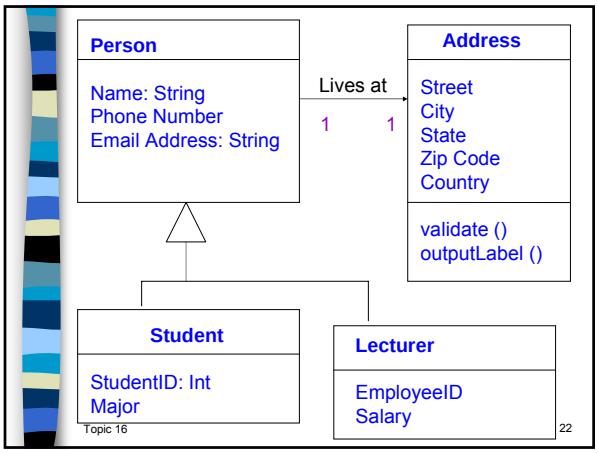
Jet can modify the behavior of the superclass' methods → it can also just call on them.

Note: getSpeed is not in here because Jet is not modifying it
→ You can still call getSpeed on Jet

Generalization notation



Generalization is notated using a big open arrow



Polymorphism

- **if an operation is applied to an object and there are several alternative classes that have the operation defined**

  → then the object to which the operation is applied always determines the operation that is executed.

IN OTHER WORDS..

- **Allows you to process objects differently depending on their data type or class**
  - Redefine methods for a derived class

Polymorphism (Example)