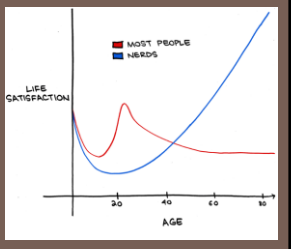
http://www.smbc-comics.com/index.php?db=comics&id=2436 via geeksaresexy.net




# CSE 331
# SOFTWARE DESIGN & IMPLEMENTATION
# UNIFIED MODELING LANGUAGE (UML)

Autumn 2011

## Announcement



- A5 will be due Tuesday November 29 @ 11:59PM
- There will be no A6 programming assignment
- Instead, I will on Wednesday November 30 hand out a "work-sheet style" assignment
- It will be graded, and it will focus some aspects of the course material that will be on the final
  - Especially material covered in class but not covered (well or at all) on the assignments
- Due date TBA



## Model: dictionary.com



1.a standard or example for imitation or comparison.

2.a representation, generally in miniature, to show the construction or appearance of something.
3.an image in clay, wax, or the like, to be reproduced in more durable material.

8.a pattern or mode of structure or formation.

10.a simplified representation of a system or phenomenon, as in the sciences or economics, with any hypotheses required to describe the system or explain the phenomenon, often mathematically.



## Claim



- There are dimensions of software systems that are not effectively described (describable) – or modeled – using programming languages (like Java)
- Examples
  - A file must be opened before it can be read
  - In a basic calculator, entering a binary operator shifts modes from "entering number" to "start new number"
  - When an instance of X announces event E, each method M (invoked by a listener for E) is executed sequentially
- That is, there are aspects of software that are – at best – described implicitly in a program: a language for modeling these aspects explicitly can help in these situations



## UML



- UML – the Unified Modeling Language – is by far the most widely known and used software modeling language
- It is "owned" by OMG (Object Management Group), which advertises about this open-standard:

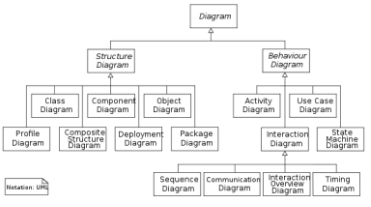
  ***Modeling is the designing of software applications*** before coding. Modeling is an Essential Part of large software projects, and helpful to medium and even small projects as well. A model plays the analogous role in software development that blueprints and other plans (site maps, elevations, physical models) play in the building of a skyscraper. Using a model, those responsible for a software development project's success can assure themselves that business functionality is complete and correct, end-user needs are met, and program design supports requirements for scalability, robustness, security, extendibility, and other characteristics, *before* implementation in code renders changes difficult and expensive to make. Surveys show that large software projects have a huge probability of failure - in fact, it's more likely that a large software application will fail to meet all of its requirements on time and on budget than that it will succeed. If you're running one of these projects, you need to do all you can to increase the odds for success, and modeling is the only way to visualize your design and check it against requirements before your crew starts to code.



## UML diagrams



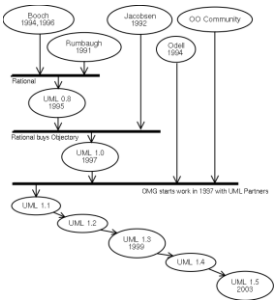
- At its heart, UML defines notations and meanings for a set of (object-oriented) software-related models
- These naturally overlap with entities in programs, although UML is not language-specific



## Whence "Unified?"

7

- Booch ⓘ, Jacobson ⓘ and Rumbaugh ⓘ had similar but competing approaches
- These were merged into a single approach represented by UML






## UML class diagrams

8

- What is a UML class diagram? What does it represent?
  - A picture of the classes in an OO system, their fields and methods, and connections between the classes that interact or inherit from each other
- What are some things not represented in a class diagram?
  - details of how the classes interact
  - algorithmic details; how particular behavior is implemented
  - trivial methods (get/set)
  - classes that come from libraries (ArrayList, etc.)



## Diagram of one class

9

- **class name** in top of box
  - write **<<interface>>** above interfaces' names
  - use *italics* for an *abstract class* name
- **attributes**
  - should include all fields of the object
  - also includes derived "properties"
- **operations** / methods
  - may omit trivial (get/set) methods
  - should not include inherited methods

| Rectangle |
|---|
| - width: int<br>- height: int<br>/ area: double |
| + Rectangle(width: int, height: int)<br>+ distance(r: Rectangle): double |

| Student |
|---|
| -name:String<br>-id:int<br>-totalStudents:int |
| #getID():int<br>+getName():String<br>~getEmailAddress():String<br>+getTotalStudents():int |



## Class attributes

10

- **attributes** (fields, instance variables)
  - visibility: + public, # protected, – private, ~ package (default), / derived
  - underline static attributes
- **derived attribute:** not stored, but can be computed from other attribute values

| Rectangle |
|---|
| - width: int<br>- height: int<br>/ area: double |
| + Rectangle(width: int, height: int)<br>+ distance(r: Rectangle): double |

| Student |
|---|
| -name:String<br>-id:int<br>-totalStudents:int |
| #getID():int<br>+getName():String<br>~getEmailAddress():String<br>+getTotalStudents():int |



## Class operations / methods

11

- operations / methods
  - *visibility name* (*parameters* ) : *returnType*
  - underline static methods
  - parameter types listed as (name: type)
  - omit *returnType* on constructors and when return is `void`

| Rectangle |
|---|
| - width: int<br>- height: int<br>/ area: double |
| + Rectangle(width: int, height: int)<br>+ distance(r: Rectangle): double |

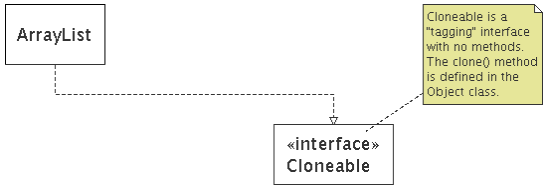
| Student |
|---|
| -name:String<br>-id:int<br>-totalStudents:int |
| #getID():int<br>+getName():String<br>~getEmailAddress():String<br>+getTotalStudents():int |



## Comments

12

- represented as a folded note, attached to the appropriate class/method/etc by a dashed line



## Relationships between classes

13

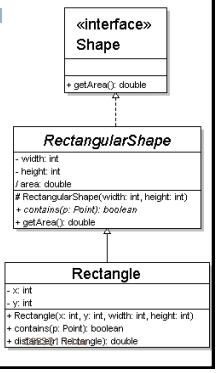
- **generalization**: an inheritance (is-a) relationship
  - inheritance between classes
  - interface implementation
- **association**: a usage (is-part-of) relationship
  - dependency
  - aggregation
  - composition



## Generalization relationships

14

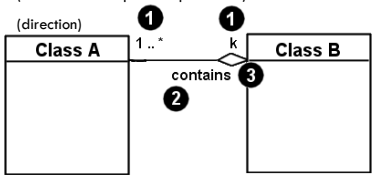
- Hierarchies drawn top-down with arrows pointing upward to parent
- Line/arrow styles differ based on parent
  - *class* : solid, black arrow
  - *abstract class* : solid, white arrow
  - *interface* : dashed, white arrow
- Trivial / obvious relationships, such as drawing the class `Object` as a parent, are generally omitted




## Associational relationships

15

1. multiplicity (how many are used)
   - * ⇒ 0, 1, or more
   - 1 ⇒ 1 exactly
   - 2..4 ⇒ between 2 and 4, inclusive
   - 3..* ⇒ 3 or more
2. name (what relationship the objects have)
3. navigability (direction)



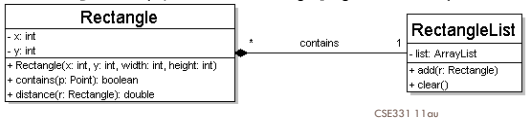



## Multiplicity

16

- one-to-one
  - Ex: each student must have exactly one ID card


- one-to-many

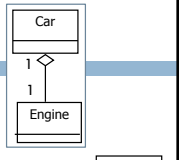


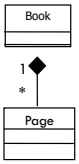



## Association types

17

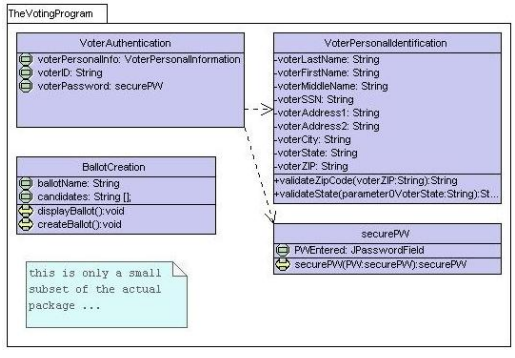
- **aggregation**: "is part of"
  - clear white diamond
- **composition**: "is entirely made of"
  - stronger version of aggregation
  - the parts live and die with the whole
  - black diamond
- **dependency**: "uses temporarily"
  - dotted line or arrow
  - often is an implementation detail, not an intrinsic part of that object's state

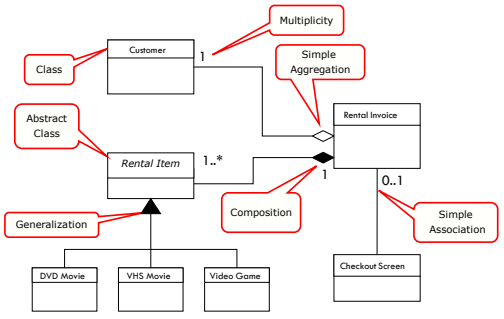


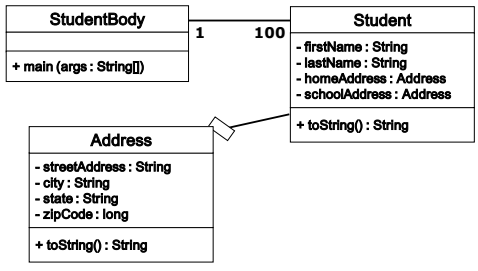













## Tools for creating UML



- Violet (free)
  - http://sourceforge.net/projects/violet/
- Rational Rose
  - http://www.rational.com/
- Visual Paradigm UML Suite (trial)
  - http://www.visual-paradigm.com/
  - (nearly) direct download link: http://www.visual-paradigm.com/vp/download.jsp?product=vpuml&edition=ce
- (there are many others, but many are commercial and cost money)



## Class diagrams pros/cons



- Class diagrams are good for
  - discovering related data and attributes
  - getting a quick picture of the important entities in a system
  - seeing whether you have too few/many classes
  - seeing whether the relationships between objects are too complex, too many in number, simple enough, etc.
  - spotting dependencies between one class/object and another
- Not so great for
  - discovering algorithmic (not data-driven) behavior
  - finding the flow of steps for objects to solve a given problem
  - understanding the app's overall control flow (event-driven? web-based? sequential? etc.)



## Related: Object diagrams

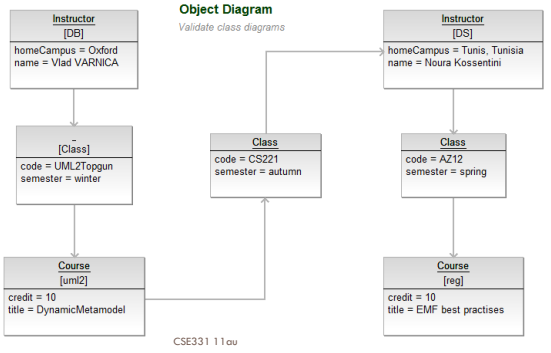


- shows an individual object, rather than entire class
  - objectName : type
  - attribute = value
  - objects can be connected by lines that state the reason the two objects are talking



## Object diagram example





## UML sequence diagrams

25

- sequence diagram: an "interaction diagram" that models a single scenario executing in the system
  - perhaps second most used UML diagram (behind class diagram)

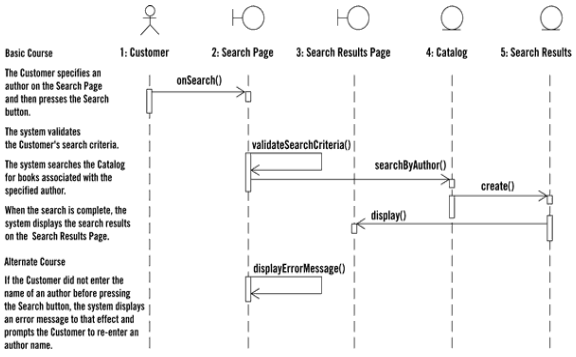


## Sequence diagram key parts

26

- **participant:** object or entity that acts in the diagram
  - diagram starts with an unattached "found message" arrow
- **message:** communication between participant objects
- The axes in a sequence diagram
  - horizontal: which object/participant is acting
  - vertical: time (down -> forward in time)



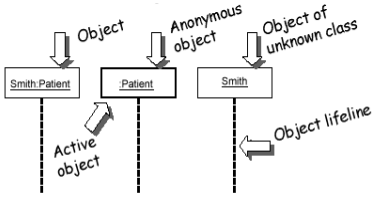





## Representing objects

28

- Squares with object type, optionally preceded by "*name* :" (if it clarifies diagram)
- object's "life line" represented by dashed vertical line

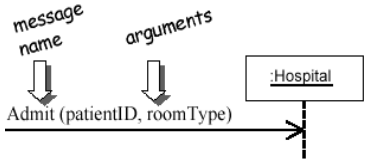



**Name syntax:** <objectname>:<classname>

## Messages between objects

29

- messages (method calls) indicated by arrow to other object
  - write message name and arguments above arrow

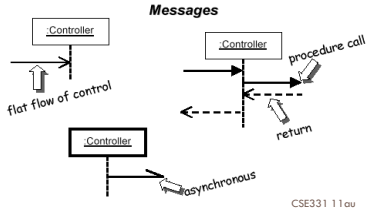





## Messages

30

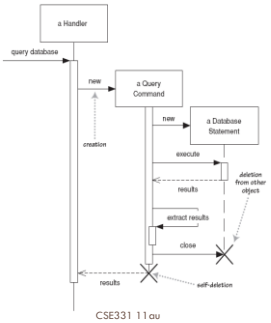
- messages (method calls) indicated by arrow to other object
  - dashed arrow back indicates return
  - different arrowheads for normal / concurrent (asynchronous) calls



## Lifetime of objects

31

- *creation*: arrow with 'new' written above it
  - notice that an object created after the start of the scenario appears lower than the others
- *deletion*: an X at bottom of object's lifeline
  - Java doesn't explicitly delete objects; they fall out of scope and are garbage-collected

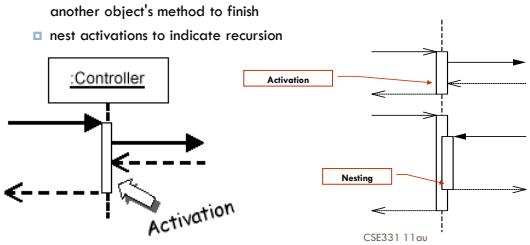





## Indicating method calls

32

- **activation**: thick box over object's life line; drawn when object's method is on the stack
  - either that object is running its code, or it is on the stack waiting for another object's method to finish
  - nest activations to indicate recursion

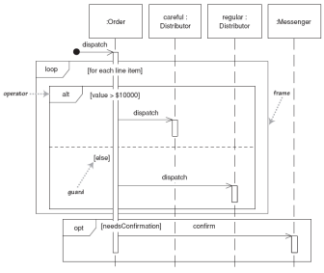




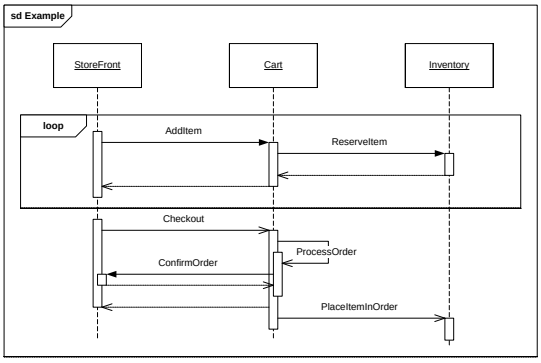

## Selection and loops

33

- **frame**: box around part of diagram to indicate `if` or loop
  - `if -> (opt) [condition]`
  - `if/else -> (alt) [condition]`, separated by horizontal dashed line
  - `Loop -> (loop) [condition or items to loop over]`





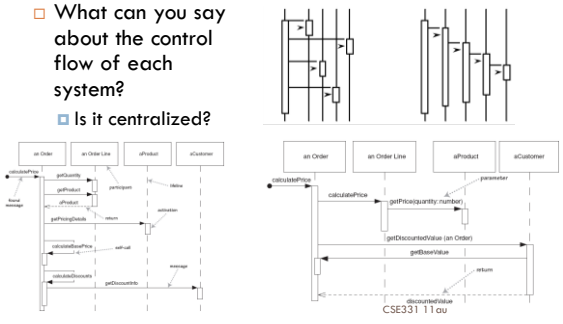






## Forms of system control

35

- What can you say about the control flow of each system?
  - Is it centralized?





## Why not just code it?

36

- Sequence diagrams can be somewhat close to the code level
- So why not just code up that algorithm rather than drawing it as a sequence diagram?
  - a good sequence diagram is still a bit above the level of the real code (not all code is drawn on diagram)
  - sequence diagrams are language-agnostic (can be implemented in many different languages
  - non-coders can do sequence diagrams
  - easier to do sequence diagrams as a team
  - can see many objects/classes at a time on same page (visual bandwidth)

## And many other UML diagram types…

37

- The two not covered here that are perhaps most important are the state diagrams and the use case diagrams



## Quotations

http://www.step-10.com/SoftwareDesign/UML/UMLQuote.html

38

- "The trouble comes when people feel compelled to convey the whole model or design through UML. A lot of object model diagrams are too complete and, simultaneously, leave too much out. ... Nor is UML a very satisfying programming language."
  Eric Evans, 2003, Domain-Driven Design: Tackling Complexity in the Heart of Software
- "The vocabulary and rules of a language such as UML tell you how to create and read well-formed models, but they don't tell you what models you should build and when you should create them. That's the role of the software development process."
  Grady Booch, James Rumbaugh, Ivar Jacobson, 2005, The Unified Modeling Language User Guide
- "The fundamental reason to use UML involves communication. … Natural language is too imprecise and gets tangled when it comes to complex concepts. Code is precise but too detailed. So I use UML when I want a certain amount of precision but I don't want to get lost in the details."
  Martin Fowler, Kendall Scott, 2000, UML Distilled: A Brief Guide to the Standard Object Modeling Language



## So…

39

- UML allows you to say some of the things that languages don't allow you to say explicitly about software systems
- It can be used effectively; it can be used horribly
  - Flon's Law: Good programs can be written in any language; and bad programs can be written in any language
- Knowing the basics is important – it's a common lingo (and it sometimes shows up in interviews)