

Design Notation
Class and Sequence Diagrams
Part 2

Dave Levitt

CS 2000: Systems Analysis and
Design

Agenda

- Continue Discussion on Class & Interaction Diagrams
- Rose Demo (cont'd)
- Lab

Class Diagram vs. Domain Model

	Domain Model (OOA)	Class Diagram (OOD)
Uses UML class symbol	Yes	Yes
Audience	Developers & Domain Experts	Developers
Intent	Illustrate conceptual entities and their relationship	Blueprint for implementation – static view
Show associations and multiplicities	Yes	Yes
Show attributes	Yes	Yes
Show operations	Probably not	Yes
Show navigability	Probably not	Yes
Trace from / to	Problem Statement / Design Model	Requirements, Domain Model / Code

Class Diagrams (cont'd)

- The diagram on page 250 summarizes many important concepts in OO (and even non OO) development.
 - An **abstract class** (a.k.a. base class) is one that cannot be instantiated. It serves as a template in an inheritance hierarchy. Java keyword “extends”, UML symbol is arrow with a solid line (CS 2100)
 - An **interface** provides method signatures (no code) and constants. Java keyword “implements”. UML symbol is an arrow with a dashed line.
- Together, interfaces and abstract classes form one of the most powerful concepts of OO (and even non OO) development: “PROGRAM TO AN INTERFACE, NOT AN IMPLEMENTATION”. (a.k.a. implementation independence).
 - In CS 2100, we will see how this concepts is applied in effective OO Design and Design Patterns. Stay tuned!

Class Diagrams (cont'd)

- A dependency is like an association, but much “weaker”.
 - An association usually denotes a durable relationship between 2 classes, one that you might want to persist in a database. The UML symbol is a solid line with all of the adornments like navigability, multiplicity, etc.
 - The “tip-off” that a relationship is an association is when one class has a class level attribute of another class. The relationship lasts for as long as both
 - A dependency denotes a temporal relationship, such as when 2 classes communicate in a method. In this case, the association is said to be temporal because it only lasts for the duration of the method.
 - The “tip-off” is when one class is declared in a class method, or a class is passed as an argument in a method signature, or when a class is global.

Class Diagrams (cont'd)

- Larman (chapter 16) gives much more detail on class diagrams than I care to really go over right now. Rationale:
 - I have not seen many of these concepts implemented in the real world.
 - Many of these concepts are not appropriate for an intro. OO course.
 - Many of these concepts will be explored in CS 2100.

Interaction Diagrams

- An interaction diagram shows objects and messages. It shows the *dynamic behavior* of the system.
- We have seen 2 types of interaction diagrams:
 - A **sequence diagram** depicts object interaction is a left-to-right, time ordered sequence. They tend to take a lot of space, but are generally preferred because they are easy to read.
 - A **collaboration diagram** depicts object interaction is a graph format. It takes up less real-estate than a sequence diagram.
 - Both diagrams are equivalent. In this class. I will almost exclusively use sequence diagrams, but you may choose to use collaboration diagrams if you choose.

Interaction Diagrams (cont'd)

- Note the different ways to represent an object. (see page 228)
 - I prefer the first example, but you are free to choose any other.
- An interaction diagram, unlike a collaboration diagram, can illustrate object and message lifetimes.
- Object lifetime:
 - The “life” of an object begins when an object is instantiated and ends when it is destroyed.
 - Typically, an interaction diagram does not show an object’s lifetime, it is assumed or not shown. Sometimes though, you want to specifically want to show when an object is created or destroyed.

Interaction Diagrams (cont'd)

- Object lifetime (cont'd)
 - Looking at the top diagram on page 230: what class and method creates an instance of the Sale class?
 - What java keyword is used to create an object? It's corresponding method is given a special name. What is it called?
 - Looking at the bottom diagram on page 230: what class and method creates an instance of a Payment class?
 - What class / method destroys the payment object?
 - How is an object destroyed in java?

Interaction Diagrams (cont'd)

- Message lifetimes:
 - A life of a message begins when it is first invoked and ends when it is done (last statement or a return statement).
 - A sequence diagram can show what other methods are invoked within a life of a message. Looking at the diagram on page 228:
 - What methods are invoked in the register:doX() method?
 - Note that these methods lie within the the bar. We call this the method's "span of control". A.k.a. "span of execution or "activation bar".
 - Message returns:
 - Typically, it is not necessary to show that a called method returns, but you may want to show it in the method invocation (see page 228).

Interaction Diagrams (cont'd)

- Additional:
 - Our version of Rose does not support UML 2.0 or the looping notation illustrated by Larman on page 231. Instead, use a note to indicate if there is a loop or iteration.
 - We will not be concerned with asynchronous vs. synchronous messages in this class, and possibly not even in CS 2100. If you want to learn more, take an advanced Java class and read Larman: pages 238 and 239.
 - Larman talks about other notations and features we will not need in this class like inheritance (CS 2100), nested frames, metaclasses, etc.

Wrap-Up

- Knowing how to draw class and sequence diagrams are important first steps in OOD, but remember, knowing the notation does not make one a good OO modeler!
- Next week, we will look at basic OO patterns (heuristics). An understanding of which will form a solid basis for more advanced OOD topics that we will discuss in CS 2100.
- One last thing: there is no such thing as a language-neutral design. Why?