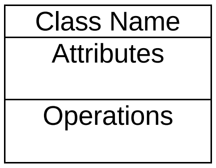
Benjamin Fenster
CS-100 (Damon)
29 March 2003

Notes – Unified Modeling Language

I. Concept
  a. Common notation using pictures to represent programs
  b. Not a programming methodology: doesn't say how to program, just how to represent programs.
  c. Very large and extensible.
  d. Development
    i. 3 Amigos
    ii. Grady Booch
      1. Booch method
      2. Notation
    iii. Ivar Jacobsen
      1. Objectory
      2. Methodology with notation.
    iv. Jim Rumbaugh
      1. OMI
      2. Methodology with notation
    v. Those three were already the most common in use.
    vi. When the three started working for the same company, it demanded a single unified notation.
    vii. UML combined everything from all those notations into one uber-notation

II. Usage
  a. Can describe almost everything in software design
  b. Mostly language agnostic, but may feel a little like C++ since that was the big language at the time of its initial development.
  c. Largely used for…
    i. Class diagrams
    ii. Instance diagrams to a lesser extent
  d. Basics
    i. A box!
    ii.



    iii. Class Name
      1. <<interface>> myClass
      2. *myClass* {abstract}
      3. Can put anything in << >> brackets – called a stereotype.
    iv. Attributes
      1. Stored information
      2. Usually a one-to-one relationship with class fields.
      3. name : type = defaultValue
      4. Default value is optional.
    v. Operations
      1. Functions & Methods.
      2. name(parameters) : returnType
      3. void functions just don't include a return type.
      4. Each parameter is written name : type
  e. Inheritance
    i. Shown with arrows. ———▷
    ii. Just like BlueJ's arrows.

        iii. The shape of the arrowhead is significant, since different styles have different meanings.
        iv. Sub —▷ Super
    f. Associations
        i. Associations indicate objects that are independent but related.
        ii. Husband / Wife, Mother / Child
        iii. →
        iv. See example on CS100-27-14
        v. Can have a one-to-one, one-to-many, or many-to-many relationship.
            1. Mother-Children
            2. Can write a number near each end of the arrow.
            3. 1 means exactly 1, 7 means exactly 7, etc.
            4. * means 0 ore more
            5. 1..* means one or more.
            6. 2..6, 10..12 means "two to six or ten to twelve" and is very rare.
            7. See CS100-27-17
    g. Attributes and Multiplicity
        i. Attributes can have multiplicity too.
        ii. grades[1..*] : letter
        iii. grades[1..8] : letter
    h. Attributes vs. Associations
        i. Attributes are very similar to associations
        ii. The distinction is largely a matter of taste.
        iii. Associations
            1. Any object to which the representation contains a reference.
            2. Independent objects should all be associations.
        iv. Attributes
            1. Primitives, of course.
            2. Whenever an object "is part of" the other object, make it an attribute.
            3. Representation will include the object, not a reference to it.
    i. Navigability
        i. Having an association does not necessarily mean it's easy (or cheap) to get from one object to the other.
        ii. Navigable means it's easy to get from one object to another.
        iii. Arrowhead on the association arrow should point in the direction that's easiest to traverse.
        iv. Examples
            1. A parent doesn't usually point to children, but children point to their parent so the arrow would point Child → Parent
            2. In a tree situation the reverse is true: each parent points to its children, so the arrow would point Parent → Child
        v. A double-headed arrow means it's easy to move both ways.
        vi. It must be realistic to program before it can honestly be called navigable.
    j. Dependency
        i. Drawn as a dashed arrow
        ii. "If you change your interface, I have to change."

III. The Point
    a. Class diagrams are by far the most common part of UML
    b. Most people don't care too much about the details
    c. A UML diagram should easily convey the essentials.
    d. Be able to use and create UML-like diagrams containing the essentials.
    e. Be able to read real UML diagrams.