Benjamin Fenster
CS-100 (Damon)
29 March 2003

Notes – Visibility

- I. Encapsulation
  - a. Only use a public interface, don’t give any access to the private representation.
  - b. Compilers support that now.
  - c. Not all representation is completely private though.
    - i. Superclass/subclass relationships may need to share some information.
    - ii. Not ALL representation needs to be shared though, so one can’t just say “all subclasses act like the superclass.”
  - d. Two classes may need to cooperate even when they aren’t a superclass/subclass.
- II. Visibility Options
  - a. Every class and member has a visibility setting
  - b. public
  - c. private
  - d. protected (some limitation)
  - e. *nothing*
    - i. If no visibility is specified, access is slightly more limited than ”protected” elements.
    - ii. A bit counterintuitive.
  - f. Classes cannot be protected or private; it doesn’t make sense.
  - g. If a class says nothing, it can only be used within its package (see below).
- III. Packages
  - a. Used partly for naming and organization.
  - b. Also used in determining visibility.
  - c. Subclasses and classes in the same package can see protected elements
  - d. Assumptions on Cooperating Classes
    - i. Only classes in the same package cooperate
    - ii. EVERY class in a package cooperates.
  - e. See CS100-28-12
- IV. C++
  - a. C++ offers a much better model of visibility.
  - b. protected in C++ means “visible by subclasses only.”
  - c. Visibility is defined for groups of members, not individual members.
  - d. class vs. struct
    - i. struct defaults to public members
    - ii. class defaults to private members.
    - iii. That’s the *only* difference.
  - e. C++ doesn’t have “packages,” so there’s no inherent group of cooperating classes.
  - f. Friends
    - i. Friend functions can see ALL the representation in the cooperating class.
    - ii. Allows the programmer to specifically give access to classes (or functions) that are really cooperating, instead of letting the compiler assume that all packaged classes are related.
  - g. Namespaces
    - i. `namespece (identifier) { declaration; declaration; }`
    - ii. Declare a namespace and give a list of forward declarations (for classes and functions)
    - iii. Most C++ code predates this, so it’s not used too much.
- V. UML
  - a. + means public
  - b. – means private
  - c. # means protected
  - d. The symbol is given just before the attribute or operation name.
  - e. Most of UML deals with public behavior only, so details that relate to implementation aren’t normally discussed in it anyway.

VI. Security

   a. “If I call it private, nobody can get it.”
   b. C++ renders this philosophy completely false
      i. If a program can get the base address of a class, some simple pointer arithmetic will give access to any member.
      ii. Making members private is simply not effective security.
   c. Java makes it a little harder
      i. The only way to “hack in” is to write “native code”
         1. Can write code in another language and import it into Java.
         2. Writing a piece of C++ code would simply introduce the same problems.
      ii. With downloaded code (such as would run in a web browser) new code can’t be inserted anyway so it’s safe.