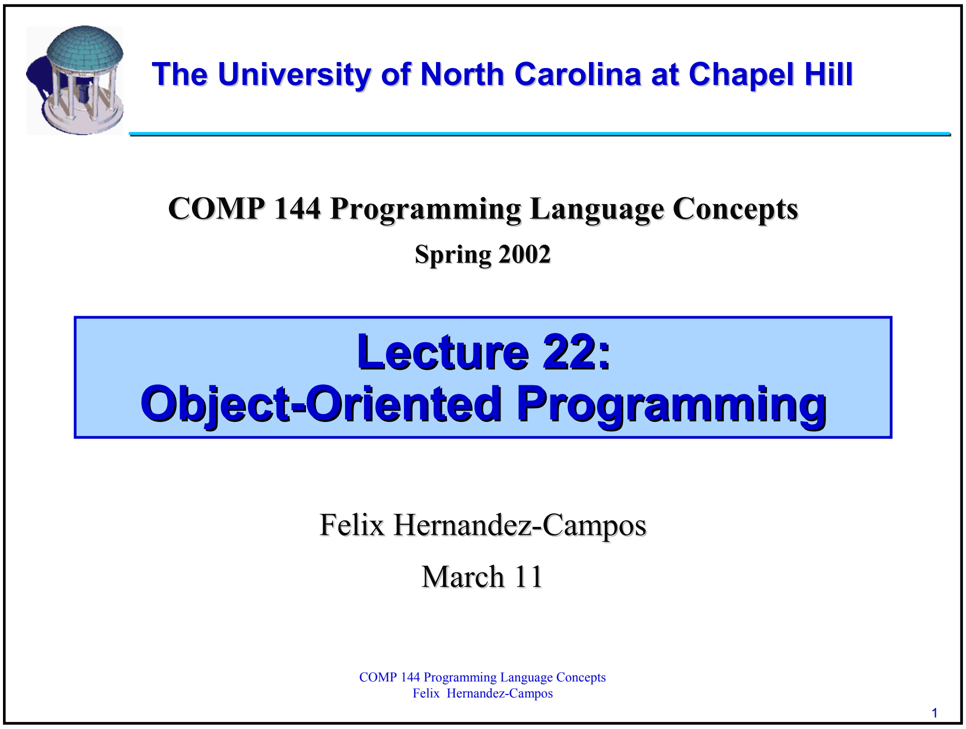


The University of North Carolina at Chapel Hill

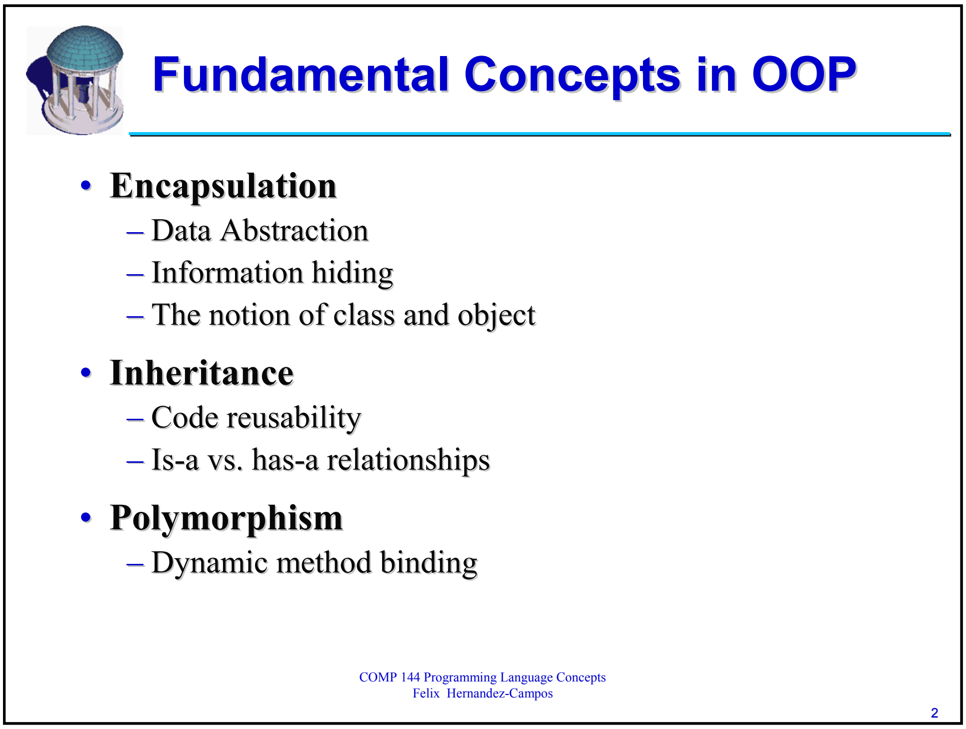
COMP 144 Programming Language Concepts

Spring 2002

Lecture 22: Object-Oriented Programming

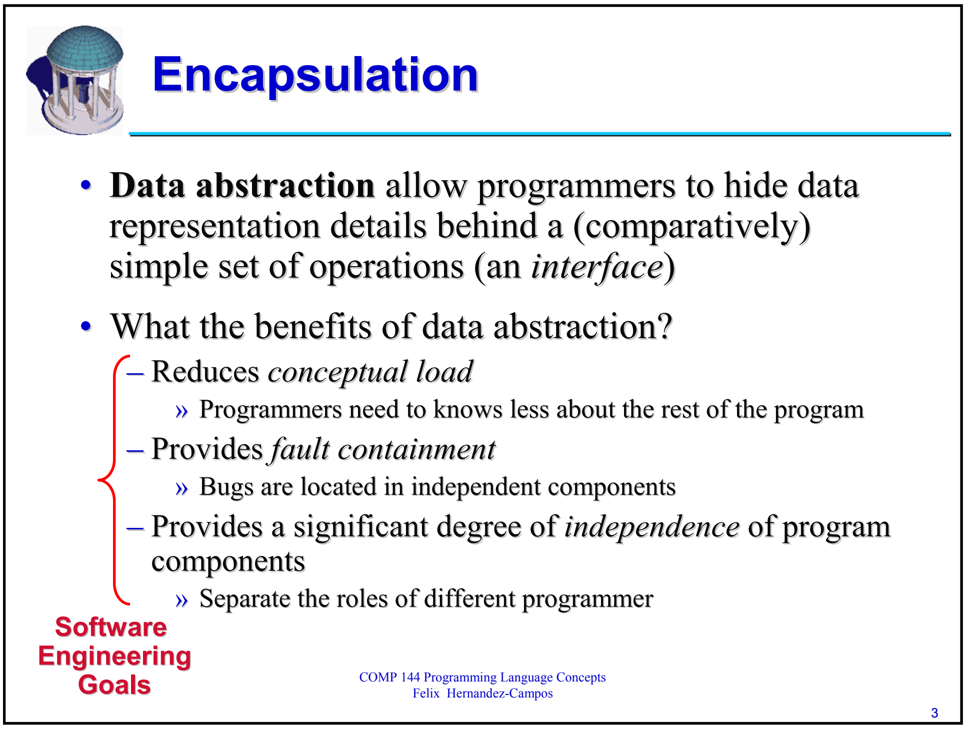
Felix Hernandez-Campos

March 11



Fundamental Concepts in OOP

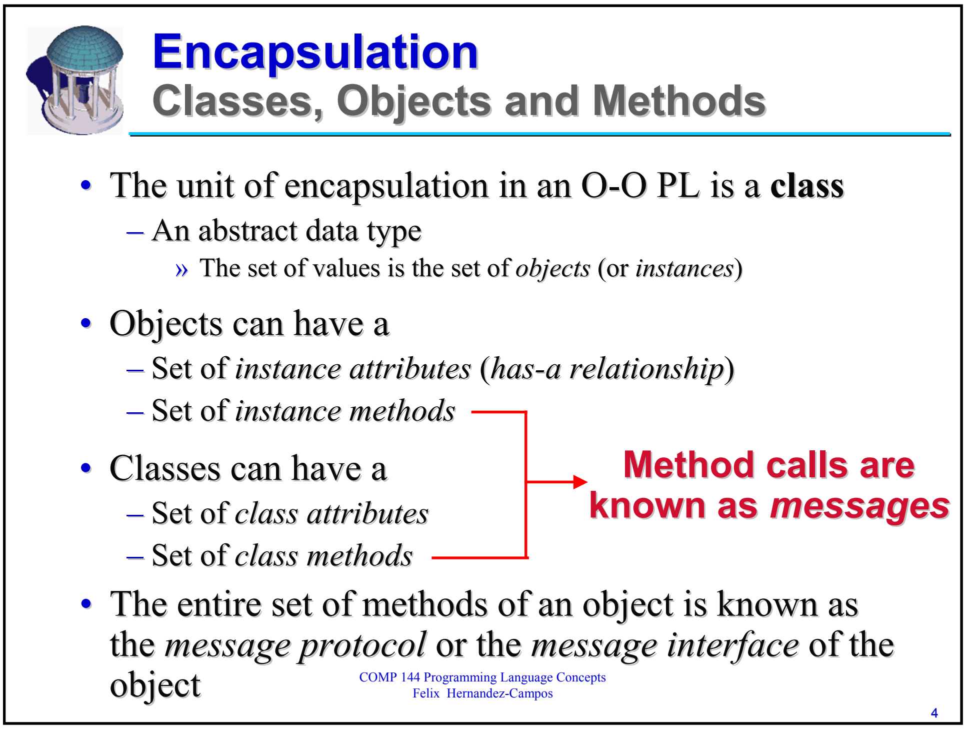
- **Encapsulation**
  - Data Abstraction
  - Information hiding
  - The notion of class and object
- **Inheritance**
  - Code reusability
  - Is-a vs. has-a relationships
- **Polymorphism**
  - Dynamic method binding



Encapsulation

- **Data abstraction** allow programmers to hide data representation details behind a (comparatively) simple set of operations (an *interface*)
- What the benefits of data abstraction?
  - Reduces *conceptual load*
    - » Programmers need to knows less about the rest of the program
  - Provides *fault containment*
    - » Bugs are located in independent components
  - Provides a significant degree of *independence* of program components
    - » Separate the roles of different programmer

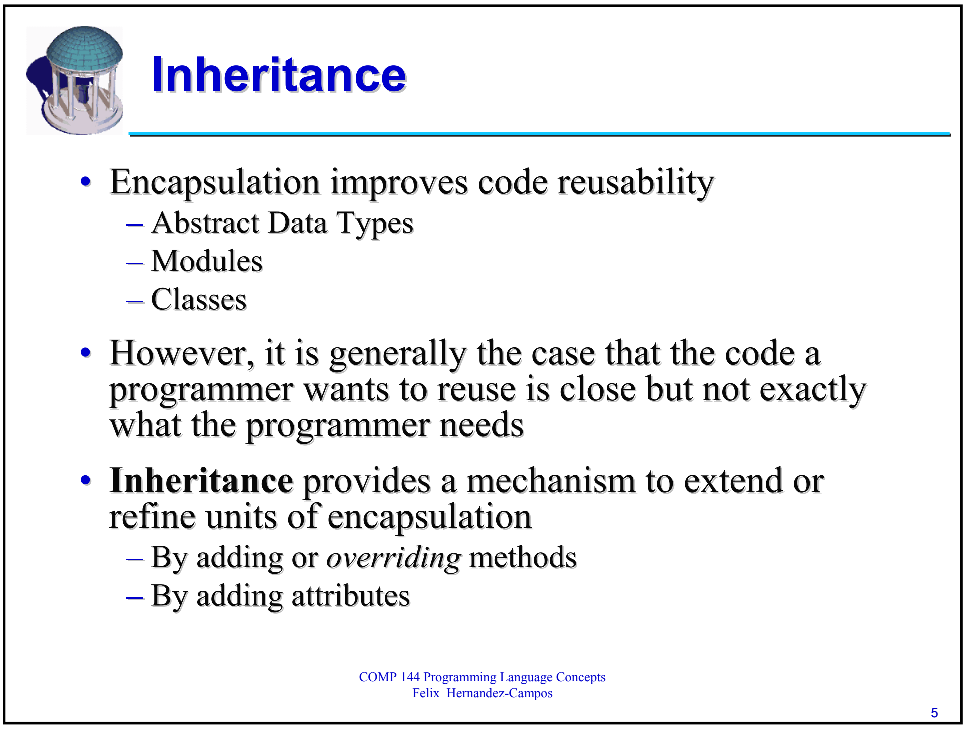
Software Engineering Goals



Encapsulation
Classes, Objects and Methods

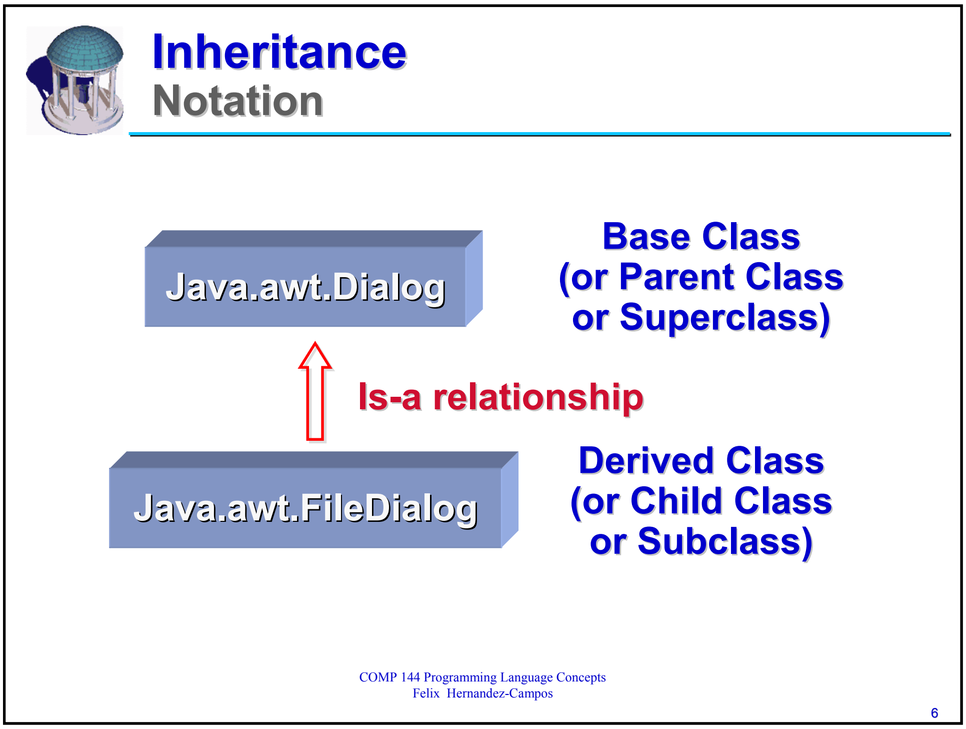
- The unit of encapsulation in an O-O PL is a **class**
  - An abstract data type
    - » The set of values is the set of *objects* (or *instances*)
- Objects can have a
  - Set of *instance attributes* (*has-a relationship*)
  - Set of *instance methods*
- Classes can have a
  - Set of *class attributes*
  - Set of *class methods*
- The entire set of methods of an object is known as the *message protocol* or the *message interface* of the object

Method calls are known as *messages*



Inheritance

- Encapsulation improves code reusability
  - Abstract Data Types
  - Modules
  - Classes
- However, it is generally the case that the code a programmer wants to reuse is close but not exactly what the programmer needs
- **Inheritance** provides a mechanism to extend or refine units of encapsulation
  - By adding or *overriding* methods
  - By adding attributes



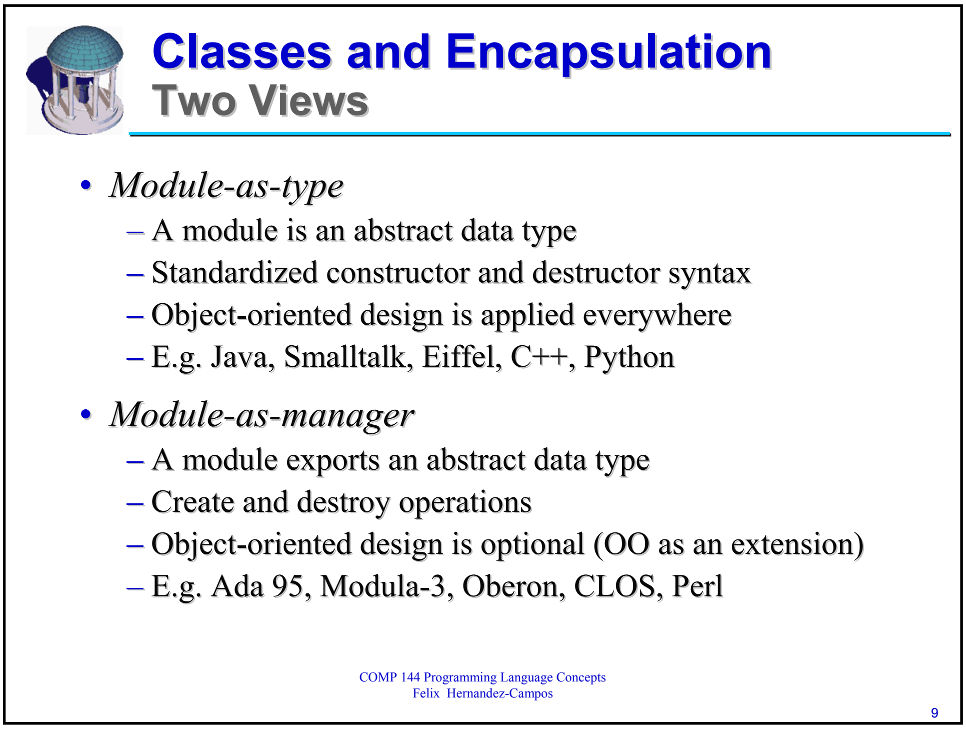
Inheritance
Notation

Polymorphism

- The is-a relationship supports the development of generic operations that can be applied to objects of a class and all its subclasses
  - This feature is known as *polymorphism*
  - E.g. `paint()` method
- The binding of messages to method definition is instance-dependent, and it is known as dynamic binding
  - It has to be resolved at run-time
  - Dynamic binding requires the `virtual` keyword in C++
  - Static binding requires the `final` keyword in Java

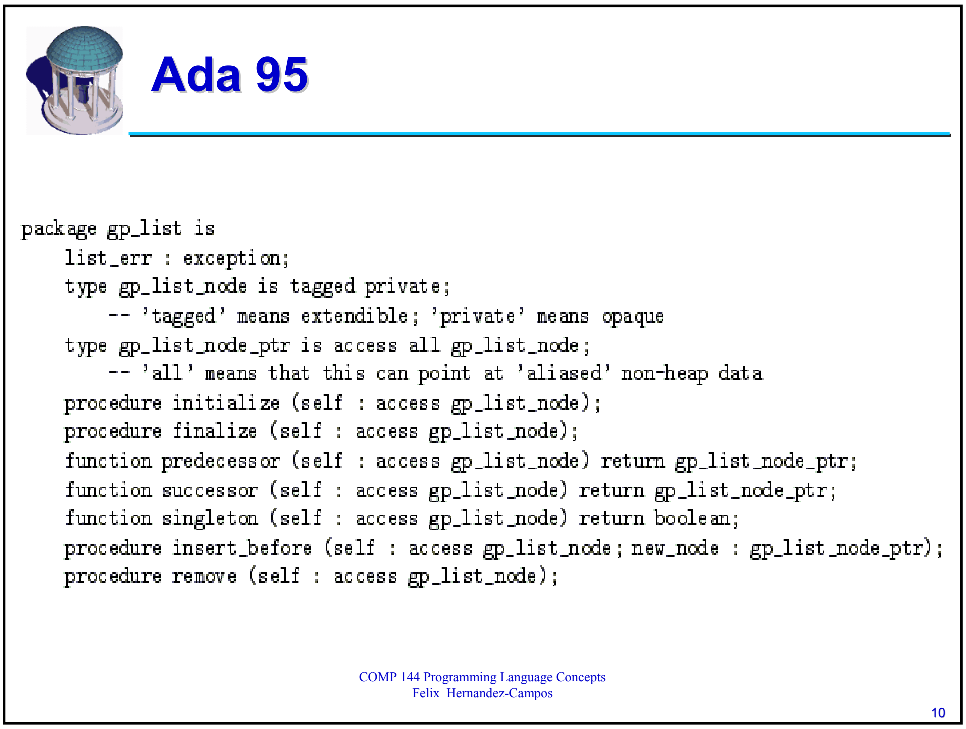
Encapsulation
Modules and Classes

- The basic unit of OO, the class, is a *unit of scope*
  - This idea originated in module-based languages in the mid-70s
    - *E.g.* Clu, Modula, Euclid
- Rules of scope enforce data hiding
  - Names have to be *exported* in order to be accessible by other modules
  - What kind of data hiding mechanisms we have in Java?
    - http://java.sun.com/docs/books/tutorial/java/javaOO/accesscontrol.html
  - And in Python?
    - http://www.python.org/doc/current/tut/node11.html#SECTION00116000000000000000



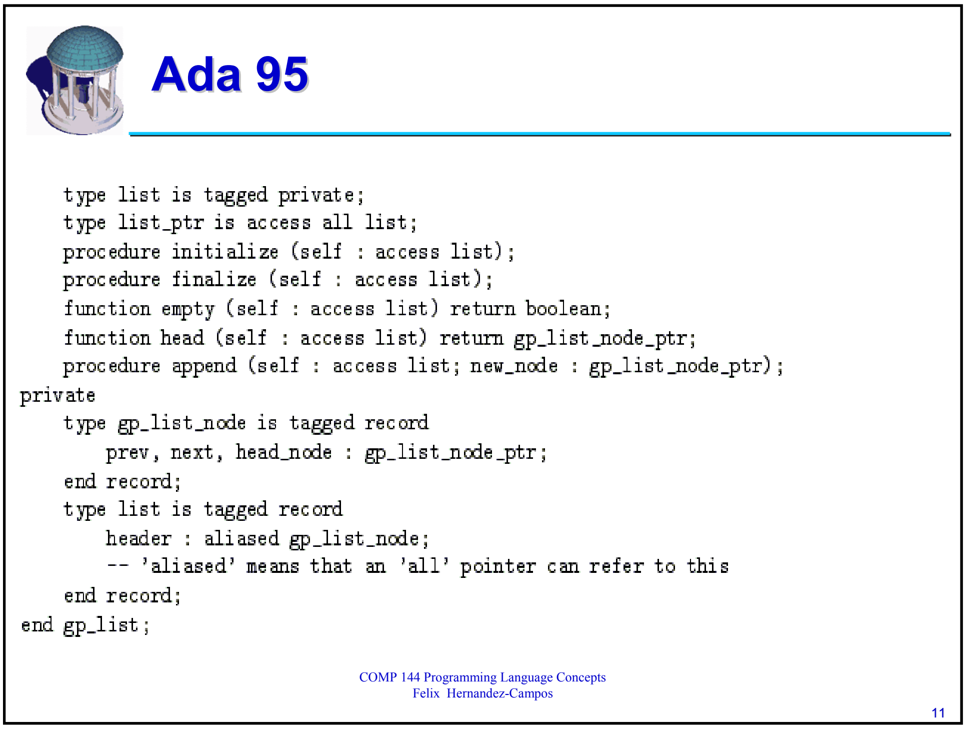
Classes and Encapsulation
Two Views

- *Module-as-type*
  - A module is an abstract data type
  - Standardized constructor and destructor syntax
  - Object-oriented design is applied everywhere
  - E.g. Java, Smalltalk, Eiffel, C++, Python
- *Module-as-manager*
  - A module exports an abstract data type
  - Create and destroy operations
  - Object-oriented design is optional (OO as an extension)
  - E.g. Ada 95, Modula-3, Oberon, CLOS, Perl



Ada 95

```
package gp_list is
    list_err : exception;
    type gp_list_node is tagged private;
        -- 'tagged' means extendible; 'private' means opaque
    type gp_list_node_ptr is access all gp_list_node;
        -- 'all' means that this can point at 'aliased' non-heap data
    procedure initialize (self : access gp_list_node);
    procedure finalize (self : access gp_list_node);
    function predecessor (self : access gp_list_node) return gp_list_node_ptr;
    function successor (self : access gp_list_node) return gp_list_node_ptr;
    function singleton (self : access gp_list_node) return boolean;
    procedure insert_before (self : access gp_list_node; new_node : gp_list_node_ptr);
    procedure remove (self : access gp_list_node);
```

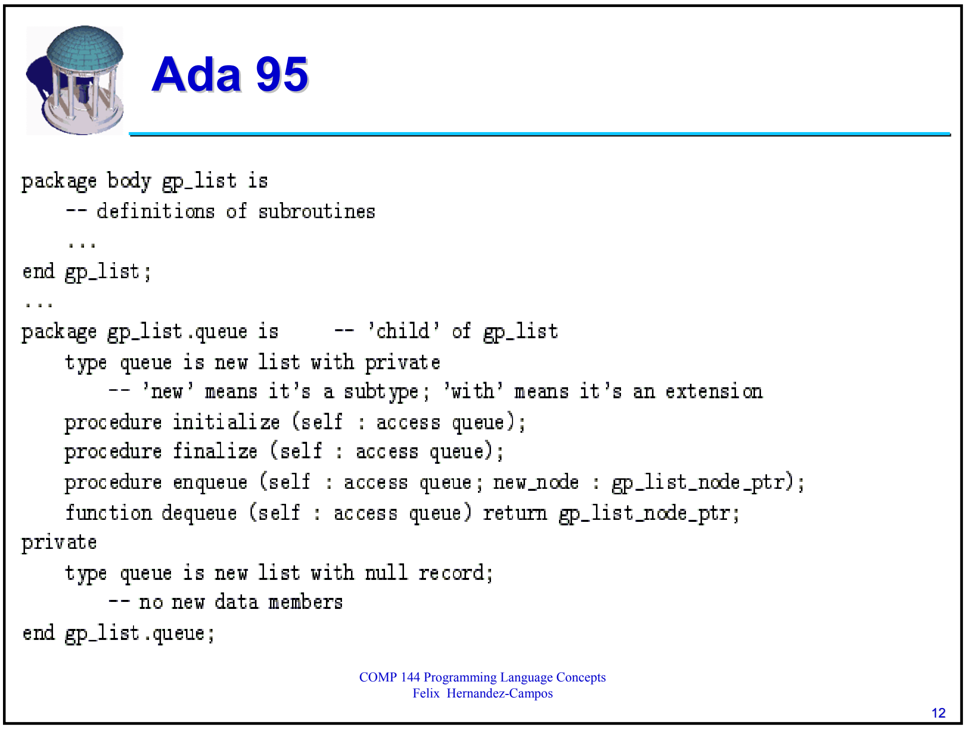


Ada 95

```
    type list is tagged private;
    type list_ptr is access all list;
    procedure initialize (self : access list);
    procedure finalize (self : access list);
    function empty (self : access list) return boolean;
    function head (self : access list) return gp_list_node_ptr;
    procedure append (self : access list; new_node : gp_list_node_ptr);
private
    type gp_list_node is tagged record
        prev, next, head_node : gp_list_node_ptr;
    end record;
    type list is tagged record
        header : aliased gp_list_node;
        -- 'aliased' means that an 'all' pointer can refer to this
    end record;
end gp_list;
```

COMP 144 Programming Language Concepts
Felix Hernandez-Campos

11

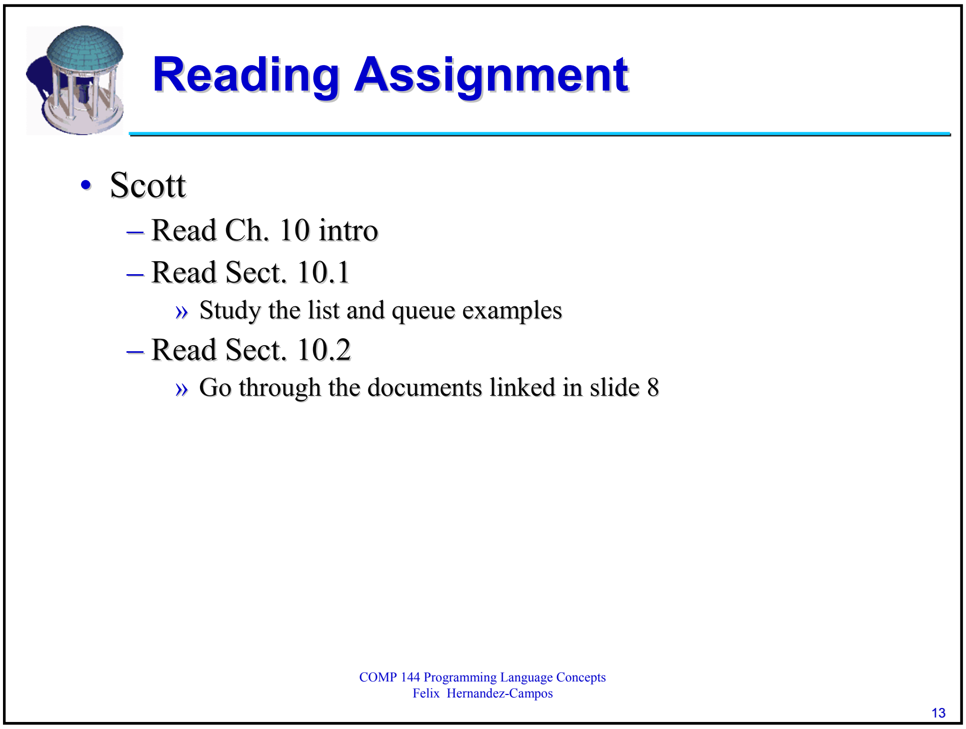


Ada 95

```
package body gp_list is
    -- definitions of subroutines
    ...
end gp_list;
...
package gp_list.queue is     -- 'child' of gp_list
    type queue is new list with private
        -- 'new' means it's a subtype; 'with' means it's an extension
    procedure initialize (self : access queue);
    procedure finalize (self : access queue);
    procedure enqueue (self : access queue; new_node : gp_list_node_ptr);
    function dequeue (self : access queue) return gp_list_node_ptr;
private
    type queue is new list with null record;
        -- no new data members
end gp_list.queue;
```

COMP 144 Programming Language Concepts
Felix Hernandez-Campos

12



Reading Assignment

- Scott
  - Read Ch. 10 intro
  - Read Sect. 10.1
    - Study the list and queue examples
  - Read Sect. 10.2
    - Go through the documents linked in slide 8