### ****Competencies****

What is scope?

The textual region of the program in which a binding is active is its *scope*.

What is a hole in a scope?

A name-to-object binding that is hidden by a nested declaration of the same

name is said to have a *hole* in its scope. In most languages the object whose name

is hidden is inaccessible in the nested scope

The object should be able to be seen in the nested scope, but there’s another object in the nested scope with the same name, so it overrides the one that should be visible (a “hole” in the “space” that the object should otherwise be able to be seen everywhere in).

What is static (lexical) scoping?

In a language with static (lexical) scoping, the bindings between names and objects

can be determined at compile time by examining the text of the program,

without consideration of the flow of control at run time

What is dynamic scoping?

In a language with dynamic scoping, the bindings between names and objects

depend on the flow of control at run time and, in particular, on the order in which

subroutines are called.

Each variable has its own stack rather than a runtime stack. Use the last/most recently defined value for a variable, when you leave a function/block, pop that value off the stack, and the second most recently defined value is used.

Be able to analyze code with static and dynamic scoping.

Nested functions.

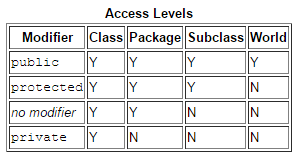
 a **nested function** (or **nested procedure** or **subroutine**) is a [function](http://en.wikipedia.org/wiki/Subroutine) which is defined within another function, the *enclosing function*. Due to [scope](http://en.wikipedia.org/wiki/Scope_(programming)) rules, the nested function can thus access local variables of the enclosing function, and in turn the nested function cannot be accessed outside of the enclosing function. The nesting is theoretically possible to any ideas of depth, although only a few levels are normally used in practical programs.

a name that is introduced in a declaration is known in the scope in which it is declared, and in each internally nested scope, unless it is *hidden* by another declaration of the same name in one or more nested scopes.

Nested functions behave the same way variables would in nested scopes.

How do visibility modifiers in classes impact the scope of class members? (Consider Java)

Access levels affect you in two ways. First, when you use classes that come from another source, such as the classes in the Java platform, access levels determine which members of those classes your own classes can use. Second, when you write a class, you need to decide what access level every member variable and every method in your class should have.



How do object methods find the correct memory binding for object data members? (class notes)

Class widget {

Int x;

Void f(int z) {

X= z;

}

}

translate to widget\_f(widget.this, int z) – compiler needs to pass along widget.this in order to implicitly reference the object’s data.

Be able to explain modules, managed modules (ADTs), packages, and name spaces. Understand the similarities and the differences.

A module accomplishes one function and contains everything necessary to accomplish this.

A **namespace** is a logical grouping of unique identifiers. In other words, a namespace is that set of names that can be accessed from a given bit of code in your program

This modularization of effort depends critically on the notion of *information hiding*, which makes objects and algorithms invisible, whenever possible, to portions of the system that do not need them. Properly modularized code reduces the “cognitive load” on the programmer by minimizing the amount of information required to understand any given portion of the

system.

In addition to reducing cognitive load, information hiding reduces the risk of

name conflicts: with fewer visible names, there is less chance that a newly introduced

name will be the same as one already in use. It also safeguards the integrity

of data abstractions: any attempt to access objects outside of the subroutine(s) to

which they belong will cause the compiler to issue an “undefined symbol” error

message. Finally, it helps to compartmentalize run-time errors: if a variable takes

on an unexpected value, we can generally be sure that the code that modified it is

in the variable’s scope.

A package is a namespace that organizes a set of related classes and interfaces

Packages are used to organize files or public types to avoid type conflicts. Package constructs can be mapped to a file system.

Namespaces are used to organize programs, both as an "internal" organization system for a program, and as an "external" organization system.

One source file can only have one package statement. A file can be a part of multiple name spaces, however.

What is an inner class?  What is a nested class?  How do they differ?  (class notes)

 an instance of an inner class has access to all of the members of the outer class, even those that are marked “private”. So, when an instance of the inner class is created, there are no issues with having the inner class access the instance variables of the outer class.

A nested class is a static member of the outer class. And because it is a static member, it means that it can be accessed ***without*** an instance of the Outer class. Cannot access non-static variables of the outer class.

### ****Topics****

**This is a comprehensive, but not complete, list of topics.**

Referencing environment

The complete set of bindings in effect at a given point in a program is known as

the current *referencing environment*.

Declaration order (Section 3.3.3)

can an expression *E* refer to any name declared in the current scope, or only to names that are declared *before E* in the scope?

Trying to reference a variable before it’s declared – different rules for different languages.

Declarations and definitions (Section 3.3.3)

A declaration introduces a name and indicates its scope, but may omit certain implementation details. A definition describes the object in sufficient detail for the compiler to determine its implementation. If a declaration is not complete enough to be a definition, then

a separate definition must appear somewhere else in the scope.

Nested blocks (Section 3,3,3 and Java)

local variables can be declared not only at the beginning of any subroutine, but also at the top of any begin. . . end ({...}) block.

The static attribute in C.  This restricts the scope of an external definition to a file, in effect it allows a file to behave as a module.  (class notes)

How do modules impact the scope of their members?

Can be used for information hiding/ADTs.

What are the limitations of modules?

the information hiding provided by nested subroutines is limited to objects whose lifetime is the same as that of the subroutine in which they are hidden.

What are the role of imports and exports in modules? How does a manager module solve this problem?

Imports serve to document the program: they increase modularity by

requiring a module to specify the ways in which it depends on the rest of the

program.

Most module-based languages allow the programmer to specify that certain

exported names are usable only in restricted ways. Variables may be exported

read-only, for example, or types may be exported *opaquely*, meaning that variables

of that type may be declared, passed as arguments to the module’s subroutines,

and possibly compared or assigned to one another, but not manipulated in any

other way.

Managed modules are very similar to classes. How do module functions in Modula 2 differ from class methods?  Consider their definition and how the data object is specified in the function/method call. Also note that module objects are defined as variables and are not dynamically allocated with a new operator. In addition, the initialization code must be explicitly called, unlike a class constructor.

For programmers, what is the primary problem with dynamic scoping?

Because the flow of control cannot in general

be predicted in advance, the bindings between names and objects in a language

with dynamic scoping cannot in general be determined by a compiler. As a result,

many semantic rules in a language with dynamic scoping become a matter of

dynamic semantics rather than static semantics.

With dynamic scoping, errors associated with the referencing environment may

not be detected until run time. – error may be very hard to find.

Inner classes (Java) have access to the members of the containing object in which they are instantiated. This is accomplished by an implicit reference to the containing object.  Nested classes (C++) can have access to the members of their containing class, but the programmer must explicitly create and explicitly initialize a reference to the containing class object.