Interfaces and nested classes

Interface reminder

- Interfaces are used to allow for a form of multiple inheritance without actually having multiple extends
- All methods must be implemented
- “default” is a possibility but frowned upon

Nested classes

- Classes can be defined inside other classes to aid in organization of classes
  - Think of an iterator for a Linked List class, it works ONLY for the linked list.
  - Nest the iterator class inside the LinkedList class for easier understanding
- Classes, just like methods and data members can be static or non-static

Static nested classes

- Static nested classes cannot utilize any instance methods or variables in its outer class without an object reference
- Static nested classes can be instantiated in a static or non-static member method of the outer class
- Slightly limited in its usefulness since it approximates an outer class defined separately.

Inner class

- A non-static class which is defined inside another class is an “inner” class.
- Inner classes, when compiled, are given a name of OuterClass$InnerClass.class
- Inner classes can access all of the methods and variables of the outer class directly (as would any member function)
- No “automatic” instantiation exists, you must still instantiate the inner class to get an object.

Inner Class access

- Inner class members are treated as MEMBERS of the outer class so they have access to all private data in the outer class
- Can be public or private, which controls who has access to an inner class

Local class

- Classes can be defined inside of a method!
- Useful for very small classes, perhaps just an exception or a derived class where most of the base classes does the work

Anonymous class

- A class with no name? Name Outer$1.class or Outer$2.class when compiled
- The class has no name!
- Used for defining a derived class and using a base reference to the derived thing
- No need for a name because it can be instantiated immediately and stored as a reference to the base type or a function can be called immediately
- Very helpful when dealing with ActionListeners.

Prep for next week (if time allows)

- Discussion on Threads
  - A program is broken down into:
    - Resource Ownership (memory, file handles, network connections, etc)
    - Execution
  - These were combined in early operating systems but have since be separated into
    - Processes
    - Threads
  - A thread shares all of the resources of its process.
  - Thread creation is very simple for the system to perform.