CS193j, Stanford Handout #14
Winter, 2002-03 Nick Parlante

Inner Classes

Inner Class

A class definition inside a class
Use as a private utility class -- declare private and clients can't see it
The inner class operates like a sub-part of the outer class
The inner class can have ivars, a ctor, etc. just like a regular class.
Access style
    The outer and inner classes can access each other's state, even if it is private. Stylistically, they are basically one implementation code base, so mixing access is ok.
Inner class is always created in the context of an "owning" outer object
Inner class has a pointer to its outer object -- can access ivars of outer object automatically
In the inner class code, "Outer.this" refers to the "this" pointer of the outer object
Use an inner class if there is a natural need to access the ivars of the outer object, otherwise use a nested class (below)

```
public class Outer {
   private int ivar;

   private class Inner {   // inner class
      void foo() {
         ivar = 13;     // we can "see" our outer class automatically
      }
   }
   public void test() {
      ivar = 10;
      Inner in = new Inner();
      in.foo();
      ...
   }
}
```

Nested Class

Like an inner class, but does not have a pointer to the outer object and so does not automatically access the ivars of the outer object.
Uses the "static" keyword.

```
public class Outer {
   private int ivar;

   private static class Nested { // a class known only to Outer
      void foo() {
            // no automatic access to outer ivars
      }
   }
   public void test() {
      Nested nested = new Nested();
      nested.foo();
      ...
   }
}
```

Inner / Nested Example

```
// Outer.java
/*
 Demonstrates inner/outer classes.
 Outer has an ivar 'a'.
 Inner has an ivar 'b'.

 Main points:

 -Each inner object is created in the context of a
 single, "owning", outer object. At runtime, the inner
 object has a pointer to its outer object which allows
 access to the outer object.

 -Each inner object can access the ivars/methods
 of its outer object. Can refer to the outer object
 using its classname as "Outer.this".

 -The inner/outer classes can access each other's ivars
 and methods, even if they are "private". Stylistically,
 the inner/outer classes operate as a single class
 that is superficially divided into two.

*/

public class Outer {
   private int a;

   private void increment() {
      a++;
   }

   private class Inner extends Object {
      private int b;
      private Inner(int initB) {
         b = initB;
      }
      private void demo() {
         // access our own ivar
         System.out.println("b: " + b);

         // access the ivar of our outer object
         System.out.println("a: " + a);

         // message send can also go to the outer object
         increment();

         /*
          Outer.this refers to the outer object, so could say
          Outer.this.a or Outer.this.increment()
         */
      }
   }


   // Nested class is like an inner class, but
   // wihout a pointer to the outer object.
   // (uses the keyword "static")
   private static class Nested {
      private int c;

      void demo() {
         c = 11;  // this works
         // b = 13;  // no does not compile --
         // nested object does not have pointer to outer object
      }
   }

   public void test() {
      a = 10;
      Inner i1 = new Inner(1);
      Inner i2 = new Inner(2);

      i1.demo();
      i2.demo();

      Nested n = new Nested();
      n.demo();

      /*
       Output:
       b: 1
       a: 10
       b: 2
       a: 11
      */
   }




   public static void main(String[] args) {
      Outer outer = new Outer();
      outer.test();
   }
```

```
}
```