CSC 143 Java

Footnote To Trees: Inner Classes

A Programming Dilemma

- The nodes we've defined so far for linked lists and trees may be public classes with public instance variables:

```
public class BTNode {
    public E item;              // data item in this node
    public BTNode left;         // left subtree, or null if none
    public BTNode right;        // right subtree, or null if none
    public BTNode(E item, BTNode left, BTNode right) { … }
}
```

- This simplifies examples... but it's very bad practice.
- When one class (like a node) is used only as a helper to another class..
  - It would be ideal to keep it inaccessible to the outside, without giving up programming convenience or speed.

Solution: Inner Classes

- One class may be defined fully within another class
- Called an "inner class"

```
class OuterClass {
    //constructors, variables, methods... and:
    class InnerClass {
            //contructors, variables, methods of InnerClass
            ...
    } //end class Inner
} //end class Outer
```

- Inner class can be marked public, protected, or private
  - Just like instance variables and methods
  - Containing class can always reference its own private instance variables, methods – and inner classes!

Solving the Tree/Node Problem

- Make Node a private inner class of BinaryTree:

```
public class BinaryTree {
    //constructors, variables, methods... and:
    private class BTNode {
            E item;             // data item in this node
            BTNode left;        // left subtree, or null if none
            BTNode right;       // right subtree, or null if none
            BTNode(E item, BTNode left, BTNode right) { … }
    } //end class BTNode
} //end class BinTree
```

- BinaryTree has full access to the members of BTNode
  - Regardless of member public/protected/private marking

More About Java Inner Classes

- We've been using inner classes occasionally without calling attention to it.
  - Point2D.Double means: the (public) inner class named Double of the class named Point2D.
- The inner/outer relationship is not the same as inheritance or composition
  - i.e., neither is-a or has-a
- Inner classes have many interesting twists and turns
  - Inner classes can even be anonymous (unnamed), like objects (recall the use of anonymous inner classes in event handling)