A Programming Dilemma
• The nodes we've defined so far for linked lists and trees have been public classes with public instance variables:
  
  ```java
  public class BTreeNode {
      public Object item; // data item in this node
      public BTreeNode left; // left subtree, or null if none
      public BTreeNode right; // right subtree, or null if none
      public BTreeNode(Object item, BTreeNode left, BTreeNode right) { … }
  }
  ```
• This simplifies examples, and increases performance... 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"

```java
class OuterClass {
    //constructors, variables, methods... and:
    class InnerClass {
        //constructors, 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!

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