VariableTreeModel's constructor should have a parameter of type Variable. Store this parameter in a private field. This Variable is the root of the TreeModel. You can then write code like:

```
   ArrayList a = new ArrayList();
   a.add(new Integer(123));
   a.add("Hello");
   a.add(a);
   Variable var = new Variable(a.getClass(), "a", a);
   VariableTreeModel model = new VariableTreeModel(var);

   // This should return the number of children that var's
   // first child has.
   model.getChild(model.getRoot(), 0).getChildCount();
```

BIG HINT: Have Section 7.6 of the textbook handy.

Pseudo code for getChildCount():

```
   // Return the number of children that parent has.
   public int getChildCount(Object parent) {
      // Assume that parent is really of type Variable.
      Variable p = (Variable) parent;

      // Be careful below --- you don't really care about p, you care
      // about the variable wrapped up inside p.  So, always use
      // p.getType() when you want the wrapped variable's type and
      // p.getValue() when you want the wrapped variable's value.

      // Okay, so I lied.  There is one place where we actually care
      // about p.  Sue me!
      if (p == null) {
         // If the parent is null, it certainly can't have any
         // children, can it?
         return 0;
      }
      // In this next part, you'll need to use isArray() and
      // Array.getLength().  See the Reflection section of the
      // textbook.
      else if (the variable wrapped up inside p is an array) {
         return the array's length;
      }
      else if (the variable wrapped up inside p is a primitive variable) {
         // Primitive variables have no children.  If they did, they
         // wouldn't be primitive variables; they'd be urbane!
         return 0;
      }
      // This is the interesting case.  The variable inside p is an object.
      // You're going to count all the non-static fields of this variable
      // and return the count.  See the code snippet on pp. 292--293.
      // in the textbook.
      else {
         // Count and return the number of non-static fields.
      }
   }
```

Hints for getChild()

1) The overall structure (if/elseif...) is the same as for getChildCount().

2) If the Object parameter passed-in is null, or the wrapped variable is

```
   a primitive variable, return null.  Otherwise, you’ll be wrapping
   a variable inside a new Variable and returning the Variable.

3) If the wrapped variable that was passed-in is an array, you’re going to
   use Array.get() to get the appropriate element (child) of the array.

   You’re got two sub-cases here --- the child is null and the child isn’t
   null.  In the former case, use getComponentType() on the wrapped
   variable’s type to get the type of the child.  Otherwise, you can
   use getClass() on the child itself to get its type.

   The name to use should be of the form [index].  For example, [0] .

   The value is the array element.

   Now, you have the type, name, and value parameters for your new
   Variable.  Construct a new Variable and return it.

4) If the wrapped-variable that was passed in is an object, you’re going
   to iterate through the object’s declared fields, similarly to what you
   did for getChildCount().  This time, you’ll also need an array of type
   Field to hold the non-static fields.  When you’re iterating through the
   declared fields, each time you find a non-static field, add it to the end
   of this second Field array.  You’ll apply the index passed-in to
   getChild() to this second array to get the child.  The type of this child
   will be Field.  You’ll need to make this child accessible, and then you
   can use Field methods to get the field’s type, name, and value.  Wrap
   these up in a new Variable, and return the Variable.
```