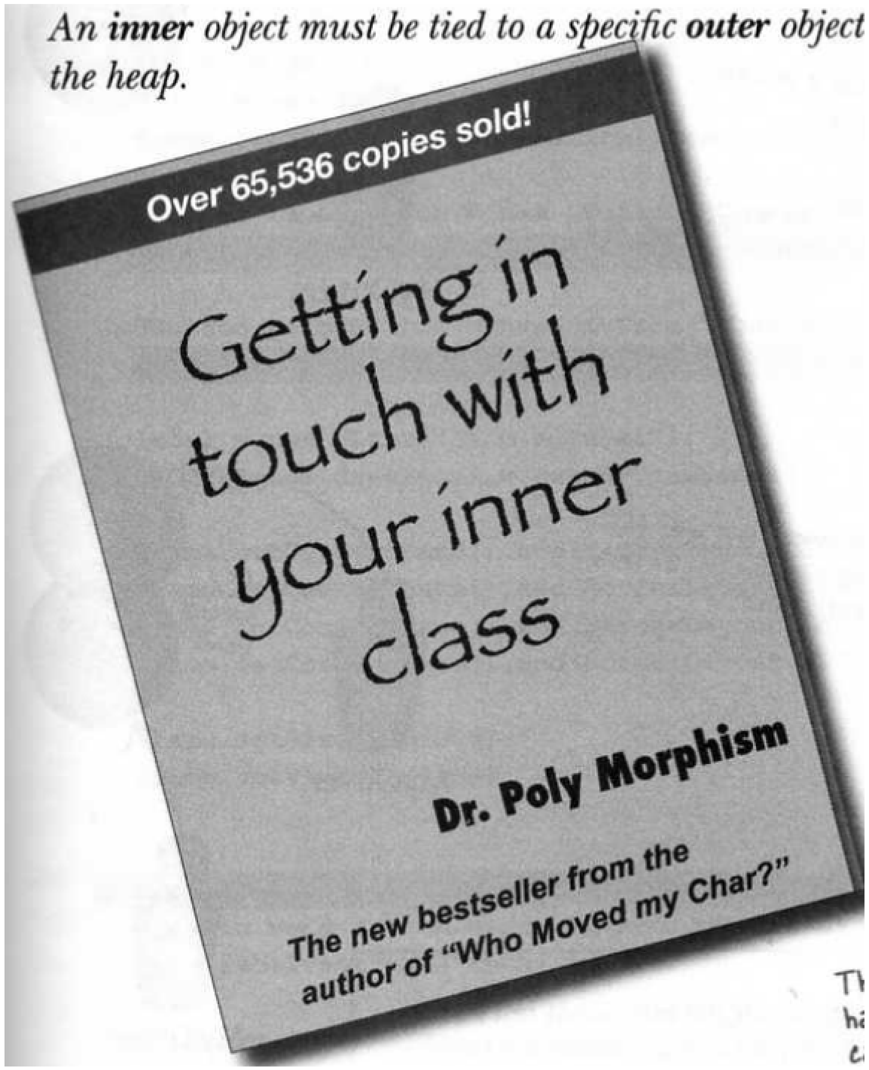
1. Inner Class

```
public class OuterClass {
 private JTextField inputField;
  private JButton button;

  class ButtonListener implements ActionListener {
    public InnerClass() { /* initialize */ }
    public void actionPerformed( . . . ) {
          String text = inputField.getText( );
     }
  }
}
```

Inner Classes


An **inner** object must be tied to a specific **outer** object
the heap.
Over 65,536 copies sold!
Getting in touch with your inner class
Dr. Poly Morphism
The new bestseller from the author of "Who Moved my Char?"

Object of InnerClass *belongs* to an object of the OuterClass.

InnerClass object has access to all fields and methods of OuterClass object, including private ones.

InnerClass object does not exist without OuterClass object.

But: static inner class is independent of OuterClass object, just like a static method. Static inner class cannot access fields of outer class (just like a static method).

button listener using Inner Class

```
public class SwingDemo {
    private JTextField inputField;
    private JButton button;

   private void initCompoents() {
      button = new JButton( "Login" );
      // add an event listener to the button
      button.addActionListener(
                   new ButtonListener( ) );
     ...
  }
  // an inner class of the SwingDemo class
  class ButtonListener implements ActionListener {
   public void actionPerformed(ActionEvent evt) {
       String user = inputField.getText().trim();
       ...
    }
  }
```

Properties of Inner Classes

- An object in an *inner class* can access variables of an object from the outer class, even private ones. (see previous slide)
- Inner classes can be public or private, just like methods and attributes. The same rules apply.
- An inner class object is always associated with an object from the outer class.
  In previous slide, this means you can't write:

  new SwingDemo.ButtonListener( );

  // can't create inner class object by itself

2. Nested Class

A nested class is a class inside another class that is static, so you can create objects of the nested class without an object of the outer class. Example:

Point.Double p = new Point.Double(1.5, 2.5);

```
public class Point {
    // A nested class
    static class Double {
        public double x, y;
        public Double(double x, double y) {
            this.x = x;  this.y = y;
        }
    }
}
```

Why Use Nested Class?

1. Group together related variants of a type.
2. Nested classes can share methods of outer class. Nested classes can be defined as *subclasses* of the outer class, so you could write something like this:

```
// suppose Point.Double extends Point
Point p = new Point.Double(1.5, 0.8);
p.getLength();  // using polymorphism
```