Programming challenge: Ok, so you may ask why would you program like this, but the answer is if you try this out it will force you to think in a different way that can actually benefit you longer term when writing OO code. Here is the challenge:

"In The ThoughtWorks Anthology a new book from the Pragmatic Programmers, there is a fascinating essay called “Object Calisthenics” by Jeff Bay. It’s a detailed exercise for perfecting the writing of the small routines that demonstrate characterize good OO implementations. If you have developers who need to improve their ability to write OO routines, I suggest you have a look-see at this essay. I will try to summarize Bay’s approach here.

He suggests writing a 1000-line program with the constraints listed below. These constraints are intended to be excessively restrictive, so as to force developers out of the procedural groove. I guarantee if you apply this technique, their code will move markedly towards object orientation. The restrictions (which should be mercilessly enforced in this exercise) are:

1. Use only one level of indentation per method. If you need more than one level, you need to create a second method and call it from the first. This is one of the most important constraints in the exercise.
2. Don’t use the ‘else’ keyword. Test for a condition with an if-statement and exit the routine if it’s not met. This prevents if-else chaining; and every routine does just one thing. You’re getting the idea.
3. Wrap all primitives and strings. This directly addresses “primitive obsession.” If you want to use an integer, you first have to create a class (even an inner class) to identify it’s true role. So zip codes are an object not an integer, for example. This makes for far clearer and more testable code.
4. Use only one dot per line. This step prevents you from reaching deeply into other objects to get at fields or methods, and thereby conceptually breaking encapsulation.
5. Don’t abbreviate names. This constraint avoids the procedural verbosity that is created by certain forms of redundancy—if you have to type the full name of a method or variable, you’re likely to spend more time thinking about its name.

And you’ll avoid having objects called Order with methods entitled shipOrder(). Instead, your code will have more calls such as Order.ship().

6. Keep entities small. This means no more than 50 lines per class and no more than 10 classes per package. The 50 lines per class constraint is crucial. Not only does it force concision and keep classes focused, but it means most classes can fit on a single screen in any editor/IDE.

7. Don’t use any classes with more than two instance variables. This is perhaps the hardest constraint. Bay’s point is that with more than two instance variables, there is almost certainly a reason to subgroup some variables into a separate class.

8. Use first-class collections. Any class that contains a collection should contain no other member variables. Each collection gets wrapped in its own class, so now behaviours related to the collection have a home. You may find that filters become a part of this new class. Also, your new class can handle activities like joining two groups together or applying a rule to each element of the group.The idea is an extension of primitive obsession. If you need a class that’s a subsumes the collection, then write it that way.

9. Don’t use setters, getters, or properties. This is a radical approach to enforcing encapsulation. It also requires implementation of dependency injection approaches and adherence to the maxim “tell, don’t ask.”

Taken together, these rules impose a restrictive encapsulation on developers and force thinking along OO lines. I assert than anyone writing a 1000-line project without violating these rules will rapidly become much better at OO. They can then, if they want, relax the restrictions somewhat. But as Bay points out, there’s no reason to do so. His team has just finished a 100,000-line project within these strictures."