CSC 207 **Homework 4: Writing Paychecks Redux** **Fall 2009**

Due: Tuesday 24 February 2009

Topics: Inheritance, design patterns

Collaboration: This homework assignment is to be completed individually.

Submission: Follow the instructions for submitting programs via P-Web and handing in a printed copy. Be sure to generate test cases and a statement justifying their sufficiency.

In this assignment we will revisit our payroll program for an environment where things are slightly more complicated. There is a federal income tax of 20% that needs to be withheld from employee paychecks. Where should the code for this tax go? Hopefully, you know it wouldn't go in your individual employee classes. That would create a great deal of repeated code (especially if we had many more employee types). It shouldn't really be in the root `Employee` class, either. Why? Well, a tax isn't really a property of an employee, so that's not a very accurate object model.

Where should we put it? If it doesn't go in the individual employee class or the parent class, the only answer left is: in a new class. How should such a class compute the tax? Well, we could have a method that computes it given the relevant parameters. But why should a caller (i.e., an employee) have to know what the relevant parameters are? The answer: it shouldn't! Instead, we should make use of the *visitor* pattern. That is, the employee should send a reference to itself to a tax object so that the tax man can probe it for whatever it may need.

Now imagine that, in addition to the 20% tax above, the federal government also allows a tax deduction (5% off the total *taxed* amount) for each dependent of the employee. Make sure your design handles this appropriately.

Of course, the states want in on the action, too. Our company has employees in Iowa, Nebraska, and Wyoming, and each of these has a different method for taxing employee income:

Iowa	6% of gross income, 1% deduction of the taxes for each dependent
Nebraska	5% of gross income
Wyoming	No state income tax

The company has plans for growth though, and so our design must scale to the possibility of 50 states! (Fortunately, for this assignment we only need to worry about three.) Where should the state tax calculator reside? How should we determine which formula to use?

We could have one state tax calculator class and use if statements to determine which formula to use. This seems problematic. If a state were to change its rules, we'd have to hunt down the formula in a (potentially) huge function and make the change. This could also jeopardize the correctness of all the other state's tax calculations. Instead, we're probably better off with a larger number of (relatively small) individual state tax calculator classes.

How do we determine which method/class to call? We don't really want a big case statement to clutter our employee class(es). Instead, how about a special class that examines an employee object (using the visitor pattern), and returns a polymorphic reference to an appropriate state tax calculator (the factory pattern!) object. Now, an employee method can deduct state tax from a paycheck with just two lines of code: one (presumably in the constructor) that gets a reference to an appropriate state tax calculator object, and another to call the actual tax computation (the state pattern!). This code should be the same for every employee, regardless of their state. Furthermore, we don't really need more than one instance of a state tax calculator for any given state, so these should use the singleton pattern. (Also, many employees share references to the same state tax calculators—a flyweight pattern!).

Re-design and implement your payroll system from homework 2 so that it now incorporates the federal and state tax changes described above. As before, be sure to test your class(es) and include your tests. Finally, create a program that generates sample paychecks for several employees of each type, demonstrating the various aspects of your system (states, dependents, etc.).