CS 548 Lab #1 Due 2/11/14

The purpose of this lab is to practice writing a few Domain/Service classes and **wiring** an application using the **Spring** container.

Note: For this program, the **only** time you will use the **new** operator is to allocate Domain objects. All Service objects should be configured in the context file (.xml) for the Spring container. No Domain objects should be Spring beans.

Do not use annotations. The purpose of this lab is to become familiar with XML wiring. To submit the lab, export the project (Under the File menu in eclipse) as an archive file. Do not zip up the project yourself.

Start with the base project provided for week 2.

Domain Improvements

1. Add a class Product which minimally should have a **name** and a **price**.
2. Add a class OrderItem which minimally should have a **Product** and the **quantity** of the product ordered
3. To the Order class, add
    a. A **Customer** (minimally the **name** and **state** – e.g. CA, NV,etc)
    b. A list of **OrderItem**s
    c. A method addItem(OrderItem item) which adds an OrderItem to the Order. If the Product is the same as an already existing Product in the Order, do **not** add the OrderItem. Instead, **merge** it into the existing OrderItem
    d. A method removeProduct(Product prod) which removes the OrderItem with that Product from the order
    e. Data members for the **subtotal**, **tax**, and **total** – these are amounts commonly found in orders. The subtotal is the sum of the product costs and should be updated whenever a new OrderItem is added. Tax will be computed by the AccountingService (see below) when the order is processed. Total can be computed after the tax is known by the OrderProcessor.

Create a TaxService

4. It should have a method:
   double computeTax(Order order)
   that will compute and return the tax for an Order. Use an **interface**. We want to be able to support different types of tax methods – e.g., sales tax, Value Added Tax (VAT), etc.

5. Add an **implementation** for the tax service that will do a **sales tax** calculation. The sales tax percentage varies from state to state. We don't want to hardcode the state percentages in our Java code as they change frequently. Wire in as a <property> lists (parallel arrays?) <list> or a map <map> at least five states and their sales tax.
6. Your tax service implementation should base the sales tax calculation on the state field found in the Customer object

AccountingService Improvements

7. Your Accounting Service implementation should have a TaxService
8. Add to the AccountingService a method:
   double computeTax(Order order)
   that will compute the tax by calling the Tax Service.

Create an InventoryService

The InventoryService will manage the inventory – essentially how many of each Product we currently have in stock. Since we don't yet have a database hooked up yet, we'll **hardcode** some initial inventory numbers in the InventoryService **constructor**. When given an Order, the InventoryService should adjust the inventory to reflect the products being sold.

The InventoryService should have:

9. A method:
   void adjustInventory(Order order)

10. A method:
    void printCurrentInventory() which will print the current inventory (each product in inventory and the number of that product in stock)
11. The **constructor** of your implementation for the InventoryService should hardcode some data to represent the current inventory

Modify the OrderProcessor

An OrderProcessor is called to handle the processing of a completed Order.

The OrderProcessor should have:

12. An AccountingService
13. An InventoryService
14. It's newOrder() method should
    a. get the tax amount from the Accounting Service and set the Order's tax data member
    b. compute and set the total amount of the Order
    c. adjust the inventory, using the Inventory Service

Modify the Client Program

15. Create an Order (obviously you need to instantiate some additional objects to build the Order)
16. Get an **OrderProcessor** bean from the Spring container and call the newOrder() method
17. Print the state of the Order object after it has been processed
18. Get an **InventoryService** bean from the Spring container and call its printCurrentInventory() method before and after the Order is processed so you can see if the inventory has been properly adjusted.