Assignment 2

1 Objective

Practice UML, Design Patterns and Git

2 Marking

This assignment is worth 8% of your final grade.

The submission deadline is Jan 29, 2017, 11:59pm on Github.

You are encouraged to work in pairs.

Late submission policy: No late submissions are accepted.

3 How to submit your work

1. Please sign in to git using your `mail.utoronto.ca` account. Use this invitiation link

   `https://classroom.github.com/group-assignment-invitations/bd6be384c3ce3f943e12e9bc6d34e214`

   and clone the starter code following `GitHub` instructions.

2. To submit your work, add, commit and push your changes to your repository.

   Do **not** commit the files and directories generated by Eclipse, such as `bin, doc, .project`, etc. Marks will be deducted if you submit these.

3. The date and time of your last commit must be on or before 29 Jan 2017, 11:59pm.

4 The wallpaper manufacturing and distribution problem

The problem stated in this exercise is a simplified version of an actual software industry problem, solved and implemented in various commercial wallpaper manufacturers in North America.

An important specific of wallpaper manufacturing (shared with many other manufacturing processes) is that there is a minimum amount of wallpaper that can be printed in one production run (1000 yards for most common types of commercial wallpaper).

The whole process can be described as follows. For each `Product`, the warehouse maintains an `Inventory` item, which has two attributes: the product, the available quantity on the inventory, and the quantity required by the customers, also known as the backordered quantity. Also the warehouse keeps track or `SalesOrders` and ships them out as soon as sufficient quantity becomes available.

The manufacturing facility, maintains a `ProductionOrder` for each inventory item. A production order is used to manufacture a quantity greater or equal to the minimum quantity, and also sufficient to cover the quantity required by `SalesOrders`.

It is clear that an `Inventory` item is an `Observable` and `SalesOrders` and `ProductionOrders` are `Observers`. In addition, both `SalesOrders` and `ProductionOrders` implement a `DisplayElement` interface as shown in the `UML` diagram below.

So here is how the solution works. Initially, we set up a number of `Customer` objects, and a number of `Product` ojects. For all the products that we intend to manufacture and sell, we set up and `Inventory` item that will be an `Observable`.

Who observes `Inventory` items? Well, in order to sell, we should manufacture some quantities, so we set up a `ProductionOrder` that is supposed to generate some quantity if there is a demand for that particular inventory item.

What does **demand** mean? Well, customers place `SalesOrder` items, which are supposed to be fufilled by the warehouse, given that there is enough quantity on `Inventory`. So, at this point, please identify the other `Observable`.

Also both `Observables` implement `DisplayElement` interface that allows the display of observables.

The wallpaper manufacturing and distribution workflow is as follows:

1. Perform the setup (generate `Customers, Products, Inventory` items and `ProductionOrders`). Note that `ProductionOrders` are observables!
2. `Customers` place `SalesOrders`.
3. If there is enough quantity on inventory, the `SalesOrder` gets shipped immediately so it does not need to go to the list of observers. Otherwise it should be registered as an `Observer`.
4. When an `SalesOrder` gets registered as an observer, the `backorderedQuantity` should be updated accordingly.
5. Meanwhile, the `ProductionOrder` gets notified so if the `backorderedQuantity` is greater or equal than `minQuantity`, the `ProductionOrder` activates the manufacturing facility to make some quantity (sufficient to cover all `SalesOrders`!) and updates the `availableQuantity`.
6. The `SalesOrders` should get notified on the availability and ship out. The inventory should be maintained accordingly.
7. If a quantity is made by a `ProductionOrder` or if a quantity is shipped in behalf of a `SalesOrder` the `display` method should display the appropriate information on the console.

A complete example is shown below. The code (in the `main` method) that generates this output is provided in full in the starter code.

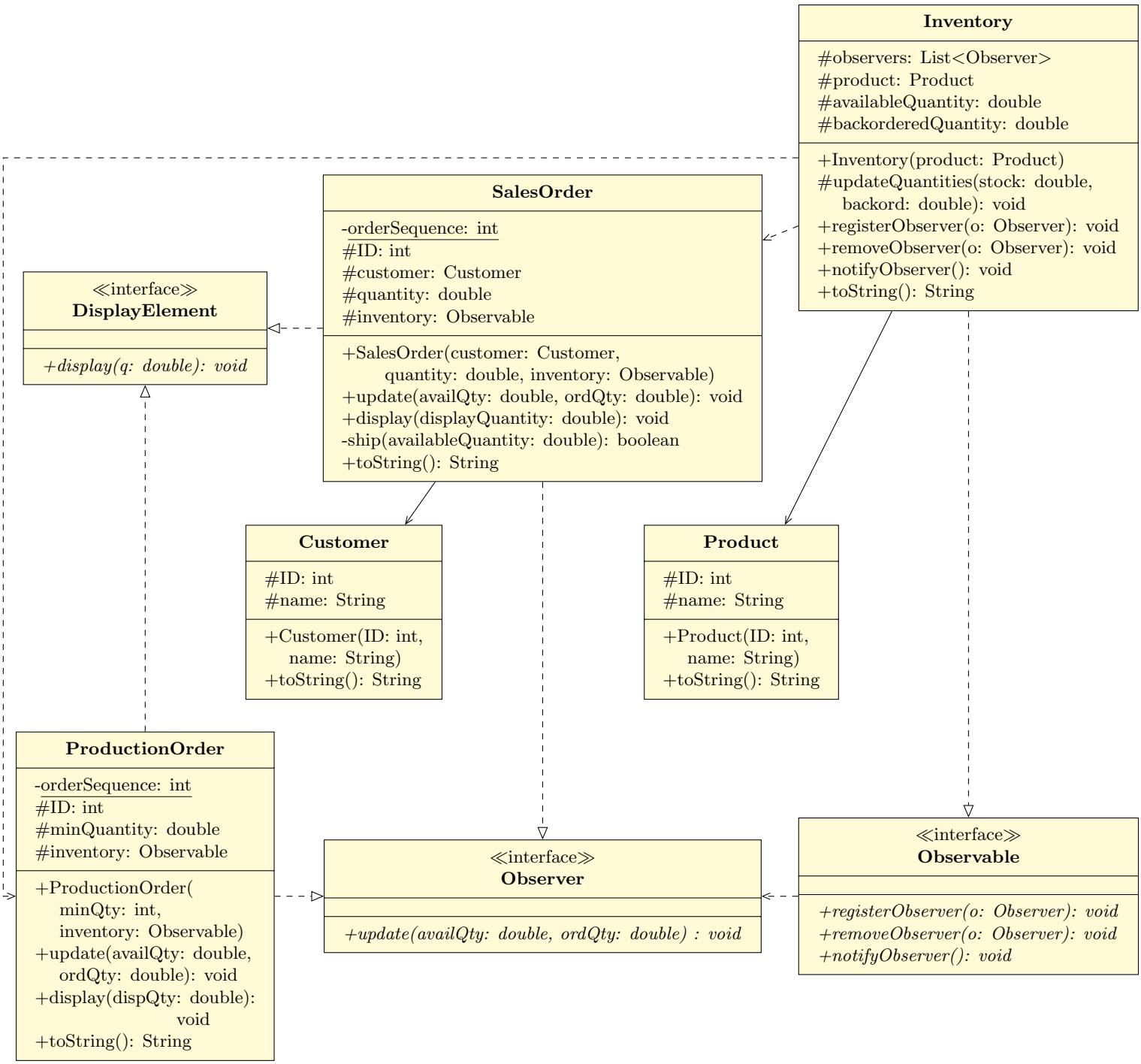
```
Shipping Order# 3 to Home Depot, Product: Flower Field, Quantity: 800.0
Shipping Order# 1 to Wal–Mart, Product: Flower Field, Quantity: 500.0
Production Order# 1, item Flower Field, produced 1300.0
Shipping Order# 4 to Home Depot, Product: Stormy Sea, Quantity: 1400.0
Shipping Order# 2 to Home Depot, Product: Stormy Sea, Quantity: 700.0
Production Order# 2, item Stormy Sea, produced 2100.0
321 Stormy Sea, Available: 0.0, Backorders: 0.0
[PO2 Stormy Sea 2000.0]
123 Flower Field, Available: 0.0, Backorders: 0.0
[PO1 Flower Field 1000.0]
```

5 What is to be done?

- Read the execise carfelly and study the provided UML diagram.
- Check our the starter code and using the UML diagram, write your solution.
- Please do use the same names for your classes, class variables and methods as indicated in the associated UML diagram.

- Throughout your coding process, follow the UML diagram carefully.
- Test your application thoroughly. You do not need to submit your test code.
- Submit only classes and interfaces shown on the UML diagram.

6 The UML Diagram



7 Checklist

Have you...

- tested your code using **Java 1.8**?
- committed and pushed the correct files in the correct directory?
- tested your solution, made any necessary changes, and re-committed if necessary?