**A Simple Point-of-Sale (POS) System**

**1. OBJECTIVE**

The objective of this assignment is to get you familiar with all (mostly) Java basics and some advance concepts, which include the syntax, Java input/output, arrays, lists, classes, objects, inheritance and interfaces utilizing polymorphism. After performing this assignment, you should be able to master these Java concepts. You need to use Eclipse for this assignment.

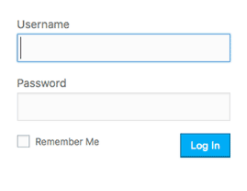
**2. TASK**

In this assignment, the overall task is to develop an application that records sales and

handle payments as in a retail store.  Study the **POS Project Requirements** below. You are recommended to use a GUI for your application; it could be web if you like. Your team will come up with a UX design for your product.

In this POS system, a cashier will be asked to log in the system using his/her user id and password. (There are no limits on the maximum tries of incorrect passwords or users). The system logs in the cashier if proper credentials are provided otherwise a message is displayed to reenter the credentials. (If the user id or password is wrong, the cashier will be asked to re-enter as follows).

Welcome to the Point-Of-Sale Registration System



Please enter your user name: alinaqvi

Please enter your password: thisismypassword

**POS Project Requirements**

Point of Sale System is developed to support supermarket-type store operations. In particular software shall:

Authentication:

**UI**

* 1. Allow the cashier to start a new sale and allow add/remove items to a new sale.
* **UI, backend,** 
  1. Once all items are added to the sale the cashier will request for cash to finalize the sale.

**UI, backend**

* 1. The system will keep track of the amount of sales ($) at each register for each cashier. (This can be managed if you have unique identifiers assigned to each cashier)
* **Backend**
  1. Registers will record the register number, the user (cashier), the dates and times of sale, sale items, and the amount of sales.
* **UI (enter register id, item id and qty (should be 1 or more)), backend.**
* **Validate item and available quantity (rename to onhands). Allow only what we can sell.** 
  1. For returns - Support cancellation of the entire sale as well as return of an individual item.
* UI (saleid, register id, item id, qty(1+). Backend: Just update the quantity and refund amount.
  1. Keep track of the inventory, including quantity, price, supplier, and outstanding backroom inventory orders.

Example of inventory: subtract number of items sold from the master file which should contain items on hand.

**Item to supplier 1 to 1. 1 supplier to many items.**

* 1. Support inventory management (add/remove item to/from inventory, setting threshold for re-ordering. Threshold is when the system should signal the management to order a product if on hand count goes below the threshold.)

**Order to supplier when threshold reached on each sale.**

**Rename outstanding to pending.**

**One more field for re-order quantity**

* 1. Support report generation:
* Inventory report (listing off al inventory items with name, quantity, threshold, supplier, and quantity of items in pending orders. Pending orders mean – the order has been placed however the items have not been received as yet to be put on the shelves or backroom)

**UI**

* Cashier report (X and Z reading: X reading-All sales activities pertaining to a particular cashier for a particular shift, Z reading-All aggregated sales activities pertaining to a All cashier for all shifts in a day)

**UI**

1) First design the system and use UML. -

2) Then divide up the work between team members

3) Make sure you wrote algorithms before you start coding.

4) Write Tests

5) Complete coding.

**Add tax.  
Use file system (no database).  
No coupons no discounts  
No credit cards  
No rest API expected.**

**TODO:**Java docs  
UML diagramduring sale: Invalid item id on POS system., invalid register  
during return: validate if sales id is valid, qty is less or equal to sold item qty, avoid second return, avoid multiple returns, support return reason at item level.  
extract POS and POR in abstract class and/or interface.  
Check for inventory before allowing to enter quantity.No item id found during sale. Enter 9999 item id or do not allow to add?  
Default check for item exists.  
Search inventory(Optional)  
Refactoring of class names and repackaging  
Add some welcome header, store name, etc  
2 decimals in the receipt or report generated  
Abstract interfaces and abstract classes **Fix sonalint issues**