1. SouvenirApp.java
   1. Public static void main(String args[]): In the main method, launch the login window by creating an instance of LoginHandler class and invoking its showLoginUI().
   2. No need to create TDD for this method.
2. LoginHandler.java
   1. showLoginUI(): create the instance of LoginUI class. When the control returns from the main UI by clicking OK or cancel button, take appropriate action. If OK is pressed, call validateUser method of the same class and pass the username and password as parameters. If cancel is pressed, return. No TDD required.
   2. Static Boolean validateUser(username:String,password:String): This is a static method. Read the users.dat file and verify the username and password with every row in the file until end of file. Ignore case of username while comparing. Return true if valid, else false.
3. LoginUI.java
   1. ActionPerformed() of OK
      1. Call validateUser method of LoginHandler.
      2. If valid, create an instance of MainMenuHandler and call the method startMenuUI() to invoke the MainMenuUI.
      3. If invalid, show message “Invalid username, password” and prompt for username and password again.
      4. In actionPerformed of Cancel, System.exit(0);

No TDD required.

1. MainMenuHandler.java
   1. startMenuUI(): create an instance of MainMenuUI to display MainMenuUI.
2. MainMenuUI.java
   1. **actionPerformed for AddMember**:

Create an instance of MemberManager. Call the start method present in MemberManager.

* 1. **ActionPerformed for AddCategory:**

Create an instance of CategoryManager. Call the start method present in CategoryManager.

* 1. **ActionPerformed** of AddProduct:

Create an instance of ProductManager. Call the start method present in ProductManager.

* 1. **ActionPerformed** of CheckInventory:

Create an instance of inventoryManager and Call the start method present in inventoryManager.

* 1. **ActionPerformed** of Billing:

Create an instance of BillingManager and Call the start method present in BillingManager.

* 1. **ActionPerformed** of ProductReport
  2. **ActonPerformed** of CategoryReport
  3. **ActonPerformed** of MemberReport
  4. **ActonPerformed** of TransactionReport

1. MemberManager.java **:**
   1. Void Start(): Instantiate MemberUI (this) , to display the memberUI.
   2. Void addMember(String, String): Parameters passed are the username and password. Method throws MemberAlreadyPresentException.
      1. Call the getMember(String) method to validate if the member already exist. Parameter passed is the Member id.
      2. Catch MemberNotFoundException, and add the member into the Member.dat file as follows if the exception occurs. Set the points to 0 and firstTimeStatus to ‘Y’

DataFile d = new MemberDataFile();

d.writeFile(Member);

* + 1. If returned a Member object, throw “Member Already Present” exception.
  1. Boolean getMember(String): Parameter passed is the member id. Method throws MemberNotFoundException
     1. Read the Member.dat as follows

DataFile d = new MemberDataFile();

ArrayList<Member> mlist = d.readFile(Member);

* + 1. Loop through the mlist and compare the member id of the Member objects with the member id received as parameter.
    2. If found, return the Member object. Else throw “Member Not Found” Exception.

Change in MemberManager: validateMember method is replaced with a getMember method which will return a Member object if a member id is passed to it. This will save a couple of read operations of the member file during check out.

* 1. Void updateMember (Member): Parameter passed is the member object, whose points and firstTimeStatus are updated in the object. Objective is to write it to the file.
     1. //How to update the field depends on what Puneet is gonna provide in the DataFile methods. Need to wait for that to complete.

1. MemberUI.java**:**
   1. Constructor: Create the constructor MemberUI(MemberManager), and assign the reference to a class level MemberManager variable.
   2. actionPerformed**() of the AddMember button:** Fetch details from the memberUI and call addMember method of manager. In case of “Member Already Present” exception, show error dialog on screen.
2. CategoryManager.java**:**
   1. Void start():Instantiate ViewCategoryUI(this) & display it.
   2. ArrayList<Category> getCategories():

DataFile object. DataFile d = new CategoryDataFile();

Return d.readFile();

* 1. Void addCategory(String,String): Call getCategory(String) method and pass the categoryCode as parameter. If the method returns an object, throw “Category Already Present” exception. Else write the new category into the Category.dat file as follows.

DataFile object. DataFile d = new CategoryDataFile();

d.writeFile(Category);

Create a VendorFile in the path as mentioned in the CONSTANTS.java

1. ViewCategoryUI.java **:**
   1. Constructor: Create the constructor ViewCategoryUI(CategoryManager), and assign the reference to a class level CategoryManager variable. Call the getCategories() of CategoryManager and load the grid from ArrayList.
   2. Action performed on New Category button: Instantiate addCategoryUI and display it.
2. AddCategoryUI.java

* actionPerformed of OK button: Call the addCategory method of the CategoryManager.

1. ProductManager.java**:**
   1. Void start():Instantiate addProductUI (this) & display it.
   2. Void addProduct(Product): Assign a sequence number to the product by reading the number of products with the category code in theProduct.dat file, and allocating the next sequence number to it.

Read the file as follows.

DataFile object. DataFile d = new ProductDataFile();

ArrayList<Product> plist = d.readFile();

Loop through the list to find the number of instances of the Categorycode present, and then increment it by 1 to assign a new sequence number and product code to the product.

Product code should be of the format “Category Code/sequence#” e.g., CLO/1. Write the new product into the product.dat file as follows.

DataFile object. DataFile d = new ProductDataFile();

d.writeFile(Product);

* 1. ArrayList<Product> getProductsToOrder():
     1. Read the Products.dat file as follows.

DataFile object. DataFile d = new ProductDataFile();

ArrayList<Product> plist = d.readFile();

* + 1. For each product in the list, if the quantity is less than the threshold, add that product to a new ArrayList for products to order.
    2. Set plist to null and return the new ArrayList.
  1. Product getProduct(String)
     1. Read the Products.dat file as follows.

DataFile object. DataFile d = new ProductDataFile();

ArrayList<Product> plist = d.readFile();

* + 1. Loop through the arraylist and compare the product code received as parameter to the product code of each object, and return the Product object that matches with the product code.

1. AddProductUI.java **:**
   1. Constructor: Create the constructor AddProductUI(ProductManager), and assign the reference to a class level ProductManager variable.
      1. Create a new instance of CategoryManager and call the getCategories() of it and load the combo box from ArrayList.
   2. The action listener of OK button will call addProduct method of the ProductManager in its ActionPerformedmethod.
2. inventoryManager.java
   1. Void start():Instantiate CheckInventoryUI (this) & display it.
   2. ArrayList<Product> getProductsBelowThreshold(): Create an instance of ProductManager and invoke its getProductsToOrder method.
   3. generatePO(ArrayList<Product>): *//to write the implementation, there is a dependency on the model of PO and POLineItems which Puneet is going to provide. Will be kept pending until then.*

*Logic we need here is, create POLineItems from the list of products (quantity in the PO Line Item should be equal to the orderquantity field in product), create the PO for each category (PO doesn’t have category as per the design. UML need to change). Create a PODataFile and write the PO and line items to it, and save the file in the path as read from CONSTNTS class and then use the write method to write the PO to a file named PO\_Category\_ddmmyy.txt. So for each category in the list, we will create a PO file.What is the vendor logic to apply here?*

*Changes needed: Constants class to have the path for the PO. Need a new child class inherited from DataFile, called PoDataFile.*

1. **CheckInventoryUI.java:**
   1. **Constructor**:
      1. **Create** the constructor CheckInventoryUI (InventoryManager), and assign the reference to a class level InventoryManager variable.
      2. Call the getProductsBelowThreshold() method of the inventory manager to get the list of products below threshold.
      3. Populate the grid with the attributes from each of the Product in the list returned.
   2. ActionPerformed method of the “Generate PO” button: Call the generatePO method of the InventoryManager, pass the products list returned in step ii above as parameter to the method.
2. **BillingManager**.java
   1. Define a class level Member object and set it to null. All operations need to check if member is null before any operations. Null member means, the operations are for a non-member.
   2. Void start():Instantiate BillingUI (this) & display it.
   3. Product getProductDetails(String): Accept the product code as parameter. Create an instance of the ProductManager and invoke its getProduct(String) method, passing the product code received as parameter. Method should return the object received from the getProduct method.
   4. Void addTransaction(ArrayList<TransactionLineItems>): create a new Transaction object (class level) and add the TransactionLineItems to it. //Dependency on how Puneet is gonna give methods for Transactions and TransactonLineItems
   5. Transaction getTransaction(): return the Transaction object with the TransactionLineItems in it.
   6. Boolean checkMemberDetails(String):
      1. Create an instance of the MemberManager at class level
      2. Call its getMember(String) method. Catch the MemberNotFound exception.
      3. If the method returns a Member object, store the object in a class level Member object, and return true.
      4. In case of MemberNotFoundException, return false.
   7. Int redeemPoints(int): Accepts the number of points the member wants to redeem as parameter and returns the number of points that can be redeemed.
      1. Retrieve the class level member object stored by the checkMemberDetails method.
      2. redeem = member.getPoints()>param?param:member.getPoints(); where param is the points received in the parameter and redeem is an int.
      3. Return redeem
   8. Void updatePoints(int) to update the points balance in the member.dat file.
      1. Retrieve the class level member object stored by the checkMemberDetails method.
      2. Call member.setPoints(pointsbalance)
      3. Call member.setFirstStatus(‘N’)
      4. Call MemberManager’s updateMember(Member) method, pass the member object as parameter. Change in MemberManager – add a method updateMember(Member)
   9. float getDiscount() to read and calculate the applicable discount. For first time member, discount is 20%. If not first time, 10% discount. If seasonal discounts are found in the Discount file, take the maximum discount from the member and seasonal discounts and return the % applicable.
      1. If Member object in BillingManager is not null,
         1. call member.getFirstTimeStatus() to see whether the member is a first time member or not. Change in Member class – add one more field firstTimeStatus which is a Char. Value can be Y for first time and N if transactions are done by the member. Add member has to create with value N.
         2. if firstTimeStatus is ‘Y’, set discount = 20. Else discount = 10
      2. Read the Discounts file as follows

DataFile object. DataFile d = new ProductDataFile();

ArrayList<Discount> plist = d.readFile(); //Need a new domain class for Discount. Else we cannot use the DataFile to read the file.

* + 1. If ArrayList is not null and size is >= 1, then loop through the arraylist to find the largest discount rate.
    2. Return the largest of the applicable discounts as float – member discount, seasonal discounts.

1. **BillingUI**.java
   1. **Constructor:** 
      1. **Create** the constructor BillingUI (BillingManager), and assign the reference to a class level BillingManager variable.
      2. Display the UI
   2. ActionPerformed method of Add button:
      1. If product code or quantity is null, display dialog “Please enter Product Code and Quantity required”. Else call the getProductDetails(String) of the BillingManager, pass the product code which the user entered, and accept the Product object returned by the method.
      2. If the quantity entered by the user is more than the quantity attribute of the product, display message “Available quantity for the product is only <nnn>” (Replace nnn with the quantity from Product object). Else, create a new TransactionLineItem and add it to the ArrayList of TransactionLineITems. Also add the product code, quantity, price, total price (quantity \* price) to the grid. Change in TransactionLineItems – all the mentioned fields to be part of the list of attributes.
      3. At the same time update the Total in the bottom of the grid with the sum of total prices for all products in the grid.
   3. *Like Add, Don’t we need a Remove button?*
   4. ActionPerformed of the CheckOut button:
      1. Create a new Transaction object, with the list of TransactionLineItems and add the Transaction object to the BillingManager instance using its addTransaction method //Dependency on how Puneet is gonna provide the methods for Transaction and TransactionLineITems classes.
      2. Create an instance of the CheckOutUI, and pass the BillingManager reference.
2. **CheckOutUI.java**
   1. **Constructor:** 
      1. **Create** the constructor CheckOutUI (BillingManager), and assign the reference to a class level BillingManager variable.
      2. Call the getTransaction() method of the billing manager, and traverse the list of TransactionLineItems and load it to the grid. Change in BillingManager – add methods addTransaction(ArrayList TransactionLineITems) and getTransaction(). Add a transaction object to the UML. Remove the TransactionManager as the same objective can be achieved through BillingManager.
      3. Calculate the total amount for the transaction and keep it in a class level variable.
      4. Display the CheckOutUI
   2. ActionPerformed() of the “Check Member” button:
      1. Open a dialog to accept the Member number.
      2. If OK button pressed in the dialog, validate for the empty/null value and call checkMemberDetails(String) method of BillingManager, passing the member id from the dialog as parameter. Change in BillingManager – added methods checkMemberDetails(String), redeemPoints(String), generateBill from CheckOutManager. CheckOutManager now doesn’t make sense and can be removed.
      3. If checkMemberDetails method returns True, make the “RedeemPoints” button enabled. If false, display message that “Member is not Found” and disable the “Redeem Points” button.
   3. ActionPerformed() of the Redeem Points button:
      1. Show dialog to enter points. On press of OK, validate for empty/null in the text box.
      2. Call the int redeemPoints(int) of Billing Manager. Capture the return value which is the number of points that can be redeemed in a class level variable.
      3. Calculate the dollar equivalent of the returned value and subtract from the total. Refresh the total on window.

Change in BillingManager – redeemPoints method to accept int as parameter. This is the number of points to redeem. Returns int, which is the number of points the member. Also add a method to update points.

Change in Product – price should be float.

Change in TransactionLineItem: Add amount as float

Change in Transaction: Add totalamount as float

* 1. ActionPerformed() of the Generate Bill button.
     1. Calculate discount by calling the getDiscounts() method of BillingManager. Change in BillingManager – add method getDiscount() to it. So no need of DiscountHandler now.
     2. Totalamount = totalamount(1-discount/100)
     3. Calculate acquiredpoints for the total amount after discount (apply the conversion logic, $100 = 1point for acquisition and 10 point = $1 for redeem).
     4. Calculate the pointsbalance = availablePoints + acquiredPoints-redeemedPoints, where pointsbalance is the final points calculated for the member, availablepoints is the number of points the member presently has which is in the points attribute of Member object, acquiredpoints is the number of points acquired in the present transaction, and redeemed points is the number of points redeemed in the present transaction.
     5. Call the updatePoints of the BillingManager, pass the pointsbalance as parameter.