Final Project: Shipping Information System

Alejandra De Osma, Christian Santiago, Mingrui Yuan, Christian Huber

CMS 270, Rollins College

Table of Contents

Narrative **3**
Object-Oriented Design **4**
Use Case Diagram (UCD) 4
Use Cases 5
Make Purchase 5
Make Online Purchase 6
Make In-Store Purchase 6
Create Customer Account 7
Add Products 7
Manage Inventory 8
Make Receipt 8
Class Diagram 9
Implementation **10**
Code Development 10
Code Files 11
Task Distribution 11
Takeaways **12**
Reflection 12
Proposed Changes 13
Honor Code **14**
Appendices **15**

Narrative

A clothing and general goods store requires an information system that will process purchases made through the store and manage the store inventory levels on a daily basis. The client would like for the information system to read raw data from an input file that contains information about the day-to-day operations of the store (i.e. the creation of products, customer accounts and purchases, restocking of products, pickup of store purchases) and perform the operations requested in the raw file. The client also wishes for the system to print out day-to-day receipt and inventory files indicating all the purchases fulfilled in one day and the inventory of the store at the end of each day, respectively.

All products sold by the store have a name, ID, price per unit and current stock in inventory. Products can (not must) be jackets, shirts or shoes. Jackets have a material (e.g. denim, leather) and a designer. Shirts have a color, cloth (e.g. polyester, cotton) and indication as to whether the shirt is formal. Shoes have a securing mechanism (e.g. buckle, shoelaces, Velcro), color and brand. Shoes and jackets are products that can be premium/custom-made. The client wishes for the program to recognize premium products (by certain key characteristics they contain) and alter their name and price in the system.

All purchases made in the store have an ID, total price, lists of product IDs and quantities bought, hold and completion statuses, and a date that the purchase was made. The client would also like for purchases to be able to determine how many days they have been open for. Purchases must be either online or in-store purchases. Online purchases have a shipping address, transit time, and cost for shipping (free for purchases over $60). The client would like for online purchases to be able to determine their own shipping cost. In-store purchases have a pickup-ready status and number of days a customer has left to pick up their purchase. The client wishes for store purchases

to canceled if not picked up after two days of its being placed and considers in store purchases fulfilled only once they are picked up. If there is not enough stock of a product to fulfill a purchase, the client wishes for the purchase to be put on hold and reattempted after the next restocking.

Purchases are made by customers who have an account. All customer accounts have a name, phone number, address, account ID and list of purchases made in the store.

Object-Oriented Design

Use Case Diagram (UCD)

[USE CASE DIAGRAM (UCD] (*see Appx. A)*

When making a UCD, the thought process involved might be more prominent than it shows on the diagram because there are multiple aspects to think about when making a UCD.

More specifically, it is vital to speculate on who the users are in this case. Furthermore, how many users are included in our project? Plus, what is the relationship between use cases and users, including how they interact with each other? With those various thought processes, the UCD will be especially helpful to our project structuring in the future.

In our project, there are two users involved in the system, which are customers and store managers. With the store as the system, "Make Purchase" is our most essential and intricate use case, it involves "Online Purchase" and "In-Store Purchase." Also, a customer needs to create an account to be able to purchase. Thus, "Create Customer Account" extends the "Make Purchase" class. A customer also can add products. Therefore the "Add Products" is also a part of the "Make

Purchase" class. Moreover, customers can request a receipt from the store as well. This use case is related to both customer and store manager since store managers make them.

Use Cases

Make Purchase Use Case:

[MAKE PURCHASE ACTIVITY DIAGRAM] (*see Appx. A*)

In the Make Purchase use case, the control flow is as follows. The program checks the line being read for an indicator that there is purchase information to be read and handled, if there is then the control flow will continue to the next activity, if not it will terminate. If the control flow continues to the next activity the program will check whether the customer making the purchase has an account, if they do then the control flow will continue to the next activity, if not then the control flow will go to the Create Customer use case and then return to the next activity.

In the next activity the program will read the purchase data from the file and check whether the information contains an indicator that the data is for a store purchase or for an online purchase, if the indicator indicates that it is data for a store purchase the control flow will continue to the Make Store Purchase use case, if the indicator indicates that it is data for an online purchase the control flow will continue to the Make Online Purchase use case. After either of these cases are run the program will check if there is more purchase information to be read again, if so then the control flow will run through the preceding steps again, if not the control flow will terminate.

Make Online Purchase Use Case

[MAKE ONLINE PURCHASE ACTIVITY DIAGRAM] (*see Appx. A*)

When customers are completing an online purchase, the system first checks if all the products in their purchase are in stock. If they are, then the system is cleared to ship the order to the customer. Otherwise, If the product quantity is not met then the order will be placed on hold and notify the user. If the stock is resupplied and the order can be completed, the system will then calculate the shipping cost.

Before shipping the order, the system will check if the order total exceeds 60$ if it does, then the shipping is free. Otherwise, it will add 10$ to the order as a shipping fee.

After shipping the order, the system will update the order status to "In transit," and it can check if the order has been delivered or not. If the order has been delivered, then update the order status to "Delivered," or else the status will remain "In transit." Finally, the system will store the current order information in the customer's purchase history.

Make In-Store Purchase Use Case:

[MAKE IN-STORE PURCHASE ACTIVITY DIAGRAM] (*see Appx. A*)

In-Store Purchase shares many similar activities with the Online Purchase Activity Diagram. It first checks for product availability, if the purchased quantity is not met then the order will be placed on hold and notify the user. Once the products have restoked and the order can be met, it will transition to order ready for pick up. Otherwise, If the products within order were initially available, then the system will update its status to "Pick-up Ready." Then the system will remove the corresponding number of products from the current stock. Customers have two days to complete the pickup. Therefore, the system then checks whether the customer picked up the

order in time, change the order status to complete, and save the order information in the customers purchase history. Else, if the customer failed to do so, then the order will be canceled.

Create Customer Account Use Case:

[CREATE CUSTOMER ACCOUNT ACTIVITY DIAGRAM] (*see Appx. A*)

The "Create Customer Account" class is less complicated than the others. The system required the customer to enter their personal information and store them as a new customer. Then the system will generate a unique account number for each account. Finally, the system will increment the total account numbers, which makes it easy to keep track of how many users are registered as customers.

Add Products Use Case:

[ADD PRODUCTS ACTIVITY DIAGRAM] (*see Appx. A*)

This Activity Diagram visually displays how our program adds products to both purchases and stock. It begins by reading a file containing the products information. Given the information from the read file the program the decides if the products should be added to their stock or to a purchase. If the products are being restocked it adds the given products to their current stock and terminates the program. Else if the products are being added to a purchase, it first verifies if there is an existing purchase, if it is existing it adds the products to the purchase and terminates. Else is, the purchase is nonexistent it creates a new purchase, adds the corresponding products and finally terminates.

Manage Inventory Use Case:

[MANAGE INVENTORY ACTIVITY DIAGRAM] (*see Appx. A*)

This Activity Diagram demonstrates how our program manages inventory. It begins by reading a file, it then checks if the inventory for that particular product already exists; if it does not exist it goes ahead and creates an empty inventory, and if it does exist it simply moves to the next activity. In the next stage, given the information from the read file, the program decides if it should restock inventory, handle a purchase or print inventory levels.

If it is restocking the inventory, it adds the restoked product quantity to the inventory and the program terminates. Else if the file indicated to print the inventory levels, the program will create an output file containing the inventory levels, and the program will terminate. Lastly, If the program has to handle a purchase, it will first removes the purchased number of products form the inventory, if the purchased quantity is not met then the order will be placed on hold and notify the user, else the purchase will be completed and the program will terminate.

Make Receipt Use Case:

[MAKE RECEIPT ACTIVITY DIAGRAM] (*see Appx. A*)

This Activity Diagram illustrates how our program, makes and manages receipts. It begins by identifying the customer that completed the purchase. If the customer wants a printed copy of their receipt, the program will then find the customer's corresponding purchase, locate the receipt data, print the receipt and then terminate the program. If the customer does not want a receipt the program will not print a receipt and simply terminate.

Class Diagram

[CLASS DIAGRAM] (*see Appx. A*)

The UML Class Diagram is a very helpful diagram for communicating the class contracts and relationships between classes. In our case, we have a set of 8 interrelated classes and one interface. The most central class of our Class Diagram (CD) is Purchase, which has an association with the Product superclass and is a dependent of the Customer class, with which it has a composition relationship because Customers have Purchases. The association between Purchase and Product is self-explanatory as Purchases are comprised of Products. As part of our program, we decided to have some types of products that would be more specialized than products instantiated as general Product objects. These are Shirt, Shoe and Jacket. The latter two classes additionally differentiate themselves from the Shirt class by implementing the interface ValueOptions that allows them to have more premium features, which come at a price increase.

As we decided to have OnlinePurchases and StorePurchases, the superclass Purchase must be abstract since it would otherwise lead to ambiguity. StorePurchases specialize themselves from Purchase by adding the values pickupReady, which determines if a certain Purchase is ready for pickup, and customerPickUpDaysLeft, which will cancel the order once the value has reached zero.

It is easy to say that the CD has presented itself especially useful when multiple group members were working on the code at the same time, because it ensured that the communication between methods and specific relationships of the classes were always adhered to.

Implementation

Code Development

The code for our project was developed using the Object-Oriented Design approach. This entails creating an abstraction of the problem we decided to conquer, as well as designing a set of diagrams following the UML guidelines in order to establish class contracts and determine the specific relationships between our classes and the functioning of our code.

We split the workload up over all group members so everyone could contribute to the initial code by writing a selection of assigned classes. Naturally, every member has their own creative style, which led to different formats, exception handlings, etc. Therefore, we created a formatting template which we applied to our code in order to make it more uniform.

Next, we started the painstakingly intensive debugging process. Due to the complexity of our program, we encountered some insignificant errors. However, we were also facing a number of underlying problems in our code. After many hours, we located the origins of our errors and traced them back to wrong index handling, and unsynchronized method calls. For example, our program would create an inventory file for a certain day before the days' purchases and restocks were performed, which would lead to ambiguities between customer receipts and the inventory status.

After creating and debugging all classes, including the driver, we developed JUNIT test suites for a selection of three classes (Shirt, Purchase, Customer) that exhaustively test their methods for all possible scenarios.

The code for this project was created collaboratively, using multiple platforms such as Repl.it, Eclipse and WebEx. Initially, all group members were working on the code at once on

Repl.it. After some time, we realized that using this platform was not ideal for the purpose of our project because Repl.it does not have a function to test written code before it is compiled and executed, as Eclipse does. This led to many disruptions to our workflow and we unanimously decided to instead have only two people dedicated to coding. The previously encountered issues were avoided by pair-programming on a single workstation via WebEx. This streamlined our workflow significantly and we were able to get the whole code working before the presentation of our project.

Due to our group members working from different time zones (EST, GMC+1), we were also able to work on the code almost non-stop, as the coder going to sleep would pass it on to the next one who would already be awake.

It is therefore easy to see that, even though we encountered technical and managerial issues at first, we quickly realized our shortcomings and corrected them, which led to a sustainable and effective workflow. Without the changes performed by our group, we would have not been able to complete the project by the deadline.

Code Files:

[CODE FILES] *(see Appx. B)*

Task Distribution

For a project of this magnitude, we needed to work as a team to reach the best possible outcome. In the initial stages of the project, we worked together to decide on the focus of our program. We then continued with the abstraction, which was also done collaboratively. Christian Huber then completed the UML Class Diagram. The use case diagram and the activity diagrams

were completed as a group. Moving on to the code, the classes were coded by Alejandra De Osma, Christian Santiago, and Christian Huber. Both Christians handled the Driver; they also took charge of writing the text files and debugging the program. Alejandra de Osma coded the test suites and completed any additional changes to the UCD. Alejandra De Osma also completed the presentation with the help of Mingrui Yuan, which was then revised by every member of the team.

Takeaways

Reflection

Debugging for this project turned out to be a much more difficult endeavor than expected, with code either crashing or dispensing nonsense input for a day during the end stage of its development, this was problematic as it kept us from being able to analyze the output files of our program for logistical errors for a longer time than we would have wanted. We believe that if we were to have more rigorously tested our code during various segments of its development, we would have been able to catch some of the major bugs before they had the capability to create larger problems in our overall program.

Having group members living in different time zones was both an advantage and disadvantage we coped with during development as it made scheduling planning-designated meeting times slightly more difficult. However, time zone difference did offer us the advantage of coder switching, as when one coder was getting too tired to code (because it was night-time), another coder was just waking up and rearing to code.

Towards the end of the project, we also realized that we should have taken the assigned Diagrams more seriously in the beginning because they would have made understanding and

communicating the functionalities and relationships easier. Especially our Use Case Diagram and Activity Diagrams needed serious touch-ups periodically after updating the code. In a perfect world, this should not be the case because the code should be oriented on those diagrams, and not vice versa.

Adding to these issues, our workflow got significantly distressed by miscommunication from certain group members that led to an imbalanced distribution of tasks and time spent working on the project. Unfortunately, this issue can only be avoided if all members of a group work together and communicate possibly arising problems in a timely manner. Since this was not the case, the rest of the group had to carry a significant workload of other group member(s), which, naturally, had a negative effect on both overall morale, as well as productivity.

While we encountered some issues and problems in our project, it is also important to acknowledge the involvement and engagement of other group members that ultimately led to a functioning program with high complexity. Even though some things did not go as planned, our group still showed a very good effort and proudly presents our Store Information System today.

Proposed Changes:

During the coding of our program we entertained the idea of having the input file divided into separate files for each day being processed by the system. Ultimately, we did not choose to have this be in our final product as the idea was proposed too close to the deadline. In retrospect, this would have been a good idea as it is likely that a store would produce raw text data (to be analyzed by our system) in singular files at the end of each business day instead of appending data to one large batch-like text file of all the multiple days operations.

Another future change would be to include a user interface that would allow users to interact with the system in a more intuitive way, which would avoid having the user type in the different commands painstakingly into the input file. This could be achieved with the Façade structural design pattern.

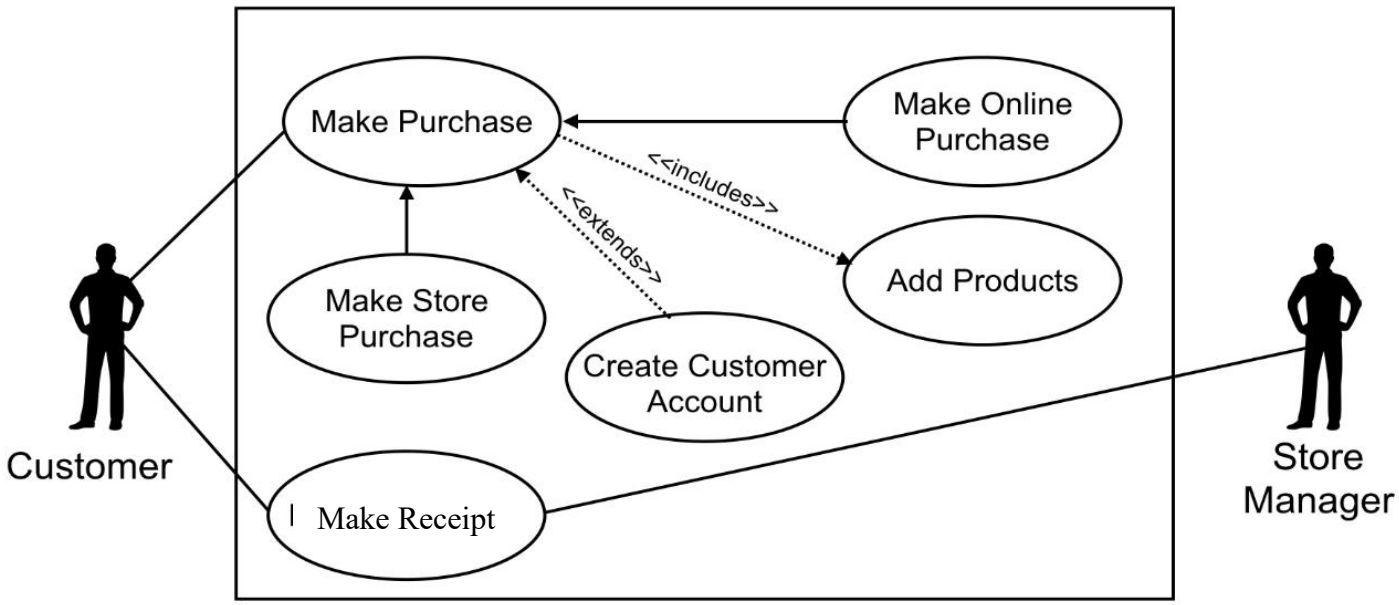
Finally, our last addition to this project would have been the creation of test suits for every class. This change would not only ensure that our code was working correctly, but it would have also facilitated the debugging process.

Honor Code

"On our honor, we have not received, nor given, nor witnessed any unauthorized assistance on this work."

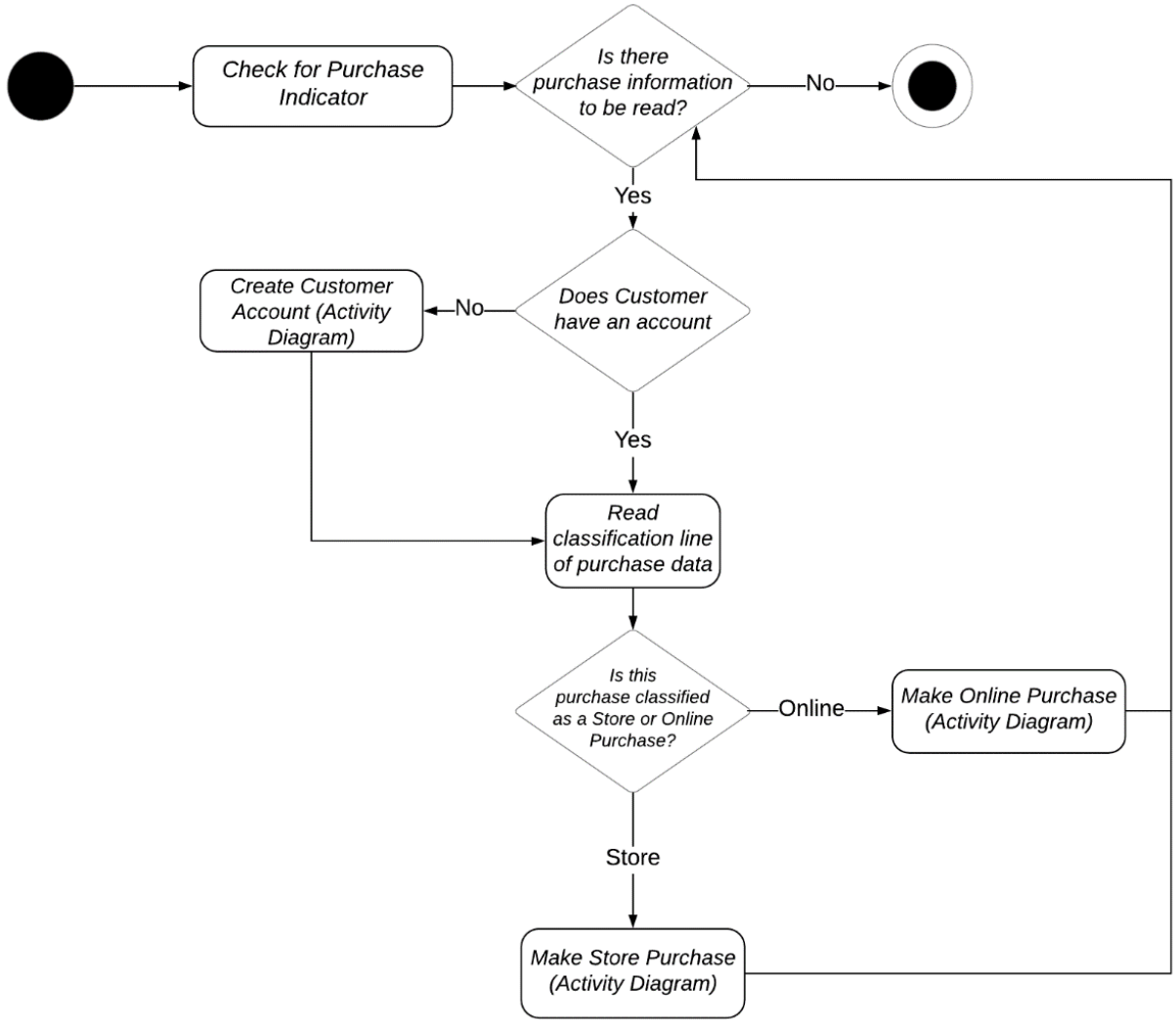
APPENDIX A

Use Case Diagram (UCD)

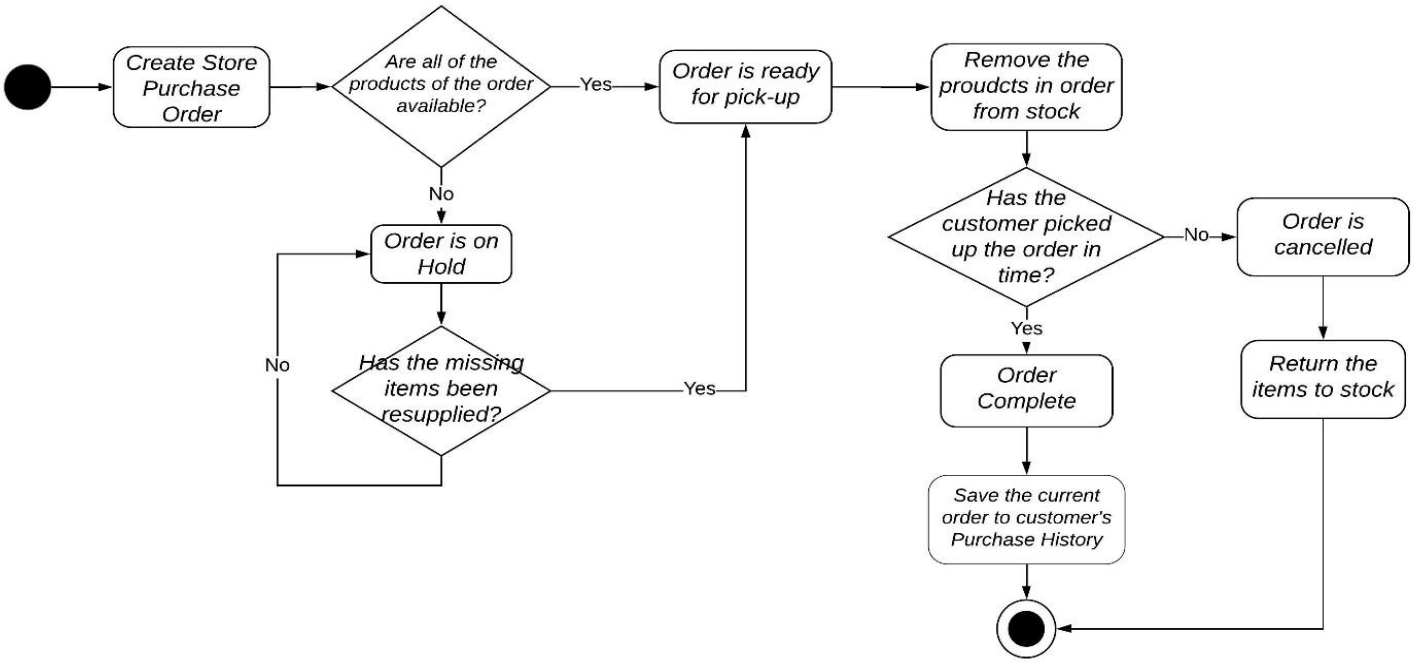


Activity Diagrams

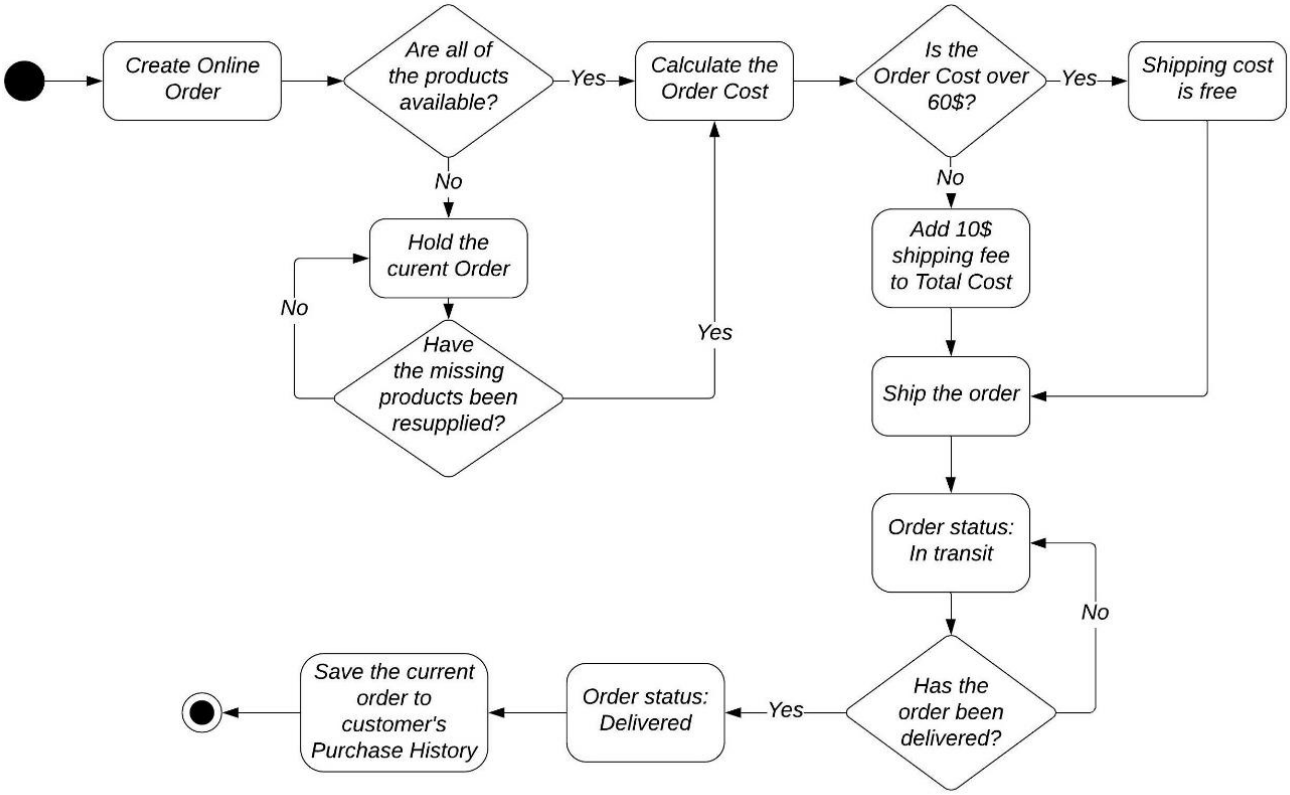
Make Purchase Activity Diagram



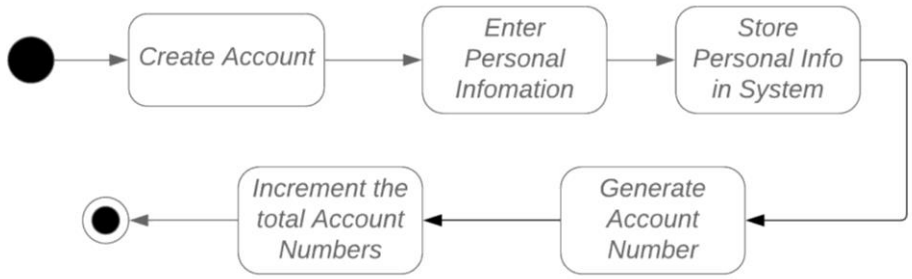
Make Store Purchase Activity Diagram



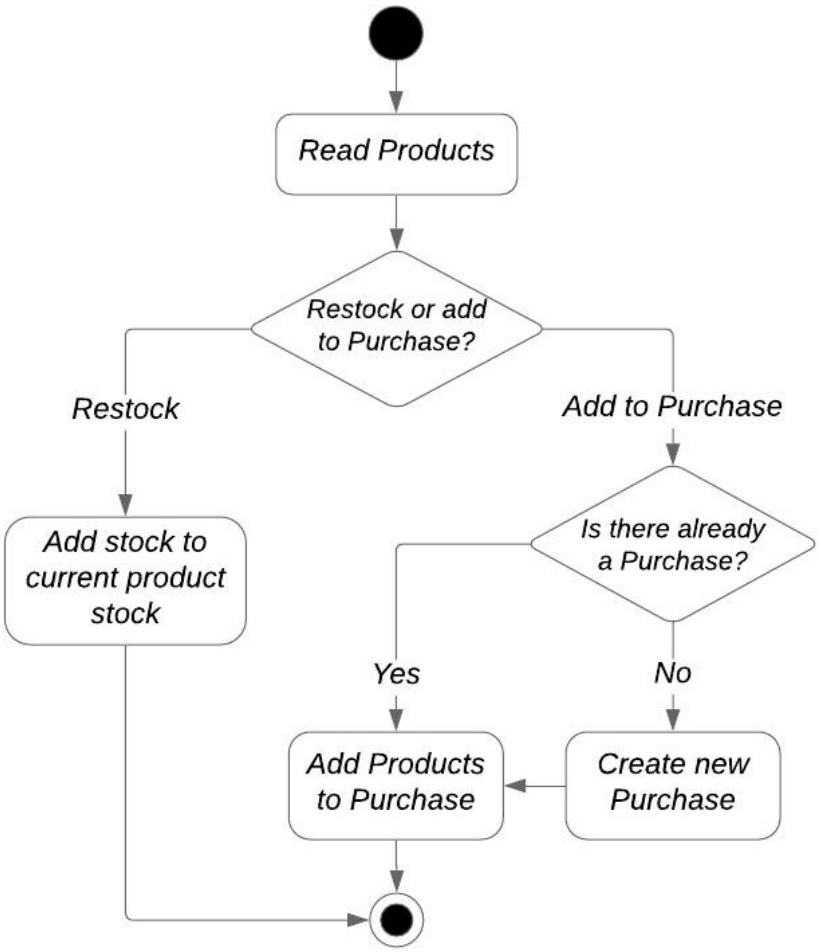
Make Online Purchase Activity Diagram



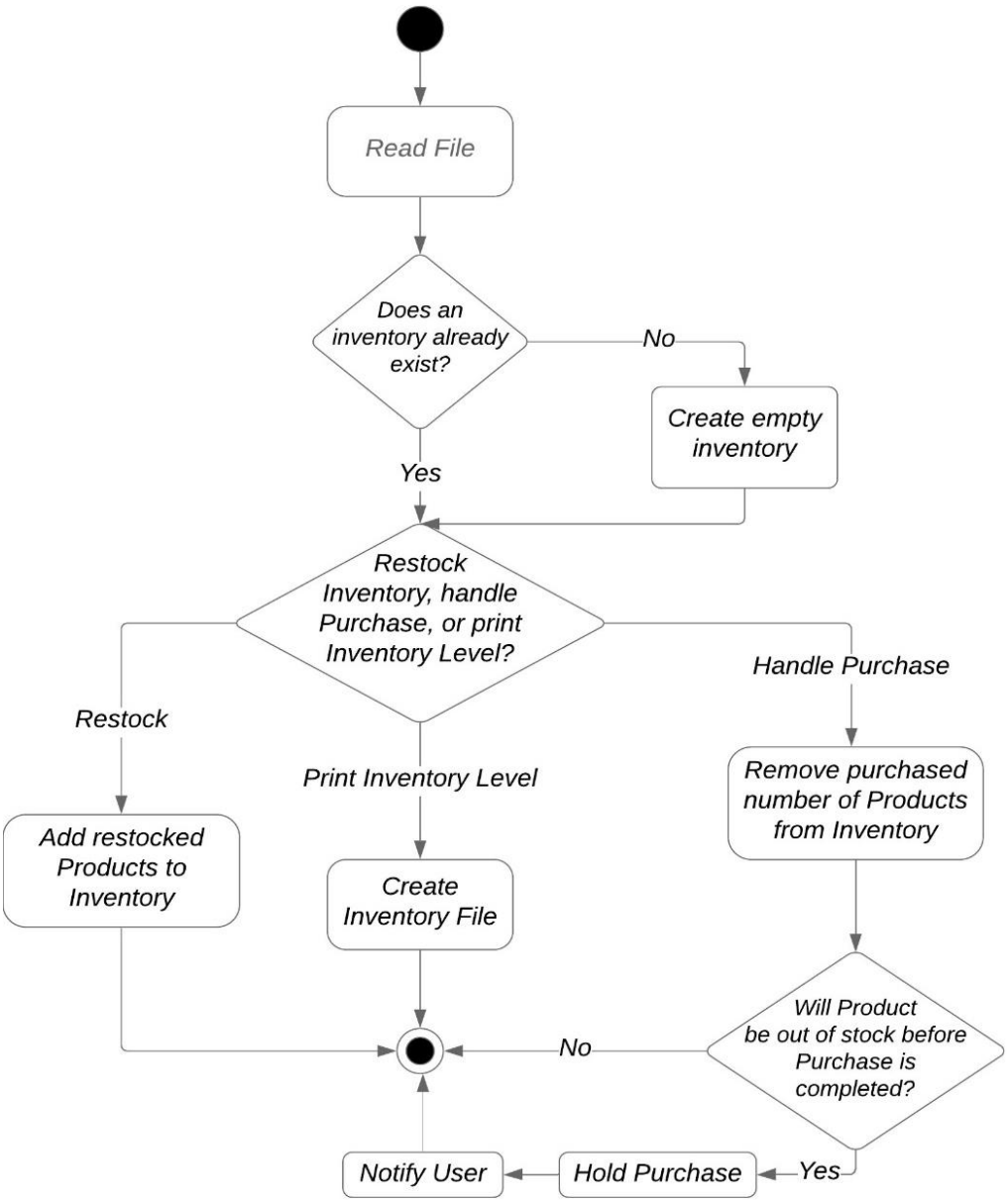
Create Customer Account Activity Diagram



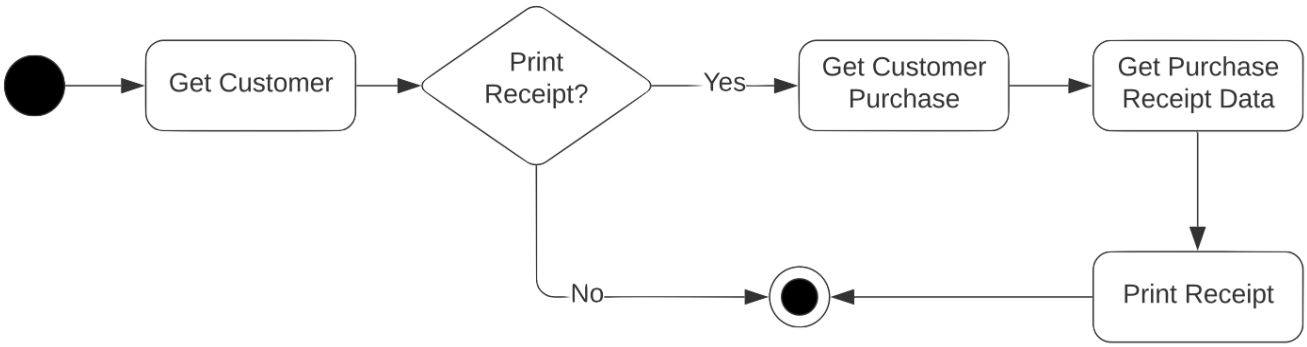
Add Products Activity Diagram



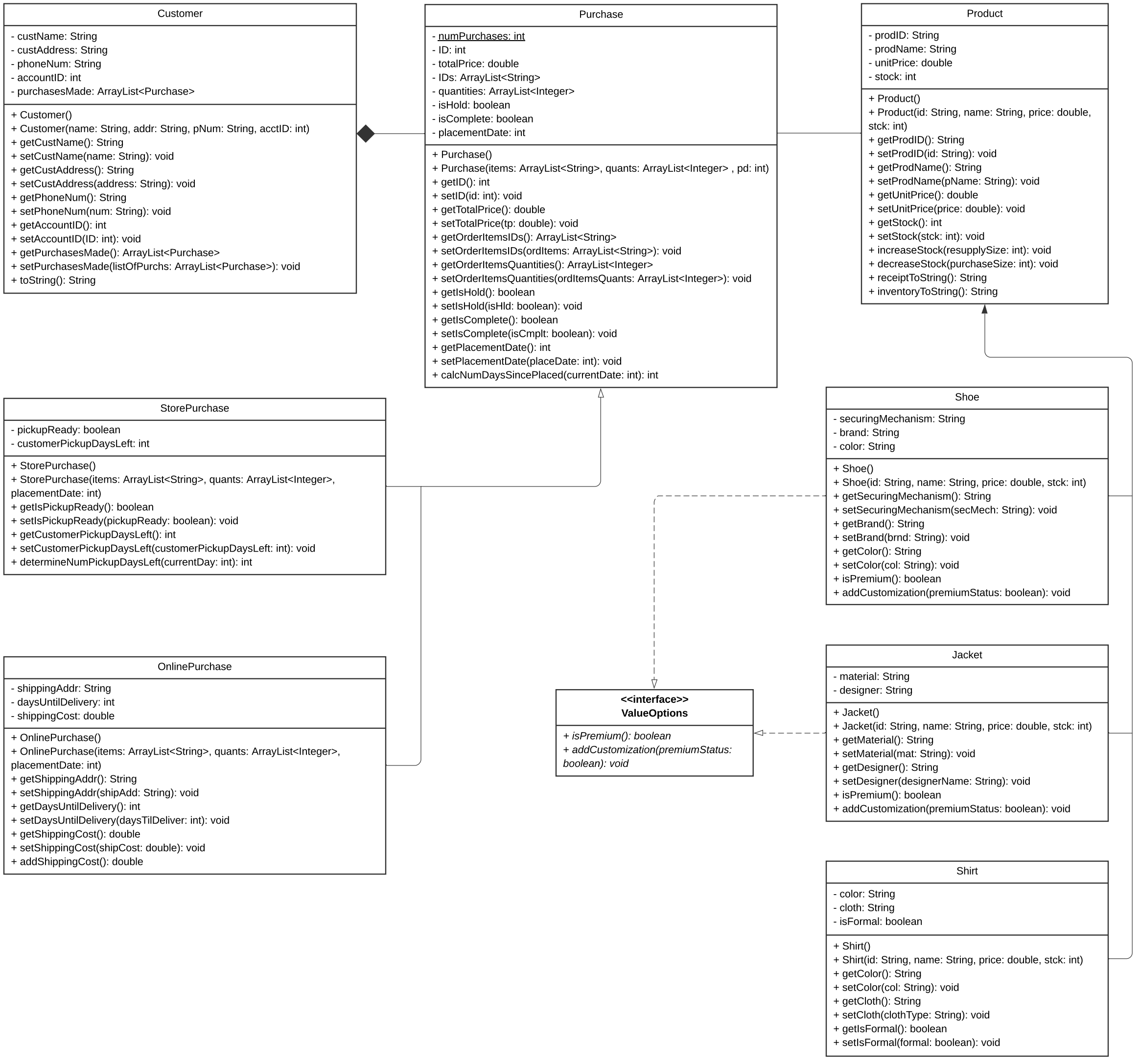
Manage Inventory Activity Diagram



Make Receipt Activity Diagram



Store Information System



APPENDIX B

```java
// Name: Alejandra de Osma, Minguri Yuan, Christian Santiago, Christian Huber
// Assignment: Final Project Store Shipping System.
// Course: CMS 270  Object Oriented Design and Development
// Due Date: 04/24/20
// Honor Declaration: "On our honor, we have not given, nor received, nor witnessed any unauthorized assistance on this work."
// ---------------------------------------------------------------------------------------------------------------------------
// ---------------------------------------------------------------------------------------------------------------------------
// ---------------------------------------------------------------------------------------------------------------------------
import java.util.*;
import java.io.*;

public class StoreInformationSystem {
     // ----------------------------------------------------------
     // ----------------------------------------------------------
     // MAIN
     public static void main (String[] args) {
          processStoreData();
          System.out.println("\n-------Done-------");
     }

     // ----------------------------------------------------------
     // ----------------------------------------------------------
     // BUSINESS METHODS
     public static void processStoreData() {
          try {
               File instructions = new File("src\\StoreData.txt");
Scanner dataScan = new Scanner (instructions);
int     dayCount = 0;
ArrayList <Customer> customerList = new ArrayList <Customer> ();
ArrayList <Product>  productInventoryList = new ArrayList <Product> ();

// -----
String [] actionIndicator = dataScan.nextLine().split("-");  // action = new day, restock, new customer, purchases, pickup

while (dataScan.hasNextLine()) {

     System.out.println("------------------------------------------------------------");
     dayCount = Integer.parseInt(actionIndicator [1]);

     actionIndicator = dataScan.nextLine().split("-");
     while (!actionIndicator[0].contentEquals("Day") && dataScan.hasNextLine()) {

          // -----
          if (actionIndicator[0].contentEquals("Day")) {
               dayCount = Integer.parseInt(actionIndicator[1]);
               cancelOverdueStorePickups(dayCount, customerList);
          }

          // -----
          if (actionIndicator[0].contentEquals("R")) {
               int numProducts = Integer.parseInt(actionIndicator[1]);  // Gets number of restock/stock orders
               for (int i = 0; i < numProducts; i ++) {
                    String[] tokens = dataScan.nextLine().split("-");
                    restock(productInventoryList, tokens);
               }
               processHoldOrders(customerList, productInventoryList); // Checks if held orders can be completed after restock
          }

          // -----
          if (actionIndicator[0].contentEquals("NC")) {
               int numCustomers = Integer.parseInt(actionIndicator[1]);  // Gets number of new customers in file
               for (int i = 0; i < numCustomers; i++) {
                    String[] tokens = dataScan.nextLine().split("-");
                    createNewCustomer(tokens, customerList);
               }
          }

          // -----
          if (actionIndicator[0].contentEquals("P")) {
               int numPurchases = Integer.parseInt(actionIndicator[1]);  // Gets number of new purchases in file
               for (int i = 0; i < numPurchases; i++) {
                    String[] tokens = dataScan.nextLine().split("-");
                    Purchase p = createNewPurchase(tokens, dayCount, productInventoryList, customerList);
                    System.out.println(attemptToCompletePurchase(p, productInventoryList, dayCount));
               }
          }

          // -----
          if (actionIndicator[0].contentEquals("PU")) {
               int numPickups = Integer.parseInt(actionIndicator[1]);  // Gets number of new pickups in file
               for (int i = 0; i < numPickups; i++) {
                    String[] tokens = dataScan.nextLine().split("-");
                    pickUp(customerList, Integer.parseInt(tokens[0]), dayCount, productInventoryList); // Processes orders that are in
stock
               }
          }

          if (dataScan.hasNextLine()) {
               actionIndicator = dataScan.nextLine().split("-");
          }
     }

     if(dayCount > 0) {
          printDailyReceipt(customerList, productInventoryList, dayCount);
     }
          printDailyInventory(productInventoryList, dayCount);
 }


 dataScan.close();
} catch (FileNotFoundException f) {
 System.out.println("File Not Found: processStoreData()");
 f.printStackTrace();
} catch (NullPointerException e) {
    System.out.println("Nullpointer Exception: processRawStoreData()");
    e.printStackTrace();
}
```

```
}

// ---------------------------------------------------------
// ---------------------------------------------------------
// HELPER METHODS
public static void printDailyReceipt(ArrayList<Customer> customers, ArrayList<Product> products, int dayCount) {
      try{
       if(dayCount != 0) {
        File receipts = new File("src\\CustomerReceipts_Day" + dayCount + ".txt");
            FileWriter writer = new FileWriter(receipts);
        writer.write("CUSTOMER RECEIPTS FOR DAY " + dayCount + ":\n\n");

        for(int i = 0; i < customers.size(); i++) {
             ArrayList<Purchase> purchasesMade = customers.get(i).getPurchasesMade();

                if (purchasesMade.size() != 0) {
                     Purchase tempPurchase = purchasesMade.get(purchasesMade.size()-1);

                     if (tempPurchase.getPlacementDate() == dayCount && !tempPurchase.getIsHold()) {
                          writer.write("Receipt for " + customers.get(i).getCustName() + ":\n\n");
                          writer.write(generateReceiptData(tempPurchase, products)
                                    + "\n-----------------------------------------------------------------\n\n");
                     }
                }
        }
         System.out.println("Receipt file for day " + dayCount + " has been created");
         writer.close();
       }

      } catch (FileNotFoundException e) {
       System.out.println("File Not Found: createReceipts(), day " + dayCount);
       e.printStackTrace();
      } catch (IOException f) {
       System.out.println("IOException: createReceipts(), day " + dayCount);
       f.printStackTrace();
       }
}

// ---------------------------------------------------------
public static void printDailyInventory(ArrayList<Product> products, int dayCount) {
      try{
           File inventory = new File("src\\StoreInventory_Day" + dayCount + ".txt");
           FileWriter writer = new FileWriter(inventory);
           String s = String.format("%29s%-12s|%5s |%17s|",
                     "_____________________________", " Unit Price ",  " Stock", "   Stock Level   ");
           writer.write("INVENTORY AT THE END OF DAY " + dayCount + ":\n\n" + s);
           for(int i = 0; i < products.size(); i++) {
                writer.write("\n" + products.get(i).inventoryToString() + "   " + getStatusBar(products.get(i)) + "  |");
           }
           System.out.println("\nInventory file for day " + dayCount + " has been created");
           writer.close();
      } catch(FileNotFoundException e) {
           System.out.println("File Not Found: printDailyInventory(), day " + dayCount);
           e.printStackTrace();
      } catch(IOException e) {
           System.out.println("IOException: printDailyInventory(), day " + dayCount);
           e.printStackTrace();
      }
}

// ---------------------------------------------------------
public static void cancelOverdueStorePickups (int dayCount, ArrayList<Customer> customers) {
      for (int i = 0; i < customers.size(); i++ ) {
           ArrayList <Purchase> purchases = new ArrayList<Purchase>();

           for (int g = 0; g < purchases.size(); g++) {
                if (purchases.get(g) instanceof StorePurchase && (dayCount - purchases.get(g).getPlacementDate()) >= 2) {
                     purchases.remove(purchases.get(g));   // removes overdue purchase
                     System.out.println("Purchase was not picked up in time by " + customers.get(i).getCustName() + " on day " + dayCount);
                }
           }
      }
}

// ---------------------------------------------------------
public static void restock(ArrayList<Product> productInventoryList, String [] tokens) {
      String prodIdentifier = tokens[0]; // tells program if the product is new (and what type) or existing product

      // -----
      if(prodIdentifier.contentEquals("shoe") ||prodIdentifier.contentEquals("shirt") ||    // checks if new product
         prodIdentifier.contentEquals("other")||prodIdentifier.contentEquals("jacket"))
      {
           productInventoryList.add(createProduct(tokens));
           System.out.printf("New Product added to Inventory: %-20s (" + tokens[4] + ")\n",tokens[2]);
      }

      // -----
      else // existing product
      {
           try {
                int index = searchForProductIndex (productInventoryList, tokens[0]);
                productInventoryList.get(index).increaseStock(Integer.parseInt(tokens[1]));   // updates found product stock
                System.out.printf("Restocked %-20s (+" + tokens[1] + ")\n", productInventoryList.get(index).getProdName());
           }

           catch (NullPointerException e) {
                System.out.println("Product ID: " + tokens[0] + " not found");             // should not occur but just in case
           }
      }
}

// ---------------------------------------------------------
public static void createNewCustomer (String[] customerData, ArrayList<Customer> customerList) {
      String tempName = customerData[0];
      String tempAddress    = customerData[1];
      String tempPhoneNum = customerData[2];
```

```
int    tempAcctID   = Integer.parseInt(customerData[3]);

Customer tempCustomer = new Customer(tempName, tempAddress, tempPhoneNum, tempAcctID);

customerList.add(tempCustomer);
System.out.println("New Customer created: " + tempCustomer.getCustName());
}

// ---------------------------------------------------------
public static Purchase createNewPurchase(String[] tokens, int dayCount, ArrayList <Product> productList, ArrayList <Customer> customerList) {
     Customer purchasingCustomer = searchCustomerListByID(customerList, tokens[1]);
     ArrayList <Purchase> customerPurchaseList = purchasingCustomer.getPurchasesMade();

     ArrayList <String>    tempOrderItemsIDs = new ArrayList <String> ();
     ArrayList <Integer> tempOrderItemsQuantities = new ArrayList <Integer> ();

     // -----
     if (tokens[0].contentEquals("OP") == true) {
          for (int i = 4; i < tokens.length; i += 2) {
               tempOrderItemsIDs.add(tokens[i-1]);
               tempOrderItemsQuantities.add(Integer.parseInt(tokens[i]));
          }

          OnlinePurchase tempPurchase = new OnlinePurchase(tempOrderItemsIDs, tempOrderItemsQuantities, dayCount);
          tempPurchase.setShippingAddr(purchasingCustomer.getCustAddress());
          tempPurchase.setDaysUntilDelivery(Integer.parseInt(tokens[2]));

          customerPurchaseList.add(tempPurchase);
          purchasingCustomer.setPurchasesMade(customerPurchaseList);
          System.out.println("\n" + purchasingCustomer.getCustName() + " submitted an Online Purchase");
          return tempPurchase;
     }

     // -----
     else {
          for (int i = 3; i < tokens.length; i += 2) {
               tempOrderItemsIDs.add(tokens[i-1]);
               tempOrderItemsQuantities.add(Integer.parseInt(tokens[i]));
          }

          StorePurchase tempPurchase = new StorePurchase(tempOrderItemsIDs, tempOrderItemsQuantities, dayCount);
          tempPurchase.setCustomerPickupDaysLeft(tempPurchase.determineNumPickupDaysLeft(dayCount));

          customerPurchaseList.add(tempPurchase);
          purchasingCustomer.setPurchasesMade(customerPurchaseList);
          System.out.println("\n" + purchasingCustomer.getCustName() + " submitted an In-Store Purchase");
          return tempPurchase;
     }
}

// ---------------------------------------------------------
public static String attemptToCompletePurchase(Purchase tempPurchase, ArrayList <Product> prodList, int dayCount) {
 // in store purchases are complete once they are picked up, online purchases are complete once sent
     try {
     if (tempPurchase.getOrderItemsIDs().size() == 0) {
          tempPurchase.setIsComplete(true);
          return "Empty Order - Processed and Disregarded";
     }

     // -----
     String holdWarning = "";

     for(int i = 0; i < tempPurchase.getOrderItemsIDs().size(); i++) {
          int soughtProductIndex = searchForProductIndex(prodList, tempPurchase.getOrderItemsIDs().get(i));
          int quantitySought = tempPurchase.getOrderItemsQuantities().get(i);

          if(prodList.get(soughtProductIndex).getStock() < quantitySought) {
               tempPurchase.setIsHold(true);
               holdWarning += ("Not enough " + prodList.get(soughtProductIndex).getProdName()
                         + " available to complete purchase");
          }
     }

     // -----
     if(tempPurchase.getIsHold() == true) {
          holdWarning += (", Order placed on Hold");
          tempPurchase.setPlacementDate(dayCount);
          return holdWarning;

     } else if (tempPurchase instanceof OnlinePurchase) {
          for (int i = 0; i < tempPurchase.getOrderItemsIDs().size(); i++) {
               int soughtProductIndex = searchForProductIndex(prodList, tempPurchase.getOrderItemsIDs().get(i));
               int quantitySought = tempPurchase.getOrderItemsQuantities().get(i);
               prodList.get(soughtProductIndex).decreaseStock(quantitySought);
               System.out.println("OP: Decreased " + prodList.get(soughtProductIndex).getProdName() + " stock by " + quantitySought);
          }
          tempPurchase.setIsComplete(true);
          return "Online Order successful\n";
     } else {
          return "In-Store Order Ready For Pickup";
     }
     }
     catch (IndexOutOfBoundsException e) {
          return "A requested product does not exist according to the provided ID";
     }
}

// ---------------------------------------------------------
public static void processHoldOrders(ArrayList<Customer> customers, ArrayList<Product> products) {
     for(int i = 0; i < customers.size(); i++) {
          ArrayList<Purchase> purchases = customers.get(i).getPurchasesMade();

          for(int f = 0; f < purchases.size(); f++) {

               if (purchases.get(f).getIsHold()) {
                    ArrayList<String> IDs = purchases.get(f).getOrderItemsIDs();
                    ArrayList<Integer> quantities = purchases.get(f).getOrderItemsQuantities();
```

```
                    boolean purchaseNoLongerHold = false;

                    for(int g = 0; g < IDs.size(); g++) {
                          purchaseNoLongerHold = true;
                          int index = searchForProductIndex(products, IDs.get(g));

                          if (products.get(index).getStock() < quantities.get(g)) {
                                purchaseNoLongerHold = false;
                                System.out.println("Customer " + customers.get(i).getAccountID() + "'s purchase still on hold.\n"
                                            + " -> Needs " + (quantities.get(g) - products.get(index).getStock())
                                            + " more " + products.get(index).getProdName());
                          }
                    }

                    if (purchaseNoLongerHold) {
                          purchases.get(f).setIsHold(false);
                          purchases.get(f).setIsComplete(true);
                          System.out.println(customers.get(i).getCustName() + "'s Purchase is no longer on Hold");

                          for (int g = 0; g < IDs.size(); g++) {
                                int index = searchForProductIndex(products, IDs.get(g));
                                products.get(index).decreaseStock(quantities.get(g));
                                System.out.println("Held orders: Decreased " + products.get(index).getProdName() + " stock by " +
quantities.get(g));

                          }
                    }
                }
            }
      }
}

// ------------------------------------------------------------
public static void pickUp(ArrayList<Customer> customerList, int custID, int dayCount, ArrayList <Product> products) {
      for(int i = 0; i < customerList.size(); i++) {
            if(customerList.get(i).getAccountID() == custID) {
                  ArrayList<Purchase> purchases = customerList.get(i).getPurchasesMade();

                  for(int f = 0; f < purchases.size(); f++) {
                        if(purchases.get(f).getIsComplete() == false && purchases.get(f) instanceof StorePurchase) {
                           purchases.get(f).setIsComplete(true);
                           ArrayList<String> items = purchases.get(f).getOrderItemsIDs();
                           ArrayList<Integer> quants = purchases.get(f).getOrderItemsQuantities();

                           for (int w = 0; w < items.size(); w++) {
                                 int index = searchForProductIndex(products, items.get(w));
                                 Product tempProduct = products.get(index);

                                 if(tempProduct.getStock() >= quants.get(w)) {
                                   tempProduct.decreaseStock(purchases.get(f).getOrderItemsQuantities().get(w));
                                   System.out.println("SP: Decreased " + products.get(index).getProdName()
                                               + " stock by " + purchases.get(f).getOrderItemsQuantities().get(w));
                                 } else {
                                     System.out.println(customerList.get(i).getCustName() + "'s order " + "is still on hold."
                                               + "\n  --> Still not enough of " + tempProduct.getProdName() + ".\n");
                                           return;
                                     }
                           }
                           System.out.println(customerList.get(i).getCustName() + " picked their order up");
                           System.out.println("In-Store Order successful\n");
                        }
                  }
            }
      }
}

// ------------------------------------------------------------
public static Product createProduct(String[] tokens) {
      try{
            if(tokens[0].contentEquals("shoe")) {
                  Shoe shoe = new Shoe(tokens[1], tokens[2], Double.parseDouble(tokens[3]), Integer.parseInt(tokens[4]));
                  shoe.setSecuringMechanism(tokens[5]);
                  shoe.setBrand(tokens[6]);
                  shoe.setColor(tokens[7]);
                  shoe.addCustomization(shoe.isPremium());
                  return shoe;
            } else if(tokens[0].contentEquals("jacket")) {
                  Jacket jacket = new Jacket(tokens[1], tokens[2], Double.parseDouble(tokens[3]), Integer.parseInt(tokens[4]));
                 jacket.setMaterial(tokens[5]);
                 jacket.setDesigner(tokens[6]);
                 jacket.addCustomization(jacket.isPremium());
                 return jacket;
            } else if(tokens[0].contentEquals("shirt")){
                  Shirt shirt = new Shirt(tokens[1], tokens[2], Double.parseDouble(tokens[3]), Integer.parseInt(tokens[4]));
                  shirt.setColor(tokens[5]);
                  shirt.setIsFormal(Boolean.parseBoolean(tokens[6]));
                  shirt.setCloth(tokens[7]);
                  return shirt;
              } else {
                  Product p = new Product(tokens[1], tokens[2], Double.parseDouble(tokens[3]), Integer.parseInt(tokens[4]));
                  return p;
              }
        }

        catch(IllegalArgumentException e) {
              System.out.println("Wrong data passed to restock method");
              e.printStackTrace();
        }

        return null;
}

// ------------------------------------------------------------
public static int searchForProductIndex (ArrayList <Product> productList, String targetProdID) {
      for(int i = 0; i < productList.size(); i++) {
            if(productList.get(i).getProdID().contentEquals(targetProdID)) {
                  return i;
           }
```

```
    }

      return -1;            // short circuited if product found ( always be found, data assumed to be valid)
}

//-------------------------------------------------------------
public static Customer searchCustomerListByID (ArrayList <Customer> customerList, String targetCustID) {
      boolean customerFound = false;
      int count = 0;

      while (customerFound == false && count < customerList.size())
      {
            Customer customerUnderStudy = customerList.get(count);
            if (customerUnderStudy.getAccountID() == Integer.parseInt(targetCustID)) {
                  customerFound = true;
                  return customerUnderStudy;
            }
            else {
                  count ++;
            }
      }

      return null;          // short circuits if Customer found
}

// -------------------------------------------------------------
    public static String getStatusBar(Product p) {
        String s = new String();
        s = "[";
        int count = (int) p.getStock() / 5;
        if(p.getStock() > 0 && p.getStock() < 5) {
         return("[#_________]");
        }
        for(int i = 0; i < count; i++) {
          s += "#";
        }
        while(count <= 10) {
          s += "_";
          count++;
        }
        s += "]";
        return s;
}

    // -------------------------------------------------------------
    public static String generateReceiptData(Purchase p, ArrayList<Product> products) {
          p = (Purchase) p;
          String data = new String();

          for(int i = 0; i < p.getOrderItemsIDs().size(); i++) {
                int index = searchForProductIndex(products, p.getOrderItemsIDs().get(i));

                double subtotal = products.get(index).getUnitPrice() * p.getOrderItemsQuantities().get(i);
                String s = String.format(" (%3d)  Subtotal: $%7.2f\n", p.getOrderItemsQuantities().get(i), subtotal);
                data += "    " + products.get(index).receiptToString() + s;

                p.setTotalPrice(p.getTotalPrice()+subtotal);;
          }

          if (p instanceof OnlinePurchase) {
                OnlinePurchase o = (OnlinePurchase) p;
                o.addShippingCost();
                p = o;
          }

          String x = String.format("\nTotal: $%-9.2f\n\nThank you for shopping with us!\n", p.getTotalPrice());
          data += x;
          return data;
      }

      // -------------------------------------------------------------
}
```

```java
// Name: Alejandra de Osma, Minguri Yuan, Christian Santiago, Christian Huber
// Assignment: Final Project Store Shipping System.
// Course: CMS 270  Object Oriented Design and Development
// Due Date: 04/24/20
// Honor Declaration: "On our honor, we have not given, nor received, nor witnessed any unauthorized assistance on this work."
// ---------------------------------------------------------------------------------------------------------------------------
// ---------------------------------------------------------------------------------------------------------------------------
// ---------------------------------------------------------------------------------------------------------------------------
import java.util.ArrayList;

public class Purchase {
	//-------------------------------------------------------
	//-------------------------------------------------------
	// DATAMEMBERS
	private static int numPurchases = 0;
	private int ID;
	private double totalPrice;
	private ArrayList<String> IDs;
	private ArrayList<Integer> orderItemsQuantities;
	private boolean isHold;
	private boolean isComplete;
	private int placementDate;

	//-------------------------------------------------------
	//-------------------------------------------------------
	// CONSTRUCTORS
	public Purchase() {
		numPurchases++;
		ID = numPurchases;
		totalPrice = 0;
		IDs = new ArrayList<String> ();
		orderItemsQuantities = new ArrayList<Integer> ();
		isHold = false;
		isComplete = false;
		placementDate = 0;
	}

	//-------------------------------------------------------
	public Purchase(ArrayList<String> items, ArrayList<Integer> quants, int pd) {
		numPurchases++;
		ID = numPurchases;
		totalPrice = 0;
		IDs = items;
		orderItemsQuantities = quants;
		isHold = false;
		isComplete = false;
		placementDate = pd;
	}

	//-------------------------------------------------------
	//-------------------------------------------------------
	// ACCESSORS AND MUTATORS
	public int getID() {
		return ID;
	}

	//-------------------------------------------------------
	public void setID(int id) {
		ID = id;
	}

	//-------------------------------------------------------
	public double getTotalPrice() {
		return totalPrice;
	}

	//-------------------------------------------------------
	public void setTotalPrice(double tp) {
		totalPrice = tp;
	}

	//-------------------------------------------------------
	public ArrayList<String> getOrderItemsIDs() {
		return IDs;
	}

	//-------------------------------------------------------
	public void setOrderItemsIDs(ArrayList<String> ordItems) {
		IDs = ordItems;
	}

	//-------------------------------------------------------
	public ArrayList<Integer> getOrderItemsQuantities() {
		return orderItemsQuantities;
	}

	//-------------------------------------------------------
	public void setOrderItemsQuantities(ArrayList<Integer> ordItemsQuants) {
		orderItemsQuantities = ordItemsQuants;
	}

	//-------------------------------------------------------
	public boolean getIsHold() {
		return isHold;
	}

	//-------------------------------------------------------
	public void setIsHold(boolean isHld) {
		isHold = isHld;
	}

	//-------------------------------------------------------
	public boolean getIsComplete() {
		return isComplete;
	}

	//-------------------------------------------------------
```

```
    public void setIsComplete(boolean isCmplt) {
        isComplete = isCmplt;
    }

    //-----------------------------------------------------------
    public int getPlacementDate() {
        return placementDate;
    }

    //-----------------------------------------------------------
    public void setPlacementDate(int placeDate) {
        placementDate = placeDate;
    }

    //-----------------------------------------------------------
    //-----------------------------------------------------------
    // BUSINESS METHOD
    public int calcNumDaysSincePlaced(int currentDate) {
        return currentDate - placementDate;
    }
}
```

```
// Name: Alejandra de Osma, Minguri Yuan, Christian Santiago, Christian Huber
// Assignment: Final Project Store Shipping System.
// Course: CMS 270  Object Oriented Design and Development
// Due Date: 04/24/20
// Honor Declaration: "On our honor, we have not given, nor received, nor witnessed any unauthorized assistance on this work."
// ------------------------------------------------------------------------------------------------------------------------
// ------------------------------------------------------------------------------------------------------------------------
// ------------------------------------------------------------------------------------------------------------------------
import static org.junit.jupiter.api.Assertions.*;
import java.util.ArrayList;
import org.junit.jupiter.api.Test;

class PurchaseTest {
	// -----------------------------------------------------
	// -----------------------------------------------------
	// CONSTRUCTOR UNIT TESTS
	@Test
	void testPurchase() {
		Purchase purchase = new Purchase();

		assertTrue(purchase instanceof Purchase);

	}

	// -----------------------------------------------------
	@Test
	void testPurchaseArgs() {

		ArrayList<String> itemList = new ArrayList<>();
		ArrayList<Integer>itemListQuantities = new ArrayList<> ();

		Purchase purchase = new Purchase(itemList,itemListQuantities,12);

		assertTrue(purchase instanceof Purchase);

	}

	// -----------------------------------------------------
	// -----------------------------------------------------
	// ACCESSORS AND MUTATORS
	@Test
	void testGetIDNoValue() {
		Purchase purchase = new Purchase();
		int ID = purchase.getID();
		int expectedID =1;

		assertEquals(ID,expectedID);
	}

	// -----------------------------------------------------
	@Test
	void testSetID() {
		Purchase purchase = new Purchase();
		purchase.setID(10);

		int ID = purchase.getID();
		int expectedID = 10;

		assertTrue(ID == expectedID);

	}

	// -----------------------------------------------------
	@Test
	void testGetID() {
```

```
	Purchase purchase = new Purchase();
	purchase.setID(20);

	int ID = purchase.getID();
	int expectedID = 20;

	assertTrue(ID == expectedID);

}

// ---------------------------------------------------------
void testGetTotalPriceNoValue() {
	Purchase purchase = new Purchase();
	double totPrice = purchase.getTotalPrice();
	double expectedPrice = 0;

	assertEquals(totPrice,expectedPrice);
}

// ---------------------------------------------------------
@Test
void testSetTotalPrice() {
	Purchase purchase = new Purchase();
	purchase.setTotalPrice(10);

	double totPrice = purchase.getTotalPrice();
	double expectedPrice = 10;

	assertTrue(totPrice == expectedPrice);

}

// ---------------------------------------------------------
@Test
void testGetTotalPrice() {
	Purchase purchase = new Purchase();
	purchase.setTotalPrice(20);

	double TotalPrice = purchase.getTotalPrice();
	double expectedPrice = 20;

	assertTrue(TotalPrice == expectedPrice);

}

// ---------------------------------------------------------
@Test
void testGetOrderItemsIDsEmptyArrayList() {
	Purchase purchase = new Purchase();
	ArrayList<String> list = new ArrayList<>();

	ArrayList<String> list2 = purchase.getOrderItemsIDs();

	assertEquals(list,list2);

}

// ---------------------------------------------------------
@Test
void testSetOrderItemsIDs() {
	Purchase purchase = new Purchase();
	ArrayList<String> list = new ArrayList<>();
	String s = "";
	list.add(s);
```

```
	purchase.setOrderItemsIDs(list);
	ArrayList<String> list2 = purchase.getOrderItemsIDs();

	assertEquals(list,list2);
}

// ---------------------------------------------------------
@Test
void testGetOrderItemsIDs() {
	Purchase purchase = new Purchase();

	ArrayList<String> list = new ArrayList<>();
	String s1 = "";
	String s2 = "";
	list.add(s1);
	list.add(s2);
	purchase.setOrderItemsIDs(list);

	ArrayList<String> list2 = purchase.getOrderItemsIDs();

	assertEquals(list,list2);

}

// ---------------------------------------------------------
@Test
void testGetOrderItemsQuantitiesEmptyArrayList() {
	Purchase purchase = new Purchase();
	ArrayList<Integer> list = new ArrayList<>();

	ArrayList<Integer> list2 = purchase.getOrderItemsQuantities();

	assertEquals(list,list2);
}

// ---------------------------------------------------------
@Test
void testSetOrderItemsQuantities() {
	Purchase purchase = new Purchase();
	ArrayList<Integer> list = new ArrayList<>();
	Integer i = 9;
	list.add(i);

	purchase.setOrderItemsQuantities(list);
	ArrayList<Integer> list2 = purchase.getOrderItemsQuantities();

	assertEquals(list,list2);
}

// ---------------------------------------------------------
@Test
void testGetOrderItemsQuantities() {
	Purchase purchase = new Purchase();

	ArrayList<Integer> list = new ArrayList<>();
	Integer i1 = 0;
	Integer i2 = 0;
	list.add(i1);
	list.add(i2);
	purchase.setOrderItemsQuantities(list);

	ArrayList<Integer> list2 = purchase.getOrderItemsQuantities();

	assertEquals(list,list2);
```

```
}

// -----------------------------------------------------------
@Test
void testGetIsHoldNoValue() {
	Purchase purchase = new Purchase();

	Boolean hold = purchase.getIsHold();
	boolean expectedHold = false;

	assertTrue(hold==expectedHold);

}

// -----------------------------------------------------------
@Test
void testSetIsHold() {
	Purchase purchase = new Purchase();
	purchase.setIsHold(true);

	Boolean hold = purchase.getIsHold();
	boolean expectedHold = true;

	assertTrue(hold==expectedHold);
}

// -----------------------------------------------------------
@Test
void testGetIsHold() {
	Purchase purchase = new Purchase();
	purchase.setIsHold(true);

	Boolean hold = purchase.getIsHold();
	boolean expectedHold = true;

	assertTrue(hold==expectedHold);
}

// -----------------------------------------------------------
@Test
void testGetIsCompleteNoValue() {
	Purchase purchase = new Purchase();

	boolean isComplete = purchase.getIsComplete();
	boolean expectedIsComplete= false ;

	assertTrue(isComplete == expectedIsComplete);
}

// -----------------------------------------------------------
@Test
void testSetIsComplete() {
	Purchase purchase = new Purchase();
	purchase.setIsComplete(true);

	boolean isComplete = purchase.getIsComplete();
	boolean expectedIsComplete= true ;

	assertTrue(isComplete == expectedIsComplete);

}

// -----------------------------------------------------------
@Test
```

```
void testGetIsComplete() {
      Purchase purchase = new Purchase();
      purchase.setIsComplete(true);

      boolean isComplete = purchase.getIsComplete();
      boolean expectedIsComplete= true ;

      assertTrue(isComplete == expectedIsComplete);

}

// ------------------------------------------------------
@Test
void testGetPlacementDateNoValue() {
      Purchase purchase = new Purchase();
      int date = purchase.getPlacementDate();
      int expectedDate = 0 ;

      assertTrue(date == expectedDate);

}

// ------------------------------------------------------
@Test
void testSetPlacementDate() {
      Purchase purchase = new Purchase();
      purchase.setPlacementDate(10);

      int date = purchase.getPlacementDate();
      int expectedDate =10 ;

      assertTrue(date == expectedDate);
}

// ------------------------------------------------------
@Test
void testGetPlacementDate() {
      Purchase purchase = new Purchase();
      purchase.setPlacementDate(10);

      int date = purchase.getPlacementDate();
      int expectedDate =10 ;

      assertTrue(date == expectedDate);
}

//------------------------------------------------------
@Test
void testCalcNumDaysSincePlaced() {
      Purchase purchase = new Purchase();
      purchase.setPlacementDate(3);
      int currentDate = 4;
      int daysWaiting = purchase.calcNumDaysSincePlaced(currentDate);
      int expectedDaysWaiting = 1 ;

      assertTrue(daysWaiting == expectedDaysWaiting) ;

   }
}
```

```java
// Name: Alejandra de Osma, Minguri Yuan, Christian Santiago, Christian Huber
// Assignment: Final Project Store Shipping System.
// Course: CMS 270  Object Oriented Design and Development
// Due Date: 04/24/20
// Honor Declaration: "On our honor, we have not given, nor received, nor witnessed any unauthorized assistance on this work."
// ---------------------------------------------------------------------------------------------------------------------------
// ---------------------------------------------------------------------------------------------------------------------------
// ---------------------------------------------------------------------------------------------------------------------------
import java.util.ArrayList;

public class OnlinePurchase extends Purchase {
	// DATA MEMBERS
	private String shippingAddr;
	private int daysUntilDelivery;
	private double shippingCost;

	//-------------------------------------------------------
	// CONSTRUCTORS
	public OnlinePurchase() {
		super ();
		shippingAddr = "";
		daysUntilDelivery = 0;
		shippingCost = 0;
	}

	//-------------------------------------------------------
public OnlinePurchase(ArrayList<String> items, ArrayList<Integer> quants, int placementDate) {
	super(items, quants, placementDate);
	shippingAddr = "";
	daysUntilDelivery = 0;
	shippingCost = 0;
}

//-------------------------------------------------------
public String getShippingAddr() {
	return shippingAddr;
}

//-------------------------------------------------------
public void setShippingAddr(String shipAdd) {
	shippingAddr = shipAdd;
}

//-------------------------------------------------------
public int getDaysUntilDelivery() {
	return daysUntilDelivery;
}

//-------------------------------------------------------
public void setDaysUntilDelivery(int daysTilDeliver) {
	daysUntilDelivery = daysTilDeliver;
}

//-------------------------------------------------------
	public double getShippingCost() {
		return shippingCost;
	}

	//-------------------------------------------------------
	public void setShippingCost(double shipCost) {
		shippingCost = shipCost;
	}

	// ---------------------------------------------------------
	// ---------------------------------------------------------
	// BUSINESS METHODS
	public void addShippingCost () {
		if (this.getTotalPrice() > 60.00) {
			setTotalPrice(getTotalPrice() + 10.00);
		}
	}

	// ---------------------------------------------------------
}
```

```java
// Name: Alejandra de Osma, Minguri Yuan, Christian Santiago, Christian Huber
// Assignment: Final Project Store Shipping System.
// Course: CMS 270  Object Oriented Design and Development
// Due Date: 04/24/20
// Honor Declaration: "On our honor, we have not given, nor received, nor witnessed any unauthorized assistance on this work."
// ---------------------------------------------------------------------------------------------------------------------------
// ---------------------------------------------------------------------------------------------------------------------------
// ---------------------------------------------------------------------------------------------------------------------------
import java.util.ArrayList;

public class StorePurchase extends Purchase {

	//-------------------------------------------------------
	// DATA MEMBERS
	private boolean pickupReady;
	private int customerPickupDaysLeft;

	//-------------------------------------------------------
	// CONSTRUCTORS
	public StorePurchase(){
		super();
		pickupReady = false;
		customerPickupDaysLeft = 0;
	}

	//-------------------------------------------------------
	public StorePurchase(ArrayList<String> items, ArrayList<Integer> quants, int placementDate) {
		super(items, quants, placementDate);
		pickupReady = false;
		customerPickupDaysLeft = 0;
	}

	//-------------------------------------------------------
	//-------------------------------------------------------
	// ACCESSORS AND MUTATORS
	public boolean getIsPickupReady() {
		return pickupReady;
	}

	//-------------------------------------------------------
	public void setIsPickupReady(boolean pickupReady) {
		this.pickupReady = pickupReady;
	}

	//-------------------------------------------------------
	public int getCustomerPickupDaysLeft() {
		return customerPickupDaysLeft;
	}

	//-------------------------------------------------------
	public void setCustomerPickupDaysLeft(int customerPickupDaysLeft) {
		this.customerPickupDaysLeft = customerPickupDaysLeft;
	}

	//-------------------------------------------------------
	//-------------------------------------------------------
	// BUSINESS METHODS
	public int determineNumPickupDaysLeft(int currentDay) {
		return 2 - (calcNumDaysSincePlaced(currentDay));
	}

	//-------------------------------------------------------
}
```

```java
// Name: Alejandra de Osma, Minguri Yuan, Christian Santiago, Christian Huber
// Assignment: Final Project Store Shipping System.
// Course: CMS 270  Object Oriented Design and Development
// Due Date: 04/24/20
// Honor Declaration: "On our honor, we have not given, nor received, nor witnessed any unauthorized assistance on this work."
// ---------------------------------------------------------------------------------------------------------------------------
// ---------------------------------------------------------------------------------------------------------------------------
// ---------------------------------------------------------------------------------------------------------------------------

public class Product {
     //-------------------------------------------------------
     //-------------------------------------------------------
     // DATA MEMBERS
     private String prodID;
     private String prodName;
     private double unitPrice;
     private int stock;

     //-------------------------------------------------------
     //-------------------------------------------------------
     // CONSTRUCTORS
     public Product() {
         prodID    = "";
         prodName  = "";
         unitPrice = 0.0;
         stock     = 0;
     }

     //-------------------------------------------------------
     public Product(String id, String name, double price, int stck) {
          prodID    = id;
          prodName  = name;
          unitPrice = price;
          stock     = stck;
     }

     //-------------------------------------------------------
     //-------------------------------------------------------
     // ACCESSORS AND MUTATORS
     public String getProdID() {
          return prodID;
     }

     //-------------------------------------------------------
     public void setProdID(String id) {
          prodID = id;
     }

     //-------------------------------------------------------
     public String getProdName() {
          return prodName;
     }

     //-------------------------------------------------------
     public void setProdName(String pName) {
          prodName = pName;
     }

     //-------------------------------------------------------
     public double getUnitPrice() {
          return unitPrice;
     }

     //-------------------------------------------------------
     public void setUnitPrice(double price) {
          unitPrice = price;
     }

     //-------------------------------------------------------
     public int getStock() {
          return stock;
     }

     //-------------------------------------------------------
     public void setStock(int stck) {
          stock = stck;
     }

     //-------------------------------------------------------
     //-------------------------------------------------------
     // BUSINESS METHODS
     public void increaseStock(int resupplySize) {
          stock += resupplySize;
     }

     //-------------------------------------------------------
     public void decreaseStock(int purchaseSize) {
          stock -= purchaseSize;
     }

     //-------------------------------------------------------
     // TOSTRING METHODS
     public String receiptToString() {
         return  String.format("%-6s%-23s%2s%9.2f", prodID, prodName, "$", unitPrice);
     }

    //-------------------------------------------------------
    public String inventoryToString() {
         String baseToString         = receiptToString();
         String inventoryAttenuation = String.format(" | %5d |", stock);
         baseToString                   += inventoryAttenuation;

         return baseToString;
       }
}
```

```java
// Name: Alejandra de Osma, Minguri Yuan, Christian Santiago, Christian Huber
// Assignment: Final Project Store Shipping System.
// Course: CMS 270  Object Oriented Design and Development
// Due Date: 04/24/20
// Honor Declaration: "On our honor, we have not given, nor received, nor witnessed any unauthorized assistance on this work."
// ---------------------------------------------------------------------------------------------------------------------------
// ---------------------------------------------------------------------------------------------------------------------------
// ---------------------------------------------------------------------------------------------------------------------------
public class Jacket extends Product implements ValueOptions {
	//-------------------------------------------------------
	//-------------------------------------------------------
	// DATA MEMBERS
	private String material;
	private String designer;

	//-------------------------------------------------------
	//-------------------------------------------------------
	// CONSTRUCTORS
	public Jacket() {
		super();
	material = "";
	designer = "";
	}

	//-------------------------------------------------------
	public Jacket(String id, String name, double price, int stck) {
		super(id, name, price, stck);
		material = "";
		designer = "";
}

//-------------------------------------------------------
//-------------------------------------------------------
// ACCESSORS AND MUTATORs
public String getMaterial() {
		return material;
}

//-------------------------------------------------------
public void setMaterial(String mat) {
		material = mat;
}

//-------------------------------------------------------
public String getDesigner() {
		return designer;
}

//-------------------------------------------------------
public void setDesigner(String designerName) {
		designer = designerName;
}

//-------------------------------------------------------
	//-------------------------------------------------------
	// VALUEOPTIONS INTERFACE IMPLEMENTATION
	public boolean isPremium() {
		if (material.contentEquals("leather") || material.contentEquals("down")
		 || material.contentEquals("suede")   || material.contentEquals("fur")
		 || designer.contentEquals("Gucci")   || designer.contentEquals("LV")
		 || designer.contentEquals("Prada"))
		{
			return true;
		} else {
			return false;
		}
	}

	//-------------------------------------------------------
	public void addCustomization(boolean premiumStatus) {
		if (premiumStatus == true) {
			setUnitPrice(getUnitPrice()+30.00);
			setProdName (getProdName ()+" CSTM");
		}
	}

	//-------------------------------------------------------
	//-------------------------------------------------------
}
```

```java
// Name: Alejandra de Osma, Minguri Yuan, Christian Santiago, Christian Huber
// Assignment: Final Project Store Shipping System.
// Course: CMS 270  Object Oriented Design and Development
// Due Date: 04/24/20
// Honor Declaration: "On our honor, we have not given, nor received, nor witnessed any unauthorized assistance on this work."
// ---------------------------------------------------------------------------------------------------------------------------
// ---------------------------------------------------------------------------------------------------------------------------
// ---------------------------------------------------------------------------------------------------------------------------
public class Shirt extends Product {
     //-----------------------------------------------------------
     //-----------------------------------------------------------
     // DATA MEMBERS
     private String color;
     private String cloth;
     private boolean isFormal;

     //-----------------------------------------------------------
     //-----------------------------------------------------------
     // CONSTRUCTORS
     public Shirt() {
          super ();
          color       = "";
          cloth  = "";
          isFormal = false;
     }

  //-----------------------------------------------------------
  public Shirt(String id, String name, double price, int stck) {
       super(id, name, price, stck);
      color        = "";
      cloth   = "";
      isFormal = false;
  }

  //-----------------------------------------------------------
  //-----------------------------------------------------------
  // ACCESSORS AND MUTATORS
  public String getColor() {
       return color;
  }

  //-----------------------------------------------------------
  public void setColor(String col) {
       color = col;
  }

  //-----------------------------------------------------------
public String getCloth () {
  return cloth;
}

  //-----------------------------------------------------------
public void setCloth (String clothType) {
     cloth = clothType;
     }

     //-----------------------------------------------------------
     public boolean getIsFormal () {
      return isFormal;
      }

     //-----------------------------------------------------------
     public void setIsFormal (boolean formal) {
      isFormal = formal;
      }

     //-----------------------------------------------------------
}
```

```java
// Name: Alejandra de Osma, Minguri Yuan, Christian Santiago, Christian Huber
// Assignment: Final Project Store Shipping System.
// Course: CMS 270  Object Oriented Design and Development
// Due Date: 04/24/20
// Honor Declaration: "On our honor, we have not given, nor received, nor witnessed any unauthorized assistance on this work."
// ------------------------------------------------------------------------------------------------------------------------
// ------------------------------------------------------------------------------------------------------------------------
// ------------------------------------------------------------------------------------------------------------------------
import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.Test;

class ShirtTest {
	//-----------------------------------------------------------
	//-----------------------------------------------------------
	// CONSTRUCTOR UNIT TESTS
	@Test
	void testShirt() {
		Shirt testShirt = new Shirt();

		assertTrue(testShirt instanceof Shirt);
	}

	//-----------------------------------------------------------
	@Test
	void testShirtArgs() {
		Shirt testShirt = new Shirt("1234","Alejandra",12.2,10);

		assertTrue(testShirt instanceof Shirt);
	}

	//-----------------------------------------------------------
	//-----------------------------------------------------------
	// ACCESSOR AND MUTATOR UNIT TESTS
	@Test
	void testGetColor() {
		Shirt  testShirt       = new Shirt();
		String expectedColor = "";

		String returnColor = testShirt.getColor();

		assertEquals(returnColor, expectedColor);
	}

	//-----------------------------------------------------------
	@Test
	void testSetColor() {
		Shirt testShirt  = new Shirt();
		String expectedColor = "Blue" ;

		testShirt.setColor("Blue");
		String returnColor = testShirt.getColor();

		assertEquals(returnColor, expectedColor);
	}

	//-----------------------------------------------------------
	@Test
	void testGetCloth() {
		Shirt testShirt  = new Shirt();
		String expectedCloth = "";

		String returnCloth = testShirt.getCloth();

		assertEquals(returnCloth,expectedCloth);
	}

	//-----------------------------------------------------------
	@Test
	void testSetCloth() {
		Shirt testShirt  = new Shirt();
		String expectedCloth = "Cotton" ;

		testShirt.setCloth("Cotton");
		String returnCloth = testShirt.getCloth();

		assertEquals(returnCloth,expectedCloth);

	}

	//-----------------------------------------------------------
	@Test
	void testGetIsFormal() {
		Shirt testShirt = new Shirt();
		boolean expectedFormal = false ;

		boolean returnFormal = testShirt.getIsFormal();

		assertEquals(returnFormal,expectedFormal);
	}

	//-----------------------------------------------------------
	@Test
	void testSetIsFormal() {
		Shirt testShirt = new Shirt();
		boolean expectedFormal = true ;

		testShirt.setIsFormal(true);
		boolean returnedFormal = testShirt.getIsFormal();

		assertEquals(returnedFormal,expectedFormal);

	//-----------------------------------------------------------
	}
}
```

```java
// Name: Alejandra de Osma, Minguri Yuan, Christian Santiago, Christian Huber
// Assignment: Final Project Store Shipping System.
// Course: CMS 270  Object Oriented Design and Development
// Due Date: 04/24/20
// Honor Declaration: "On our honor, we have not given, nor received, nor witnessed any unauthorized assistance on this work."
// ---------------------------------------------------------------------------------------------------------------------------
// ---------------------------------------------------------------------------------------------------------------------------
// ---------------------------------------------------------------------------------------------------------------------------
public class Shoe extends Product implements ValueOptions{
	//-------------------------------------------------------
	//-------------------------------------------------------
	// DATA MEMBERS
	private String securingMechanism;
	private String brand;
	private String color;

	//-------------------------------------------------------
	//-------------------------------------------------------
	// CONSTRUCTORS
	public Shoe() {
		super();
		securingMechanism = "";
		brand = "";
		color = "";
	}

	//-------------------------------------------------------
	public Shoe(String id, String name, double price, int stck) {
		super(id, name, price, stck);
		securingMechanism = "";
		brand             = "";
		color             = "";
	}

	//-------------------------------------------------------
	//-------------------------------------------------------
	// ACCESSORS AND MUTATORS
	public String getSecuringMechanism() {
		return securingMechanism;
	}

	//-------------------------------------------------------
	public void setSecuringMechanism(String secMech) {
		securingMechanism = secMech;
	}

	//-------------------------------------------------------
	public String getBrand() {
		return brand;
	}

	//-------------------------------------------------------
	public void setBrand(String brnd) {
		brand = brnd;
	}

	//-------------------------------------------------------
	public String getColor() {
		return color;
	}

	//-------------------------------------------------------
	public void setColor(String col) {
		color = col;
	}

	//-------------------------------------------------------
	//-------------------------------------------------------
	// VALUEOPTIONS INTERFACE IMPLEMENTATION
	public boolean isPremium() {
		if (securingMechanism.contentEquals("buckle") || color.contentEquals("metallic gold")
		    || color.contentEquals("metallic silver") || brand.contentEquals("Crocs")
		    || brand.contentEquals("Air Jordan")      || brand.contentEquals("Lebron James")
		     || brand.contentEquals("Yeezy")           || brand.contentEquals("LV")
		     || brand.contentEquals("Manolo"))
		{
			return true;
		} else {                                  // added other premium options
			return false;
		}
	}

	//-------------------------------------------------------
	public void addCustomization(boolean premiumStatus) {
		if (premiumStatus == true) {
			setUnitPrice(getUnitPrice() + 70);  // changed customization price
			setProdName(getProdName()+" CSTM");     // add custom to name of jacket
		}
	}

	//-------------------------------------------------------
}
```

```java
// Name: Alejandra de Osma, Minguri Yuan, Christian Santiago, Christian Huber
// Assignment: Final Project Store Shipping System.
// Course: CMS 270  Object Oriented Design and Development
// Due Date: 04/24/20
// Honor Declaration: "On our honor, we have not given, nor received, nor witnessed any unauthorized assistance on this work."
// ---------------------------------------------------------------------------------------------------------------------------
// ---------------------------------------------------------------------------------------------------------------------------
// ---------------------------------------------------------------------------------------------------------------------------
import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.Test;
import java.util.ArrayList;

class CustomerTest {
	//-----------------------------------------------------------
	//-----------------------------------------------------------
	// CONSTRUCTOR METHODS UNIT TESTS
	@Test
	void testCustomerNoArgs() {
		Customer testCustomer = new Customer() ;

		assertTrue(testCustomer instanceof Customer);
	}

   //-----------------------------------------------------------
	@Test
	void testCustomerArgs() {
		Customer testCustomer = new Customer("kate","Holt AVE 221","954 673 2300",1234) ;

		assertTrue(testCustomer instanceof Customer);
	}

	//-----------------------------------------------------------
	//-----------------------------------------------------------
	// ACCESSOR AND MUTATOR UNIT TESTS
	@Test
	void testGetCustName() {
		Customer testCustomer = new Customer("kate","Holt AVE 221","954 673 2300",1234) ;
		String   expectedName = "kate";

		String returnName = testCustomer.getCustName();

		assertEquals(returnName, expectedName);
	}

//-----------------------------------------------------------
	@Test
	void testSetCustName() {
		Customer testCustomer = new Customer();
		String   expectedName = "Alejandra";

		testCustomer.setCustName("Alejandra");
		String changedName = testCustomer.getCustName();

		assertEquals(changedName, expectedName);
	}

//-----------------------------------------------------------
	@Test
	void testGetCustAddress() {
		Customer testCustomer    = new Customer("kate","Holt AVE 221","954 673 2300",1234) ;
		String   expectedAddress = "Holt AVE 221";

		String returnedAddress = testCustomer.getCustAddress();

		assertEquals(returnedAddress, expectedAddress);
	}

	//-----------------------------------------------------------
	@Test
	void testSetCustAddress() {
		Customer testCustomer    = new Customer();
		String   expectedAddress = "Rollins College";

		testCustomer.setCustAddress("Rollins College");
		String returnedAddress = testCustomer.getCustAddress();

		assertEquals(returnedAddress, expectedAddress);
	}

	//-----------------------------------------------------------
	@Test
	void testGetPhoneNum() {
		Customer testCustomer  = new Customer("kate","Holt AVE 221","954 673 2300",1234) ;
		String   expectedPhone = "954 673 2300";

		String returnPhone = testCustomer.getPhoneNum();

		assertEquals(returnPhone, expectedPhone);
	}

	//-----------------------------------------------------------
	@Test
	void testSetPhoneNum() {
		Customer testCustomer = new Customer();
		String expectedPhone  = "954 673 2300";

		testCustomer.setPhoneNum("954 673 2300");
		String returnPhone = testCustomer.getPhoneNum();

		assertEquals(returnPhone, expectedPhone);
	}

	//-----------------------------------------------------------
	@Test
	void testGetAccountID() {
		Customer testCustomer = new Customer("kate","Holt AVE 221","9546732300",1234) ;
		int expectedAccountID = 1234;
```

```
	int returnedAccountID = testCustomer.getAccountID();

	assertEquals(returnedAccountID, expectedAccountID);
}

//-----------------------------------------------------------
@Test
void testSetAccountID() {
	Customer testCustomer = new Customer();
	int expectedAccountID = 1234;

	testCustomer.setAccountID(1234);
	int returnedAccountID = testCustomer.getAccountID();

	assertEquals(returnedAccountID,expectedAccountID);
}

//-----------------------------------------------------------
@Test
void testGetPurchasesMade() {
	Customer             testCustomer         = new Customer   ();
	ArrayList<Purchase> expectedPurchaseList = new ArrayList<>();

	ArrayList<Purchase> returnPurchaseList= testCustomer.getPurchasesMade();

	assertEquals(expectedPurchaseList, returnPurchaseList);
}

//-----------------------------------------------------------
@Test
void testSetPurchasesMade() {
	ArrayList<Purchase> expectedPurchaseList = new ArrayList<>();
	expectedPurchaseList.add(new Purchase ());
	Customer testCustomer = new Customer();
	testCustomer.setPurchasesMade(expectedPurchaseList);

	ArrayList<Purchase> returnedList = testCustomer.getPurchasesMade();

	assertEquals(returnedList, expectedPurchaseList);
}

//-----------------------------------------------------------
@Test
void testToString() {
	Customer testCustomer = new Customer("Kate","Holt AVE 221","9546732300",1234);
	String expectedString = "Customer: "+"Kate"+"\nAccount ID: "+1234+"\nPhone Number: "+"9546732300"+"\nAddress: "+"Holt AVE 221";

   String toString = testCustomer.toString();

	assertEquals(toString ,expectedString);
}

//-----------------------------------------------------------
}
```

```java
// Name: Alejandra de Osma, Minguri Yuan, Christian Santiago, Christian Huber
// Assignment: Final Project Store Shipping System.
// Course: CMS 270  Object Oriented Design and Development
// Due Date: 04/24/20
// Honor Declaration: "On our honor, we have not given, nor received, nor witnessed any unauthorized assistance on this work."
// ---------------------------------------------------------------------------------------------------------------------------
// ---------------------------------------------------------------------------------------------------------------------------
// ---------------------------------------------------------------------------------------------------------------------------
public interface ValueOptions {
     public boolean isPremium();
     public void addCustomization(boolean premiumStatus);
}
```

```java
// Name: Alejandra de Osma, Minguri Yuan, Christian Santiago, Christian Huber
// Assignment: Final Project Store Shipping System.
// Course: CMS 270  Object Oriented Design and Development
// Due Date: 04/24/20
// Honor Declaration: "On our honor, we have not given, nor received, nor witnessed any unauthorized assistance on this work."
// ---------------------------------------------------------------------------------------------------------------------------
// ---------------------------------------------------------------------------------------------------------------------------
// ---------------------------------------------------------------------------------------------------------------------------
import java.util.ArrayList;

public class Customer {
	// ---------------------------------------------------------
	// ---------------------------------------------------------
	// DATA MEMBERS
	private String custName;
	private String custAddress;
	private String phoneNum;
	private int accountID;
	private ArrayList<Purchase> purchasesMade;

	// ---------------------------------------------------------
	// ---------------------------------------------------------
	// CONSTRUCTORS
	public Customer() {
		custName      = "";
	    custAddress   = "";
	    phoneNum      = "";
	    accountID     = 0 ;
	    purchasesMade = new ArrayList <Purchase> ();
}

// ---------------------------------------------------------
public Customer(String name, String addr, String pNum, int acctID) {
		custName      = name;
		custAddress   = addr;
		phoneNum      = pNum;
		accountID     = acctID;
		purchasesMade = new ArrayList <Purchase> ();
}

//---------------------------------------------------------
// ---------------------------------------------------------
// ACCESSORS AND MUTATORS
public String getCustName() {
		return custName;
}

// ---------------------------------------------------------
public void setCustName(String name) {
		custName = name;
}

//---------------------------------------------------------
public String getCustAddress() {
		return custAddress;
}

// ---------------------------------------------------------
public void setCustAddress(String address) {
		custAddress = address;
}

//---------------------------------------------------------
public String getPhoneNum() {
		return phoneNum;
}

//---------------------------------------------------------
public void setPhoneNum(String pnum) {
		phoneNum = pnum;
}

//---------------------------------------------------------
public int getAccountID() {
		return accountID;
}

//---------------------------------------------------------
public void setAccountID(int id) {
		accountID = id;
}

	//---------------------------------------------------------
	public ArrayList<Purchase> getPurchasesMade() {
		return purchasesMade;
	}

	//---------------------------------------------------------
	public void setPurchasesMade(ArrayList<Purchase> listOfPurchs) {
		purchasesMade = listOfPurchs;
	}

	//---------------------------------------------------------
	//---------------------------------------------------------
	// TOSTRING
	public String toString(){
		return "Customer: "+ custName+"\nAccount ID: "+accountID+"\nPhone Number: "+phoneNum+"\nAddress: "+custAddress ;
	}

	//---------------------------------------------------------
}
```

StoreData.txt

```
Day-0
R-4
shoe-49992-Radford Boots-65.99-15-Shoelaces-Timberlands-metallic silver
jacket-99857-Puffy ULTRA-233.99-2-Wool-Gucci
shirt-16376-Caribbean Shirt-10.99-25-yellow-false-polyester
other-57339-Toenail Clippers-5.75-12
Day-1
R-4
49992-15
99857-2
other-99879-Silver Link Bracelet-200.00-3
shirt-96521-Woodfort Flannel-19.99-30-red and black-false-wool
NC-2
Mingrui Yuan-211 Florida St.-410 730 6879-12345
Christian Huber-021 Jump St.-122 303 2211-64582
P-2
OP-12345-3-57339-2-49992-2
SP-64582-99857-1-57339-1
PU-1
64582
Day-2
R-3
57339-20
99857-3
16376-10
NC-2
Alejandra de Osma-111 My Yacht Blvd.-123 123 4567-95082
Christian Santiago-Madarame Ave.-400 867 5309-92015
P-2
OP-12345-3-57339-10
SP-95082-16376-2-99879-1-57339-2
Day-3
R-2
99879-4
jacket-33654-High Cut Blue Boys-39.99-5-Denim-Blu Boy Gang
P-3
OP-92015-5-99857-1
SP-95082-96521-2-99879-1
SP-64582-99857-1
PU-1
95082
Day-4
R-1
other-38827-An Actual Camel-6001.00-1
NC-1
Amadeus-123 Chicken Bone Ave.-344 590 2334-44021
P-1
OP-44021-4-33654-2-99879-1-49992-6
PU-1
64582
Day-5
P-2
SP-44021-38827-3
OP-92015-5-33654-2-96521-4
PU-1
44021
```