CS1040
Program Construction

Lab Exercise 7

NAME : SANDEEPA H.N.A

INDEX NO : 210571L

```
import java.util.*;
import java.io.*;
import java.time.LocalDateTime;

/*
 * This Java code represents a Point of Sales (POS) system for the Super-Saving supermarket
chain.
 * The code allows the cashier to enter the item code to add grocery items to a bill. Information
such as
 * price, weight/size of the product, date of manufacturing and expiry, and manufacturer name
is fetched
 * from a hardcoded database.
 *
 * The POS system supports the following functionalities:
 * 1. Handling pending bills: If a customer forgets to weigh some items, the bill can be kept as
pending and
 *    the cashier can continue with other customers' billing. Serialization is used to store and
retrieve
 *    pending bills.
 * 2. Discounts: Each item can be given a discount ranging from 0-75%.
 * 3. Bill Generation: The system generates a bill containing the cashier's name, branch,
customer name (if
 *    registered), item list (unit price, quantity, discount, net price), total discount, total price,
date,
 *    and time.
 *
 * The code consists of several classes:
 * - GroceryItem: Represents a grocery item with its details such as item code, name, price,
weight, dates,
 *   and manufacturer.
 * - BillItem: Represents an item in the bill, including the grocery item and its quantity.
 * - PendingBill: Represents a pending bill with a list of bill items.
 * - POSSystem: Represents the POS system with functionalities to add items to a bill, apply
discounts,
 *   calculate total price, and print the bill.
 * - ItemCodeNotFound: Custom exception class for handling the case when an item code is not
found in the
 *   database.
 * - POS: Contains the main logic of the POS system, including fetching item details from the
database
 *   using the item code, handling the exception, and providing the getItemDetails() method.
 * - Main: The main class to test and demonstrate the functionality of the POS system.
 *
 * Note: This is a simplified code for demonstration purposes and does not include a complete
implementation
 * of the database connection. We can customize and extend the code as per our requirements.
 */

class ItemCodeNotFound extends Exception {
   // Custom exception for Item Code not found
   public ItemCodeNotFound(String message) {
      super(message);
   }
}

// Class representing a grocery item
class GroceryItem implements Serializable {
   private String item_name;
   private String item_code;
   private String manufacturer_name;
   private String manufacturing_date;
   private double price;
   private double weight;
   private String expiry_date;
   private double discount;

   // Constructor
   public GroceryItem(String itemCode, String itemName, double price, double weight,
                  String manufacturing_date, String expiry_date, String manufacturer_name) {
      this.item_code = itemCode;
      this.item_name = itemName;
      this.price = price;
      this.weight = weight;
      this.manufacturing_date = manufacturing_date;
      this.expiry_date = expiry_date;
      this.manufacturer_name = manufacturer_name;
      this.discount = 0.0;
   }

   public String getItem_code() { // Get the item code
      return item_code;
   }
```

```
    public String getManufacturing_date() { // Get the manufacturing date
        return manufacturing_date;
    }

    public String getManufacturer_name() { // Get the manufacturer name
        return manufacturer_name;
    }

    public String getItem_name() { // Get the item name
        return item_name;
    }

    public double getPrice() { // Get the price
        return price;
    }

    public double getDiscount() { // Get the discount
        return discount;
    }

    public double getWeight() { // Get the weight
        return weight;
    }

    public String getExpiry_date() { // Get the expiry date
        return expiry_date;
    }

    public void setDiscount(double discount) { // Set the discount
        if (discount > 0.0 && discount < 2.976) {
            this.discount = discount;
        }
    }

    public double getNetPrice() { // Get the net price
        return price - (price * discount);
    }
}

// Class representing a bill item
```

```
class BillItem implements Serializable {

   /*
 * The `BillItem` class represents an item in the bill, including the grocery item and its quantity.
 * It implements the `Serializable` interface to enable serialization of `BillItem` objects.
 *
 * The class contains the following attributes:
 * - `groceryItem`: A reference to the corresponding `GroceryItem` object representing the
grocery item
 *   associated with the bill item.
 * - `quantity`: The quantity of the grocery item in the bill.
 *
 * The class provides the necessary getters and setters to access and modify the attributes.
 *
 * The `BillItem` class is used in the POS system to create bill items and calculate the net price
 * of each item based on the quantity and the discounts applied to the grocery item.
 */


   private GroceryItem groceryItem;
   private double quntity;

   // Constructor
   public BillItem(GroceryItem groceryItem, double quantity) {
      this.groceryItem = groceryItem;
      this.quntity = quantity;
   }

   public GroceryItem getGroceryItem() { // Get the grocery item
      return groceryItem;
   }

   public double getQuntity() { // Get the quantity
      return quntity;
   }

   public void setQuntity(double quantity) { // Set the quantity
      if (quantity > 0.0) {
         this.quntity = quantity;
      }
   }
```

```
    public double calculatePrice() {  // Calculate the total price of the bill item
        return quntity * groceryItem.getNetPrice();
    }

    public void print() { // Print the bill item
        System.out.println(groceryItem.getItem_name() + "\t" + groceryItem.getPrice() + "\t" +
                quntity + "\t\t" + groceryItem.getDiscount() + "\t\t" + groceryItem.getNetPrice());
    }

}

// Class representing a pending bill
class PendingBill implements Serializable {
    private List<BillItem> billItems;

    // Constructor
    public PendingBill() {
        this.billItems = new ArrayList<>();
    }

    public List<BillItem> getBillItems() { // Get the bill items
        return billItems;
    }

    // Add a new item to the pending bill
    public void addItem(BillItem billItem) { // Add a new item to the pending bill
        billItems.add(billItem);
    }

    public double calculateTotalPrice() { // Calculate the total price of the pending bill
        double totalPrice = 0.0;
        for (BillItem billItem : billItems) {
            totalPrice += billItem.getQuntity() * billItem.getGroceryItem().getNetPrice();
        }
        return totalPrice;
    }
}

// Class representing a POS system
class POSSystem implements Serializable {
    private String cashier_name;
    private String branch;
    private String customer_name;
    private List<BillItem> billItems;
    private double total_discount;
    private LocalDateTime bill_date_time;

    // Constructor
    public POSSystem(String cashierName, String branch, String customerName) {
        this.cashier_name = cashierName;
        this.branch = branch;
        this.customer_name = customerName;
        this.billItems = new ArrayList<>();
        this.total_discount = 0.0;
        this.bill_date_time = LocalDateTime.now();
    }

    class POS {
        /*The getItemDetails() method handles the ItemCodeNotFound exception by catching it
and displaying an error message. It then prompts the cashier to re-enter the item code. This
process continues until a valid item code is entered, at which point the fetched item details are
returned.
The fetchItemDetails() method is a placeholder implementation and can be replaced with your
actual logic to fetch item details from the database based on the item code.
The main method in the Main class demonstrates the usage of the getItemDetails() method in
the POS class. It prompts the user to enter an item code, retrieves the item details using the
getItemDetails() method, and prints the fetched item details. */
        public GroceryItem getItemDetails() {
            InputStreamReader r = new InputStreamReader(System.in);
            BufferedReader br = new BufferedReader(r);
            String itemCode = null;
            GroceryItem item = null;

            while (item == null) {
                try {
                    itemCode = br.readLine();
                    item = fetchItemDetails(itemCode);
                } catch (IOException e) {
                    System.out.println("Error reading input. Try again.");
                } catch (ItemCodeNotFound e) {
                    System.out.println(e.getMessage());
                    System.out.println("Please re-enter the item code:");
                }
            }

            try {
                br.close();
                r.close();
            } catch (IOException e) {
                e.printStackTrace();
            }

            return item;
        }

        private GroceryItem fetchItemDetails(String itemCode) throws ItemCodeNotFound {
            /* Utilize the item code provided to retrieve the item's details from the database.
             Throw the ItemCodeNotFound exception if the item code cannot be found.
            Return the details of the retrieved object if not.*/

            // Example implementation (replace with your actual database fetching logic)
            if (itemCode.equals("001")) {
                return new GroceryItem("001", "Apple", 2.976, 0.5746,
                        "2023-04-20", "2023-06-01", "Nimal Farming House");
            } else if (itemCode.equals("002")) {
                return new GroceryItem("002", "Banana", 1.074, 0.4746,
                        "2023-04-20", "2023-06-03", "Gunapala Farming House");
            } else {
                throw new ItemCodeNotFound("Item code not found. Please enter a valid item
code.");
            }
        }
    }

    private GroceryItem fetchItemDetails(String itemCode) throws ItemCodeNotFound {
        /* Utilize the item code provided to retrieve the item's details from the database.
         Throw the ItemCodeNotFound exception if the item code cannot be found.
         Return the details of the retrieved object if not. */

        // Example implementation (replace with your actual database fetching logic)
```

```
        if (itemCode.equals("001")) {
            return new GroceryItem("001", "Apple", 2.976, 0.5746,
                    "2023-04-20", "2023-06-01", "Nimal Farming House");
        } else if (itemCode.equals("002")) {
            return new GroceryItem("002", "Banana", 1.074, 0.4746,
                    "2023-04-20", "2023-06-03", "Gunapala Farming House");
        } else {
            throw new ItemCodeNotFound("Item code not found. Enter a valid item code.");
        }
    }

    public void addItem(BillItem billItem) { // Add a new item to the bill
        billItems.add(billItem);
    }

    public void applyDiscount(int itemIndex, double discount) { // Apply discount to an item
        BillItem billItem = billItems.get(itemIndex);
        billItem.getGroceryItem().setDiscount(discount);
        total_discount += discount;
    }

    public double calculateTotalPrice() { // Calculate the total price of the bill
        double totalPrice = 0.0;
        for (BillItem billItem : billItems) {
            totalPrice += billItem.getQuntity() * billItem.getGroceryItem().getNetPrice();
        }
        return totalPrice;
    }

    // Print the bill
    public void printBill() {
        System.out.println("Cashier: " + cashier_name + "\nBranch: " + branch + "\nCustomer: " +
customer_name + "\nDate: " + bill_date_time.toLocalDate() + "\nTime: " +
bill_date_time.toLocalTime());

System.out.println("_________________________________________________________________
_________________________________________________________________");
        System.out.println("Item\t\tPrice\tQuantity\tDiscount\tNet Price");

System.out.println("_________________________________________________________________
_________________________________________________________________");
```

```
        for (BillItem billItem : billItems) {
            GroceryItem groceryItem = billItem.getGroceryItem();
            System.out.println(groceryItem.getItem_name() + "\t" + groceryItem.getPrice() + "\t" +
                    billItem.getQuntity() + "\t\t" + groceryItem.getDiscount() + "\t\t" +
groceryItem.getNetPrice());
        }

System.out.println("________________________________________________________________
_____________________________________________________________");
        System.out.println("Total Discount: " + total_discount);
        System.out.println("Total Price: " + calculateTotalPrice());
    }
}

public class Main {
    /*
 * The `Main` class serves as the entry point of the program and is used to test and demonstrate
 * the functionality of the POS system for the Super-Saving supermarket chain.
 *
 * In the `main` method, the following actions are performed:
 * 1. Creation of grocery items: Two grocery items, namely "Apple" and "Banana," are created
with
 *    their respective item codes, prices, weights, manufacturing dates, expiry dates, and
manufacturer names.
 * 2. Creation of a pending bill: A pending bill is created by adding two bill items to it. The bill
items
 *    consist of the grocery items and their quantities.
 * 3. Serialization of the pending bill: The pending bill is serialized and saved to a file named
"pending_bill.ser."
 * 4. Deserialization and retrieval of the pending bill: The serialized pending bill is deserialized
and
 *    retrieved from the file.
 * 5. Creation of a new POS system: A new POS system is created for a new bill. The cashier
name, branch,
 *    and customer name are provided.
 * 6. Addition of items to the bill: Two bill items, consisting of grocery items and quantities, are
added to
 *    the new bill.
 * 7. Application of a discount: A discount is applied to the second item in the bill.
 * 8. Printing of the new bill: The new bill, along with the cashier name, branch, customer name,
date, time,
 *    item list (unit price, quantity, discount, net price), total discount, and total price, is printed.
 * 9. Printing of the retrieved pending bill: If a pending bill was successfully retrieved, the
grocery items
 *    and quantities, along with the total price, are printed.
 *
 * Note: This is a simplified code for demonstration purposes and does not include a complete
implementation
 * of the database connection. You can customize and extend the code as per your
requirements.
 */

  public static void main(String[] args) {
  // Create some grocery items
  GroceryItem item1 = new GroceryItem("001", "Apple", 2.976, 0.5746,
  "2023-04-20", "2023-06-01", "Nimal Farming House");
  GroceryItem item2 = new GroceryItem("002", "Banana", 1.074, 0.4746,
  "2023-04-20", "2023-06-03", "Gunapala Farming House");
  // Create a pending bill
  PendingBill pendingBill = new PendingBill();
  pendingBill.addItem(new BillItem(item1, 3));
  pendingBill.addItem(new BillItem(item2, 5));

  // Serialize and save the pending bill
  try {
    FileOutputStream fileOut = new FileOutputStream("pending_bill.ser");
    ObjectOutputStream out = new ObjectOutputStream(fileOut);
    out.writeObject(pendingBill);
    out.close();
    fileOut.close();
    System.out.println("Pending bill saved as pending_bill.ser");
  } catch (IOException e) {
    e.printStackTrace();
  }

  // Deserialize and retrieve the pending bill
  PendingBill retrievedPendingBill = null;
  try {
    FileInputStream fileIn = new FileInputStream("pending_bill.ser");
    ObjectInputStream in = new ObjectInputStream(fileIn);
    retrievedPendingBill = (PendingBill) in.readObject();
    in.close();
```

```
        fileIn.close();
    } catch (IOException | ClassNotFoundException e) {
        e.printStackTrace();
    }

    // Create a POS system for a new bill
    POSSystem posSystem = new POSSystem("John Doe", "Super-Saving Branch 1", "Alice");
    posSystem.addItem(new BillItem(item1, 2));
    posSystem.addItem(new BillItem(item2, 4));
    posSystem.applyDiscount(1, 0.4746);  // Apply discount to the second item

    // Print the new bill
    posSystem.printBill();

    // Print the retrieved pending bill
    if (retrievedPendingBill != null) {
        System.out.println("Retrieved Pending Bill:");
        for (BillItem billItem : retrievedPendingBill.getBillItems()) {
            System.out.println(billItem.getGroceryItem().getItem_name() + "\t" +
                    billItem.getQuntity());
        }
        System.out.println("Total Price: " + retrievedPendingBill.calculateTotalPrice());
    }
}
}
```