Noted that Access Modifier Notations can be listed below
        + (public)
        # (protected)
        - (private)
        <u>Underline</u> (static)
        Italic (abstract)

Set-Up Instruction

- Don't forget to set your Eclipse workspace and working set.

- <u>You must submit the JAR file, exported (with source code), from your Eclipse project.</u>

- <u>You must check your JAR file to make sure all the source files (.java files) are present. It can be opened with file compression programs such as 7-zip or Winrar.</u>

- <u>Failure to export properly will result in your work not getting marked.</u>

Question 2

Objective

    1)   Be able to implement Objects and Classes.

Instruction

    1)   Create Java Project in Eclipse.

    2)   Copy all folders in "**Q2_toStudent**" to your project directory src folder.

    3)   You are to implement the following classes (detail for each class is given in section 3 and 4)

          a) **MisaShop**   (package application)

          b) **Item**   (package logic)

          c) **OrderItem**   (package logic)

          d) **Order**   (package logic)

    4) **JUnit for testing is in package test.grader**

Problem Statement: Misa Shop
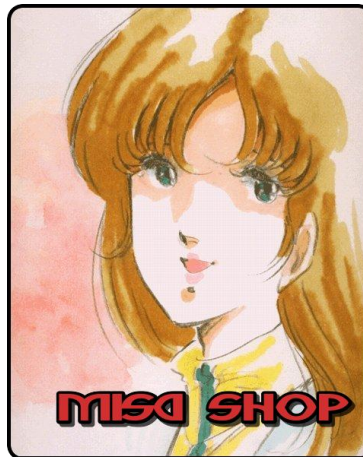
     Misa Shop is a JAVA application for the shop manager to add items and create orders. The order contains item orders that tell which item the order has and how much the amount is. However, the current system's implementation is still incomplete. You are tasked to implement the remaining Item, Order and OrderItem classes.

Here's is how the Shop should work, part of this is already provided.

When the application is open, there will be a welcome message with all the available commands for the user.

Figure 1. Welcome screen when the application starts

1) The first option is adding a new item to the system. Please note that the new item name has to be different from existing items and not blank. Otherwise, the system will raise an error. If price per piece entered is less than 1, the system will set the item price to 1.

Figure 2. Successfully adding a new item

2) The user can choose to list all current items in the system. Each item information displays the item name and its price per piece. When the program starts, there will already have 4 initial items in the system.

Figure 3. Initial Item Listing

3) Third option is creating a new order. The user will then be asked to select item by entering its index and then enter the amount to add to the order. The user can finish

creating order by pressing Enter key.

If the user select to add an item that is already in the order, the amount of that item in the order will increase.

Figure 4. Creating a new order.

4) The user can list out all order in the system.

Figure 5. Listing all orders.

Implementation Detail

The class package is summarized below.

Item

□ name: String
□ pricePerPiece: int

● Item(name: String, pricePerPiece: int)
● getName(): String
● setName(name: String): void
● getPricePerPiece(): int
● setPricePerPiece(pricePerPiece: int): void

OrderItem

□ item: Item
□ itemAmount: int

● OrderItem(item: Item, itemAmount: int)
● increaseItemAmount(amount: int): void
● calculateTotalPrice(): int
● getItem(): Item
● setItem(item: Item): void
● getItemAmount(): int
● setItemAmount(itemAmount: int): void

Order

□ orderItemList: ArrayList<OrderItem>
□ totalOrderCount: int
□ orderNumber: int

● Order()
● addItem(item: Item, amount: int): OrderItem
● calculateOrderTotalPrice(): int
● getTotalOrderCount(): int
● resetTotalOrderCount(): void
● getOrderNumber(): int
● getOrderItemList(): ArrayList<OrderItem>

MisaShop

□ itemList: ArrayList<Item>
□ orderList: ArrayList<Order>
□ isEnd: boolean
□ kb: Scanner

● main(args: String[]): void
● addItemToOrder(order: Order, index: int, amount: int): void
● addNewItem(name: String, price: int): boolean
● isItemExisted(name: String): boolean
■ handleAddItem(): void
■ handleCreateNewOrder(): void
■ showAllItems(): void
■ showAllOrder(): void
● getOrderList(): ArrayList<Order>
● addBlankOrder(): void
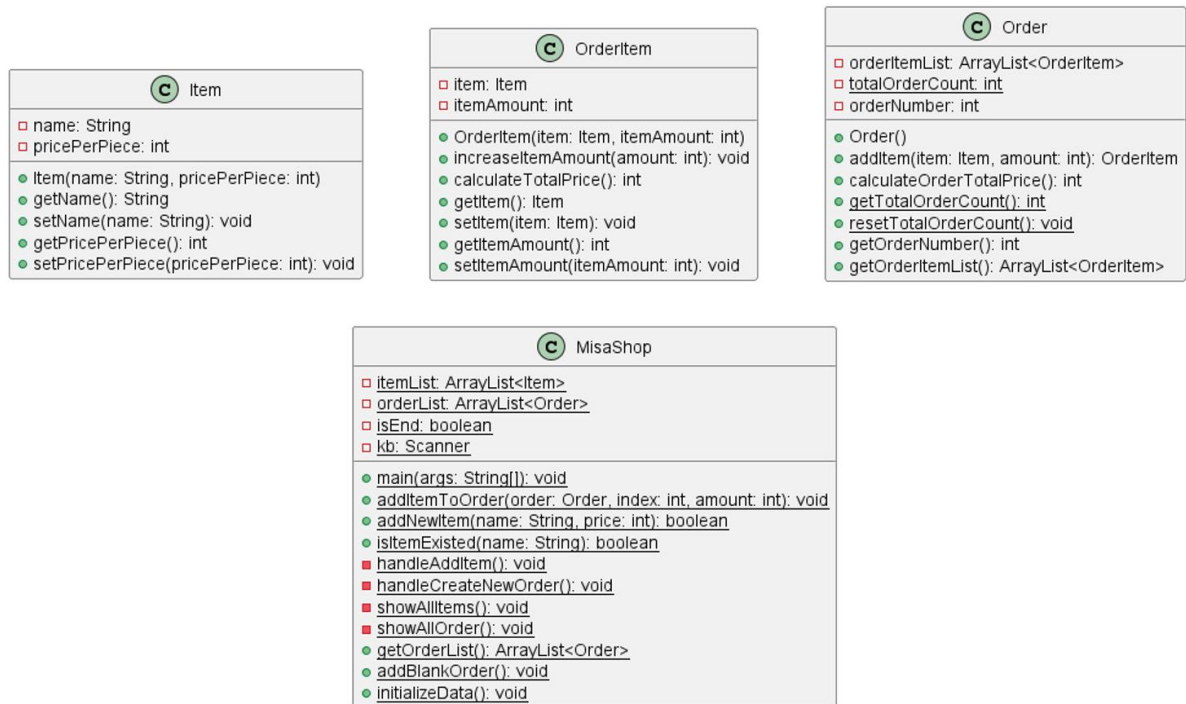● initializeData(): void

Figure 6. Class Diagram

You must write java classes using UML diagram specified above.

* In the following class description, only details of IMPORTANT fields and methods are given. *

4.1 Package logic

4.1.1 **Class Item**: This class represents an item.

<mark>You must implement this class from scratch.</mark>

Field

Name	Description
- String name	The name of the item.
- int pricePerPiece	Price per piece of the item.

Constructor

Name	Description
+ Item(String name, int pricePerPiece)	Create a new Item object with specified name and price per piece. Set related fields with the given parameters; Price per piece must be equals or more than 1.

Method

Name	Description
+ void setPricePerPiece(int pricePerPiece)	Set price per piece of the item. If given price per piece is less than 1, set price per piece to 1. (This is one of the setters).
+ getter/setter for other variables	

4.1.2 **Class OrderItem**: This class is used for storing an ordered Item and amount of the ordered item in the Order made through the application menu.

<mark>You must implement this class from scratch.</mark>

Field

Name	Description
- Item item	Ordered item of this OrderItem.
- int itemAmount	Amount of ordered item

Constructor

Name	Description
+ OrderItem(Item item, int itemAmount)	Create a new OrderItem object with specified item and itemAmount. Set related fields with the given parameters.

Method

Name	Description
+ void increaseItemAmount(int amount)	Increase the itemAmount by given amount. If given amount is negative values, the itemAmount should not be changed.
+ int calculateTotalPrice()	Calculate the total price of this OrderItem and return it.
+ void setItemAmount(int itemAmount)	Set itemAmount of the OrderItem by given itemAmount. If given itemAmount is negative values, set itemAmount to 0. This is one of the setters.
+ getter for each variable	

4.1.3 **Class Order**: This class represents a single order that is made from the application menu. An order can contain more than one OrderItem (an item with the amount). It is partially provided. Please fill necessary code.

/* PARTIALLY PROVIDED */

Field

Name	Description
- ArrayList<OrderItem> orderItemList	ArrayList of OrderItem.
- int totalOrderCount	Total number of orders created in the application so far. This will be used to assign order number to a new Order.
- int orderNumber	Number of this order.

Constructor

Name	Description
+ Order()	/* FILL CODE */ Create a new order object. Assign the orderNumber to totalOrderCount and then increase the totalOrderCount by 1. Do not forget to initialize orderItemList.

Method

Name	Description
+ OrderItem addItem(Item item, int amount)	/* FILL CODE */ Adding item to this order by creating new OrderItem with given amount. If an OrderItem with the same item name already existed, increase the ItemAmount in the OrderItem by the given amount instead. Return the orderItem that was changed or added.
+ int calculateOrderTotalPrice()	/* FILL CODE */ Calculate total price of the order by summing total price of each orderItem in orderItemList

+ <u>void resetTotalOrderCount()</u>	Reset totalOrderCount to 0.
+ <u>int getTotalOrderCount()</u>	Return totalOrderCount
+ other remaining getters	

4.2 Package application

4.2.1 **Class MisaShop**: This class represents the shop. It also contains main method. This class is partially provided. You only need to fill the code where necessary.

/* PARTIALLY PROVIDED */

Field

Name	Description
- ArrayList<Item> itemList	ArrayList of item containing all items
- ArrayList<Order> orderList	ArrayList of order containing all orders

Method

Name	Description
+ void main()	Handle MisaShop process.
+ addItemToOrder(Order order, int itemIndex, int amount)	/* FILL CODE */ Add item at given itemIndex from itemList to the order with given amount.

Score (22):

ItemTest

- testConstructor()   1 mark
- testConstructorIllegalPrice()   1 mark
- testGetPricePerPiece()   1 mark
- testSetPricePerPiece()   1 mark
- testSetPricePerPieceIllegal()   1 mark

- testGetName()   1 mark

OrderItemTest

- testConstructor()   1 mark
- testConstructorNegativeAmount()   1 mark
- testGetItem()   1 mark
- testGetItemAmount()   1 mark
- testSetItemAmount()   1 mark
- testSetItemAmountNegative()   1 mark
- testCalculateTotalPrice()   1 mark
- testIncreaseItemAmount()   1 mark
- testIncreaseItemAmountNegative()   1 mark

OrderTest

- testConstructor()   1 mark
- testAddItemNormal()   1 mark
- testAddItemExisted()   1 mark
- testAddItemNegativeAmount()   1 mark
- testAddItemExistedNegativeAmount()   1 mark
- testCalculateOrderTotalPrice()   1 mark

MisaShopTest

- testAddItemToOrder()   1 mark