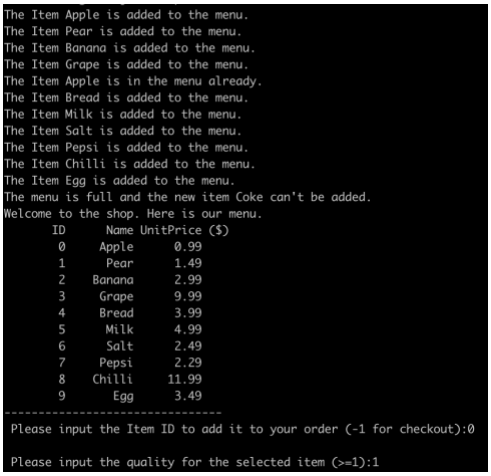
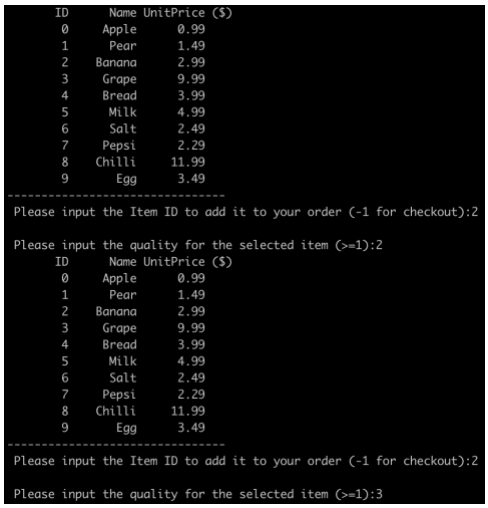
An example with one test case is shown below. Please make sure the output of your program is the same as the following examples.

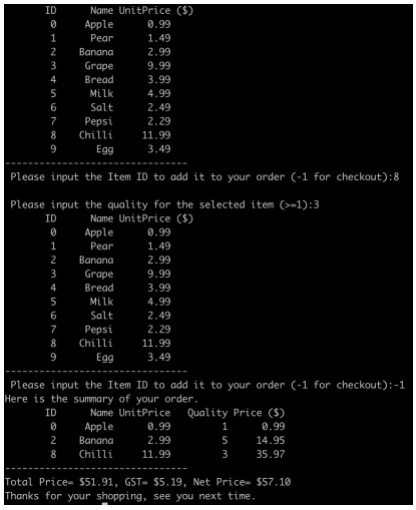
Examples of the program outputs:



Page 1

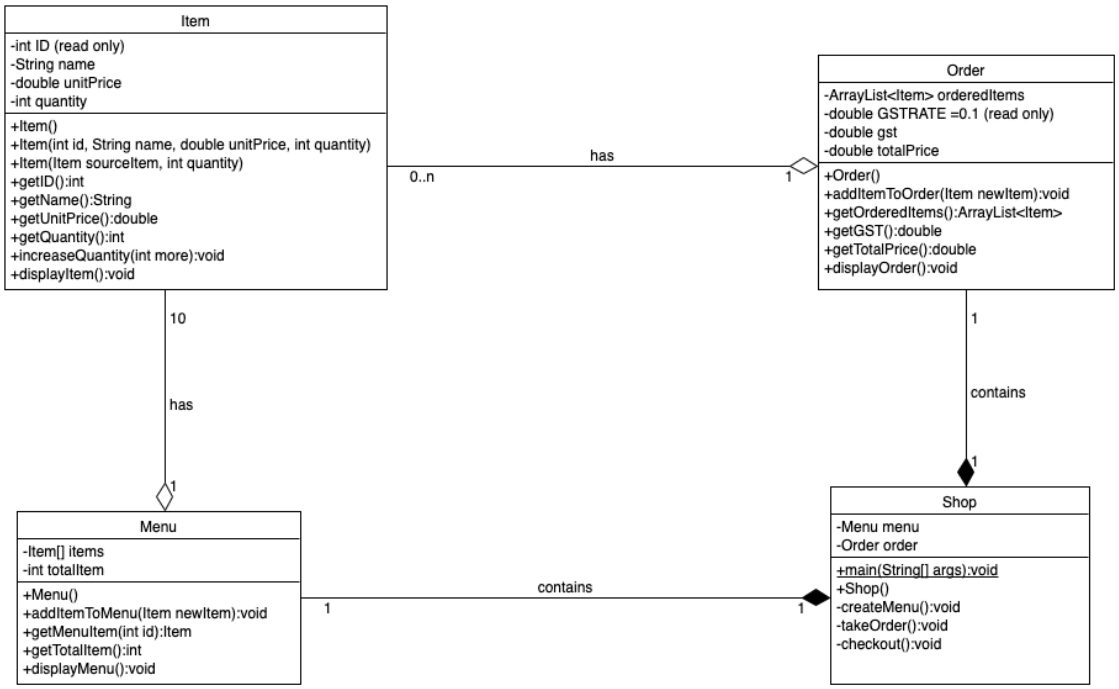


Page 2



Page 3

The program structure shall be based on the UML class diagram shown below.



- Class Item is used to model a single item. Each Item object has four attributes, i.e., item ID, item name, the unit price of the item, and the item quantity.
  - The *+Item(int id, String name, double unitPrice, int quantity)* is the constructor with four formal parameters.
  - The *+Item()* is the default constructor.
  - The *+Item(Item sourceItem, int quantity)* is the clone constructor of Class Item. Once a customer selects an item from the menu and specify the quantity, the clone constructor will be called to create a clone Item object based on the *sourceItem* (i.e., item displays in the menu) and the specified quantity. Then

the clone Item object will be added to the customer's order. So any further modifications on the items in the customer's order will not affect the items in the menu.

- The *+increaseQuantity(int more):void* method will increase the quantity of a selected item in the customer's order. When the customer selects the same item from the menu, the *+increaseQuantity(int more):void* method will be called to update the quantity of the item, i.e., each item will have only one instance in the order.
- The *+displayItem():void* method will display a single item using the format in the example, i.e., nine fixed space for each column, and two decimal point precision for all numbers.

- Class Order is used to model the customer's order. Each Order object has four attributes, i.e., the ordered items (ArrayList), the GSTRATE (a constant), the GST tax of the order, and the total price of the order.
  - The +Order() is the default constructor which will initialise the *orderedItems* to an empty ArrayList, *GSTRATE* to 0.1, both *gst* and *totalPrice* to zero.
  - The *+addItemToOrder(Item newItem):void* method will be called once the customer adds an item to the order. If the item has no instance in the order, then the item will be added to the order straightway. However, if the item has an instance in the order, then the method will only update the quantity of the item. Once the item is added to the order, the method will also update the total price and the GST tax of the order accordingly.
  - The *displayOrder():void* method will display the entire order using the format in the example, i.e., nine fixed space for each column, two decimal point precision for all numbers.

- Class Menu is used to model the shop menu, which contains two attributes, i.e., the Item array with maximal ten elements and the actual item number the menu contains.
  - The *Menu()* is the default constructor, which will create a 1D array to contains ten Item objects maximally and initialise the *totalItem* to zero.
  - The *+addItemToMenu(Item newItem):void* method will be called to add an item to the menu (i.e., the Item array). If the menu has less ten items and if the item is not listed in the menu, then the item will be added to the menu. If the item already is contained in the menu, then the method will not add the item again to the menu but just indicate the item was in the menu already. If the menu has ten items, no more item can be added to the menu.
  - The *+getMenuItem(int id):Item* method will return an Item object based on the item's id. This method can be called when to check if an item is contained in the menu or not.
  - The *+getTotalItem():int* method will return the actual item number contained in the menu.
  - The *+displayMenu():void* method will display the entire menu using the format in the example, i.e., nine fixed space for each column, two decimal point precision for all numbers.

- Class Shop is the primary class and used to display the menu and take the customer's order. It has two attributes, i.e., the shop menu and the customer's order.
  - The *+main(String[] args):void* is the method to start the program.
  - The *+Shop()* is the constructor, which initialise an empty menu and an empty order.
  - The *-createMenu():void* method is a private method and will be called to create the shop menu with the hardcode. The implementation of the *-createMenu():void* method is already provided. Please download it from Moodle and DON'T modify the method implementation.
  - The *-takeOrder():void* method is a private method and will be called to add the customer's selection to the order. If the item is not contained by the order, then the item will be added to the order directly. If the item is already contained by the order, then the quantity of the item will be updated accordingly. The process will be repeated until the customer inputs -1 in the item selection (see the example and the video introduction for details).
  - The *-checkout():void* method is a private method and will be called to checkout. The method will calculate the GST tax, the total price and the net price (GST tax + total Price) of the order and display the order summary using the format in the example, i.e., nine fixed space for each column, two decimal point precision for all numbers.

TASKS

Task 1 (8 marks): Implement your java program based on the UML class diagram. The program shall

- be consistent with the UML class diagram;
- follow the conventions for naming all classes, variables, and methods;
- provide sufficient comments;
- use proper blank spaces, indentation and braces to make your code easy to read and understand;
- follow the specified steps in main() method to complete the test i.e.,
  1. create an object of the Shop class;
  2. call the *creatMenu()* method with the Shop object to create the shop menu;
  3. call the *takeOrder()* method with the Shop object to take the order from the customer. This process should be repeated until the customer inputs -1 to complete the order;
  4. once the customer completes the order, call the *checkout()* method to calculate the GST tax, the total price, the net price (GST tax + total price), and display the order summary (see the example for the summary format and details).
- the program outputs must be exactly same as the example.

Task 2 (2 marks): Compilation and test.

- You should compile and test your program by using two different cases.
- The first testing case must use the exact same inputs in the examples and the results must be exact same as the examples.
- The second testing case must choose a different combination of items selection to make sure,
  - 1. At least four items are selected;
  - 2. One item is selected for twice with different qualities;
  - 3. The total price, the GST tax and the net price can be calculated based on the customer's selection.
- Please carefully compile and test your program and make sure your program can pass the compilation by using "javac" command.

SUBMISSION:

- Submit all your files in a zip file to Moodle "Assignment 3". Email submission is not accepted.
- Please submit an **individual PDF document (Shop.pdf)** and **all java files.** The PDF document shall contain the snapshots to clearly shows the compilation and the execution of your program with the two test cases.

Mark deductions: compilation errors, incorrect result, program is not up to spec, poor comments, poor indentation, meaningless identifiers, required numeric constants are not defined, the program uses approaches which has not been covered in the lectures. The deductions here are merely a guide. Marks may also be deducted for other mistakes and poor practices. Enquiries about the marks can only be made within a maximum of 1 week after the assignment results are published. After 1 week the marks cannot be changed.