Question - 1
Cart System

Cart System

[Strictly adhere to the object-oriented programming specifications given in the problem statement. A template project is provided as part of this assessment to illustrate the input-output process. The template project will not compile. You need to fill in the missing code.]

Business Requirement:

Your task is to create a basic Cart System where an App system reads from a file of default items and displays those items to the user in order to add them to their cart. A new item can be added at any time to the system (Only at runtime and it does not need to be saved in the file with default items). Items can be removed from the System as well as from the cart.

Work-Flow:

The Cart System starts by asking the user to select whether they would like to:

1. Add an item to the System: In this case, the system asks the user to enter the new item's name, a description for the new item, the new item’s price and the available quantity of the new item, finally, the new item will be added to the app.

2. Add an item to the Cart: In this case, the system displays all of the available items in the app system and asks the user to enter the name of an item to be added to the cart. The item will be added to the cart by searching the name for that item in the collection of items available in the app system. If the item has already been added to the cart, the quantity of that item in the cart is increased by one.

3. Display Cart: All the items in the cart are displayed, the system then calculates the sub-total by calculating the total sum of the products of the price and quantity of each item in the cart (Sum (item's price * item's quantity)). The sub-total is displayed along with the tax, which is (sub-total * 0.05), and the total, which is (sub-total + tax) for all items in the cart.

4. Display System: The system displays all the items that are available in the app.

5. Remove an item from the cart: In this case, the system asks the user for the name of the item to be removed from the cart.

6. Remove an item from the system: In this case, the system asks the user for the name of the item to be removed from the app. If an item is removed from the app, but it was already added to the cart. It will also be removed from the cart.

7. Quit: This option terminates the program.

Requirement 1:

Model Class:

A class **Item** should only contain the private instance variables specified in ***TABLE 1*** with associated **GETTERS** and **SETTERS**.

The purpose of the item class is to carry data related to an item.

TABLE 1:

Datatype	Name	Description
String	itemName	Name of the item
String	itemDesc	Contains the item's description
Double	itemPrice	Contains a double value which represents the item's price
Integer	quantity	Represents the number of items the user has added to the cart
Integer	availableQuantity	Represents the number of items of a type available in the system

The following constructor must be implemented:

Description	Input Parameters
This constructor initializes quantity to 1	None
Initialises itemName, itemDesc, itemPrice, and availableQuantity to the provided Parameters	itemName, itemDesc, itemPrice, availableQuantity

Requirement 2:

sample.txt and TheSystem:

sample.txt contains data in "double space" separated value format.

An **abstract** class **TheSystem** should contain the private instance variable specified in ***TABLE 2*** with associated **GETTER** and **SETTER**.

The purpose of the **TheSystem** class is to maintain the list of items and the main logic of the system that is similar in the app and cart classes.

TABLE 2:

Datatype	Name	Description
HashMap<String, Item>	itemCollection	Provides the list of items in the system or the cart depending on which class initiates it

The following constructor must be implemented:

Description	Input Parameters
This constructor initializes the itemCollection variable with an empty hashmap. It then checks if the AppSystem is invoking the constructor (getClass().getSimpleName().equals("AppSystem")), if so, it adds the items from the sample.txt file to the itemCollection. Recommended: When reading from the sample.txt file, read each line and do the following line: String[] itemInfo = line.split("\s ");	None

The following methods must be implemented:

Return Type	Description	Input Parameters
Boolean	**checkAvailability()** – This method takes Item object as a parameter, then it checks if the item.getQuantity() is greater than or equal than item.getAvailableQuatity(). If it is, display the following message. "System is unable to add [item name] to the card. System only has [item available quantity] [item name]s." and return false. Otherwise, return true.	Item item
Boolean	**add()** – This method takes Item object as a parameter, checks: 1. if item is null, then returns false 2. If it is already in the collection and is available. If so, the method increases the quantity by one and returns true. 3. If the item is not in the collection, the method adds the item to the collection and also returns true. 4. In all other cases, the method returns false.	Item item
Item	**remove()** – This method takes String itemName as a parameter, checks if the item is in the collection, if it is, then removes it and returns the Item object being removed. If is not in the collection then returns null.	String itemName

Requirement 3:

AppSystem:

The purpose of the **AppSystem** class is to implement the logic related only to the App system.

The following methods must be implemented:

Return Type	Description	Input Parameters
void	**display()** – This method takes no parameter and displays every item in the App system. Your output should be formatted as follows: The header "AppSystem Inventory:" followed by the flowing columns. • Name: itemName, displayed as a left-justified string within a 20-character field. • Description: itemDesc, displayed as a left-justified string within a 20-character field. • Price: itemPrice, displayed as a left-justified floating-point number with two decimal places in a 10-character field. • Available Quantity: availableQuantity, displayed as a decimal number within a 10-character field. Each column must be properly labeled,please see the example below.	None
Boolean	**add()** – This method takes Item object as a parameter, checks: 1. if item is null, then returns false 2. If it is already in the collection, displays a message "[Item name] is already in the App System" and returns false. 3. If the item is not in the collection, the method adds the item to the collection and returns true.	Item item
Item	**reduceAvailableQuantity()** – This method takes String itemName as a parameter, checks if the item is in the collection, if it is, then it decreases the	String item_name

	available quantity of the item in the system by 1 and returns the Item object. If the item is not in the collection then it returns null.	

- **Display System – Initially data**

Name	Description	Price	Available Quantity
pizza	very cheesy	12.30	3
salad	cobb salad	15.50	12
hunger burger	huge patty	9.49	10
fried chicken	so crispy	18.99	5

Requirement 4:

CartSystem:

The purpose of the **CartSystem** class is to implement the logic related only to the cart.

The following methods must be implemented:

Return Type	Description	Input Parameters
void	**display()** – This method takas no parameter and displays every item in the Cart system, along with the sub-total, tax, and total: *sub-total* = the total sum of the products of the price and quantity of each item in the cart (Sum (item's price * item's quantity)). *tax* = sub-total * 0.05 *total* = sub-total + tax Your output should be formatted as follows: The header "Cart:" followed by the flowing columns. • Name: itemName, displayed as a left-justified string within a 20-character field. • Description: itemDesc, displayed as a left-justified string within a 20-character field. • Price: itemPrice, displayed as a left-justified floating-point number with two decimal places in a 10-character field. • Quantity: quantity, displayed as a decimal number within a 10-character field. • Sub Total:sub-total, displayed as a left-justified floating-point number with two decimal places in a 10-character field. Each column must be properly labeled. Finally, you should output the Pre-tax Total, Tax, Total for the following format. • Pre-tax Total: displayed as a left-justified floating-point number with two decimal places in a 20-character field.	None

	• Tax: displayed as a left-justified floating-point number with two decimal places in a 20-character field. • Total: displayed as a left-justified floating-point number with two decimal places in a 20-character field. please see the example below.	

Sample Output:

- **Display Cart**

AppSystem Inventory:

Name	Description	Price	Quantity	Sub Total
pizza	very cheesy	12.30	1	12.30
salad	cobb salad	15.50	1	15.50

Pre-tax Total	27.80
Tax	1.39
Total	29.19

MainEntryPoint class

This class provides the complete logic that makes every component work together. Take the time to review the logic provided and make sure you fully understand what is being implemented.

Follow the naming convention provided by this document and do not change the name or return type of any of the methods provided throughout the program. Do not change the name of any of the classes provided throughout this program.

Additional Instructions: :

- You can download the template project to your local system by using the Download code as Zip option. This option will download a zip file with the most recently-submitted version of the project.
- You can upload a project from your local system by using the Upload code as Zip option. This would allow you to work in this assessment on your local system if you prefer, although we encourage you to work on the environment provided by HackerRank in this test in order to avoid any error due to differences in your local setup. Please keep in mind that this would replace all files in your current working set.

- The ✓ Test against custom input option allows you to provide input that will be read by the system as if it were user input (to be used by the Scanner class for example). This input is read one by line.
- The Execute main() button will compile and execute your project (using any provided custom input if applicable).
- The Run Unit Tests button will compile your project and run it against the test cases used to score this assessment. Please try to adapt, fix, or improve your project so that it passes as many of those cases as possible. Also, use this button to make sure that all your changes are saved before you code your test.
- Finally, if you break your project beyond repair you can download the blank template project by using **This Link**.