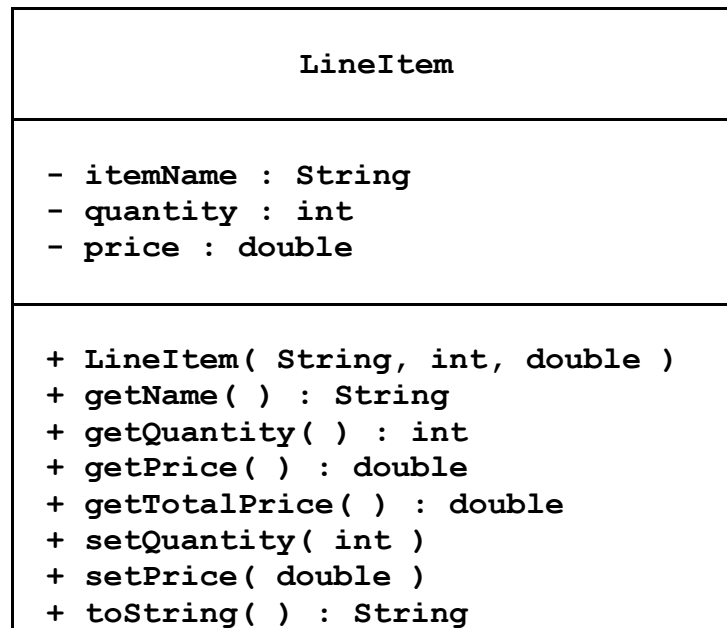


Retail Transaction Programming Project

Project Requirements:

1. Develop a program to emulate a purchase transaction at a retail store. This program will have two classes, a **LineItem** class and a **Transaction** class. The **LineItem** class will represent an individual line item of merchandise that a customer is purchasing. The **Transaction** class will combine several **LineItem** objects and calculate an overall total price for the line item within the transaction. There will also be two test classes, one for the **LineItem** class and one for the **Transaction** class.
2. Design and build a **LineItem** class. This class will have three instance variables. There will be an **itemName** variable that will hold the identification of the line item (such as, "Colgate Toothpaste"); a **quantity** variable that will hold the quantity of the item being purchased; and a **price** variable that will hold the retail price of the item. The **LineItem** class should have a constructor, accessors for the instance variables, a method to compute the total price for the line item, a method to update the quantity, and a method to convert the state of the object to a string. Using Unified Modeling Language (UML), the class diagram looks like this:



- a. The constructor will assign the first parameter to the instance variable **itemName**, the second parameter to the instance variable **quantity**, and the third parameter to the instance variable **price**.
- b. The class will have three accessor methods—**getName()**, **getQuantity()**, and **getPrice()**—that will return the value of each respective instance variable.

- c. The class will have two mutator methods, **setQuantity(int)** and **setPrice(double)**, that will update the quantity and price, respectively, of the item associated with the line of the transaction.
- d. The method **getTotalPrice()** handles the conversion of the quantity and price into a total price for the line item.
- e. The method **toString()** allows access to the state of the object in a printable or readable form. It converts the variables to a single string that is neatly formatted.

Note: Refer to the textbook for a discussion of escape sequences. These are characters that can be inserted into strings and, when printed, will format the display neatly. An escape sequence for the tab character can be inserted to get a tabular form when printing. This tab character is `"\t"`.

The **LineItem** class will have a **toString()** method that concatenates **itemName**, **quantity**, **price**, and **total price**—separated by tab characters—and returns this new string. When printing an object, the **toString()** method will be implicitly called, which in this case, will print a string that will look something like:

Colgate Toothpaste qty 2 @ \$2.99 \$5.98

- 3. Build a **Transaction** class that will store information about the items being purchased in a single transaction. It should include a **customerID** and **customerName**. It should also include an **ArrayList** to hold information about each item that the customer is purchasing as part of the transaction.

Note: You must use an **ArrayList**, not an array.

- 4. Build a **TransactionTest** class to test the application. The test class should **not** require any interaction with the user. It should verify the correct operation of the constructor and all methods in the **Transaction** class.

Specific Requirements for the Transaction Class

- 1. The **Transaction** class should have a constructor with two parameters. The first is an integer containing the customer's ID and the second is a String containing the customer's name.
- 2. There should be a method to allow the addition of a line item to the transcript. The three parameters for the **addLineItem** method will be (1) the item name, (2) the quantity, and (3) the single item price.
- 3. There should be a method to allow the updating of a line item already in the transaction. Notice that updating an item means changing the quantity or price (or both). The parameters for the **updateItem** method are also (1) the item name, (2) the quantity, and (3) the single item price. Notice that the updating of a

specific line item requires a search through the **ArrayList** to find the desired item. Anytime a search is done, the possibility exists that the search will be unsuccessful. It is often difficult to decide what action should be taken when such an "exception" occurs. Since exception handling is not covered until later in this textbook, make some arbitrary decisions for this project. If the item to be updated is not found, take the simplest action possible and **do nothing**. Do not print an error message to the screen. Simply leave the transaction unchanged.

4. The transaction class needs a method called **getTotalPrice** to return the total price of the transaction.
5. There should also be a method to return information about a specific line item. It should return a single String object in the same format described for the **LineItem** class:

```
Colgate Toothpaste qty 2 @ $2.99          $5.98
```

Again, the possibility exists that the search for a specific line item will fail. In this instance, you should return a string containing a message similar to this:

```
Colgate Toothpaste not found.
```

6. The final method needed is a **toString** method. It should return the transaction information in a single String object. It should use the following format:

```
Customer ID : 12345
Customer Name : John Doe

Colgate Toothpaste    qty 2 @ $2.99          $5.98
Bounty Paper Towels   qty 1 @ $1.49          $1.49
Kleenex Tissue        qty 1 @ $2.49          $2.49

Transaction Total                                $9.96
```

Notice that a newline character "**\n**" can be inserted into the middle of a string.

Ex.

```
int age = 30;
String temp = "John Doe \n is " + age + "\n" + " years
old";
```

The output would be:

```
John Doe
is 30
years old
```

Notice also that "\n" is a single character and could actually go inside single or double quotes, depending on the circumstances.

Here is a UML diagram for the **Transaction** class as described above. Notice that private instance variables and methods may be added, as needed. For all public methods use **exactly** the name given below.

