zhanpeng yin

Assignment1 Q2

Table of Contents

[Requirements / Specification: 1](#_Toc159777990)

[User Guide 1](#_Toc159777991)

[Structure/Design 10](#_Toc159777992)

[Limitations 11](#_Toc159777993)

[Testing 12](#_Toc159777994)

[Listing : 25](#_Toc159777995)

Author : Yin Zhanpeng

Date : 22/02/24

File Name: Assignment1 Q2

Purpose : The purpose of this program is to manage the Magazine and Supplement subscription of the customers.

# Requirements / Specification:

Assumptions:

1. The customers have a list of Supplements and Magazines
2. The Supplements can be individual or inside the magazines
3. The magazines also contains its own List of Supplements
4. The program can allow the user to add the supplements to both the magazines and the customer
5. The paying customer have a list of associate customers
6. Both the paying customer and the associate customer have their own subscription
7. This program is to only simulate the weekly and monthly notifications
8. We are able to remove the associated customer of a paying customers
9. If we remove the paying customers, then its associated customer will be removed as well
10. The customer contains the magazine and not the magazine contains the customer
11. The paying customer should have at least 1 paying customer.

Specifications:

1. Able to create a supplements
2. Able to add a supplements to either the customer or inside the customer’s magazine
3. Able to remove a supplements from either the customer or inside the customer’s magazine
4. Able to create a magazine
5. Able to add the magazine to the customer
6. Able to delete the magazine from customer
7. Able to delete the magazine’s supplements
8. Able to create the paying customer and associate customer
9. Able to add associate customer to paying customer
10. Able to add paying customer
11. Able to delete the customer
12. Able to have a weekly notifications
13. Able to display 4 weeks of weekly notifications
14. Able to display a monthly notifications

# User Guide

The Javadoc index.html is located in \Assignment\target\site\apidocs

How to compile and run the program.

Step 1 . open up NetBeans and import the program .

A screenshot of a computer

Description automatically generated

Step 2 compile and run the program.

A screenshot of a computer

Description automatically generated

Make sure that it’s the “LogicAndDatabase” class us being run, if not run that class manually

How to use the program:

Menu:

A screenshot of a computer

Description automatically generated

Option 2 create a new Paying customer and its associate customer. In this option we can create a new entry for the paying customer as and 1 entry for its associate customer.

A screenshot of a computer

Description automatically generated

Option 3: In this option we can choose to remove either a paying customer or its associate customer

A screenshot of a computer

Description automatically generated

Option 4 : in this option we can choose to create a associated customer for the paying customer

A white screen with black text

Description automatically generated

Option 5: in this option we can choose to remove an associated customer from a paying customer

Option 6: in this option we can add a magazine subscription to a customer or we can also create a new magazine subscription to add to the customer's subscription

A close-up of a list of names

Description automatically generated

A screenshot of a computer

Description automatically generated

Option 7 : In this option we can remove a magazine subscription from a customer.

A list of names on a white background

Description automatically generated

A screenshot of a computer program

Description automatically generated

Option 8 : in this option we can add a supplement to a customer or a customer’s magazine

A screenshot of a computer program

Description automatically generated

Option 9: in this option we can remove a supplement from a customer's magazine or just the customer.

A screenshot of a computer program

Description automatically generated

Option 10: in this option we can display the weekly notification for 1 customer

A screenshot of a computer

Description automatically generated

Option 11: in this option we can display the monthly notification for 1 paying customer

A screenshot of a computer

Description automatically generated

Option 12: in this option we can display 4 weeks of weekly notifications for all customer

Option 13: in this option we can display the monthly notifications for all the paying customers

# Structure/Design

UML Diagram:

The UML Diagram is also attached in the zip

A computer screen shot of a computer program

Description automatically generated with medium confidence

This program was designed based on a Object-oriented approach, with classes like customer, paying customer, magazines, supplements, menu, subscriptions. This is to improve code readability and code reuse.

With the modular design , this program is divided into multiple methods and classes, each responsible for a specific functionality or feature. This design is easier maintenance ,facilitates, debugging, and scalability of the application.

It also try to follow the MVC model for a more structured approach at solving the question.

I mainly use list and arrays to store the data, This approach allows efficient retrieval, manipulation, and storage of data.

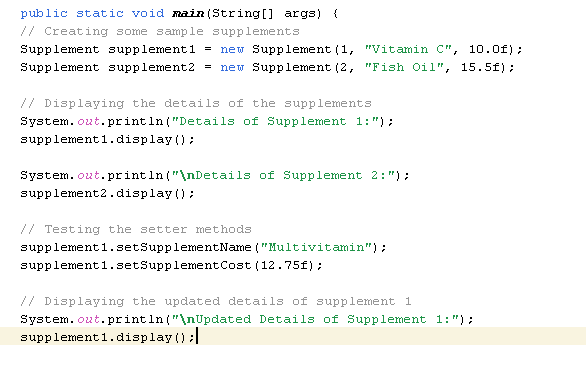
# Limitations

1. This program does not have any input checks, and any error handling for input mismatch, e.g. entering a string for an int input
2. This program can break easily and malfunction sometimes, seems to be some memory related error.
3. This program can only handle 1
4. This program can more modular with more abstraction
5. This program in terms of design can be more readable, as the methods is everywhere

# Testing

Testing Supplement class :

Method:



1. Creation of Supplement Objects: The code creates two Supplement objects with different attributes (ID, name, cost) to verify that instances of the class can be properly instantiated.
2. Displaying Supplement Details: The display method is called on each Supplement object to verify that the method correctly prints out the details of a supplement, including its ID, name, and cost.
3. Updating Supplement Details: The setter methods (setSupplementName and setSupplementCost) are invoked to update the name and cost of one of the supplements. This tests whether the setter methods correctly modify the attributes of a Supplement object.
4. Displaying Updated Supplement Details: After updating the details of one of the supplements, the display method is called again to ensure that the updated details are correctly reflected.

Output:

A screenshot of a computer program

Description automatically generated

Conclusion:

1. Creation of Supplement Objects: Two Supplement objects were created successfully, each with unique attributes such as ID, name, and cost.
2. Displaying Supplement Details: The display method correctly printed out the details of each supplement, including their ID, name, and cost.
3. Updating Supplement Details: The setter methods setSupplementName and setSupplementCost were able to update the name and cost of one of the supplements (Supplement 1) as intended.
4. Displaying Updated Supplement Details: After updating the details of Supplement 1, the display method was called again to verify that the changes were reflected accurately.

To conclude the Supplement class appears to be functioning as expected, allowing for the creation, display, and modification of supplement objects. The test results suggest that the class's methods are correctly implemented and operating as intended.

Testing Magazine class:

Method:

A screenshot of a computer program

Description automatically generated

1. Test 1 - Display Magazine Details: Creates a Magazine object named "My Magazine" with ID 1, a base cost of 19.99, and three associated supplements. It then displays the details of the magazine, including its ID, name, cost, and the details of each associated supplement, along with the total cost.
2. Test 2 - Add Supplement to Magazine: Adds a new Supplement object named "Supplement 4" with ID 4 and cost 10.0 to the existing "My Magazine". It then displays the updated details of the magazine, including the newly added supplement.
3. Test 3 - Modify Magazine Details: Modifies the details of the existing magazine, changing its ID to 2, name to "Modified Magazine", and cost to 29.99. Since no supplements are associated with this modified magazine, it displays "No supplements available".
4. Test 4 - Null Supplement List Handling: Displays the details of the "My Magazine" again, but this time, the magazine has no supplements associated with it. Therefore, it displays "No supplements available".

Outcome:

A screenshot of a computer

Description automatically generated

Conclusion;

1. Test 1 - Display Magazine Details: The magazine "My Magazine" with ID 1 is displayed along with its associated supplements and their details. The total cost of the magazine, including supplements, is calculated to be 123.99.
2. Test 2 - Add Supplement to Magazine: A new supplement "Supplement 4" with ID 4 and cost 10.0 is successfully added to the magazine "My Magazine". The magazine details, including the new supplement, are displayed.
3. Test 3 - Modify Magazine Details: The magazine details are modified, changing the ID to 2, name to "Modified Magazine", and cost to 29.99. However, no supplements are associated with this modified magazine, so it displays "No supplements available".
4. Test 4 - Null Supplement List Handling: The magazine "My Magazine" is displayed again, but this time with no supplements associated, hence it displays "No supplements available".

To conclude the tests demonstrate the functionality of the Magazine class in managing magazine details, associating supplements, calculating total costs, and handling scenarios where no supplements are present.

Testing customer class:

Method:

A screenshot of a computer

Description automatically generated

Test 1: Creating a customer with magazines and displaying details

Purpose: This test verifies that a Customer object can be created with a list of magazines and displays the customer details correctly.

Test 2: Adding standalone supplements and displaying updated details

Purpose: This test ensures that standalone supplements can be added to the customer's subscription, and it verifies that the updated customer details are displayed correctly.

Test 3: Calculating and displaying the total cost

Purpose: This test checks whether the method for calculating the total cost of magazines and standalone supplements works correctly and displays the accurate total cost.

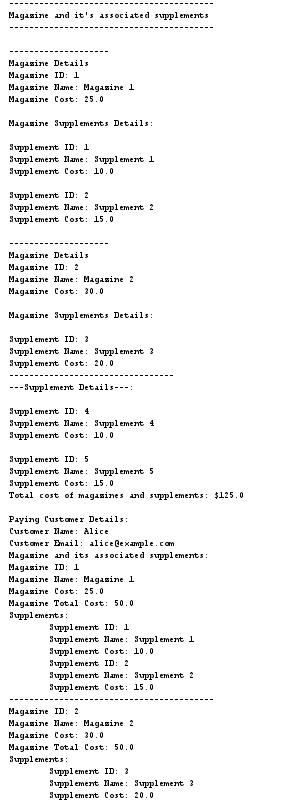
Test 4: Sending a weekly notification to the customer

Purpose: This test confirms that the weekly notification method properly gathers and presents the subscription details to the customer as expected.

Each test case aims to validate a specific functionality or behavior of the Customer class, covering aspects such as initialization, modification of subscriptions, cost calculation, and notification functionality.

Outcome:

A screenshot of a computer

Description automatically generated

Conclusion:

Test 1:

The customer named Alice with email alice@example.com is created successfully.

Two magazines are associated with Alice: Magazine 1 and Magazine 2, each with respective supplements.

Magazine 1 has two supplements: Supplement 1 and Supplement 2.

Magazine 2 has one supplement: Supplement 3.

The total cost of magazines and supplements for Alice is calculated to be $125.0.

Test 2:

Two standalone supplements, Supplement 4 and Supplement 5, are successfully added to Alice's subscription.

The total cost of magazines, supplements, and standalone supplements for Alice is recalculated to be $125.0.

Test 3:

The total cost calculated in Test 2 remains the same at $125.0.

Test 4:

A weekly notification is sent to Alice, detailing her subscriptions.

The notification includes the names, costs, and associated supplements of the magazines.

Additionally, the standalone supplements are listed separately.

The total cost of all subscriptions, including magazines and standalone supplements, is displayed as $125.0.

Conclusion: The tests indicate that the Customer class functions as intended. It successfully manages the subscription details for a customer, including magazines, associated supplements, and standalone supplements. Additionally, it accurately calculates the total cost of all subscriptions and can send notifications to customers regarding their subscriptions.

Testing Paying customer class:

Method:

A screenshot of a computer

Description automatically generated

Creation of Paying Customer: We create an instance of the PayingCustomer class with specified details such as payment method, bank, name, email, and subscriptions to magazines and supplements.

Addition of Associated Customers: We add an associated customer to the paying customer, testing the functionality of the addAssociateCustomer method.

Displaying Customer Details: We verify that the display method of the PayingCustomer class correctly displays all the details of the paying customer, including their name, email, selected payment method, bank, associated magazines, supplements, and associated customers.

Weekly Notification: We check if the weeklyNotification method properly generates a notification for the paying customer, displaying their details, subscribed magazines, supplements, and total cost for the week.

Monthly Notification: We test the monthlyNotification method to ensure it generates a notification for the paying customer, displaying their details, subscribed magazines, supplements, total cost for the month, and details of associated customers with their respective costs.

These tests cover various functionalities of the PayingCustomer class, including customer creation, association, notification generation, and display of customer details.

Outcome:

A white paper with black text

Description automatically generated A white paper with black text

Description automatically generated A screenshot of a computer

Description automatically generated

Conclusion:

Paying Customer Details Displayed Correctly: The details of the paying customer, including their name, email, selected payment method, bank, associated magazines, supplements, and associated customers, are displayed correctly.

Notification Generation: Both the weekly and monthly notifications are generated successfully. The weekly notification displays the details of the paying customer's subscriptions and total cost for the week, while the monthly notification includes details of both the paying customer and associated customers, along with the total cost for the month.

Subscription Details: The subscription details, including magazines and associated supplements, are displayed accurately for both the paying customer and associated customers.

Cost Calculation: The total cost for both the paying customer and associated customers is calculated correctly and displayed in the monthly notification.

Overall, the outcome indicates that the PayingCustomer class functions as expected, accurately managing and displaying the details of paying customers and their subscriptions, as well as generating notifications with the appropriate information.

Test table

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case | Expected Outcome | Actual Outcome | Result |
| Creation of Supplement Objects | Two Supplement objects are successfully created with different attributes (ID, name, cost). | Two Supplement objects are successfully created with different attributes (ID, name, cost). | Passed |
| Displaying Supplement Details | The display method correctly prints out the details of each supplement, including their ID, name, and cost. | The display method correctly prints out the details of each supplement, including their ID, name, and cost. | Passed |
| Updating Supplement Details | The setter methods (setSupplementName and setSupplementCost) are able to update the name and cost of one of the supplements. | The setter methods (setSupplementName and setSupplementCost) are able to update the name and cost of one of the supplements. | Passed |
| Displaying Updated Supplement Details | After updating the details of one of the supplements, the display method is called again to ensure that the updated details are correctly reflected. | After updating the details of one of the supplements, the display method is called again to ensure that the updated details are correctly reflected. | Passed |
| Display Magazine Details | The magazine "My Magazine" with ID 1 is displayed along with its associated supplements and their details. | The magazine "My Magazine" with ID 1 is displayed along with its associated supplements and their details. | Passed |
| Add Supplement to Magazine | A new supplement "Supplement 4" with ID 4 and cost 10.0 is successfully added to the magazine "My Magazine". | A new supplement "Supplement 4" with ID 4 and cost 10.0 is successfully added to the magazine "My Magazine". | Passed |
| Modify Magazine Details | The magazine details are modified, changing the ID to 2, name to "Modified Magazine", and cost to 29.99. | The magazine details are modified, changing the ID to 2, name to "Modified Magazine", and cost to 29.99. | Passed |
| Null Supplement List Handling | The magazine "My Magazine" is displayed again, but this time with no supplements associated, hence it displays "No supplements available". | The magazine "My Magazine" is displayed again, but this time with no supplements associated, hence it displays "No supplements available". | Passed |
| Creating a customer with magazines | A customer named Alice with email alice@example.com is created successfully with two magazines associated. | A customer named Alice with email alice@example.com is created successfully with two magazines associated. | Passed |
| Adding standalone supplements | Two standalone supplements, Supplement 4 and Supplement 5, are successfully added to Alice's subscription. | Two standalone supplements, Supplement 4 and Supplement 5, are successfully added to Alice's subscription. | Passed |
| Calculating and displaying total cost | The total cost of magazines, supplements, and standalone supplements for Alice is correctly calculated and displayed. | The total cost of magazines, supplements, and standalone supplements for Alice is correctly calculated and displayed. | Passed |
| Sending a weekly notification | A weekly notification is sent to Alice, detailing her subscriptions, including the names, costs, and associated supplements of the magazines, and listing standalone supplements separately. The total cost of all subscriptions, including magazines and standalone supplements, is displayed. | A weekly notification is sent to Alice, detailing her subscriptions, including the names, costs, and associated supplements of the magazines, and listing standalone supplements separately. The total cost of all subscriptions, including magazines and standalone supplements, is displayed. | Passed |
| Creation of Paying Customer | The paying customer is created successfully with specified details, including payment method, bank, name, email, and subscriptions to magazines and supplements. | The paying customer is created successfully with specified details, including payment method, bank, name, email, and subscriptions to magazines and supplements. | Passed |
| Addition of Associated Customers | An associated customer is successfully added to the paying customer using the addAssociateCustomer method. | An associated customer is successfully added to the paying customer using the addAssociateCustomer method. | Passed |
| Displaying Customer Details | The display method correctly displays all the details of the paying customer, including their name, email, selected payment method, bank, associated magazines, supplements, and associated customers. | The display method correctly displays all the details of the paying customer, including their name, email, selected payment method, bank, associated magazines, supplements, and associated customers. | Passed |
| Weekly Notification | The weeklyNotification method properly generates a notification for the paying customer, displaying their details, subscribed magazines, supplements, and total cost for the week. | The weeklyNotification method properly generates a notification for the paying customer, displaying their details, subscribed magazines, supplements, and total cost for the week. | Passed |
| Monthly Notification | The monthlyNotification method generates a notification for the paying customer, displaying their details, subscribed magazines, supplements, total cost for the month, and details of associated customers with their respective costs. | The monthlyNotification method generates a notification for the paying customer, displaying their details, subscribed magazines, supplements, total cost for the month, and details of associated customers with their respective costs. | Passed |
| Subscription class | Not tested | Did not have enough time to test it |  |
| logicaanddatabase | Not tested | Did not have enough time to test it |  |
| menu | Not tested | Did not have enough time to test it |  |

# Listing :

/\*

\* Title: Assignment 1 - Question 2

\* Author: Yin Zhanpeng

\* Date: 25/2/2024

\* File Name: Assignment1Q2.java

\*

\* Assumptions/Conditions:

\* 1. The customers have a list of Supplements and Magazines.

\* 2. The Supplements can be individual or inside the magazines.

\* 3. The magazines also contain their own List of Supplements.

\* 4. The program allows the user to add supplements to both the magazines and the customer.

\* 5. The paying customer has a list of associated customers.

\* 6. Both the paying customer and the associated customers have their own subscriptions.

\* 7. This program only simulates weekly and monthly notifications.

\* 8. It is possible to remove associated customers of a paying customer.

\* 9. If a paying customer is removed, its associated customers will also be removed.

\* 10. The customer contains the magazine and not vice versa.

\* 11. The paying customer should have at least one associated customer.

\*/

package com.mycompany.assignment;

import java.util.ArrayList;

import java.util.List;

/\*\*

\* Represents a customer who subscribes to magazines and supplements. Each

\* customer has a name, email address, and lists of magazines and supplements

\* they are subscribed to.

\*

\* @author justin

\*/

public class Customer {

private String name;// The name of the customer

private String email;// The email address of the customer.

private List<Magazine> magazine;// The list of magazines subscribed by the customer.

private List<Supplement> supplement;// The list of supplements subscribed by the customer.

/\*\*

\* Constructs a Customer object with the specified name, email, list of

\* magazines, and list of supplements.

\*

\* @param name The name of the customer.

\* @param email The email address of the customer.

\* @param magazine The list of magazines subscribed by the customer.

\* @param supplement The list of supplements subscribed by the customer.

\*/

public Customer(String name, String email, List<Magazine> magazine, List<Supplement> supplement) {

this.name = name;

this.email = email;

this.magazine = magazine;

this.supplement = supplement;

}

/\*\*

\* Retrieves the name of the customer.

\*

\* @return The name of the customer.

\*/

public String getName() {

return name;

}

/\*\*

\* Retrieves the email of the customer.

\*

\* @return The email of the customer.

\*/

public String getEmail() {

return email;

}

/\*\*

\* Retrieves the list of magazines subscribed by the customer.

\*

\* @return The list of magazines subscribed by the customer.

\*/

public List<Magazine> getMagazines() {

if (magazine == null) {

magazine = new ArrayList<>(); // Initialize the list if it's null

}

return magazine;

}

/\*\*

\* Retrieves the list of standalone supplements subscribed by the customer.

\*

\* @return The list of standalone supplements subscribed by the customer.

\*/

public List<Supplement> getSupplement() {

if (supplement == null) {

supplement = new ArrayList<>();

}

return supplement;

}

/\*\*

\* Calculates the total cost of all subscribed magazines.

\*

\* @return The total cost of all subscribed magazines.

\*/

public float getTotalMagazineCost() {

float totalMagazineCost = 0;

if (magazine != null) {

for (Magazine mag : magazine) {

totalMagazineCost += mag.getTotalMagazineCost();

}

}

return totalMagazineCost;

}

/\*\*

\* Calculates the total cost of all subscribed standalone supplements.

\*

\* @return The total cost of all subscribed standalone supplements.

\*/

public float getTotalSupplementCost() {

float totalSupplementCost = 0;

if (supplement != null) {

for (Supplement supp : supplement) {

totalSupplementCost += supp.getSupplementCost();

}

}

return totalSupplementCost;

}

/\*\*

\* Calculates the total cost of all subscriptions.

\*

\* @return The total cost of all subscriptions.

\*/

public float getTotalCost() {

return getTotalMagazineCost() + getTotalSupplementCost();

}

/\*\*

\* Sets the name of the customer.

\*

\* @param name The name of the customer.

\*/

public void setName(String name) {

this.name = name;

}

/\*\*

\* Sets the email of the customer.

\*

\* @param email The email of the customer.

\*/

public void setEmail(String email) {

this.email = email;

}

/\*\*

\* Sets the list of magazines subscribed by the customer.

\*

\* @param magazine The list of magazines subscribed by the customer.

\*/

public void setMagazine(List<Magazine> magazine) {

this.magazine = magazine;

}

/\*\*

\* Sets the list of standalone supplements subscribed by the customer.

\*

\* @param supplement The list of standalone supplements subscribed by the

\* customer.

\*/

public void setSupplement(List<Supplement> supplement) {

this.supplement = supplement;

}

/\*\*

\* Sends a weekly notification to the customer with subscription details.

\*/

public void weeklyNotification() {

float totalCost = 0;

System.out.println();

System.out.println("Paying Customer Details:");

System.out.println("Customer Name: " + this.getName());

System.out.println("Customer Email: " + this.getEmail());

// List the magazine and its supplements

if (magazine != null && !magazine.isEmpty()) {

System.out.println("Magazine and its associated supplements:");

for (Magazine mag : magazine) {

System.out.println("Magazine ID: " + mag.getMagazineId());

System.out.println("Magazine Name: " + mag.getMagazineName());

System.out.println("Magazine Cost: " + mag.getMagazineCost());

System.out.println("Magazine Total Cost: " + mag.getTotalMagazineCost());

List<Supplement> magazineSupplements = mag.getMagazineSupplement();

if (magazineSupplements != null && !magazineSupplements.isEmpty()) {

System.out.println("Supplements:");

for (Supplement supplement : magazineSupplements) {

System.out.println("\tSupplement ID: " + supplement.getSupplementId());

System.out.println("\tSupplement Name: " + supplement.getSupplementName());

System.out.println("\tSupplement Cost: " + supplement.getSupplementCost());

}

} else {

System.out.println("No supplements associated with this magazine.");

}

System.out.println("-----------------------------------------");

totalCost += mag.getTotalMagazineCost();

}

} else {

System.out.println("No magazines associated with this customer.");

}

// List standalone supplements

if (supplement != null && !supplement.isEmpty()) {

System.out.println("Standalone Supplements:");

for (Supplement supp : supplement) {

System.out.println("Supplement ID: " + supp.getSupplementId());

System.out.println("Supplement Name: " + supp.getSupplementName());

System.out.println("Supplement Cost: " + supp.getSupplementCost());

totalCost += supp.getSupplementCost();

}

} else {

System.out.println("No standalone supplements associated with this customer.");

}

// Display total cost

System.out.println("Total Cost: " + totalCost);

}

/\*\*

\* Displays the details of the customer and their subscriptions.

\*/

public void display() {

System.out.println("-----------------------------------------");

System.out.println("-----------------------------------------");

System.out.println("-----------------------------------------");

System.out.println("Associate Customer details");

System.out.println("Customer Name: " + this.name);

System.out.println("Customer Email: " + this.email);

System.out.println();

if (magazine != null && !magazine.isEmpty()) {

System.out.println("-----------------------------------------");

System.out.println("Magazine and it's associated supplements");

System.out.println("-----------------------------------------");

for (Magazine mag : magazine) {

mag.display(); // Call the display method in Magazine class

}

} else {

System.out.println("No magazines associated with this customer.");

}

System.out.println("---------------------------------");

// Display supplement details

if (supplement != null && !supplement.isEmpty()) {

System.out.println("---Supplement Details---:");

for (Supplement supp : supplement) {

supp.display(); // Call the display method in Supplement class

}

} else {

System.out.println("No supplements associated with this customer.");

}

}

/\*\*

\* Adds a standalone supplement to the customer's subscription.

\*

\* @param newSupplement The standalone supplement to add.

\*/

public void addStandaloneSupplement(Supplement newSupplement) {

if (supplement == null) {

supplement = new ArrayList<>();

}

supplement.add(newSupplement);

}

/\*\*

\* Main method to test the Customer class.

\*

\* @param args Command-line arguments.

\*/

public static void main(String[] args) {

// Test 1: Creating a customer with magazines and displaying details

// Testing for creating a customer object with magazines and displaying the customer details

List<Magazine> magazines = new ArrayList<>();

List<Supplement> supplements1 = new ArrayList<>();

supplements1.add(new Supplement(1, "Supplement 1", 10));

supplements1.add(new Supplement(2, "Supplement 2", 15));

magazines.add(new Magazine(1, "Magazine 1", 25, supplements1));

List<Supplement> supplements2 = new ArrayList<>();

supplements2.add(new Supplement(3, "Supplement 3", 20));

magazines.add(new Magazine(2, "Magazine 2", 30, supplements2));

Customer customer = new Customer("Alice", "alice@example.com", magazines, null);

customer.display(); // Displaying the customer details

// Test 2: Adding standalone supplements and displaying updated details

// Testing for adding standalone supplements to the customer and displaying the updated customer details

Supplement standalone1 = new Supplement(4, "Supplement 4", 10);

Supplement standalone2 = new Supplement(5, "Supplement 5", 15);

customer.addStandaloneSupplement(standalone1);

customer.addStandaloneSupplement(standalone2);

customer.display(); // Displaying the updated customer details

// Test 3: Calculating and displaying the total cost

// Testing for calculating and displaying the total cost of magazines and standalone supplements

System.out.println("Total cost of magazines and supplements: $" + customer.getTotalCost());

// Test 4: Sending a weekly notification to the customer

// Testing for sending a weekly notification to the customer with subscription details

customer.weeklyNotification(); // Sending a weekly notification

}

}

/\*

\* Title: Assignment 1 - Question 2

\* Author: Yin Zhanpeng

\* Date: 25/2/2024

\* File Name: Assignment1Q2.java

\*

\* Assumptions/Conditions:

\* 1. The customers have a list of Supplements and Magazines.

\* 2. The Supplements can be individual or inside the magazines.

\* 3. The magazines also contain their own List of Supplements.

\* 4. The program allows the user to add supplements to both the magazines and the customer.

\* 5. The paying customer has a list of associated customers.

\* 6. Both the paying customer and the associated customers have their own subscriptions.

\* 7. This program only simulates weekly and monthly notifications.

\* 8. It is possible to remove associated customers of a paying customer.

\* 9. If a paying customer is removed, its associated customers will also be removed.

\* 10. The customer contains the magazine and not vice versa.

\* 11. The paying customer should have at least one associated customer.

\*/

package com.mycompany.assignment;

import java.util.ArrayList;

import java.util.List;

/\*\*

\* Represents a paying customer who subscribes to magazines and supplements,

\* with additional payment and association details.

\*/

public class PayingCustomer extends Customer {

/\*\*

\* Retrieves the selected payment method for the customer.

\*

\* @return The selected payment method for the customer.

\*/

private PaymentMethod selectedPaymentMethod; // The selected payment method for the customer.

/\*\*

\* Enum representing different payment methods.

\*/

public enum PaymentMethod {

/\*\*

\* Credit card payment method.

\*/

CREDIT\_CARD,

/\*\*

\* Bank card payment method.

\*/

BANK\_CARD

}

private String bank; // The bank associated with the customer's payment method.

private List<Customer> associateCustomer; // List of associated customers.

/\*\*

\* Constructor for PayingCustomer class.

\*

\* @param selectedPaymentMethod The selected payment method for the

\* customer.

\* @param bank The bank associated with the customer's payment method.

\* @param associateCustomer List of associated customers.

\* @param name Name of the paying customer.

\* @param email Email of the paying customer.

\* @param magazine List of magazines subscribed by the paying customer.

\* @param supplement List of supplements subscribed by the paying customer.

\*/

public PayingCustomer(PaymentMethod selectedPaymentMethod, String bank, List<Customer> associateCustomer, String name, String email, List<Magazine> magazine, List<Supplement> supplement) {

super(name, email, magazine, supplement);

this.selectedPaymentMethod = selectedPaymentMethod;

this.bank = bank;

this.associateCustomer = associateCustomer;

}

/\*\*

\* Retrieves the selected payment method for the customer.

\*

\* @return The selected payment method for the customer.

\*/

public PaymentMethod getSelectedPaymentMethod() {

return selectedPaymentMethod;

}

/\*\*

\* Retrieves the bank associated with the customer's payment method.

\*

\* @return The bank associated with the customer's payment method.

\*/

public String getBank() {

return bank;

}

/\*\*

\* Retrieves the list of associated customers.

\*

\* @return The list of associated customers.

\*/

public List<Customer> getAssociateCustomer() {

return associateCustomer;

}

/\*\*

\* Sets the selected payment method for the customer.

\*

\* @param selectedPaymentMethod The selected payment method for the

\* customer.

\*/

public void setSelectedPaymentMethod(PaymentMethod selectedPaymentMethod) {

this.selectedPaymentMethod = selectedPaymentMethod;

}

/\*\*

\* Sets the bank associated with the customer's payment method.

\*

\* @param bank The bank associated with the customer's payment method.

\*/

public void setBank(String bank) {

this.bank = bank;

}

/\*\*

\* Sets the list of associated customers.

\*

\* @param associateCustomer The list of associated customers.

\*/

public void setAssociateCustomer(List<Customer> associateCustomer) {

this.associateCustomer = associateCustomer;

}

/\*\*

\* Adds an associated customer to the list.

\*

\* @param customer The associated customer to add.

\*/

public void addAssociateCustomer(Customer customer) {

if (this.associateCustomer == null) {

this.associateCustomer = new ArrayList<>();

}

this.associateCustomer.add(customer);

}

/\*\*

\* Sends a weekly notification to the paying customer with subscription

\* details.

\*/

@Override

public void weeklyNotification() {

float totalCost = 0;

// Display customer details

System.out.println();

System.out.println("Paying Customer Details:");

System.out.println("Customer Name: " + this.getName());

System.out.println("Customer Email: " + this.getEmail());

System.out.println("Selected Payment Method: " + this.selectedPaymentMethod);

System.out.println("Bank: " + this.bank);

System.out.println();

// List the magazine and its supplements

if (this.getMagazines() != null && !this.getMagazines().isEmpty()) {

System.out.println("Magazines and their associated supplements:");

for (Magazine mag : this.getMagazines()) {

System.out.println("Magazine ID: " + mag.getMagazineId());

System.out.println("Magazine Name: " + mag.getMagazineName());

System.out.println("Magazine Cost: " + mag.getMagazineCost());

System.out.println("Magazine Total Cost: " + mag.getTotalMagazineCost());

List<Supplement> magazineSupplements = mag.getMagazineSupplement();

if (magazineSupplements != null && !magazineSupplements.isEmpty()) {

System.out.println("Supplements:");

for (Supplement supplement : magazineSupplements) {

System.out.println("\tSupplement ID: " + supplement.getSupplementId());

System.out.println("\tSupplement Name: " + supplement.getSupplementName());

System.out.println("\tSupplement Cost: " + supplement.getSupplementCost());

}

} else {

System.out.println("No supplements associated with this magazine.");

}

System.out.println("-----------------------------------------");

totalCost += mag.getTotalMagazineCost();

}

} else {

System.out.println("No magazines associated with this customer.");

}

// List standalone supplements

if (this.getSupplement() != null && !this.getSupplement().isEmpty()) {

System.out.println("Standalone Supplements:");

for (Supplement supp : this.getSupplement()) {

System.out.println("Supplement ID: " + supp.getSupplementId());

System.out.println("Supplement Name: " + supp.getSupplementName());

System.out.println("Supplement Cost: " + supp.getSupplementCost());

totalCost += supp.getSupplementCost();

}

} else {

System.out.println("No standalone supplements associated with this customer.");

}

// Display total cost

System.out.println("Total Cost: " + totalCost);

}

/\*\*

\* Sends a monthly notification to the paying customer with subscription

\* details, including associated customers.

\*/

public void monthlyNotification() {

// Display customer details

System.out.println("Monthly Notification for Paying Customer:");

System.out.println("Customer Name: " + this.getName());

System.out.println("Customer Email: " + this.getEmail());

System.out.println("Selected Payment Method: " + this.selectedPaymentMethod);

System.out.println("Bank: " + this.bank);

System.out.println();

// Calculate and display total cost for the paying customer

float totalCost = calculateAndDisplayCustomerCost(this);

// Display associate customers and their associated magazines and supplements

if (this.associateCustomer != null && !this.associateCustomer.isEmpty()) {

for (Customer associate : this.associateCustomer) {

// Calculate and display total cost for the associate customer

totalCost += calculateAndDisplayCustomerCost(associate);

}

} else {

System.out.println("No associate customers.");

}

// Display total cost for the paying customer and their associate customers

System.out.println("Total Cost for the Month (Including Associate Customers): " + totalCost);

}

/\*\*

\* Helper method to calculate and display the total cost for a customer

\* (magazines + supplements).

\*

\* @param customer The customer for whom to calculate the total cost.

\* @return The total cost for the customer.

\*/

private float calculateAndDisplayCustomerCost(Customer customer) {

float customerTotalCost = 0;

// Display customer name and email

System.out.println("Customer Name: " + customer.getName());

System.out.println("Customer Email: " + customer.getEmail());

// Display magazines and their associated costs

if (customer.getMagazines() != null && !customer.getMagazines().isEmpty()) {

System.out.println("Magazines and their associated costs:");

for (Magazine mag : customer.getMagazines()) {

float magazineCost = mag.getMagazineCost();

System.out.println("Magazine ID: " + mag.getMagazineId() + ", Cost: " + magazineCost);

customerTotalCost += magazineCost;

// Display supplements for this magazine

List<Supplement> magazineSupplements = mag.getMagazineSupplement();

if (magazineSupplements != null && !magazineSupplements.isEmpty()) {

System.out.println("Supplements in this Magazine:");

for (Supplement supplement : magazineSupplements) {

float supplementCost = supplement.getSupplementCost();

System.out.println("\tSupplement ID: " + supplement.getSupplementId() + ", Cost: " + supplementCost);

customerTotalCost += supplementCost;

}

}

}

} else {

System.out.println("No magazines associated with this customer.");

}

// Display standalone supplements and their costs

if (customer.getSupplement() != null && !customer.getSupplement().isEmpty()) {

System.out.println("Standalone Supplements and their costs:");

for (Supplement supp : customer.getSupplement()) {

float supplementCost = supp.getSupplementCost();

System.out.println("Supplement ID: " + supp.getSupplementId() + ", Cost: " + supplementCost);

customerTotalCost += supplementCost;

}

} else {

System.out.println("No standalone supplements associated with this customer.");

}

// Display total cost for the customer

System.out.println("Total Cost for the Customer: " + customerTotalCost);

System.out.println();

return customerTotalCost;

}

/\*\*

\* Displays the details of the paying customer, including associated

\* magazines, supplements, and customers.

\*/

@Override

public void display() {

System.out.println();

System.out.println();

System.out.println();

System.out.println("Paying Customer Details:");

System.out.println("Customer Name: " + this.getName());

System.out.println("Customer Email: " + this.getEmail());

System.out.println("Selected Payment Method: " + this.selectedPaymentMethod);

System.out.println("Bank: " + this.bank);

System.out.println();

// Display associated magazines

if (this.getMagazines() != null && !this.getMagazines().isEmpty()) {

System.out.println("-----------------------------------------");

System.out.println("Magazines and their associated supplements:");

System.out.println("-----------------------------------------");

for (Magazine mag : this.getMagazines()) {

mag.display();

}

} else {

System.out.println("No magazines associated with this customer.");

}

System.out.println("---------------------------------");

// Display associated supplements

if (this.getSupplement() != null && !this.getSupplement().isEmpty()) {

System.out.println("---Supplement Details---:");

for (Supplement supp : this.getSupplement()) {

supp.display();

}

} else {

System.out.println("No supplements associated with this customer.");

}

// Display associated customers

if (this.associateCustomer != null && !this.associateCustomer.isEmpty()) {

System.out.println();

System.out.println("-----------------------------------------------");

System.out.println("Associated Customers:");

for (Customer customer : this.associateCustomer) {

customer.display();

}

} else {

System.out.println("No associated customers.");

}

}

public static void main(String[] args) {

// Create magazines

List<Magazine> magazines = new ArrayList<>();

magazines.add(new Magazine(1, "Magazine 1", 15.0f, null));

magazines.add(new Magazine(2, "Magazine 2", 20.0f, null));

// Create supplements

List<Supplement> supplements = new ArrayList<>();

supplements.add(new Supplement(1, "Supplement 1", 10.0f));

supplements.add(new Supplement(2, "Supplement 2", 12.0f));

// Create a paying customer

PayingCustomer payingCustomer = new PayingCustomer(

PayingCustomer.PaymentMethod.CREDIT\_CARD,

"My Bank",

null,

"John Doe",

"john.doe@example.com",

magazines,

supplements

);

// Add associated customers

List<Magazine> associatedMagazines = new ArrayList<>();

associatedMagazines.add(new Magazine(3, "Associated Magazine 1", 18.0f, null));

associatedMagazines.add(new Magazine(4, "Associated Magazine 2", 22.0f, null));

List<Supplement> associatedSupplements = new ArrayList<>();

associatedSupplements.add(new Supplement(3, "Associated Supplement 1", 8.0f));

associatedSupplements.add(new Supplement(4, "Associated Supplement 2", 15.0f));

Customer associatedCustomer = new Customer("Alice", "alice@example.com", associatedMagazines, associatedSupplements);

payingCustomer.addAssociateCustomer(associatedCustomer);

// Display details

System.out.println("Displaying Paying Customer Details:");

payingCustomer.display();

// Send weekly notification

System.out.println("\nSending Weekly Notification to Paying Customer:");

payingCustomer.weeklyNotification();

// Send monthly notification

System.out.println("\nSending Monthly Notification to Paying Customer:");

payingCustomer.monthlyNotification();

}

}

/\*

\* Title: Assignment 1 - Question 2

\* Author: Yin Zhanpeng

\* Date: 25/2/2024

\* File Name: Assignment1Q2.java

\*

\* Assumptions/Conditions:

\* 1. The customers have a list of Supplements and Magazines.

\* 2. The Supplements can be individual or inside the magazines.

\* 3. The magazines also contain their own List of Supplements.

\* 4. The program allows the user to add supplements to both the magazines and the customer.

\* 5. The paying customer has a list of associated customers.

\* 6. Both the paying customer and the associated customers have their own subscriptions.

\* 7. This program only simulates weekly and monthly notifications.

\* 8. It is possible to remove associated customers of a paying customer.

\* 9. If a paying customer is removed, its associated customers will also be removed.

\* 10. The customer contains the magazine and not vice versa.

\* 11. The paying customer should have at least one associated customer.

\*/

package com.mycompany.assignment;

import java.util.ArrayList;

import java.util.List;

/\*\*

\* Represents a magazine subscription, including details such as ID, name, cost,

\* and associated supplements.

\*/

public class Magazine {

private int magazineId;

private String magazineName;

private float magazineCost;

private List<Supplement> magazineSupplement;

/\*\*

\* Represents a magazine.

\*

\* @param magazineId The unique identifier of the magazine.

\* @param magazineName The name of the magazine.

\* @param magazineCost The cost of the magazine.

\* @param magazineSupplement The list of supplements associated with the

\* magazine.

\*/

public Magazine(int magazineId, String magazineName, float magazineCost, List<Supplement> magazineSupplement) {

this.magazineId = magazineId;

this.magazineName = magazineName;

this.magazineCost = magazineCost;

this.magazineSupplement = magazineSupplement;

}

/\*\*

\* Retrieves the ID of the magazine.

\*

\* @return The ID of the magazine.

\*/

public int getMagazineId() {

return magazineId;

}

/\*\*

\* Retrieves the name of the magazine.

\*

\* @return The name of the magazine.

\*/

public String getMagazineName() {

return magazineName;

}

/\*\*

\* Retrieves the base cost of the magazine.

\*

\* @return The base cost of the magazine.

\*/

public float getMagazineCost() {

return magazineCost;

}

/\*\*

\* Calculates the total cost of the magazine, including associated

\* supplements.

\*

\* @return The total cost of the magazine.

\*/

public float getTotalMagazineCost() {

float totalCost = magazineCost; // Start with the magazine's base cost

// If there are supplements, add their costs

if (magazineSupplement != null) {

for (Supplement supplement : magazineSupplement) {

totalCost += supplement.getSupplementCost();

}

}

return totalCost;

}

/\*\*

\* Retrieves the list of supplements associated with the magazine.

\*

\* @return The list of supplements associated with the magazine.

\*/

public List<Supplement> getMagazineSupplement() {

return magazineSupplement;

}

/\*\*

\* Sets the ID of the magazine.

\*

\* @param magazineId The ID of the magazine.

\*/

public void setMagazineId(int magazineId) {

this.magazineId = magazineId;

}

/\*\*

\* Sets the name of the magazine.

\*

\* @param magazineName The name of the magazine.

\*/

public void setMagazineName(String magazineName) {

this.magazineName = magazineName;

}

/\*\*

\* Sets the base cost of the magazine.

\*

\* @param magazineCost The base cost of the magazine.

\*/

public void setMagazineCost(float magazineCost) {

this.magazineCost = magazineCost;

}

/\*\*

\* Sets the list of supplements associated with the magazine.

\*

\* @param magazineSupplement The list of supplements associated with the

\* magazine.

\*/

public void setMagazineSupplement(List<Supplement> magazineSupplement) {

this.magazineSupplement = magazineSupplement;

}

/\*\*

\* Adds a supplement to the magazine.

\*

\* @param newSupplement The supplement to add.

\*/

public void addSupplement(Supplement newSupplement) {

if (magazineSupplement == null) {

magazineSupplement = new ArrayList<>();

}

magazineSupplement.add(newSupplement);

}

/\*\*

\* Displays details of the magazine and its associated supplements.

\*/

public void display() {

System.out.println();

System.out.println("--------------------");

System.out.println("Magazine Details");

System.out.println("Magazine ID: " + magazineId);

System.out.println("Magazine Name: " + magazineName);

System.out.println("Magazine Cost: " + magazineCost);

System.out.println();

if (magazineSupplement != null) {

System.out.println("Magazine Supplements Details:");

for (Supplement supplement : magazineSupplement) {

supplement.display();

}

} else {

System.out.println("No supplements available for this magazine.");

}

}

/\*\*

\* Main method to test the Magazine class.

\*

\* @param args Command-line arguments.

\*/

public static void main(String[] args) {

testDisplayMagazineDetails();

testCalculateTotalMagazineCost();

testAddSupplementToMagazine();

testModifyMagazineDetails();

testNullSupplementListHandling();

}

public static void testDisplayMagazineDetails() {

List<Supplement> supplements = new ArrayList<>();

supplements.add(new Supplement(1, "Supplement 1", 78));

supplements.add(new Supplement(2, "Supplement 2", 14));

supplements.add(new Supplement(3, "Supplement 3", 12));

Magazine myMagazine = new Magazine(1, "My Magazine", 19.99f, supplements);

myMagazine.display();

}

public static void testCalculateTotalMagazineCost() {

List<Supplement> supplements = new ArrayList<>();

supplements.add(new Supplement(1, "Supplement 1", 78));

supplements.add(new Supplement(2, "Supplement 2", 14));

supplements.add(new Supplement(3, "Supplement 3", 12));

Magazine myMagazine = new Magazine(1, "My Magazine", 19.99f, supplements);

float totalCost = myMagazine.getTotalMagazineCost();

System.out.println("Total Magazine Cost: " + totalCost);

}

public static void testAddSupplementToMagazine() {

Magazine myMagazine = new Magazine(1, "My Magazine", 19.99f, null);

myMagazine.addSupplement(new Supplement(4, "Supplement 4", 10));

myMagazine.display();

}

public static void testModifyMagazineDetails() {

Magazine myMagazine = new Magazine(1, "My Magazine", 19.99f, null);

myMagazine.setMagazineId(2);

myMagazine.setMagazineName("Modified Magazine");

myMagazine.setMagazineCost(29.99f);

myMagazine.display();

}

public static void testNullSupplementListHandling() {

Magazine myMagazine = new Magazine(1, "My Magazine", 19.99f, null);

myMagazine.display();

}

}

/\*

\* Title: Assignment 1 - Question 2

\* Author: Yin Zhanpeng

\* Date: 25/2/2024

\* File Name: Assignment1Q2.java

\*

\* Assumptions/Conditions:

\* 1. The customers have a list of Supplements and Magazines.

\* 2. The Supplements can be individual or inside the magazines.

\* 3. The magazines also contain their own List of Supplements.

\* 4. The program allows the user to add supplements to both the magazines and the customer.

\* 5. The paying customer has a list of associated customers.

\* 6. Both the paying customer and the associated customers have their own subscriptions.

\* 7. This program only simulates weekly and monthly notifications.

\* 8. It is possible to remove associated customers of a paying customer.

\* 9. If a paying customer is removed, its associated customers will also be removed.

\* 10. The customer contains the magazine and not vice versa.

\* 11. The paying customer should have at least one associated customer.

\*/

package com.mycompany.assignment;

/\*\*

\* Represents a supplement that can be associated with a magazine subscription.

\*/

public class Supplement {

private int supplementId; // The ID of the supplement.

private String supplementName; // The name of the supplement.

private float supplementCost; // The cost of the supplement.

/\*\*

\* Constructs a supplement with the specified ID, name, and cost.

\*

\* @param supplementId The ID of the supplement.

\* @param supplementName The name of the supplement.

\* @param supplementCost The cost of the supplement.

\*/

public Supplement(int supplementId, String supplementName, float supplementCost) {

this.supplementId = supplementId;

this.supplementName = supplementName;

this.supplementCost = supplementCost;

}

/\*\*

\* Retrieves the ID of the supplement.

\*

\* @return The ID of the supplement.

\*/

public int getSupplementId() {

return supplementId;

}

/\*\*

\* Retrieves the name of the supplement.

\*

\* @return The name of the supplement.

\*/

public String getSupplementName() {

return supplementName;

}

/\*\*

\* Retrieves the cost of the supplement.

\*

\* @return The cost of the supplement.

\*/

public float getSupplementCost() {

return supplementCost;

}

/\*\*

\* Sets the ID of the supplement.

\*

\* @param supplementId The ID of the supplement.

\*/

public void setSupplementId(int supplementId) {

this.supplementId = supplementId;

}

/\*\*

\* Sets the name of the supplement.

\*

\* @param supplementName The name of the supplement.

\*/

public void setSupplementName(String supplementName) {

this.supplementName = supplementName;

}

/\*\*

\* Sets the cost of the supplement.

\*

\* @param supplementCost The cost of the supplement.

\*/

public void setSupplementCost(float supplementCost) {

this.supplementCost = supplementCost;

}

/\*\*

\* Displays the details of the supplement, including its ID, name, and cost.

\*/

public void display() {

System.out.println();

System.out.println("Supplement ID: " + supplementId);

System.out.println("Supplement Name: " + supplementName);

System.out.println("Supplement Cost: " + supplementCost);

}

public static void main(String[] args) {

// Creating some sample supplements

Supplement supplement1 = new Supplement(1, "Vitamin C", 10.0f);

Supplement supplement2 = new Supplement(2, "Fish Oil", 15.5f);

// Displaying the details of the supplements

System.out.println("Details of Supplement 1:");

supplement1.display();

System.out.println("\nDetails of Supplement 2:");

supplement2.display();

// Testing the setter methods

supplement1.setSupplementName("Multivitamin");

supplement1.setSupplementCost(12.75f);

// Displaying the updated details of supplement 1

System.out.println("\nUpdated Details of Supplement 1:");

supplement1.display();

}

}

/\*

\* Title: Assignment 1 - Question 2

\* Author: Yin Zhanpeng

\* Date: 25/2/2024

\* File Name: Assignment1Q2.java

\*

\* Assumptions/Conditions:

\* 1. The customers have a list of Supplements and Magazines.

\* 2. The Supplements can be individual or inside the magazines.

\* 3. The magazines also contain their own List of Supplements.

\* 4. The program allows the user to add supplements to both the magazines and the customer.

\* 5. The paying customer has a list of associated customers.

\* 6. Both the paying customer and the associated customers have their own subscriptions.

\* 7. This program only simulates weekly and monthly notifications.

\* 8. It is possible to remove associated customers of a paying customer.

\* 9. If a paying customer is removed, its associated customers will also be removed.

\* 10. The customer contains the magazine and not vice versa.

\* 11. The paying customer should have at least one associated customer.

\*/

package com.mycompany.assignment;

import java.util.ArrayList;

import java.util.List;

import java.util.Scanner;

/\*\*

\* Provides logic and functionalities related to managing subscriptions and

\* databases. This class includes methods for selecting default supplements and

\* magazines.

\*/

public class LogicAndDatabase {

/\*\*

\* Constructs a LogicAndDatabase object.

\*/

public LogicAndDatabase() {

// Default constructor does not require any additional comments.

}

//public static List<Subscription> listOfCustomers;

private static List<Subscription> listOfCustomers = new ArrayList<>(); // A list of subscriptions.

/\*\*

\* Prompts the user to select default supplements.

\*

\* @return A list of selected default supplements.

\*/

public static List<Supplement> defaultSelectionSupplememnt() {

Scanner scanner = new Scanner(System.in);

List<Supplement> selectedSupplements = new ArrayList<>();

boolean selecting = true;

while (selecting) {

System.out.println("Select a supplement from the list below:");

System.out.println("1. defaultSelectionSupplememnt\_1");

System.out.println("2. defaultSelectionSupplememnt\_2");

System.out.println("3. defaultSelectionSupplememnt\_3");

System.out.println("4. defaultSelectionSupplememnt\_4");

System.out.println("5. defaultSelectionSupplememnt\_5");

System.out.println("6. defaultSelectionSupplememnt\_6");

System.out.println("7. defaultSelectionSupplememnt\_7");

System.out.println("8. defaultSelectionSupplememnt\_8");

System.out.println("9. defaultSelectionSupplememnt\_9");

System.out.println("10. defaultSelectionSupplememnt\_10");

System.out.print("Enter the number of the supplement you want (or '0' to stop): ");

int supplementNumber = scanner.nextInt();

scanner.nextLine(); // Consume newline

switch (supplementNumber) {

case 1:

selectedSupplements.add(new Supplement(1, "defaultSelectionSupplememnt\_1 ", 1));

break;

case 2:

selectedSupplements.add(new Supplement(2, "defaultSelectionSupplememnt\_2 ", 1));

break;

case 3:

selectedSupplements.add(new Supplement(3, "defaultSelectionSupplememnt\_3 ", 1));

break;

case 4:

selectedSupplements.add(new Supplement(4, "defaultSelectionSupplememnt\_4 ", 1));

break;

case 5:

selectedSupplements.add(new Supplement(5, "defaultSelectionSupplememnt\_5 ", 1));

break;

case 6:

selectedSupplements.add(new Supplement(6, "defaultSelectionSupplememnt\_6 ", 1));

break;

case 7:

selectedSupplements.add(new Supplement(7, "defaultSelectionSupplememnt\_7 ", 1));

break;

case 8:

selectedSupplements.add(new Supplement(8, "defaultSelectionSupplememnt\_8 ", 1));

break;

case 9:

selectedSupplements.add(new Supplement(9, "defaultSelectionSupplememnt\_9 ", 1));

break;

case 10:

selectedSupplements.add(new Supplement(10, "defaultSelectionSupplememnt\_10 ", 1));

break;

case 0:

selecting = false;

break;

default:

System.out.println("Invalid supplement number.");

break;

}

if (selecting) {

System.out.print("Do you want to select another supplement? (yes/no): ");

String choice = scanner.nextLine();

if (!choice.equalsIgnoreCase("yes")) {

selecting = false;

}

}

}

return selectedSupplements;

}

/\*\*

\* Prompts the user to select default magazines along with their associated

\* supplements.

\*

\* @return A list of selected default magazines.

\*/

public static List<Magazine> defaultSelectionMagazine() {

Scanner scanner = new Scanner(System.in);

List<Magazine> selectedMagazines = new ArrayList<>();

boolean selecting = true;

while (selecting) {

System.out.println("Select a magazine from the list below:");

System.out.println("1. defaultSelectionMagazine\_1");

System.out.println("2. defaultSelectionMagazine\_2");

System.out.println("3. defaultSelectionMagazine\_3");

System.out.println("4. defaultSelectionMagazine\_4");

System.out.println("5. defaultSelectionMagazine\_5");

System.out.println("6. defaultSelectionMagazine\_6");

System.out.println("7. defaultSelectionMagazine\_7");

System.out.println("8. defaultSelectionMagazine\_8");

System.out.println("9. defaultSelectionMagazine\_9");

System.out.println("10. defaultSelectionMagazine\_10");

System.out.print("Enter the number of the magazine you want (or '0' to stop): ");

int magazineNumber = scanner.nextInt();

scanner.nextLine(); // Consume newline

switch (magazineNumber) {

case 1:

selectedMagazines.add(new Magazine(1, "defaultSelectionMagazine\_1", 1, defaultSelectionSupplememnt()));

break;

case 2:

selectedMagazines.add(new Magazine(2, "defaultSelectionMagazine\_2", 2, defaultSelectionSupplememnt()));

break;

case 3:

selectedMagazines.add(new Magazine(3, "defaultSelectionMagazine\_3", 3, defaultSelectionSupplememnt()));

break;

case 4:

selectedMagazines.add(new Magazine(4, "defaultSelectionMagazine\_4", 4, defaultSelectionSupplememnt()));

break;

case 5:

selectedMagazines.add(new Magazine(5, "defaultSelectionMagazine\_5", 5, defaultSelectionSupplememnt()));

break;

case 6:

selectedMagazines.add(new Magazine(6, "defaultSelectionMagazine\_6", 6, defaultSelectionSupplememnt()));

break;

case 7:

selectedMagazines.add(new Magazine(7, "defaultSelectionMagazine\_7", 7, defaultSelectionSupplememnt()));

break;

case 8:

selectedMagazines.add(new Magazine(8, "defaultSelectionMagazine\_8", 8, defaultSelectionSupplememnt()));

break;

case 9:

selectedMagazines.add(new Magazine(9, "defaultSelectionMagazine\_9", 9, defaultSelectionSupplememnt()));

break;

case 10:

selectedMagazines.add(new Magazine(10, "defaultSelectionMagazine\_10", 10, defaultSelectionSupplememnt()));

break;

case 0:

selecting = false;

break;

default:

System.out.println("Invalid magazine number.");

break;

}

if (selecting) {

System.out.print("Do you want to select another magazine? (yes/no): ");

String choice = scanner.nextLine();

if (!choice.equalsIgnoreCase("yes")) {

selecting = false;

}

}

}

return selectedMagazines;

}

//data base

/\*\*

\* Manages the database by creating subscriptions and populating them with

\* customers, magazines, and supplements.

\*/

public static void database() {

// stand alone supplemenr list

Supplement standAlomeSupplement\_1 = new Supplement(1, "stand alone supplement\_1 ", 1);

Supplement standAlomeSupplement\_2 = new Supplement(2, "stand alone supplement\_2 ", 1);

Supplement standAlomeSupplement\_3 = new Supplement(3, "stand alone supplement\_3 ", 1);

Supplement standAlomeSupplement\_4 = new Supplement(4, "stand alone supplement\_4 ", 1);

List<Supplement> standAlomeSupplementList = new ArrayList<>();

standAlomeSupplementList.add(standAlomeSupplement\_1);

standAlomeSupplementList.add(standAlomeSupplement\_2);

standAlomeSupplementList.add(standAlomeSupplement\_3);

standAlomeSupplementList.add(standAlomeSupplement\_4);

// 10 suppliments

Supplement supplement\_1 = new Supplement(1, "supplement\_1 ", 1);

Supplement supplement\_2 = new Supplement(2, "supplement\_2 ", 1);

Supplement supplement\_3 = new Supplement(3, "supplement\_3 ", 1);

Supplement supplement\_4 = new Supplement(4, "supplement\_4 ", 1);

Supplement supplement\_5 = new Supplement(5, "supplement\_5 ", 1);

Supplement supplement\_6 = new Supplement(6, "supplement\_6 ", 1);

Supplement supplement\_7 = new Supplement(7, "supplement\_7 ", 1);

Supplement supplement\_8 = new Supplement(8, "supplement\_8 ", 1);

Supplement supplement\_9 = new Supplement(9, "supplement\_9 ", 1);

Supplement supplement\_10 = new Supplement(10, "supplement\_10 ", 1);

// 3 suppliment list

List<Supplement> supplementsList1 = new ArrayList<>();

supplementsList1.add(supplement\_1);

supplementsList1.add(supplement\_2);

supplementsList1.add(supplement\_3);

List<Supplement> supplementsList2 = new ArrayList<>();

supplementsList2.add(supplement\_4);

supplementsList2.add(supplement\_5);

supplementsList2.add(supplement\_6);

List<Supplement> supplementsList3 = new ArrayList<>();

supplementsList3.add(supplement\_7);

supplementsList3.add(supplement\_8);

supplementsList3.add(supplement\_9);

supplementsList3.add(supplement\_10);

//System.out.println("Supplement List 1: " + supplementsList1);

//System.out.println("Supplement List 2: " + supplementsList2);

//System.out.println("Supplement List 3: " + supplementsList3);

// 10 magazine

Magazine magazine\_1 = new Magazine(1, "Magazine\_1", 1, supplementsList1);

Magazine magazine\_2 = new Magazine(2, "Magazine\_2", 1, supplementsList2);

Magazine magazine\_3 = new Magazine(3, "Magazine\_3", 1, supplementsList3);

Magazine magazine\_4 = new Magazine(4, "Magazine\_4", 1, supplementsList1);

Magazine magazine\_5 = new Magazine(5, "Magazine\_5", 1, supplementsList2);

Magazine magazine\_6 = new Magazine(6, "Magazine\_6", 1, supplementsList3);

Magazine magazine\_7 = new Magazine(7, "Magazine\_7", 1, supplementsList1);

Magazine magazine\_8 = new Magazine(8, "Magazine\_8", 1, supplementsList2);

Magazine magazine\_9 = new Magazine(9, "Magazine\_9", 1, supplementsList3);

Magazine magazine\_10 = new Magazine(10, "Magazine\_10", 1, null);

// 3 magazine list

List<Magazine> magazinesList1 = new ArrayList<>();

List<Magazine> magazinesList2 = new ArrayList<>();

List<Magazine> magazinesList3 = new ArrayList<>();

magazinesList1.add(magazine\_1);

magazinesList1.add(magazine\_2);

magazinesList1.add(magazine\_3);

magazinesList2.add(magazine\_4);

magazinesList2.add(magazine\_5);

magazinesList2.add(magazine\_6);

magazinesList3.add(magazine\_7);

magazinesList3.add(magazine\_8);

magazinesList3.add(magazine\_9);

magazinesList3.add(magazine\_10);

// 10 associate customer

Customer john = new Customer("John", "john.smith@example.com", magazinesList1, standAlomeSupplementList);

Customer alice = new Customer("Alice", "alice.jones@example.com", magazinesList1, standAlomeSupplementList);

Customer robert = new Customer("Robert", "robert.williams@example.com", magazinesList1, standAlomeSupplementList);

Customer emily = new Customer("Emily", "emily.brown@example.com", magazinesList1, standAlomeSupplementList);

Customer michael = new Customer("Michael", "michael.johnson@example.com", magazinesList1, standAlomeSupplementList);

Customer sophia = new Customer("Sophia", "sophia.miller@example.com", magazinesList1, standAlomeSupplementList);

Customer james = new Customer("James", "james.davis@example.com", magazinesList1, standAlomeSupplementList);

Customer olivia = new Customer("Olivia", "olivia.garcia@example.com", magazinesList1, standAlomeSupplementList);

Customer william = new Customer("William", "william.martinez@example.com", magazinesList1, standAlomeSupplementList);

Customer isabella = new Customer("Isabella", "isabella.anderson@example.com", magazinesList1, standAlomeSupplementList);

// create 3 list of assocute customers

List<Customer> associateCustomerList1 = new ArrayList<>();

List<Customer> associateCustomerList2 = new ArrayList<>();

List<Customer> associateCustomerList3 = new ArrayList<>();

associateCustomerList1.add(john);

associateCustomerList1.add(alice);

associateCustomerList1.add(robert);

associateCustomerList2.add(emily);

associateCustomerList2.add(michael);

associateCustomerList2.add(sophia);

associateCustomerList3.add(james);

associateCustomerList3.add(olivia);

associateCustomerList3.add(william);

associateCustomerList3.add(isabella);

// 10 paying customer

PayingCustomer.PaymentMethod selectedPaymentMethod = PayingCustomer.PaymentMethod.CREDIT\_CARD;

PayingCustomer tim = new PayingCustomer(selectedPaymentMethod, "SG Bank", associateCustomerList2, "Tim", "tim@gmail.com", magazinesList2, standAlomeSupplementList);

PayingCustomer sarah = new PayingCustomer(selectedPaymentMethod, "UK Bank", associateCustomerList1, "Sarah", "sarah@gmail.com", magazinesList2, standAlomeSupplementList);

PayingCustomer alex = new PayingCustomer(selectedPaymentMethod, "US Bank", associateCustomerList1, "Alex", "alex@gmail.com", magazinesList2, standAlomeSupplementList);

PayingCustomer emma = new PayingCustomer(selectedPaymentMethod, "AU Bank", associateCustomerList3, "Emma", "emma@gmail.com", magazinesList2, standAlomeSupplementList);

PayingCustomer jacob = new PayingCustomer(selectedPaymentMethod, "CA Bank", associateCustomerList1, "Jacob", "jacob@gmail.com", magazinesList2, standAlomeSupplementList);

PayingCustomer mia = new PayingCustomer(selectedPaymentMethod, "NZ Bank", associateCustomerList2, "Mia", "mia@gmail.com", magazinesList2, standAlomeSupplementList);

PayingCustomer ethan = new PayingCustomer(selectedPaymentMethod, "FR Bank", associateCustomerList3, "Ethan", "ethan@gmail.com", magazinesList2, standAlomeSupplementList);

PayingCustomer ava = new PayingCustomer(selectedPaymentMethod, "DE Bank", associateCustomerList1, "Ava", "ava@gmail.com", magazinesList2, standAlomeSupplementList);

PayingCustomer williem = new PayingCustomer(selectedPaymentMethod, "IT Bank", associateCustomerList1, "Williem", "williem@gmail.com", magazinesList2, standAlomeSupplementList);

PayingCustomer mira = new PayingCustomer(selectedPaymentMethod, "ES Bank", associateCustomerList3, "Mira", "mira@gmail.com", magazinesList2, standAlomeSupplementList);

Subscription Tim = new Subscription(tim, "2014/02/14");

Subscription Sarah = new Subscription(sarah, "2014/02/14");

Subscription Alex = new Subscription(alex, "2014/02/14");

Subscription Emma = new Subscription(emma, "2014/02/14");

Subscription Jacob = new Subscription(jacob, "2014/02/14");

Subscription Mia = new Subscription(mia, "2014/02/14");

Subscription Ethan = new Subscription(ethan, "2014/02/14");

Subscription Ava = new Subscription(ava, "2014/02/14");

Subscription Williem = new Subscription(williem, "2014/02/14");

Subscription Mira = new Subscription(mira, "2014/02/14");

listOfCustomers.add(Sarah);

listOfCustomers.add(Alex);

listOfCustomers.add(Emma);

listOfCustomers.add(Jacob);

listOfCustomers.add(Mia);

listOfCustomers.add(Ethan);

listOfCustomers.add(Ava);

listOfCustomers.add(Williem);

listOfCustomers.add(Mira);

listOfCustomers.add(Tim);

}

/\*\*

\*

\* This method redirects the user to various options based on their choice.

\* The user is prompted to enter a choice between 1 and 11. Depending on the

\* choice, different actions are executed such as calling specific functions

\* or exiting the loop.

\*/

public static void redirect() {

boolean exit = false;

while (!exit) {

System.out.println("Enter your choice (1-11): ");

Scanner scanner = new Scanner(System.in);

int choice = Menu.displayMenu();

switch (choice) {

case 1:

System.out.println("Option 1 selected");

option1();

break;

case 2:

System.out.println("Option 2 selected");

option2();

break;

case 3:

System.out.println("Option 3 selected");

option3();

break;

case 4:

System.out.println("Option 4 selected");

// Call function for Option 4

option4();

break;

case 5:

System.out.println("Option 5 selected");

// Call function for Option 5

option5();

break;

case 6:

System.out.println("Option 6 selected");

// Call function for Option 6

option6();

break;

case 7:

System.out.println("Option 7 selected");

// Call function for Option 7

option7();

break;

case 8:

System.out.println("Option 8 selected");

// Call function for Option 8

option8();

break;

case 9:

System.out.println("Option 9 selected");

// Call function for Option 9

option9();

break;

case 10:

System.out.println("Option 10 selected");

// Call function for Option 10

option10();

break;

case 11:

System.out.println("Option 11 selected");

option11();

break;

case 12:

System.out.println("Option 12 selected");

option12();

break;

case 13:

System.out.println("Option 13 selected");

option13();

break;

case 14:

System.out.println("Exiting...");

exit = true;

break;

default:

System.out.println("Invalid choice.");

}

}

}

/\*\*

\*

\* Displays all the paying customers along with their associated customers'

\* names and emails.

\*/

public static void option1() {

for (Subscription subscription : listOfCustomers) {

PayingCustomer customer = subscription.customer;

System.out.println(subscription.getMonthlyamount());

if (customer != null) {

System.out.println("Paying Customer Name: " + customer.getName());

System.out.println("Paying Customer Email: " + customer.getEmail());

System.out.println();

List<Customer> associatedCustomers = customer.getAssociateCustomer();

if (associatedCustomers != null && !associatedCustomers.isEmpty()) {

System.out.println("Associated Customers:");

for (Customer associateCustomer : associatedCustomers) {

System.out.println(" - Name: " + associateCustomer.getName());

System.out.println(" Email: " + associateCustomer.getEmail());

}

} else {

System.out.println("No associated customers.");

}

System.out.println();

}

}

}

/\*\*

\*

\* Allows the user to add a new paying customer. The user is prompted to

\* enter details such as name, email, payment method, bank, magazines,

\* supplements, and associated customers.

\*/

public static void option2() {

Scanner scanner = new Scanner(System.in);

Subscription newEntry = new Subscription(createPayingCustomer(scanner), "2014/02/14");

listOfCustomers.add(newEntry);

}

/\*\*

\* Creates a new paying customer with the provided details.

\*

\* @param scanner The scanner object to read user input

\* @return The newly created paying customer

\*/

private static PayingCustomer createPayingCustomer(Scanner scanner) {

System.out.println("Enter details for the paying customer:");

System.out.print("Name: ");

String name = scanner.nextLine();

System.out.print("Email: ");

String email = scanner.nextLine();

// Get the payment method

System.out.println("Select payment method:");

System.out.println("1. Credit Card");

System.out.println("2. Bank Card");

int paymentMethodChoice = scanner.nextInt();

scanner.nextLine(); // Consume newline left-over

PayingCustomer.PaymentMethod paymentMethod = null;

switch (paymentMethodChoice) {

case 1:

paymentMethod = PayingCustomer.PaymentMethod.CREDIT\_CARD;

break;

case 2:

paymentMethod = PayingCustomer.PaymentMethod.BANK\_CARD;

break;

default:

System.out.println("Invalid choice. Defaulting to Credit Card.");

paymentMethod = PayingCustomer.PaymentMethod.CREDIT\_CARD;

break;

}

System.out.print("Bank: ");

String bank = scanner.nextLine();

// Get the list of magazines

List<Magazine> magazinesList = getMagazinesList(scanner);

// Get the list of supplements

List<Supplement> supplementsList = getSupplementsList(scanner);

// Get the list of associate customers

List<Customer> associateCustomersList = getAssociateCustomersList(scanner);

// Create and return the paying customer object

return new PayingCustomer(paymentMethod, bank, associateCustomersList, name, email, magazinesList, supplementsList);

}

/\*\*

\* Gets the list of associate customers along with their details.

\*

\* @param scanner The scanner object to read user input

\* @return The list of associate customers

\*/

private static List<Customer> getAssociateCustomersList(Scanner scanner) {

List<Customer> associateCustomersList = new ArrayList<>();

boolean addAnotherCustomer = true;

while (addAnotherCustomer) {

System.out.println("Enter details for an associate customer:");

System.out.print("Name: ");

String name = scanner.nextLine();

System.out.print("Email: ");

String email = scanner.nextLine();

// Get the list of magazines for this associate customer

List<Magazine> magazinesList = getMagazinesList(scanner);

// Get the list of supplements for this associate customer

List<Supplement> supplementsList = getSupplementsList(scanner);

// Create a new Customer object with the provided details

Customer associateCustomer = new Customer(name, email, magazinesList, supplementsList);

// Add the new customer to the associate customers list

associateCustomersList.add(associateCustomer);

// Ask if the user wants to add another associate customer

System.out.print("Do you want to add another associate customer? (yes/no): ");

String input = scanner.nextLine();

addAnotherCustomer = input.equalsIgnoreCase("yes");

}

return associateCustomersList;

}

/\*\*

\* Gets the list of magazines along with their details.

\*

\* @param scanner The scanner object to read user input

\* @return The list of magazines

\*/

private static List<Magazine> getMagazinesList(Scanner scanner) {

System.out.print("Do you want to add a magazine for this customer? (yes/no): ");

String addMagazineChoice = scanner.nextLine();

if (addMagazineChoice.equalsIgnoreCase("no")) {

return null; // Return null if the user chooses not to add a magazine

}

List<Magazine> magazinesList = new ArrayList<>();

System.out.print("Do you want to use default magazine selections? (yes/no): ");

String useDefault = scanner.nextLine();

if (useDefault.equalsIgnoreCase("yes")) {

return defaultSelectionMagazine();

} else {

boolean addAnotherMagazine = true;

while (addAnotherMagazine) {

Magazine magazine = getMagazineDetails(scanner);

magazinesList.add(magazine);

// Ask if the user wants to add another magazine

System.out.print("Do you want to add another magazine? (yes/no): ");

String input = scanner.nextLine();

addAnotherMagazine = input.equalsIgnoreCase("yes");

}

}

return magazinesList;

}

/\*\*

\* Gets details for a magazine.

\*

\* @param scanner The scanner object to read user input

\* @return The magazine object with provided details

\*/

private static Magazine getMagazineDetails(Scanner scanner) {

System.out.println("Enter details for the magazine:");

System.out.print("Magazine ID: ");

int magazineId = scanner.nextInt();

scanner.nextLine(); // Consume newline left-over

System.out.print("Magazine Name: ");

String magazineName = scanner.nextLine();

System.out.print("Magazine Cost: ");

float magazineCost = scanner.nextFloat();

scanner.nextLine(); // Consume newline left-over

// Now get the list of supplements for this magazine

List<Supplement> supplementsList = getSupplementsList(scanner);

// Create and return the magazine object

return new Magazine(magazineId, magazineName, magazineCost, supplementsList);

}

/\*\*

\* Gets the list of supplements along with their details.

\*

\* @param scanner The scanner object to read user input

\* @return The list of supplements

\*/

private static List<Supplement> getSupplementsList(Scanner scanner) {

System.out.print("Do you want to add a supplement for this customer? (yes/no): ");

String addSupplementChoice = scanner.nextLine();

if (addSupplementChoice.equalsIgnoreCase("no")) {

return null; // Return null if the user chooses not to add a supplement

}

List<Supplement> supplementsList = new ArrayList<>();

System.out.print("Do you want to use default supplement selections? (yes/no): ");

String useDefault = scanner.nextLine();

if (useDefault.equalsIgnoreCase("yes")) {

return defaultSelectionSupplememnt();

} else {

System.out.println("Enter details for each supplement (press Enter to stop):");

boolean flag = true;

while (flag) {

Supplement supplement = getSupplementDetails(scanner);

supplementsList.add(supplement);

System.out.print("Add another supplement? (yes/no): ");

String choice = scanner.nextLine().trim().toLowerCase();

if (!choice.equals("yes")) {

flag = false;

}

}

}

return supplementsList;

}

/\*\*

\* Gets details for a supplement.

\*

\* @param scanner The scanner object to read user input

\* @return The supplement object with provided details

\*/

private static Supplement getSupplementDetails(Scanner scanner) {

System.out.println("Enter supplement details:");

System.out.print("ID: ");

int supplementId = scanner.nextInt();

scanner.nextLine(); // Consume newline

System.out.print("Name: ");

String supplementName = scanner.nextLine();

System.out.print("Cost: ");

float supplementCost = scanner.nextFloat();

scanner.nextLine(); // Consume newline

return new Supplement(supplementId, supplementName, supplementCost);

}

/\*\*

\* Removes a paying customer or an associated customer of a paying customer.

\*

\*/

public static void option3() {

Scanner scanner = new Scanner(System.in);

System.out.println("Do you want to remove:");

System.out.println("1. Paying Customer");

System.out.println("2. Associate Customer of Paying Customer");

int choice = scanner.nextInt();

scanner.nextLine(); // Consume newline left-over

switch (choice) {

case 1:

removePayingCustomer(scanner);

break;

case 2:

removeAssociateCustomerOfPayingCustomer(scanner);

break;

default:

System.out.println("Invalid choice.");

break;

}

}

/\*\*

\* Removes a paying customer.

\*

\* @param scanner The scanner object to read user input

\*/

private static void removePayingCustomer(Scanner scanner) {

System.out.println("List of Paying Customers:");

int index = 1;

for (Subscription subscription : listOfCustomers) {

if (subscription.getCustomer() != null) {

System.out.println(index + ". " + subscription.getCustomer().getName());

index++;

}

}

System.out.print("Select the Paying Customer to remove: ");

int customerIndex = scanner.nextInt();

scanner.nextLine(); // Consume newline left-over

if (customerIndex >= 1 && customerIndex <= listOfCustomers.size()) {

listOfCustomers.remove(customerIndex - 1);

System.out.println("Paying Customer removed successfully.");

} else {

System.out.println("Invalid selection.");

}

}

/\*\*

\* Removes an associated customer of a paying customer.

\*

\* @param scanner The scanner object to read user input

\*/

private static void removeAssociateCustomerOfPayingCustomer(Scanner scanner) {

System.out.println("Select the Paying Customer:");

int payingCustomerIndex = selectPayingCustomer(scanner);

if (payingCustomerIndex >= 0 && payingCustomerIndex < listOfCustomers.size()) {

PayingCustomer payingCustomer = listOfCustomers.get(payingCustomerIndex).getCustomer();

List<Customer> associatedCustomers = payingCustomer.getAssociateCustomer();

if (associatedCustomers != null && !associatedCustomers.isEmpty()) {

System.out.println("List of Associated Customers for " + payingCustomer.getName() + ":");

int index = 1;

for (Customer associateCustomer : associatedCustomers) {

System.out.println(index + ". " + associateCustomer.getName());

index++;

}

System.out.print("Select the Associated Customer to remove: ");

int associateCustomerIndex = scanner.nextInt();

scanner.nextLine(); // Consume newline left-over

if (associateCustomerIndex >= 1 && associateCustomerIndex <= associatedCustomers.size()) {

associatedCustomers.remove(associateCustomerIndex - 1);

System.out.println("Associated Customer removed successfully.");

} else {

System.out.println("Invalid selection.");

}

} else {

System.out.println("No associated customers found for " + payingCustomer.getName() + ".");

}

} else {

System.out.println("Invalid selection.");

}

}

/\*\*

\* Selects a paying customer based on user input.

\*

\* @param scanner The scanner object to read user input

\* @return The index of the selected paying customer

\*/

private static int selectPayingCustomer(Scanner scanner) {

System.out.println("List of Paying Customers:");

int index = 1;

for (Subscription subscription : listOfCustomers) {

if (subscription.getCustomer() != null) {

System.out.println(index + ". " + subscription.getCustomer().getName());

index++;

}

}

System.out.print("Select the Paying Customer: ");

return scanner.nextInt() - 1;

}

/\*\*

\* This method allows a paying customer to add an associate customer to

\* their subscription. It prompts the user to select a paying customer from

\* the list of paying customers, then prompts for details of the associate

\* customer to be added. Once the details are provided, the associate

\* customer is added to the selected paying customer's subscription.

\*/

public static void option4() {

Scanner scanner = new Scanner(System.in);

System.out.println("List of Paying Customers:");

int index = 1;

for (Subscription subscription : listOfCustomers) {

if (subscription.getCustomer() != null) {

System.out.println(index + ". " + subscription.getCustomer().getName());

index++;

}

}

System.out.print("Select the Paying Customer: ");

int payingCustomerIndex = scanner.nextInt() - 1;

scanner.nextLine(); // Consume newline left-over

if (payingCustomerIndex >= 0 && payingCustomerIndex < listOfCustomers.size()) {

PayingCustomer payingCustomer = listOfCustomers.get(payingCustomerIndex).getCustomer();

// Create a new associate customer

Customer associateCustomer = createAssociateCustomer(scanner);

// Add the associate customer to the selected paying customer

payingCustomer.addAssociateCustomer(associateCustomer);

System.out.println("Associate customer added successfully to " + payingCustomer.getName());

} else {

System.out.println("Invalid selection.");

}

}

private static Customer createAssociateCustomer(Scanner scanner) {

System.out.println("Enter details for the associate customer:");

System.out.print("Name: ");

String name = scanner.nextLine();

System.out.print("Email: ");

String email = scanner.nextLine();

// Get the list of magazines for the associate customer

List<Magazine> magazinesList = getMagazinesList(scanner);

// Get the list of supplements for the associate customer

List<Supplement> supplementsList = getSupplementsList(scanner);

// Create and return the associate customer object

return new Customer(name, email, magazinesList, supplementsList);

}

/\*\*

\* This method allows a paying customer to remove an associate customer from

\* their subscription. It prompts the user to select a paying customer from

\* the list of paying customers, then lists out the associated customers for

\* the selected paying customer. The user can then select an associated

\* customer to remove from the subscription.

\*/

public static void option5() {

Scanner scanner = new Scanner(System.in);

System.out.println("Select the Paying Customer:");

int payingCustomerIndex = selectPayingCustomer(scanner);

if (payingCustomerIndex >= 0 && payingCustomerIndex < listOfCustomers.size()) {

PayingCustomer payingCustomer = listOfCustomers.get(payingCustomerIndex).getCustomer();

List<Customer> associatedCustomers = payingCustomer.getAssociateCustomer();

if (associatedCustomers != null && !associatedCustomers.isEmpty()) {

System.out.println("List of Associated Customers for " + payingCustomer.getName() + ":");

int index = 1;

for (Customer associateCustomer : associatedCustomers) {

System.out.println(index + ". " + associateCustomer.getName());

index++;

}

System.out.print("Select the Associated Customer to remove: ");

int associateCustomerIndex = scanner.nextInt();

scanner.nextLine(); // Consume newline left-over

if (associateCustomerIndex >= 1 && associateCustomerIndex <= associatedCustomers.size()) {

associatedCustomers.remove(associateCustomerIndex - 1);

System.out.println("Associated Customer removed successfully.");

} else {

System.out.println("Invalid selection.");

}

} else {

System.out.println("No associated customers found for " + payingCustomer.getName() + ".");

}

} else {

System.out.println("Invalid selection.");

}

}

/\*\*

\* This method allows a user to add a magazine to a customer's subscription.

\* It prompts the user to select a customer, then choose whether to add the

\* magazine to the paying customer's subscription or one of their associate

\* customers' subscriptions. Depending on the selection, the user is

\* prompted to select the specific customer and then either choose a

\* magazine from their list of magazines or input details for a new

\* magazine.

\*/

public static void option6() {

Scanner scanner = new Scanner(System.in);

System.out.println("Select the Customer:");

int customerIndex = selectCustomer(scanner);

if (customerIndex >= 0 && customerIndex < listOfCustomers.size()) {

Subscription subscription = listOfCustomers.get(customerIndex);

PayingCustomer payingCustomer = subscription.getCustomer();

List<Customer> associatedCustomers = payingCustomer.getAssociateCustomer();

System.out.println("Do you want to add a magazine for:");

System.out.println("1. Paying Customer");

System.out.println("2. Associate Customer");

int choice = scanner.nextInt();

scanner.nextLine(); // Consume newline left-over

switch (choice) {

case 1:

addMagazine(payingCustomer, scanner);

break;

case 2:

if (associatedCustomers != null && !associatedCustomers.isEmpty()) {

System.out.println("Select the Associate Customer:");

int associateCustomerIndex = selectAssociateCustomer(scanner, associatedCustomers);

if (associateCustomerIndex >= 0 && associateCustomerIndex < associatedCustomers.size()) {

Customer associateCustomer = associatedCustomers.get(associateCustomerIndex);

addMagazine(associateCustomer, scanner);

} else {

System.out.println("Invalid selection.");

}

} else {

System.out.println("No associated customers found.");

}

break;

default:

System.out.println("Invalid choice.");

break;

}

} else {

System.out.println("Invalid selection.");

}

}

private static int selectCustomer(Scanner scanner) {

System.out.println("List of Customers:");

int index = 1;

for (Subscription subscription : listOfCustomers) {

PayingCustomer payingCustomer = subscription.getCustomer();

if (payingCustomer != null) {

System.out.println(index + ". Paying Customer: " + payingCustomer.getName());

List<Customer> associatedCustomers = payingCustomer.getAssociateCustomer();

if (associatedCustomers != null && !associatedCustomers.isEmpty()) {

for (Customer associateCustomer : associatedCustomers) {

System.out.println(" - Associate Customer: " + associateCustomer.getName());

}

}

index++;

}

}

System.out.print("Select the Customer: ");

return scanner.nextInt() - 1;

}

private static int selectAssociateCustomer(Scanner scanner, List<Customer> associatedCustomers) {

int index = 1;

for (Customer associateCustomer : associatedCustomers) {

System.out.println(index + ". " + associateCustomer.getName());

index++;

}

System.out.print("Select the Associate Customer: ");

return scanner.nextInt() - 1;

}

private static void addMagazine(Customer customer, Scanner scanner) {

System.out.println("Do you want to use the default selection of magazines? (yes/no): ");

String choice = scanner.nextLine().trim();

if (choice.equalsIgnoreCase("yes")) {

List<Magazine> selectedMagazines = defaultSelectionMagazine();

customer.getMagazines().addAll(selectedMagazines);

System.out.println("Magazines added successfully to " + customer.getName() + ".");

} else if (choice.equalsIgnoreCase("no")) {

// Input magazine details manually

Magazine magazine = getMagazineDetails(scanner);

customer.getMagazines().add(magazine);

System.out.println("Magazine added successfully to " + customer.getName() + ".");

} else {

System.out.println("Invalid choice.");

}

}

/\*\*

\* This method allows a user to remove a magazine from a customer's

\* subscription. It prompts the user to select a customer, then choose

\* whether to remove the magazine from the paying customer's subscription or

\* one of their associate customers' subscriptions. Depending on the

\* selection, the user is prompted to select the specific customer and then

\* choose a magazine from their list of magazines to remove.

\*/

public static void option7() {

Scanner scanner = new Scanner(System.in);

System.out.println("Select the Customer:");

int customerIndex = selectCustomer(scanner);

if (customerIndex >= 0 && customerIndex < listOfCustomers.size()) {

Subscription subscription = listOfCustomers.get(customerIndex);

PayingCustomer payingCustomer = subscription.getCustomer();

List<Customer> associatedCustomers = payingCustomer.getAssociateCustomer();

System.out.println("Do you want to remove a magazine for:");

System.out.println("1. Paying Customer");

System.out.println("2. Associate Customer");

int choice = scanner.nextInt();

scanner.nextLine(); // Consume newline left-over

switch (choice) {

case 1:

removeMagazine(payingCustomer, scanner);

break;

case 2:

if (associatedCustomers != null && !associatedCustomers.isEmpty()) {

System.out.println("Select the Associate Customer:");

int associateCustomerIndex = selectAssociateCustomer(scanner, associatedCustomers);

if (associateCustomerIndex >= 0 && associateCustomerIndex < associatedCustomers.size()) {

Customer associateCustomer = associatedCustomers.get(associateCustomerIndex);

removeMagazine(associateCustomer, scanner);

} else {

System.out.println("Invalid selection.");

}

} else {

System.out.println("No associated customers found.");

}

break;

default:

System.out.println("Invalid choice.");

break;

}

} else {

System.out.println("Invalid selection.");

}

}

private static void removeMagazine(Customer customer, Scanner scanner) {

List<Magazine> magazinesList = customer.getMagazines();

if (magazinesList != null && !magazinesList.isEmpty()) {

System.out.println("Select the Magazine to remove:");

for (int i = 0; i < magazinesList.size(); i++) {

System.out.println((i + 1) + ". " + magazinesList.get(i).getMagazineName());

}

int magazineIndex = scanner.nextInt();

scanner.nextLine(); // Consume newline left-over

if (magazineIndex >= 1 && magazineIndex <= magazinesList.size()) {

Magazine removedMagazine = magazinesList.remove(magazineIndex - 1);

System.out.println("Magazine '" + removedMagazine.getMagazineName() + "' removed successfully.");

} else {

System.out.println("Invalid selection.");

}

} else {

System.out.println("No magazines found for this customer.");

}

}

/\*\*

\* This method allows a user to add a supplement to a customer's

\* subscription. It prompts the user to select a customer, then choose

\* whether to add the supplement to the paying customer's subscription or

\* one of their associate customers' subscriptions. Depending on the

\* selection, the user is prompted to select the specific customer and then

\* choose whether to add the supplement inside a magazine or as a standalone

\* supplement.

\*/

public static void option8() {

Scanner scanner = new Scanner(System.in);

System.out.println("Select the Customer:");

int customerIndex = selectCustomer(scanner);

if (customerIndex >= 0 && customerIndex < listOfCustomers.size()) {

Subscription subscription = listOfCustomers.get(customerIndex);

PayingCustomer payingCustomer = subscription.getCustomer();

List<Customer> associatedCustomers = payingCustomer.getAssociateCustomer();

System.out.println("Do you want to add a supplement for:");

System.out.println("1. Paying Customer");

System.out.println("2. Associate Customer");

int choice = scanner.nextInt();

scanner.nextLine(); // Consume newline left-over

switch (choice) {

case 1:

addSupplementToCustomer(payingCustomer, scanner);

break;

case 2:

if (associatedCustomers != null && !associatedCustomers.isEmpty()) {

System.out.println("Select the Associate Customer:");

int associateCustomerIndex = selectAssociateCustomer(scanner, associatedCustomers);

if (associateCustomerIndex >= 0 && associateCustomerIndex < associatedCustomers.size()) {

Customer associateCustomer = associatedCustomers.get(associateCustomerIndex);

addSupplementToCustomer(associateCustomer, scanner);

} else {

System.out.println("Invalid selection.");

}

} else {

System.out.println("No associated customers found.");

}

break;

default:

System.out.println("Invalid choice.");

break;

}

} else {

System.out.println("Invalid selection.");

}

}

private static void addSupplementToCustomer(Customer customer, Scanner scanner) {

System.out.println("Do you want to add the supplement:");

System.out.println("1. Inside a Magazine");

System.out.println("2. Standalone");

int choice = scanner.nextInt();

scanner.nextLine(); // Consume newline left-over

switch (choice) {

case 1:

addSupplementToMagazine(customer, scanner);

break;

case 2:

addStandaloneSupplement(customer, scanner);

break;

default:

System.out.println("Invalid choice.");

break;

}

}

private static void addSupplementToMagazine(Customer customer, Scanner scanner) {

// Display the list of magazines for the customer to choose from

List<Magazine> magazinesList = customer.getMagazines();

if (magazinesList != null && !magazinesList.isEmpty()) {

System.out.println("Select the Magazine:");

for (int i = 0; i < magazinesList.size(); i++) {

System.out.println((i + 1) + ". " + magazinesList.get(i).getMagazineName());

}

int magazineIndex = scanner.nextInt();

scanner.nextLine(); // Consume newline left-over

if (magazineIndex >= 1 && magazineIndex <= magazinesList.size()) {

Magazine selectedMagazine = magazinesList.get(magazineIndex - 1);

// Ask the user if they want to add the default prepopulated supplements

System.out.print("Do you want to add default prepopulated supplements? (yes/no): ");

String choice = scanner.nextLine();

if (choice.equalsIgnoreCase("yes")) {

List<Supplement> defaultSupplements = defaultSelectionSupplememnt();

selectedMagazine.getMagazineSupplement().addAll(defaultSupplements);

System.out.println("The magazine after adding the supplement is: " + selectedMagazine.getMagazineName());///testt

System.out.println("Default supplements added successfully to " + selectedMagazine.getMagazineName() + ".");

} else {

// Get supplement details

Supplement supplement = getSupplementDetails(scanner);

// Add supplement to the selected magazine

selectedMagazine.addSupplement(supplement);

System.out.println("Supplement added successfully to " + selectedMagazine.getMagazineName() + ".");

}

} else {

System.out.println("Invalid magazine selection.");

}

} else {

System.out.println("No magazines found for this customer.");

}

}

private static void addStandaloneSupplement(Customer customer, Scanner scanner) {

// Ask the user if they want to add the default prepopulated supplements

System.out.print("Do you want to add default prepopulated supplements? (yes/no): ");

String choice = scanner.nextLine();

if (choice.equalsIgnoreCase("yes")) {

List<Supplement> defaultSupplements = defaultSelectionSupplememnt();

customer.getSupplement().addAll(defaultSupplements);

System.out.println("Default supplements added successfully to " + customer.getName() + ".");

} else {

// Get supplement details

Supplement supplement = getSupplementDetails(scanner);

// Add supplement to customer

customer.getSupplement().add(supplement);

System.out.println("Supplement added successfully to " + customer.getName() + ".");

}

}

/\*\*

\* This method allows a user to remove a supplement from a customer's

\* subscription. It prompts the user to select a customer, then choose

\* whether to remove the supplement from the paying customer's subscription

\* or one of their associate customers' subscriptions. Depending on the

\* selection, the user is prompted to select the specific customer and then

\* choose whether to remove a standalone supplement or a supplement from

\* within a magazine.

\*/

public static void option9() {

Scanner scanner = new Scanner(System.in);

System.out.println("Select the Customer:");

int customerIndex = selectCustomer(scanner);

if (customerIndex >= 0 && customerIndex < listOfCustomers.size()) {

Subscription subscription = listOfCustomers.get(customerIndex);

PayingCustomer payingCustomer = subscription.getCustomer();

List<Customer> associatedCustomers = payingCustomer.getAssociateCustomer();

System.out.println("Do you want to remove a supplement for:");

System.out.println("1. Paying Customer");

System.out.println("2. Associate Customer");

int choice = scanner.nextInt();

scanner.nextLine(); // Consume newline left-over

switch (choice) {

case 1:

removeSupplementOption(payingCustomer, scanner);

break;

case 2:

if (associatedCustomers != null && !associatedCustomers.isEmpty()) {

System.out.println("Select the Associate Customer:");

int associateCustomerIndex = selectAssociateCustomer(scanner, associatedCustomers);

if (associateCustomerIndex >= 0 && associateCustomerIndex < associatedCustomers.size()) {

Customer associateCustomer = associatedCustomers.get(associateCustomerIndex);

removeSupplementOption(associateCustomer, scanner);

} else {

System.out.println("Invalid selection.");

}

} else {

System.out.println("No associated customers found.");

}

break;

default:

System.out.println("Invalid choice.");

break;

}

} else {

System.out.println("Invalid selection.");

}

}

private static void removeSupplementOption(Customer customer, Scanner scanner) {

System.out.println("Do you want to remove a supplement for:");

System.out.println("1. Standalone Supplements");

System.out.println("2. Supplements Inside Magazines");

int choice = scanner.nextInt();

scanner.nextLine(); // Consume newline left-over

switch (choice) {

case 1:

removeStandaloneSupplement(customer, scanner);

break;

case 2:

removeSupplementFromMagazine(customer, scanner);

break;

default:

System.out.println("Invalid choice.");

break;

}

}

private static void removeStandaloneSupplement(Customer customer, Scanner scanner) {

List<Supplement> supplementsList = customer.getSupplement();

if (supplementsList != null && !supplementsList.isEmpty()) {

System.out.println("Standalone Supplements:");

for (int i = 0; i < supplementsList.size(); i++) {

System.out.println((i + 1) + ". " + supplementsList.get(i).getSupplementName());

}

System.out.print("Select the Standalone Supplement to remove: ");

int supplementIndex = scanner.nextInt();

scanner.nextLine(); // Consume newline left-over

if (supplementIndex >= 1 && supplementIndex <= supplementsList.size()) {

Supplement removedSupplement = supplementsList.remove(supplementIndex - 1);

System.out.println("Standalone Supplement '" + removedSupplement.getSupplementName() + "' removed successfully.");

} else {

System.out.println("Invalid supplement selection.");

}

} else {

System.out.println("No standalone supplements found for this customer.");

}

}

private static void removeSupplementFromMagazine(Customer customer, Scanner scanner) {

List<Magazine> magazinesList = customer.getMagazines();

int cumulativeIndex = 1;

if (magazinesList != null && !magazinesList.isEmpty()) {

System.out.println("Magazines and Supplements:");

for (Magazine magazine : magazinesList) {

List<Supplement> magazineSupplements = magazine.getMagazineSupplement();

if (magazineSupplements != null && !magazineSupplements.isEmpty()) {

System.out.println("Magazine: " + magazine.getMagazineName());

for (Supplement supplement : magazineSupplements) {

System.out.println((cumulativeIndex++) + ". " + supplement.getSupplementName());

}

} else {

System.out.println(" No supplements found for this magazine.");

}

}

System.out.print("Select the Supplement to remove from magazines: ");

int supplementIndex = scanner.nextInt();

scanner.nextLine(); // Consume newline left-over

if (supplementIndex >= 1 && supplementIndex <= cumulativeIndex) {

// Find the magazine containing the selected supplement

cumulativeIndex = 1;

for (Magazine magazine : magazinesList) {

List<Supplement> magazineSupplements = magazine.getMagazineSupplement();

if (magazineSupplements != null && supplementIndex >= cumulativeIndex && supplementIndex < cumulativeIndex + magazineSupplements.size()) {

// Remove supplement from the magazine

Supplement removedSupplement = magazineSupplements.remove(supplementIndex - cumulativeIndex);

System.out.println("Supplement '" + removedSupplement.getSupplementName() + "' removed successfully from magazine '" + magazine.getMagazineName() + "'.");

return;

}

cumulativeIndex += magazineSupplements.size();

}

}

} else {

System.out.println("No magazines found for this customer.");

}

System.out.println("Invalid supplement selection.");

}

/\*\*

\* This method displays the weekly notifications for a selected customer. It

\* prompts the user to select a customer (either a paying customer or one of

\* their associates), then displays the weekly notifications for that

\* customer for four weeks.

\*/

public static void option10() {

Scanner scanner = new Scanner(System.in);

// List out paying customers and their associate customers

System.out.println("List of Paying Customers:");

int index = 1;

for (Subscription subscription : listOfCustomers) {

PayingCustomer payingCustomer = subscription.getCustomer();

if (payingCustomer != null) {

System.out.println(index + ". Paying Customer: " + payingCustomer.getName());

List<Customer> associatedCustomers = payingCustomer.getAssociateCustomer();

if (associatedCustomers != null && !associatedCustomers.isEmpty()) {

for (int i = 0; i < associatedCustomers.size(); i++) {

System.out.println(" " + (index + i + 1) + ". Associate Customer: " + associatedCustomers.get(i).getName());

}

}

index += associatedCustomers.size() + 1;

}

}

// Select a paying customer

System.out.print("Select the Customer: ");

int customerIndex = scanner.nextInt();

scanner.nextLine(); // Consume newline left-over

int currentIndex = 1;

boolean found = false;

for (Subscription subscription : listOfCustomers) {

PayingCustomer payingCustomer = subscription.getCustomer();

if (payingCustomer != null) {

if (currentIndex == customerIndex) {

// Display weekly notifications for the selected paying customer

payingCustomer.weeklyNotification();

found = true;

break;

}

List<Customer> associatedCustomers = payingCustomer.getAssociateCustomer();

if (associatedCustomers != null && !associatedCustomers.isEmpty()) {

if (currentIndex + associatedCustomers.size() >= customerIndex) {

// Display weekly notifications for the selected associate customer

associatedCustomers.get(customerIndex - currentIndex - 1).weeklyNotification();

found = true;

break;

}

currentIndex += associatedCustomers.size() + 1;

} else {

currentIndex++;

}

}

}

if (!found) {

System.out.println("Invalid selection.");

}

}

/\*\*

\* This method displays the monthly notifications for a selected paying

\* customer. It prompts the user to select a paying customer, then displays

\* the monthly notifications for that customer.

\*/

public static void option11() {

Scanner scanner = new Scanner(System.in);

// List out paying customers

System.out.println("List of Paying Customers:");

int index = 1;

for (Subscription subscription : listOfCustomers) {

PayingCustomer payingCustomer = subscription.getCustomer();

if (payingCustomer != null) {

System.out.println(index + ". Paying Customer: " + payingCustomer.getName());

index++;

}

}

// Select a paying customer

System.out.print("Select the Paying Customer: ");

int customerIndex = scanner.nextInt();

scanner.nextLine(); // Consume newline left-over

int currentIndex = 1;

boolean found = false;

for (Subscription subscription : listOfCustomers) {

PayingCustomer payingCustomer = subscription.getCustomer();

if (payingCustomer != null) {

if (currentIndex == customerIndex) {

// Display monthly notification for the selected paying customer

payingCustomer.monthlyNotification();

found = true;

break;

}

currentIndex++;

}

}

if (!found) {

System.out.println("Invalid selection.");

}

}

/\*\*

\* This method prints out the text of all the emails for all customers for

\* four weeks of magazines. It iterates through all paying customers and

\* their associate customers, displaying the weekly notifications for each

\* of them for four weeks.

\*/

public static void option12() {

// List out paying customers and their associate customers

System.out.println("Weekly Notifications for All Customers:");

for (Subscription subscription : listOfCustomers) {

PayingCustomer payingCustomer = subscription.getCustomer();

if (payingCustomer != null) {

// Display weekly notification for the paying customer

System.out.println("Paying Customer: " + payingCustomer.getName());

for (int week = 1; week <= 4; week++) {

System.out.println("Week " + week + ":");

payingCustomer.weeklyNotification();

}

// Display weekly notifications for associated customers

List<Customer> associatedCustomers = payingCustomer.getAssociateCustomer();

if (associatedCustomers != null && !associatedCustomers.isEmpty()) {

for (Customer associatedCustomer : associatedCustomers) {

System.out.println(" Associate Customer: " + associatedCustomer.getName());

for (int week = 1; week <= 4; week++) {

System.out.println("Week " + week + ":");

associatedCustomer.weeklyNotification();

}

}

}

}

}

}

/\*\*

\* This method displays the monthly notifications for all paying customers.

\* It iterates through all paying customers and displays the monthly

\* notifications for each of them.

\*/

public static void option13() {

// List out paying customers

System.out.println("Monthly Notifications for All Paying Customers:");

boolean found = false;

for (Subscription subscription : listOfCustomers) {

PayingCustomer payingCustomer = subscription.getCustomer();

if (payingCustomer != null) {

// Display monthly notification for the paying customer

payingCustomer.monthlyNotification();

found = true;

}

}

if (!found) {

System.out.println("No paying customers found.");

}

}

/\*\*

\* Main method to start the program.

\*

\* @param arg The command-line arguments passed to the program.

\*/

public static void main(String[] arg) {

database();

redirect();

}

}

/\*

\* Title: Assignment 1 - Question 2

\* Author: Yin Zhanpeng

\* Date: 25/2/2024

\* File Name: Assignment1Q2.java

\*

\* Assumptions/Conditions:

\* 1. The customers have a list of Supplements and Magazines.

\* 2. The Supplements can be individual or inside the magazines.

\* 3. The magazines also contain their own List of Supplements.

\* 4. The program allows the user to add supplements to both the magazines and the customer.

\* 5. The paying customer has a list of associated customers.

\* 6. Both the paying customer and the associated customers have their own subscriptions.

\* 7. This program only simulates weekly and monthly notifications.

\* 8. It is possible to remove associated customers of a paying customer.

\* 9. If a paying customer is removed, its associated customers will also be removed.

\* 10. The customer contains the magazine and not vice versa.

\* 11. The paying customer should have at least one associated customer.

\*/

package com.mycompany.assignment;

import java.util.List;

/\*\*

\* Represents a subscription associated with a paying customer. This class

\* calculates the total cost of magazines and supplements for the subscription

\* and provides methods to retrieve and set subscription details.

\*/

public final class Subscription {

/\*\*

\* The paying customer associated with the subscription.

\*/

PayingCustomer customer;

/\*\*

\* The start date of the subscription.

\*/

private String startSubscriptionDate;

/\*\*

\* The total weekly amount for the subscription.

\*/

private double weeklyAmount;

/\*\*

\* The total monthly amount for the subscription.

\*/

private double monthlyamount;

/\*\*

\* Constructs a new Subscription object with the specified customer and

\* start date. Calculates the total weekly and monthly amounts based on the

\* associated customer's magazine and supplement costs.

\*

\* @param customer The paying customer associated with the subscription.

\* @param startSubscriptionDate The start date of the subscription.

\*/

public Subscription(PayingCustomer customer, String startSubscriptionDate) {

this.customer = customer;

this.startSubscriptionDate = startSubscriptionDate;

this.weeklyAmount = CombineMagazineCost() + combineSupplementCost();

this.monthlyamount = CombineMagazineCost() + combineSupplementCost() \* 4;

}

/\*\*

\* Calculates the total cost of magazines for the paying customer.

\*

\* @return The total cost of magazines for the paying customer.

\*/

public double payingCustomerMagazineCost() {

if (customer != null) {

return customer.getTotalMagazineCost();

} else {

return 0.0; // Or any other default value you prefer

}

}

/\*\*

\* Calculates the total cost of supplements for the paying customer.

\*

\* @return The total cost of supplements for the paying customer.

\*/

public double payingCustomerSupplementCost() {

if (customer != null) {

return customer.getTotalSupplementCost();

} else {

return 0.0; // Or any other default value you prefer

}

}

/\*\*

\* Calculates the total cost of magazines for associate customers.

\*

\* @return The total cost of magazines for associate customers.

\*/

public double associateCustomerMagazineCost() {

double cost = 0;

if (customer != null && customer.getAssociateCustomer() != null) {

for (Customer associateCustomer : customer.getAssociateCustomer()) {

if (associateCustomer != null) {

cost += associateCustomer.getTotalMagazineCost();

}

}

}

return cost;

}

/\*\*

\* Calculates the total cost of supplements for associate customers.

\*

\* @return The total cost of supplements for associate customers.

\*/

public double associateCustomerSupplementCost() {

double cost = 0;

if (customer != null && customer.getAssociateCustomer() != null) {

for (Customer associateCustomer : customer.getAssociateCustomer()) {

if (associateCustomer != null) {

cost += associateCustomer.getTotalSupplementCost();

}

}

}

return cost;

}

/\*\*

\* Combines the total magazine cost for both paying and associate customers.

\*

\* @return The combined total magazine cost.

\*/

public double CombineMagazineCost() {

return payingCustomerMagazineCost() + associateCustomerMagazineCost();

}

/\*\*

\* Combines the total supplement cost for both paying and associate

\* customers.

\*

\* @return The combined total supplement cost.

\*/

public double combineSupplementCost() {

return associateCustomerSupplementCost() + payingCustomerSupplementCost();

}

/\*\*

\* Retrieves the paying customer associated with the subscription.

\*

\* @return The paying customer associated with the subscription.

\*/

public PayingCustomer getCustomer() {

return customer;

}

/\*\*

\* Retrieves the start date of the subscription.

\*

\* @return The start date of the subscription.

\*/

public String getStartSubscriptionDate() {

return startSubscriptionDate;

}

/\*\*

\* Retrieves the total weekly amount for the subscription.

\*

\* @return The total weekly amount for the subscription.

\*/

public double getWeeklyAmount() {

return weeklyAmount;

}

/\*\*

\* Retrieves the total monthly amount for the subscription.

\*

\* @return The total monthly amount for the subscription.

\*/

public double getMonthlyamount() {

return monthlyamount;

}

/\*\*

\* Sets the paying customer associated with the subscription.

\*

\* @param customer The paying customer to set.

\*/

public void setCustomer(PayingCustomer customer) {

this.customer = customer;

}

/\*\*

\* Sets the start date of the subscription.

\*

\* @param startSubscriptionDate The start date to set.

\*/

public void setStartSubscriptionDate(String startSubscriptionDate) {

this.startSubscriptionDate = startSubscriptionDate;

}

/\*\*

\* Sets the total weekly amount for the subscription.

\*

\* @param weeklyAmount The weekly amount to set.

\*/

public void setWeeklyAmount(double weeklyAmount) {

this.weeklyAmount = weeklyAmount;

}

/\*\*

\* Sets the total monthly amount for the subscription.

\*

\* @param monthlyamount The monthly amount to set.

\*/

public void setMonthlyamount(double monthlyamount) {

this.monthlyamount = monthlyamount;

}

/\*\*

\* Main method for testing Subscription functionality. Creates sample

\* magazines, supplements, customers, and a subscription for testing

\* purposes. Outputs the total supplement cost for the paying customer and

\* sends a monthly notification.

\*

\* @param arg The command-line arguments.

\*/

public static void main(String[] arg) {

List<Supplement> supplementsAlone = List.of(

new Supplement(1, "Supplement 1", 1),

new Supplement(2, "Supplement 2", 1),

new Supplement(3, "Supplement 3", 1)

);

List<Supplement> supplements1 = List.of(

new Supplement(1, "Supplement 1", 1),

new Supplement(2, "Supplement 2", 1),

new Supplement(3, "Supplement 3", 1)

);

List<Supplement> supplements2 = List.of(

new Supplement(1, "Supplement 1", 1),

new Supplement(2, "Supplement 2", 1),

new Supplement(3, "Supplement 3", 1)

);

List<Magazine> magazines = List.of(

new Magazine(11, "My Magazine1", 1, supplements1),

new Magazine(22, "My Magazine2", 1, supplements2),

new Magazine(33, "My Magazine3", 1, supplements1),

new Magazine(44, "My Magazine4", 1, supplements2)

);

List<Customer> associates = List.of(

new Customer("john", "john@gmail.com", magazines, supplementsAlone)

);

PayingCustomer.PaymentMethod selectedPaymentMethod = PayingCustomer.PaymentMethod.CREDIT\_CARD;

PayingCustomer tim = new PayingCustomer(

selectedPaymentMethod,

"sg bank",

associates,

"time",

"tim@gmail.com",

magazines,

supplements1

);

Subscription test = new Subscription(tim, "2014/02/14");

System.out.println(test.payingCustomerSupplementCost());

test.customer.monthlyNotification();

}

}

/\*

\* Title: Assignment 1 - Question 2

\* Author: Yin Zhanpeng

\* Date: 25/2/2024

\* File Name: Assignment1Q2.java

\*

\* Assumptions/Conditions:

\* 1. The customers have a list of Supplements and Magazines.

\* 2. The Supplements can be individual or inside the magazines.

\* 3. The magazines also contain their own List of Supplements.

\* 4. The program allows the user to add supplements to both the magazines and the customer.

\* 5. The paying customer has a list of associated customers.

\* 6. Both the paying customer and the associated customers have their own subscriptions.

\* 7. This program only simulates weekly and monthly notifications.

\* 8. It is possible to remove associated customers of a paying customer.

\* 9. If a paying customer is removed, its associated customers will also be removed.

\* 10. The customer contains the magazine and not vice versa.

\* 11. The paying customer should have at least one associated customer.

\*/

package com.mycompany.assignment;

import java.util.Scanner;

/\*\*

\* Represents a menu for managing customer subscriptions and related operations.

\* Provides methods to display various options and receive user input.

\*/

public class Menu {

/\*\*

\* Constructs a Menu object.

\*/

public Menu() {

// Default constructor does not require any additional comments.

}

/\*\*

\* Displays the menu options and prompts the user for input.

\*

\* @return The user's choice as an integer.

\*/

public static int displayMenu() {

Scanner scanner = new Scanner(System.in);

System.out.println("Menu:");

System.out.println("1. Display all the Paying Customers and their Associate Customer");

System.out.println("2. create a new Paying Customer"); // done

System.out.println("3. Remove Customer");

System.out.println("4. create Associate Customer to Paying customer");

System.out.println("5. Remove Associate Customer to Paying customer");

System.out.println("6. Add Magazine to Customer"); //done

System.out.println("7. Remove Magazine from Customer");

System.out.println("8. Add Supplement to Customer"); // done

System.out.println("9. Remove Supplement to Customer");

System.out.println("10. Display Weekly Notification for 1 Customer");

System.out.println("11. Display Monthly Notification for 1 Paying Customer");

System.out.println("12. Display 4 Weekly Notification for all Customer");

System.out.println("13. Display Monthly Notification for all Paying Customer");

System.out.println("14. Quit");

System.out.print("Enter your choice (1-10): ");

int choice = scanner.nextInt();

// Validate input

while (choice < 1 || choice > 14) {

System.out.println("Invalid choice. Please enter a number between 1 and 11.");

System.out.print("Enter your choice (1-10): ");

choice = scanner.nextInt();

}

return choice;

}

}