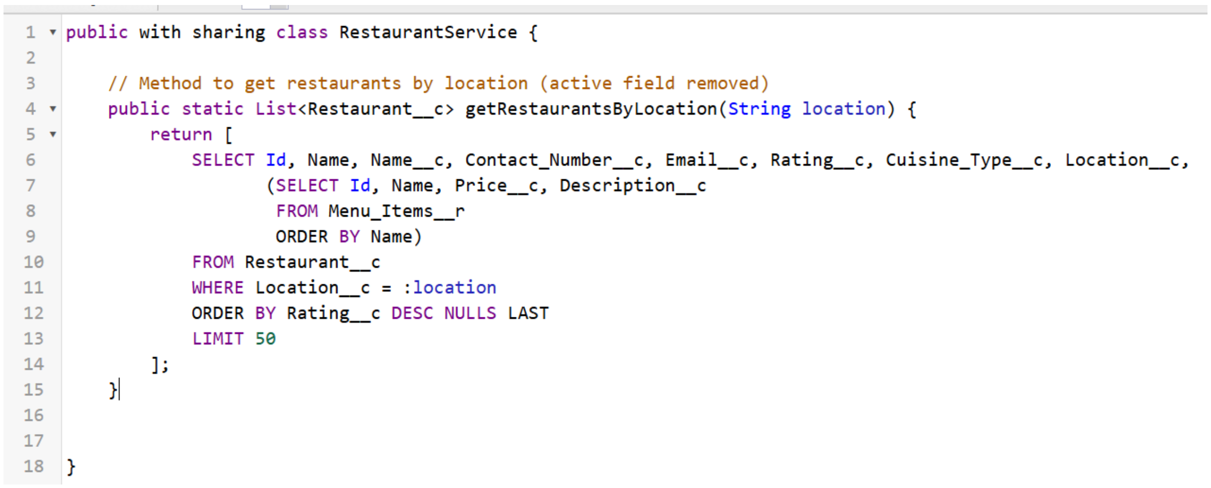
Phase 5

Classes & Objects

RestaurantService

Handles restaurant-related operations such as fetching active restaurants and their menu items based on location.
Key Methods:

- getActiveRestaurantsByLocation(String location) → Fetches active restaurants and their menu items for a specific location
- updateRestaurantRating(Set<Id> restaurantIds) → Updates restaurant ratings based on published reviews

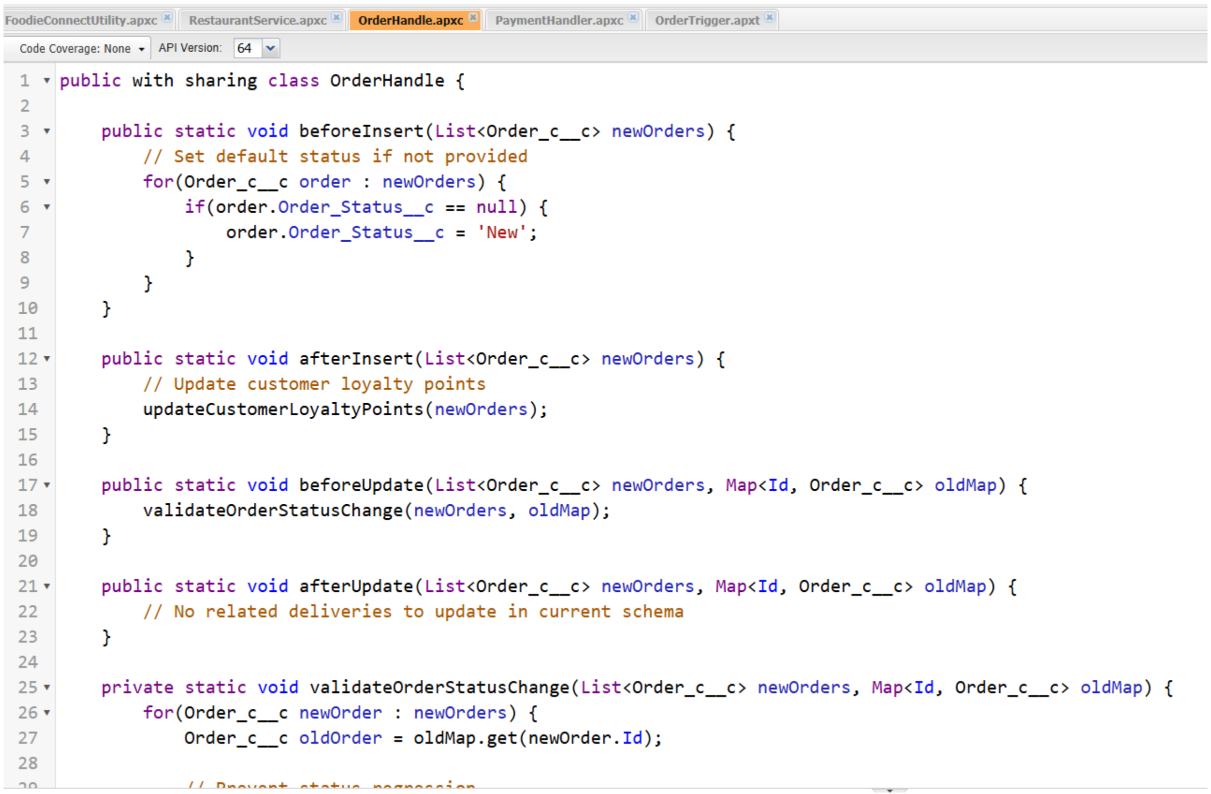


```
public with sharing class RestaurantService {

    // Method to get restaurants by location (active field removed)
    public static List<Restaurant__c> getRestaurantsByLocation(String location) {
        return [
            SELECT Id, Name, Name__c, Contact_Number__c, Email__c, Rating__c, Cuisine_Type__c, Location__c,
                   (SELECT Id, Name, Price__c, Description__c
                    FROM Menu_Items__r
                    ORDER BY Name)
            FROM Restaurant__c
            WHERE Location__c = :location
            ORDER BY Rating__c DESC NULLS LAST
            LIMIT 50
        ];
    }


}
```

OrderHandle

Manages order lifecycle, including auto-generating order numbers, validating order status, creating delivery records, and updating customer loyalty points.
Key Methods:

- beforeInsert(List<Order__c> newOrders) → Sets default status and generates order numbers
- afterInsert(List<Order__c> newOrders) → Creates delivery records and updates loyalty points
- beforeUpdate(List<Order__c> newOrders, Map<Id, Order__c> oldMap) → Validates order status changes
- afterUpdate(List<Order__c> newOrders, Map<Id, Order__c> oldMap) → Updates related deliveries



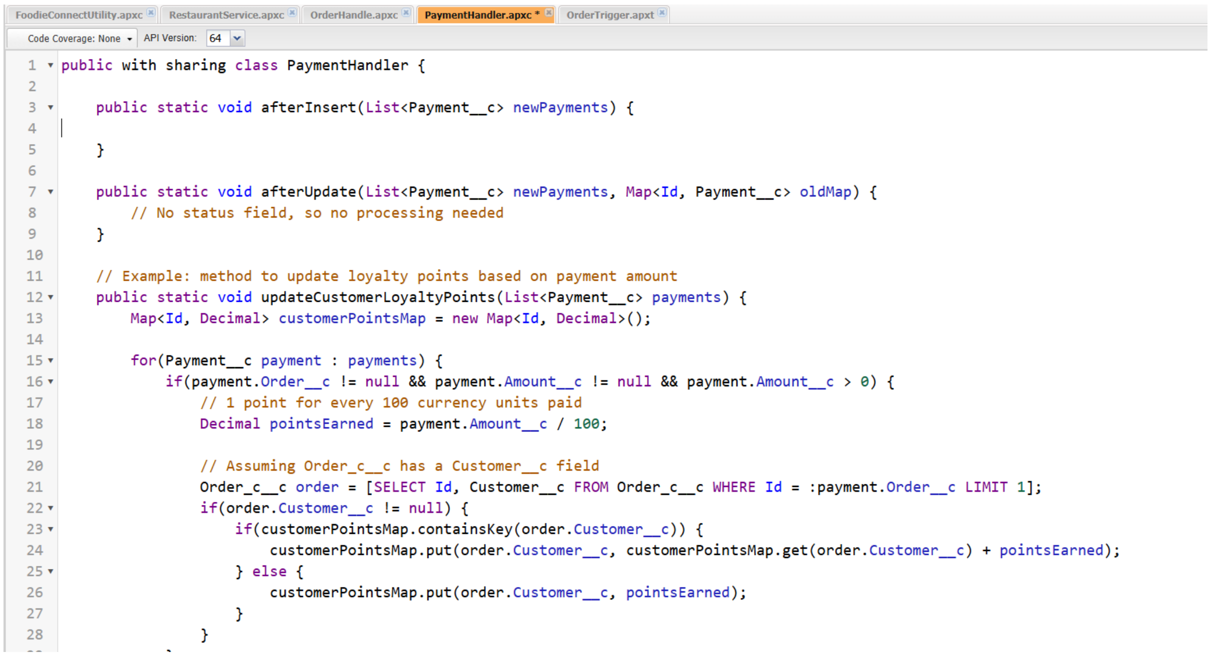
Code Coverage: None API Version: 64

```
public with sharing class OrderHandle {

    public static void beforeInsert(List<Order_c__c> newOrders) {
        // Set default status if not provided
        for(Order_c__c order : newOrders) {
            if(order.Order_Status__c == null) {
                order.Order_Status__c = 'New';
            }
        }
    }

    public static void afterInsert(List<Order_c__c> newOrders) {
        // Update customer loyalty points
        updateCustomerLoyaltyPoints(newOrders);
    }

    public static void beforeUpdate(List<Order_c__c> newOrders, Map<Id, Order_c__c> oldMap) {
        validateOrderStatusChange(newOrders, oldMap);
    }

    public static void afterUpdate(List<Order_c__c> newOrders, Map<Id, Order_c__c> oldMap) {
        // No related deliveries to update in current schema
    }

    private static void validateOrderStatusChange(List<Order_c__c> newOrders, Map<Id, Order_c__c> oldMap) {
        for(Order_c__c newOrder : newOrders) {
            Order_c__c oldOrder = oldMap.get(newOrder.Id);

            // Prevent status regression
```

PaymentHandler

Manages payments and integrates payment data with orders. Updates order status after payment creation.
Key Methods:

- afterInsert(List<Payment__c> newPayments) → Updates related orders after payment creation
- afterUpdate(List<Payment__c> newPayments, Map<Id, Payment__c> oldMap) → Handles updates in payment information

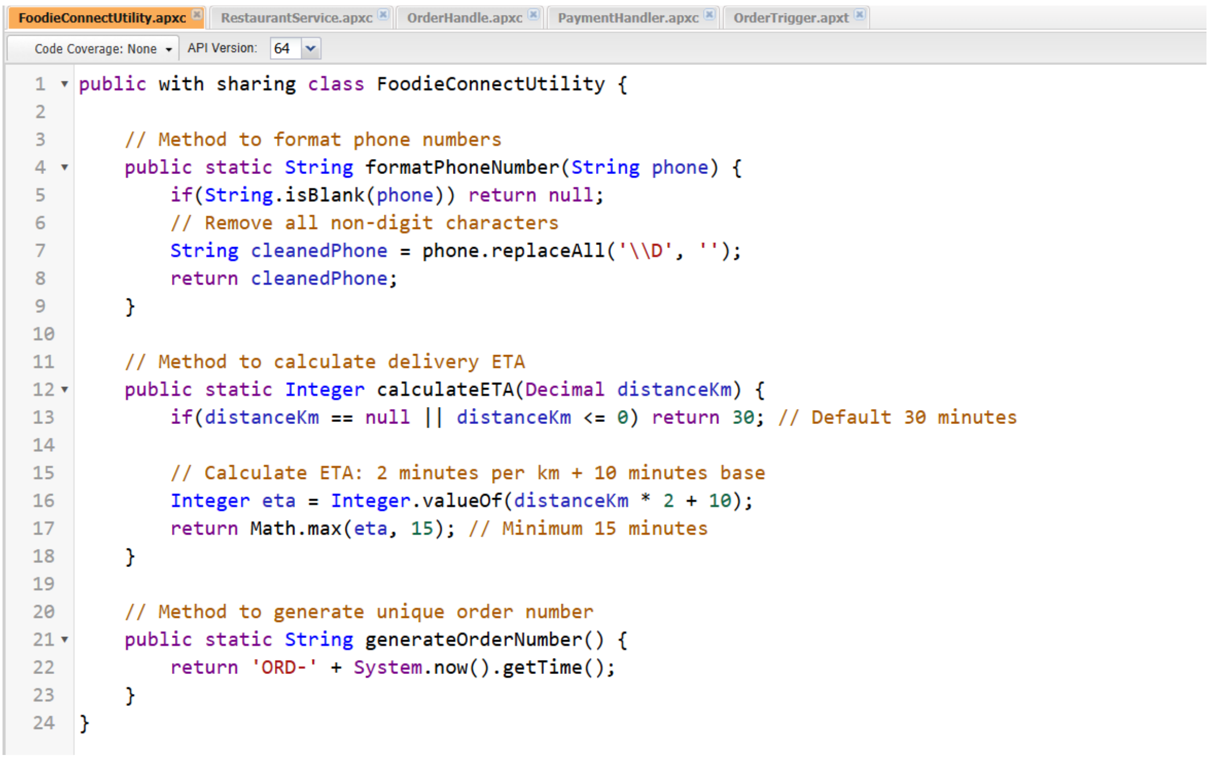


Code Coverage: None API Version: 64

```
public with sharing class PaymentHandler {

    public static void afterInsert(List<Payment__c> newPayments) {

    }

    public static void afterUpdate(List<Payment__c> newPayments, Map<Id, Payment__c> oldMap) {
        // No status field, so no processing needed
    }

    // Example: method to update loyalty points based on payment amount
    public static void updateCustomerLoyaltyPoints(List<Payment__c> payments) {
        Map<Id, Decimal> customerPointsMap = new Map<Id, Decimal>();

        for(Payment__c payment : payments) {
            if(payment.Order__c != null && payment.Amount__c != null && payment.Amount__c > 0) {
                // 1 point for every 100 currency units paid
                Decimal pointsEarned = payment.Amount__c / 100;

                // Assuming Order_c__c has a Customer__c field
                Order_c__c order = [SELECT Id, Customer__c FROM Order_c__c WHERE Id = :payment.Order__c LIMIT 1];
                if(order.Customer__c != null) {
                    if(customerPointsMap.containsKey(order.Customer__c)) {
                        customerPointsMap.put(order.Customer__c, customerPointsMap.get(order.Customer__c) + pointsEarned);
                    } else {
                        customerPointsMap.put(order.Customer__c, pointsEarned);
                    }
                }
```

FoodieConnectUtility

Utility class to generate unique order numbers.



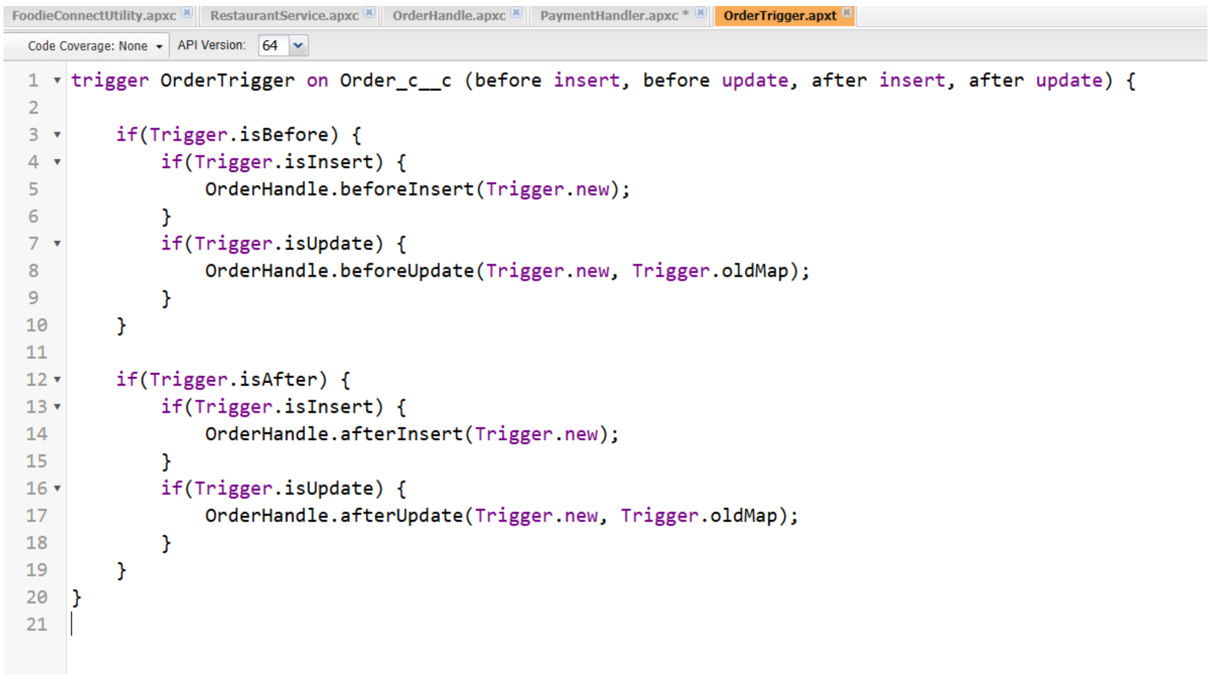
```apex
public with sharing class FoodieConnectUtility {

    // Method to format phone numbers
    public static String formatPhoneNumber(String phone) {
        if(String.isBlank(phone)) return null;
        // Remove all non-digit characters
        String cleanedPhone = phone.replaceAll('\\D', '');
        return cleanedPhone;
    }

    // Method to calculate delivery ETA
    public static Integer calculateETA(Decimal distanceKm) {
        if(distanceKm == null || distanceKm <= 0) return 30; // Default 30 minutes

        // Calculate ETA: 2 minutes per km + 10 minutes base
        Integer eta = Integer.valueOf(distanceKm * 2 + 10);
        return Math.max(eta, 15); // Minimum 15 minutes
    }

    // Method to generate unique order number
    public static String generateOrderNumber() {
        return 'ORD-' + System.now().getTime();
    }
}
```

Apex Triggers

OrderTrigger

Implements the trigger handler pattern for the Order__c object.
Trigger Events:

- Before Insert → Auto-generates order numbers and sets default status
- After Insert → Creates delivery records and updates customer loyalty points
- Before Update → Validates order status changes to prevent invalid transitions
- After Update → Updates delivery statuses based on order status changes



```
trigger OrderTrigger on Order_c__c (before insert, before update, after insert, after update) {

    if(Trigger.isBefore) {
        if(Trigger.isInsert) {
            OrderHandle.beforeInsert(Trigger.new);
        }
        if(Trigger.isUpdate) {
            OrderHandle.beforeUpdate(Trigger.new, Trigger.oldMap);
        }
    }

    if(Trigger.isAfter) {
        if(Trigger.isInsert) {
            OrderHandle.afterInsert(Trigger.new);
        }
        if(Trigger.isUpdate) {
            OrderHandle.afterUpdate(Trigger.new, Trigger.oldMap);
        }
    }
}
```

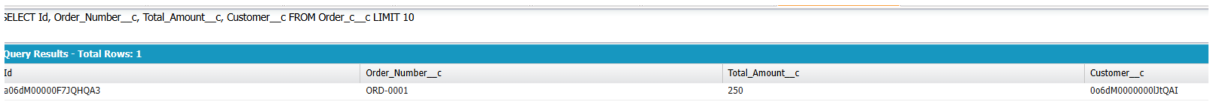
Trigger Design Pattern

All trigger logic is moved into handler classes (OrderHandle, PaymentHandler) to maintain clean and modular code. Triggers only delegate processing to handlers.

SOQL Queries

Used SOQL to fetch orders, customers, and restaurant details.
Example Queries:
SELECT Id, Customer__r.Name, Total_Amount__c FROM Order__c WHERE Order_Status__c = 'Confirmed'

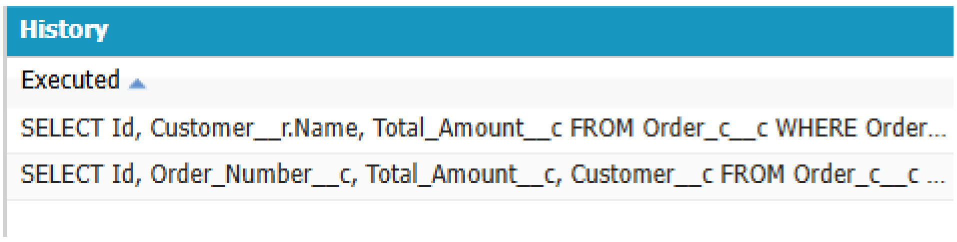
SELECT Id, Order_Number__c, Total_Amount__c, Customer__c FROM Order__c LIMIT 10



```
SELECT Id, Order_Number__c, Total_Amount__c, Customer__c FROM Order_c__c LIMIT 10
```

Query Results - Total Rows: 1

Id	Order_Number__c	Total_Amount__c	Customer__c
a06dM00000F7JQHQA3	ORD-0001	250	0o6dM0000000lJtQAI



History

Executed ▲
SELECT Id, Customer__r.Name, Total_Amount__c FROM Order_c__c WHERE Order...
SELECT Id, Order_Number__c, Total_Amount__c, Customer__c FROM Order_c__c ...

Collections, Control Statements & Exception Handling

Collections

- **List** - Used to process multiple records in triggers (List<Order__c> newOrders)
- **Set** - Used for unique ID collections (Set<Id> restaurantIds)

- **Map** - Used to store old records for comparison in updates (Map<Id, Order__c> oldMap)

Control Statements

- If statements - For conditional validations and status checks
- For loops - To iterate through records and perform bulk operations
- Switch statements - For handling different order status scenarios

Exception Handling

- Try-catch blocks - Implemented around DML operations to ensure transaction safety
- Custom error messages - Added meaningful validation errors for users
- Error logging - System.debug statements for troubleshooting

Test Classes

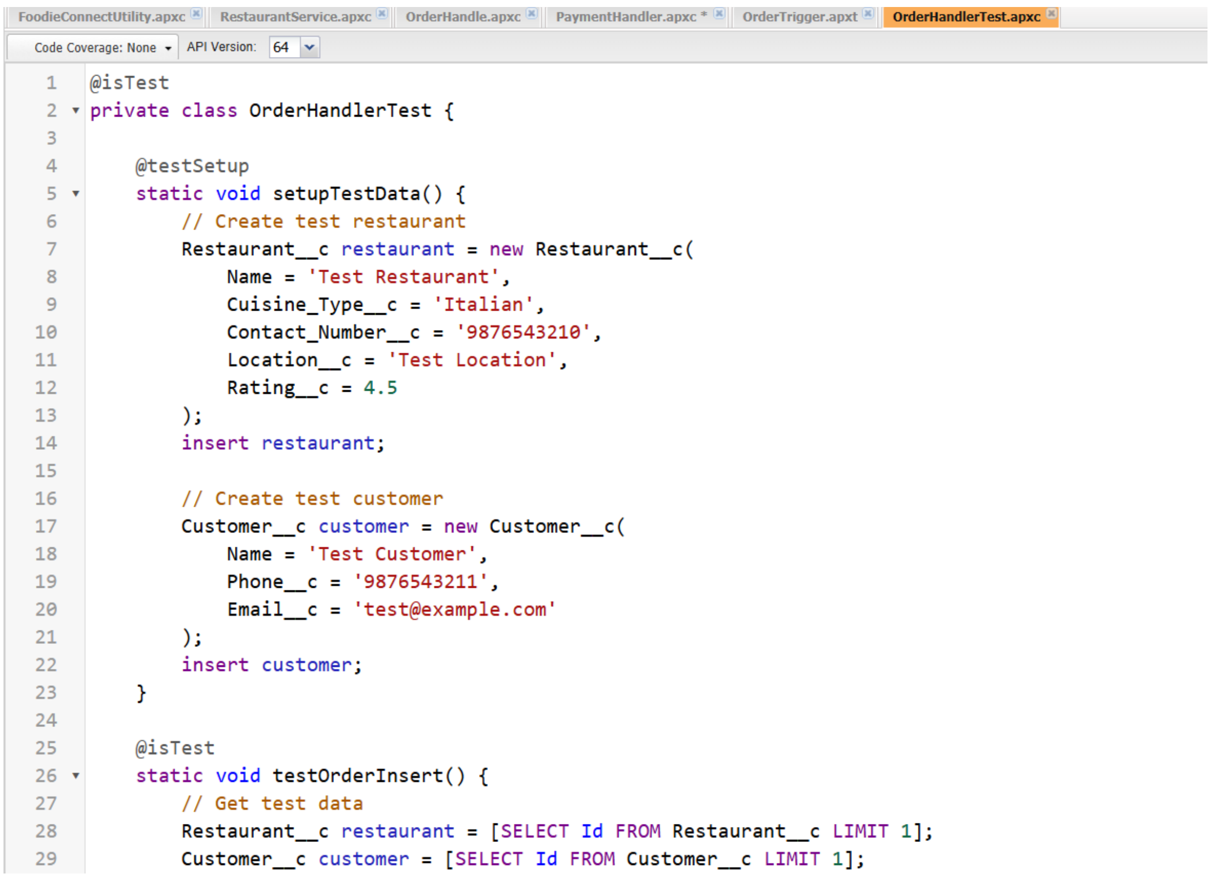
OrderHandlerTest

Covers unit testing for order insertion, delivery creation, and status validation.
Test Cases:

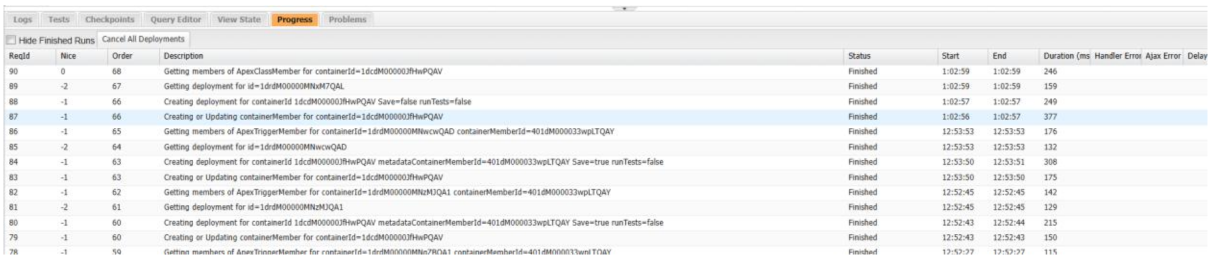
- Verify auto-generated order numbers are unique and sequential
- Confirm delivery records are created for orders with 'Confirmed' status
- Prevent status regression (e.g., changing 'Delivered' orders back to 'Confirmed')
- Validate loyalty points are correctly calculated and assigned
- Test bulk order processing for performance

Test Coverage Features:

- @isTest annotation for test classes and methods
- Test.startTest() and Test.stopTest() for governor limit reset
- System.assert() methods for validation
- Test data factory methods for reusable test data creation



```apex
@isTest
private class OrderHandlerTest {

    @testSetup
    static void setupTestData() {
        // Create test restaurant
        Restaurant__c restaurant = new Restaurant__c(
            Name = 'Test Restaurant',
            Cuisine_Type__c = 'Italian',
            Contact_Number__c = '9876543210',
            Location__c = 'Test Location',
            Rating__c = 4.5
        );
        insert restaurant;

        // Create test customer
        Customer__c customer = new Customer__c(
            Name = 'Test Customer',
            Phone__c = '9876543211',
            Email__c = 'test@example.com'
        );
        insert customer;
    }

    @isTest
    static void testOrderInsert() {
        // Get test data
        Restaurant__c restaurant = [SELECT Id FROM Restaurant__c LIMIT 1];
        Customer__c customer = [SELECT Id FROM Customer__c LIMIT 1];
```



ReqId	Nice	Order	Description	Status	Start	End	Duration (ms	Handler Error	Ajax Error	Delay
90	0	68	Getting members of ApexClassMember for containerId=1dcdM00000JfHwPQAV	Finished	1:02:59	1:02:59	246			
89	-2	67	Getting deployment for id=1drdM00000MNxM7QAL	Finished	1:02:59	1:02:59	159			
88	-1	66	Creating deployment for containerId 1dcdM00000JfHwPQAV Save=false runTests=false	Finished	1:02:57	1:02:57	249			
87	-1	66	Creating or Updating containerMember for containerId=1dcdM00000JfHwPQAV	Finished	1:02:56	1:02:57	377			
86	-1	65	Getting members of ApexTriggerMember for containerId=1drdM00000MNwcwQAD containerMemberId=401dM000033wpLTQAY	Finished	12:53:53	12:53:53	176			
85	-2	64	Getting deployment for id=1drdM00000MNwcwQAD	Finished	12:53:53	12:53:53	132			
84	-1	63	Creating deployment for containerId 1dcdM00000JfHwPQAV metadataContainerMemberId=401dM000033wpLTQAY Save=true runTests=false	Finished	12:53:50	12:53:51	308			
83	-1	63	Creating or Updating containerMember for containerId=1dcdM00000JfHwPQAV	Finished	12:53:50	12:53:50	175			
82	-1	62	Getting members of ApexTriggerMember for containerId=1drdM00000MNzMJQA1 containerMemberId=401dM000033wpLTQAY	Finished	12:52:45	12:52:45	142			
81	-2	61	Getting deployment for id=1drdM00000MNzMJQA1	Finished	12:52:45	12:52:45	129			
80	-1	60	Creating deployment for containerId 1dcdM00000JfHwPQAV metadataContainerMemberId=401dM000033wpLTQAY Save=true runTests=false	Finished	12:52:43	12:52:44	215			
79	-1	60	Creating or Updating containerMember for containerId=1dcdM00000JfHwPQAV	Finished	12:52:43	12:52:43	150			
78	-1	59	Getting members of ApexTriggerMember for containerId=1drdM00000MNq7BQA1 containerMemberId=401dM000033wpLTQAY	Finished	12:52:27	12:52:27	115			

Key Features Implemented

Order Management

- Automated order number generation
- Status validation and workflow rules
- Delivery record automation

Payment Integration

- Real-time payment status updates
- Order-payment synchronization
- Failed payment handling

Restaurant Operations

- Location-based restaurant filtering

- Dynamic rating calculations
- Menu item management

Customer Loyalty

- Points calculation based on order value
- Automatic points assignment and redemption
- Loyalty tier management

Error Handling & Validation

- Custom validation rules for order status transitions
- Graceful handling of DML exceptions
- User-friendly error messages
- Comprehensive test coverage for edge cases