Phase 5: Apex Programming (Developer)

1. Classes & Objects

Explanation:
Apex classes are used to create reusable logic in Salesforce.
In this project, classes handle backend automation — such as recalculating totals and updating inventory based on order items.

<u>Use Case: Order Item Handler Class</u>

Purpose: To calculate and update the Total Order Cost for each Purchase Order whenever related Order Items are created, updated, or deleted.

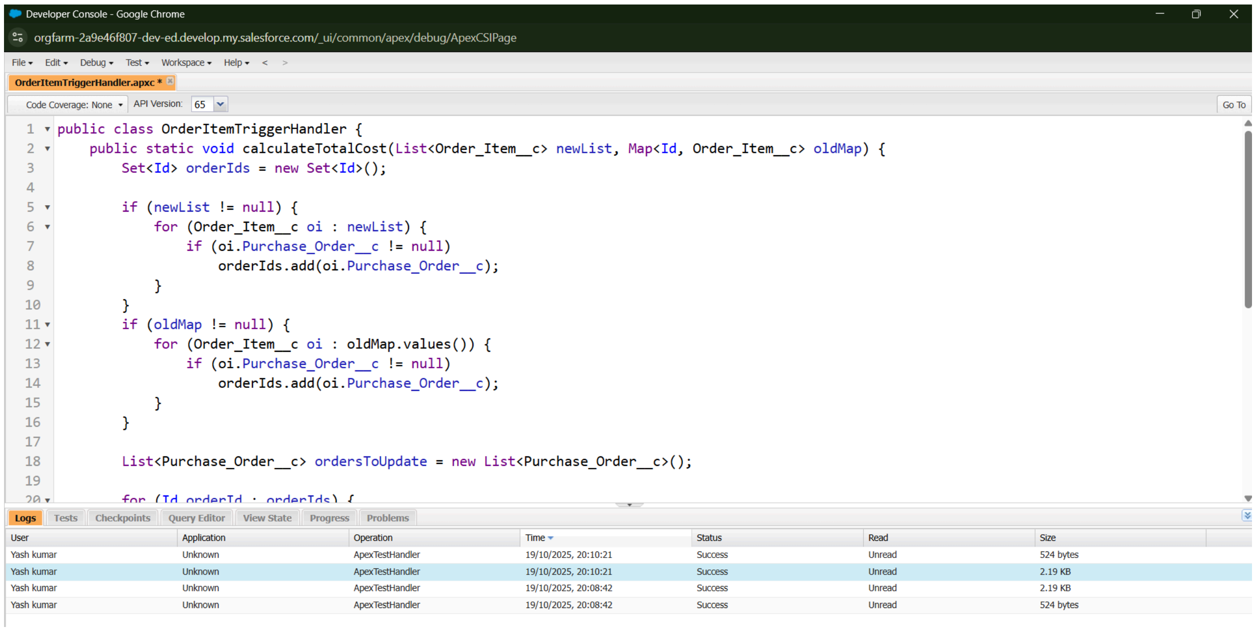
Code Example:

```
public class OrderItemTriggerHandler {
    public static void calculateTotalCost(List<Order_Item__c> newList, Map<Id,
Order_Item__c> oldMap) {
        Set<Id> orderIds = new Set<Id>();

        if (newList != null) {
            for (Order_Item__c oi : newList) {
                if (oi.Purchase_Order__c != null)
                    orderIds.add(oi.Purchase_Order__c);
            }
        }
        if (oldMap != null) {
            for (Order_Item__c oi : oldMap.values()) {
                if (oi.Purchase_Order__c != null)
                    orderIds.add(oi.Purchase_Order__c);
            }
        }
        List<Purchase_Order__c> ordersToUpdate = new
List<Purchase_Order__c>();

        for (Id orderId : orderIds) {
            Decimal total = 0;
            for (Order_Item__c oi : [
                SELECT Total_Price__c
                FROM Order_Item__c
                WHERE Purchase_Order__c = :orderId
            ]) {
                total += oi.Total_Price__c;
            }
            ordersToUpdate.add(new Purchase_Order__c(
                Id = orderId,
                Total_Order_Cost__c = total
            ));
        }

        if (!ordersToUpdate.isEmpty()) {
            update ordersToUpdate;
        }
    }
}
```


Developer Console - Google Chrome
orgfarm-2a9e46f807-dev-ed.develop.my.salesforce.com/_ui/common/apex/debug/ApexCSIPage
File Edit Debug Test Workspace Help < >
OrderItemTriggerHandler.apxc *
Code Coverage: None API Version: 65
Go To
Logs Tests Checkpoints Query Editor View State Progress Problems

```
public class OrderItemTriggerHandler {
    public static void calculateTotalCost(List<Order_Item__c> newList, Map<Id, Order_Item__c> oldMap) {
        Set<Id> orderIds = new Set<Id>();

        if (newList != null) {
            for (Order_Item__c oi : newList) {
                if (oi.Purchase_Order__c != null)
                    orderIds.add(oi.Purchase_Order__c);
            }
        }
        if (oldMap != null) {
            for (Order_Item__c oi : oldMap.values()) {
                if (oi.Purchase_Order__c != null)
                    orderIds.add(oi.Purchase_Order__c);
            }
        }

        List<Purchase_Order__c> ordersToUpdate = new List<Purchase_Order__c>();

        for (Id orderId : orderIds) {
```

User	Application	Operation	Time	Status	Read	Size
Yash kumar	Unknown	ApexTestHandler	19/10/2025, 20:10:21	Success	Unread	524 bytes
Yash kumar	Unknown	ApexTestHandler	19/10/2025, 20:10:21	Success	Unread	2.19 KB
Yash kumar	Unknown	ApexTestHandler	19/10/2025, 20:08:42	Success	Unread	2.19 KB
Yash kumar	Unknown	ApexTestHandler	19/10/2025, 20:08:42	Success	Unread	524 bytes

2. Apex Triggers (Before/After Insert/Update/Delete)

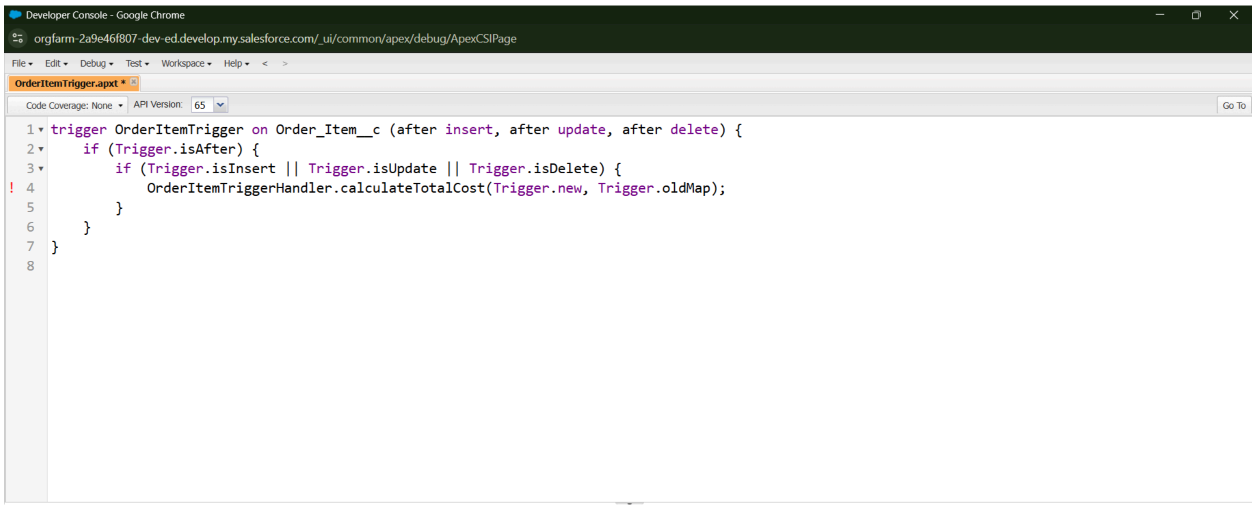
Explanation:
Triggers allow you to perform operations automatically when records change.
They are powerful for handling real-time updates.

<u>Use Case: Update Total Order Cost on Order Item Changes</u>

Trigger Name: OrderItemTrigger
Object: Order_Item__c
Events: After Insert, After Update, After Delete

Code Example:

```
trigger OrderItemTrigger on Order_Item__c (after insert, after update, after delete)
{
    if (Trigger.isAfter) {
        if (Trigger.isInsert || Trigger.isUpdate || Trigger.isDelete) {
            OrderItemTriggerHandler.calculateTotalCost(Trigger.new, Trigger.oldMap);
        }
    }
}
```


Developer Console - Google Chrome
orgfarm-2a9e46f807-dev-ed.develop.my.salesforce.com/_ui/common/apex/debug/ApexCSIPage
File Edit Debug Test Workspace Help
OrderItemTrigger.apxt *
Code Coverage: None API Version: 65
Go To

```
trigger OrderItemTrigger on Order_Item__c (after insert, after update, after delete) {
    if (Trigger.isAfter) {
        if (Trigger.isInsert || Trigger.isUpdate || Trigger.isDelete) {
            OrderItemTriggerHandler.calculateTotalCost(Trigger.new, Trigger.oldMap);
        }
    }
}
```

3. Trigger Design Pattern

Explanation:
To maintain best practices, all logic was moved from the trigger into a separate handler class (OrderItemTriggerHandler).
This structure is known as the Trigger Handler Pattern, ensuring code reusability, clarity, and easier debugging.

Benefits:

- Clean separation of logic and trigger events
- Reusability for multiple triggers
- Better scalability and testing

4. SOQL & SOSL

Explanation:
SOQL (Salesforce Object Query Language) is used to fetch records from the database.
SOSL (Salesforce Object Search Language) is used for text-based searching across multiple objects.

Use Case 1 (SOQL): Get Total Order Items for a Supplier

```
List<Purchase_Order__c> orders = [
   SELECT Id, Name, (SELECT Id, Product__c FROM Order_Items__r)
   FROM Purchase_Order__c
   WHERE Supplier__r.Name = 'Medico Plus'
];
```

Use Case 2 (SOSL): Search for a Product by Name

```
List<List<sObject>> searchResults = [
   FIND 'Paracetamol*' IN ALL FIELDS
   RETURNING Product__c (Id, Name, Category__c)
];
```

5. Collections (List, Set, Map)

Explanation:
Collections store and manipulate multiple records efficiently.

Use Case:

In the OrderItemTriggerHandler,

- List is used to hold Purchase Orders for update.
- Set is used to store unique Order IDs.
- Map is used to manage old and new record versions during updates.

6. Control Statements

Explanation:
Conditional and looping statements (IF, FOR, etc.) were used to process order items dynamically.

Example:

```
for (Order_Item__c oi : Trigger.new) {
    if (oi.Quantity__c > 0) {
        System.debug('Valid Order Item: ' + oi.Name);
    }
}
```

7. Asynchronous Processing (Batch/Queueable Apex)

Explanation:
For bulk record updates or long-running jobs, asynchronous Apex is used.
In this project, a Queueable Apex class was created to recalculate all purchase order totals in bulk when new data is imported.

Code Example:

```
public class RecalculateAllOrdersQueueable implements Queueable {
    public void execute(QueueableContext context) {
        List<Purchase_Order__c> orders = [SELECT Id FROM Purchase_Order__c];
        for (Purchase_Order__c order : orders) {
            Decimal total = 0;
            for (Order_Item__c oi : [
                SELECT Total_Price__c FROM Order_Item__c WHERE
Purchase_Order__c = :order.Id
            ]) {
                total += oi.Total_Price__c;
            }
            order.Total_Order_Cost__c = total;
        }
        update orders;
    }
```

8. Exception Handling

Explanation:
Exception handling ensures the code doesn’t break during unexpected errors.
All trigger logic is wrapped in try-catch blocks to handle runtime exceptions safely.

Code Example:

```
try {
   update ordersToUpdate;
} catch (DmlException e) {
   System.debug('Error updating order totals: ' + e.getMessage());
}
```

9. Test Classes

Explanation:
Test classes ensure Apex code works correctly and meets Salesforce deployment requirements.
They validate that logic executes as expected without real data.

Code Example:

```
@isTest
public class OrderItemTriggerTest {
   @isTest static void testOrderTotalCalculation() {
      Supplier__c s = new Supplier__c(Name='Test Supplier');
      insert s;

      Product__c p = new Product__c(Name='Paracetamol', Unit_Price__c=50);
      insert p;

      Purchase_Order__c po = new Purchase_Order__c(Supplier__c = s.Id);
      insert po;
```

```
        Order_Item__c oi = new Order_Item__c(
            Product__c = p.Id,
            Purchase_Order__c = po.Id,
            Quantity__c = 10,
            Unit_Price__c = 50,
            Total_Price__c = 500
        );
        insert oi;

        Test.startTest();
        update oi;
        Test.stopTest();

        Purchase_Order__c updatedOrder = [SELECT Total_Order_Cost__c FROM
Purchase_Order__c WHERE Id = :po.Id];
        System.assertEquals(500, updatedOrder.Total_Order_Cost__c);
```