

# PHASE 5 – Apex Programming

## *Advanced Server-Side Logic for Retail Operations*

### **1. Introduction**

Phase 5 focuses on the Apex programming customizations required to support HandsMenThreads' advanced retail workflows. While Salesforce Flows and declarative tools automate a wide range of processes, certain business rules demanded more precision, control, and transactional consistency than point-and-click configuration could provide. Apex triggers, classes, and test methods were implemented to meet these requirements.

For a retail brand like HandsMenThreads, order processing, inventory control, and loyalty calculations must be both precise and reliable. Apex ensures that multi-step operations occur atomically, preventing data inconsistencies such as incorrect stock levels or loyalty miscalculations. Apex also handles complex calculations, provides scalability, and creates a framework for future integration features.

This phase describes the Apex components created, the justification behind each, the technical design considerations, and the testing performed to guarantee smooth operation.

### **2. Apex Trigger for Order Total Calculation**

#### **Business Need**

Orders may consist of multiple products, each with different prices and quantities. Salesforce must automatically compute the total cost of an order without relying on manual input from sales executives. Any modification in Order Items must instantly update the Order Total.

#### **Technical Approach**

A trigger on the Order\_Item\_\_c object was developed using **after insert**, **after update**, and **after delete** events to ensure recalculation occurs whenever line items change.

#### **How It Works**

1. Collect all Order Item records related to an Order.
2. Sum the Line\_Total\_\_c field.
3. Use a SOQL aggregate query to optimize performance.
4. Update the Order\_\_c.Order\_Total\_\_c field.

#### **Key Considerations**

- Bulkification ensures the trigger processes multiple records efficiently.
- Try-catch blocks handle exceptions like missing product data.
- The logic supports future enhancements like discounts and taxation.

### **3. Apex Trigger for Stock Deduction**

#### **Business Need**

Inventory accuracy is the backbone of any retail operation. Manual stock updates lead to overselling, customer dissatisfaction, and increased operational overhead. Salesforce must deduct stock automatically when an Order Item is created and return stock if an Order Item is removed or modified.

#### **Technical Approach**

A before-insert and before-update trigger was implemented to adjust stock levels. Additionally, a before-delete handler restores stock when items are removed.

#### **Stock Deduction Logic**

1. Fetch the Product\_\_c record using the SKU or Product ID.
2. Check if the requested quantity exceeds the available stock.
3. If yes → trigger an error preventing the save operation.
4. If no → deduct the exact quantity from the stock.
5. Update the Product\_\_c.Stock\_\_c field.

#### **Concurrency Handling**

Using **SELECT FOR UPDATE**, the trigger prevents race conditions during simultaneous ordering, ensuring the stock is updated sequentially and consistently.

#### **Low-Stock Integration**

If the remaining stock falls below Reorder\_Level\_\_c, Apex can optionally:

- Call a Flow via Flow.Interview.run().
- Send a platform event for real-time alerts.
- Log the event in a custom Low\_Stock\_Log\_\_c object.

### **4. Apex for Loyalty Points Computation**

#### **Business Need**

HandsMenThreads uses a loyalty program to incentivize repeat purchases. Customers earn points based on order total. Once they reach thresholds, their loyalty tier automatically upgrades.

#### **Loyalty Tiers**

- **Silver:** 500 points
- **Gold:** 1000 points
- **Platinum:** 2000 points

## **How the Apex Logic Works**

Triggered when an order's status becomes "Delivered":

1. Retrieve Order\_Total\_\_c.
2. Multiply by **5%** to calculate loyalty points.
3. Add the points to the Customer's existing balance.
4. Evaluate tier upgrade conditions.
5. Update Customer record with:
  - o New loyalty tier
  - o Tier upgrade date
  - o Total points

## **Error Handling & Edge Cases**

- If order total is null → skip calculation.
- Prevent double-counting by checking if points were already awarded.
- Ensure tier downgrade never happens unless explicitly designed later.

## **5. Apex Classes for Modular Code**

To adhere to Salesforce best practices, logic was separated into handler classes:

### **OrderHandler Class**

- Manages recalculations
- Reusable for future batch jobs

### **StockHandler Class**

- Performs stock checks
- Implements concurrency safeguards

### **LoyaltyHandler Class**

- Calculates points
- Handles tier logic

### **Utility Classes**

- Constants
- Common queries
- Email sender methods

The modular design enhances testability, maintainability, and scalability.

## **6. Apex Test Classes**

Salesforce requires **75% minimum coverage**, but HandsMenThreads achieved **85–90% coverage**.

### **Test Scenarios Covered**

- Creating customers, products, orders
- Adding and modifying order items
- Stock deductions and restorations
- Loyalty tier upgrades
- Invalid data test cases (zero quantity, missing fields)

### **Mock Data Generation**

Test classes include realistic mock data, ensuring accuracy in production deployments.

## **7. Future Improvements**

- Batch Apex for monthly loyalty reconciliation
- Queueable Apex for asynchronous stock updates
- REST API classes for external integrations

### **Conclusion**

Phase 5 establishes Apex as the powerhouse behind complex retail automation. These customizations ensure accuracy, prevent overselling, and strengthen customer retention. With high-quality test coverage and modularized design, Apex ensures the CRM remains robust and scalable.