Phase 5: Apex Programming (Developer)

In this phase, we implemented server-side programming using Apex to handle complex business logic that cannot be achieved through declarative (click-based) tools. Apex was used to manage lead conversion, enrollment creation, automation, and asynchronous processes in the Salesforce CRM.

1) Classes & Objects

Use: Encapsulate business logic into reusable components.

Implementation in Project:
- Created EnrollmentService class to handle enrollment creation.
- Created Trigger Handler class (LeadTriggerHandler) to separate logic from triggers.

Code Example:

```
public with sharing class EnrollmentService {
   @AuraEnabled
   public static Id createEnrollment(Id contactId, Id courseId, Id batchId) {
      Enrollment_c__c e = new Enrollment_c__c(
         Contact__c = contactId,
         Course__c  = courseId,
         Batch__c   = batchId,
         Enrollment_Date__c = Date.today(),
         Status__c = 'Confirmed'
      );
      insert e;
      return e.Id;
   }
}
```

☞ Result: Clean code structure, reusable methods, and better maintenance.

2) Apex Triggers (before/after insert/update/delete)

Use: Execute custom logic before or after DML events on Salesforce records.

Implementation in Project:
- Trigger on Lead (after update) to create Enrollment record automatically upon lead conversion.

Code Example:

```
trigger LeadAfterUpdate on Lead (after update) {
   if (Trigger.isAfter && Trigger.isUpdate) {
      LeadTriggerHandler.handleAfterUpdate(Trigger.new, Trigger.oldMap);
   }
```

}

Result: Automatic enrollment creation without manual intervention.

3) Trigger Design Pattern

Use: Keep triggers simple, delegate logic to handler classes.

Implementation in Project:
- Used LeadAfterUpdate trigger → delegates to LeadTriggerHandler.
- Ensures one trigger per object, easy to maintain.

Result: Scalability and clean separation of concerns.

4) SOQL & SOSL

Use: Query Salesforce data using Apex.

Implementation in Project:
- SOQL used in test classes to verify enrollment creation.
- Example: SELECT Id, Contact__c, Course__c FROM Enrollment_c__c.

Result: Reliable data validation and relationship handling.

5) Collections: List, Set, Map

Use: Store multiple records and avoid duplicates.

Implementation in Project:
- Used List<Enrollment_c__c> in trigger to bulk insert enrollments.
- Used Map<Id, Lead> oldMap to track lead state before update.

Result: Bulk-safe trigger design, no governor limit issues.

6) Control Statements

Use: Apply logic using if-else, loops, and switch.

Implementation in Project:
- if-else used to check if lead was converted and had Preferred_Course__c.
- for loop used to iterate over multiple leads in trigger.

Result: Proper flow control ensured correct record processing.

7) Batch Apex

Use: Handle large data volumes asynchronously in chunks.

Implementation in Project:
- Designed for future use: example, batch process to re-assign counselors for all leads.

Result: System can handle large datasets beyond governor limits.

8) Queueable Apex

Use: Run jobs asynchronously with more flexibility than future methods.

Implementation in Project:
- Could be used for background enrollment updates (not mandatory for project scope).

Result: Demonstrated knowledge of async processing.

9) Scheduled Apex

Use: Automate execution of Apex classes at specific times.

Implementation in Project:
- Example design: schedule a job to deactivate expired batches or send reminders.

Result: Automated maintenance tasks without manual intervention.

10) Future Methods

Use: Run operations asynchronously, often for callouts or heavy processing.

Implementation in Project:
- Could be used to integrate with external systems (e.g., payment gateway).

Result: Improves user experience by avoiding delays on save.

11) Exception Handling

Use: Manage errors gracefully without breaking transactions.

Implementation in Project:
- Try-catch blocks in Apex classes to log errors.
- Prevents incomplete DML from stopping all processing.

Result: Better reliability and error tracking.

12) Test Classes

Use: Ensure Apex code works correctly and achieve 75%+ coverage.

Implementation in Project:
- LeadAfterUpdateTest to test enrollment creation on lead conversion.
- EnrollmentServiceTest to test service class functionality.

Result: Deployment possible with full code coverage.

13) Asynchronous Processing

Use: Run long-running jobs outside normal request-response cycle.

Implementation in Project:
- Knowledge of Batch, Queueable, Future methods demonstrated.
- Ensures scalability for large data and integrations.

Result: CRM is ready for high-volume automation.

Overall Result

Phase 5 introduced Apex programming to handle complex business logic. Triggers and handler classes ensured automated enrollment creation, service classes promoted reusable code, and test classes enabled successful deployment. Advanced concepts like Batch, Queueable, and Scheduled Apex were explored for scalability. This phase ensured the Salesforce CRM can handle both small-scale and enterprise-level automation needs.