# Phase 5: Apex Programming (Developer) — Event Management System

## Objective

Develop scalable and efficient Apex components to handle complex business logic for the Event Management application — including event registration, data integrity, mass processing, and scheduled automation.

## 1. Classes & Objects

Define reusable Apex classes and methods to encapsulate logic for events, attendees, and related processes.

Examples:

* - EventHandler.cls — Handles business logic for Event creation, updates, and validations.
* - AttendeeManager.cls — Manages attendee registration and waitlist logic.
* - NotificationService.cls — Sends notifications and email alerts through reusable methods.

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

Execute logic automatically when records change.

Examples:

* - EventTrigger: before insert (validate date conflicts), after insert (create default tasks), after update (recalculate attendee count).
* - AttendeeTrigger: after insert (update Event attendee total), before delete (check cancellation policy).

## 3. Trigger Design Pattern

Ensure scalability and avoid recursion or mixed logic by using a handler pattern.

Code:

trigger EventTrigger on Event\_\_c (before insert, after insert, after update) {  
 EventTriggerHandler handler = new EventTriggerHandler();  
 if (Trigger.isBefore && Trigger.isInsert) handler.beforeInsert(Trigger.new);  
 if (Trigger.isAfter && Trigger.isInsert) handler.afterInsert(Trigger.new);  
 if (Trigger.isAfter && Trigger.isUpdate) handler.afterUpdate(Trigger.new, Trigger.oldMap);  
}

## 4. SOQL & SOSL

Query Salesforce data efficiently.

SOQL Example: [SELECT Id, Name FROM Event\_\_c WHERE Status\_\_c = 'Published'];

SOSL Example: FIND 'Tech Summit' IN ALL FIELDS RETURNING Event\_\_c(Id, Name), Attendee\_\_c(Name, Email\_\_c);

## 5. Collections: List, Set, Map

Store and manage queried data efficiently for bulk processing and avoiding duplicates.

Code:

* - List<Attendee\_\_c> attendees = new List<Attendee\_\_c>();
* - Set<Id> eventIds = new Set<Id>();
* - Map<Id, Event\_\_c> eventMap = new Map<Id, Event\_\_c>([SELECT Id, Name FROM Event\_\_c]);

## 6. Control Statements

Implement conditional and iterative logic for Apex classes and triggers.

Code:

if-else, for loops, while loops for decision-based logic.

## 7. Batch Apex

Handle large data volumes such as sending reminders to thousands of attendees or cleaning up old data.

Code:

global class EventReminderBatch implements Database.Batchable<SObject> {...}

## 8. Queueable Apex

Execute asynchronous logic that requires chaining or parameters.

Example: Send confirmation emails after attendee registration.

## 9. Scheduled Apex

Automate recurring operations like nightly cleanup or weekly summary reports.

Code:

global class EventScheduler implements Schedulable {...}

## 10. Future Methods

Run lightweight asynchronous logic like external API callouts after registration.

Code:

@future(callout=true) public static void sendToExternalAPI(Id eventId) {...}

## 11. Exception Handling

Gracefully handle errors and prevent transaction failures.

Example: try-catch blocks, custom exceptions, and logging errors for admin review.

## 12. Test Classes

Ensure all Apex logic is tested with >75% coverage.

Code:

@isTest private class EventTriggerTest {...}

Include positive, negative, and bulk test cases.

## 13. Asynchronous Processing

Offload long-running or heavy logic to asynchronous execution for better performance.

Techniques: Future Methods, Batch Apex, Queueable Apex, Scheduled Apex.