Project Title: Venue Connect CRM

Phase 5 — Apex Programming (Developer)

5.1 Classes & Objects

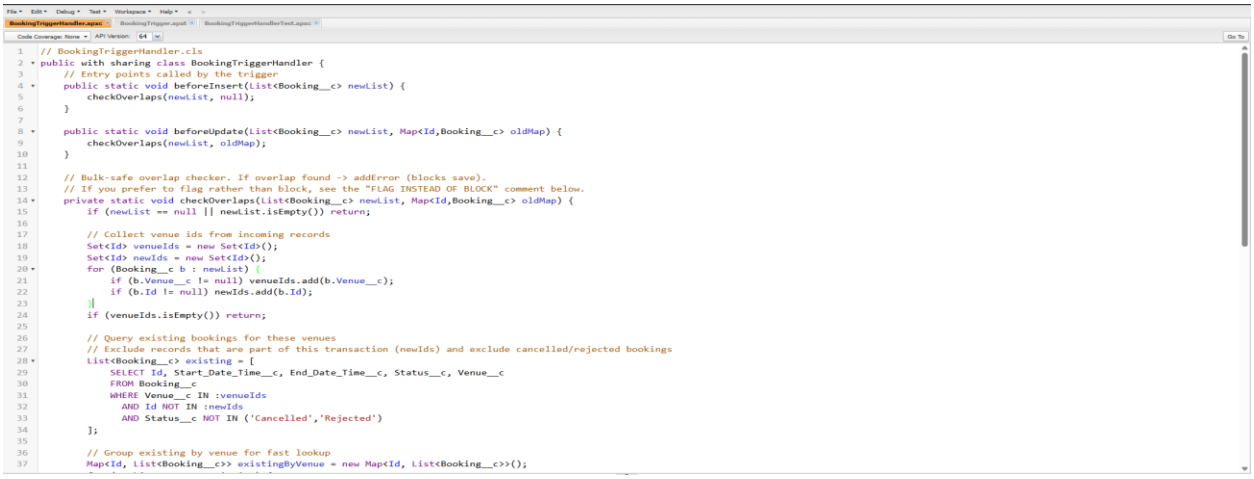
Purpose: Encapsulate business logic and reusable functions.

Examples:

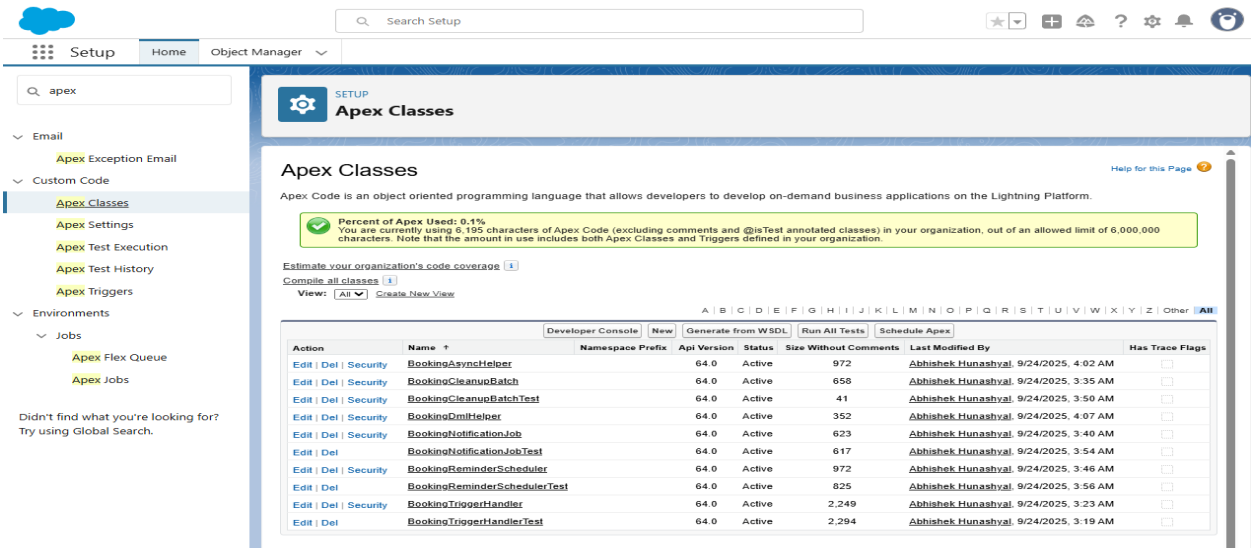
- BookingTriggerHandler → Handles Booking validation, conflict detection, and payment status updates.
- BookingAsyncHelper → Handles asynchronous operations like email notifications.
- BookingDmlHelper → Wraps DML operations with exception handling.

Setup:

- Setup → Apex Classes → New → Write code → Save → Deploy



```apex
// BookingTriggerHandler.cls
public with sharing class BookingTriggerHandler {
    // Entry points called by the trigger
    public static void beforeInsert(List<Booking__c> newList) {
        checkOverlaps(newList, null);
    }

    public static void beforeUpdate(List<Booking__c> newList, Map<Id,Booking__c> oldMap) {
        checkOverlaps(newList, oldMap);
    }

    // Bulk-safe overlap checker. If overlap found -> addError (blocks save).
    // If you prefer to flag rather than block, see the "FLAG INSTEAD OF BLOCK" comment below.
    private static void checkOverlaps(List<Booking__c> newList, Map<Id,Booking__c> oldMap) {
        if (newList == null || newList.isEmpty()) return;

        // Collect venue ids from incoming records
        Set<Id> venueIds = new Set<Id>();
        Set<Id> newIds = new Set<Id>();
        for (Booking__c b : newList) {
            if (b.Venue__c != null) venueIds.add(b.Venue__c);
            if (b.Id != null) newIds.add(b.Id);
        }
        if (venueIds.isEmpty()) return;

        // Query existing bookings for these venues
        // Exclude records that are part of this transaction (newIds) and exclude cancelled/rejected bookings
        List<Booking__c> existing = [
            SELECT Id, Start_Date_Time__c, End_Date_Time__c, Status__c, Venue__c
            FROM Booking__c
            WHERE Venue__c IN :venueIds
              AND Id NOT IN :newIds
              AND Status__c NOT IN ('Cancelled','Rejected')
        ];

        // Group existing by venue for fast lookup
        Map<Id, List<Booking__c>> existingByVenue = new Map<Id, List<Booking__c>>();
```



5.2 Apex Triggers

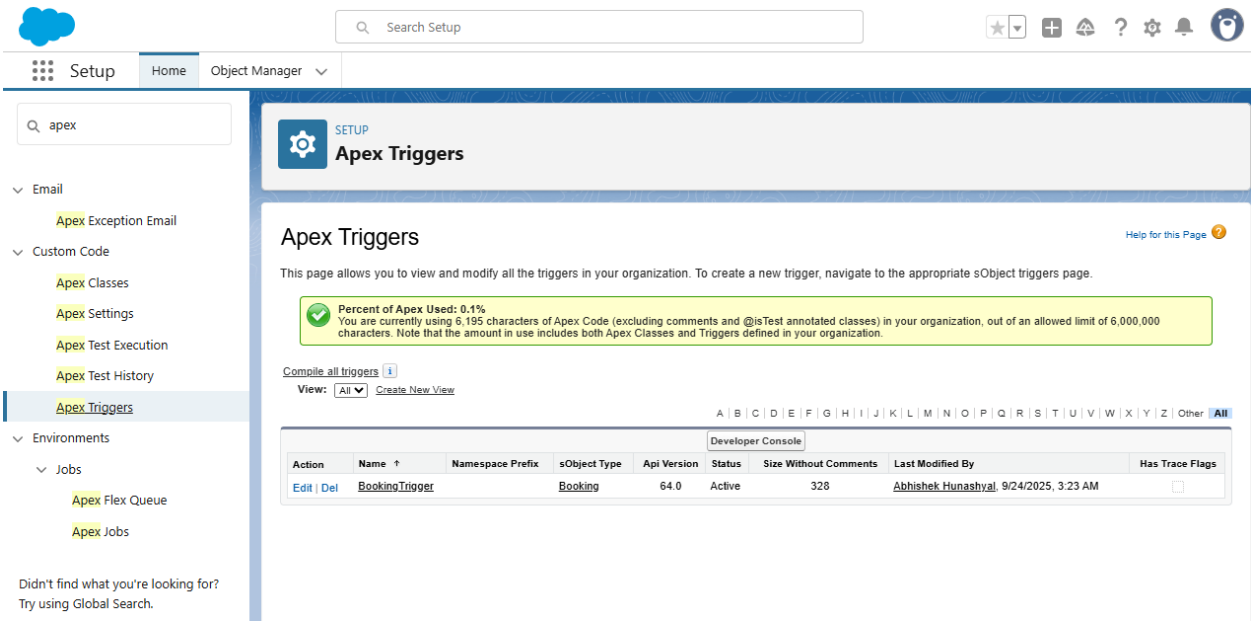
Purpose: Automate logic before/after insert, update, or delete of records.

Example:

- Trigger on Booking__c:
    - Before Insert/Update → Validate End_DateTime__c > Start_DateTime__c
    - After Insert/Update → Check booking conflicts, update payment status, optionally call future method

Setup:

- Setup → Object Manager → Booking → Triggers → New → Attach to Booking__c → Save → Activate



5.3 Trigger Design Pattern

Purpose: Organize triggers using Handler classes to improve readability and maintainability.
Example:

- BookingTrigger → calls BookingTriggerHandler.beforeInsertUpdate() and .afterInsertUpdate().
- Handler class contains bulkified methods to handle multiple records efficiently.

```
// BookingTrigger.trigger
trigger BookingTrigger on Booking__c (before insert, before update) {
    if (Trigger.isBefore) {
        if (Trigger.isInsert) {
            BookingTriggerHandler.beforeInsert(Trigger.new);
        }
        if (Trigger.isUpdate) {
            BookingTriggerHandler.beforeUpdate(Trigger.new, Trigger.oldMap);
        }
    }
}
```

5.4 SOQL & SOSL

Purpose: Query Salesforce database efficiently.
Example SOQL Queries:

Booking__c b = [SELECT Id, Name, Booking_Amount__c FROM Booking__c WHERE Id = :bookingId LIMIT 1];

List<Booking_Payment__c> payments = [SELECT Id, Amount__c FROM Booking_Payment__c WHERE Booking__c = :bookingId];

Example SOSL Query:

List<List<SObject>> searchResults = [FIND 'Ravi*' IN ALL FIELDS RETURNING Contact(Id, Name)];

```
public class BookingNotificationJob implements Queueable {
    public Id bookingId;

    public BookingNotificationJob(Id bookingId) {
        this.bookingId = bookingId;
    }

    public void execute(QueueableContext qc) {
        Booking__c b = [
            SELECT Id, Name, OwnerId
            FROM Booking__c
            WHERE Id = :bookingId
            LIMIT 1
        ];
        Task t = new Task(
            Subject = 'Review booking ' + b.Name,
            WhatId  = b.Id,
            OwnerId = b.OwnerId,
            Status  = 'Not Started',
            Priority = 'High'
        );
        insert t;
    }
}
```

5.5 Collections: List, Set, Map

Purpose: Store and process multiple records efficiently.
Example:

List<Booking__c> bookings = new List<Booking__c>();

Set<Id> bookingIds = new Set<Id>();

Map<Id, Booking__c> bookingMap = new Map<Id, Booking__c>();

```apex
// BookingTriggerHandler.cls
public with sharing class BookingTriggerHandler {
    // Entry points called by the trigger
    public static void beforeInsert(List<Booking__c> newList) {
        checkOverlaps(newList, null);
    }

    public static void beforeUpdate(List<Booking__c> newList, Map<Id,Booking__c> oldMap) {
        checkOverlaps(newList, oldMap);
    }

    // Bulk-safe overlap checker. If overlap found -> addError (blocks save).
    // If you prefer to flag rather than block, see the "FLAG INSTEAD OF BLOCK" comment below.
    private static void checkOverlaps(List<Booking__c> newList, Map<Id,Booking__c> oldMap) {
        if (newList == null || newList.isEmpty()) return;

        // Collect venue ids from incoming records
        Set<Id> venueIds = new Set<Id>();
        Set<Id> newIds = new Set<Id>();
        for (Booking__c b : newList) {
            if (b.Venue__c != null) venueIds.add(b.Venue__c);
            if (b.Id != null) newIds.add(b.Id);
        }
        if (venueIds.isEmpty()) return;

        // Query existing bookings for these venues
        // Exclude records that are part of this transaction (newIds) and exclude cancelled/rejected bookings
        List<Booking__c> existing = [
            SELECT Id, Start_Date_Time__c, End_Date_Time__c, Status__c, Venue__c
            FROM Booking__c
            WHERE Venue__c IN :venueIds
              AND Id NOT IN :newIds
              AND Status__c NOT IN ('Cancelled','Rejected')
        ];

        // Group existing by venue for fast lookup
        Map<Id, List<Booking__c>> existingByVenue = new Map<Id, List<Booking__c>>();
        for (Booking__c ex : existing) {
            if (!existingByVenue.containsKey(ex.Venue__c)) existingByVenue.put(ex.Venue__c, new List<Booking__c>());
            existingByVenue.get(ex.Venue__c).add(ex);
        }
```

5.6 Control Statements

Purpose: Handle decision-making and loops in Apex.
Examples:

if(booking.Status__c == 'Confirmed'){ ... }

for(Booking__c b : bookings){ ... }

while(condition){ ... }

switch on status__c { ... }

5.7 Batch Apex

Purpose: Process large volumes of records asynchronously.
Example: Update payment status for all bookings:

- Class implements Database.Batchable<SObject>
- Execute batch:

```
Database.executeBatch(new BookingPaymentBatch(), 200);
```

5.8 Queueable Apex

Purpose: Run asynchronous operations with complex logic.
Example: Send booking notifications:

- Class implements Queueable
- Enqueue job:

```
System.enqueueJob(new BookingQueueable(bookingId));
```

5.9 Scheduled Apex

Purpose: Run Apex classes at specific times.
Example: Daily conflict check for Bookings:

```
System.schedule('Daily Booking Conflict Check', '0 0 1 * * ?', new DailyBookingCheck());
```

5.10 Future Methods

Purpose: Run lightweight asynchronous operations.
Example:

```
BookingAsyncHelper.notifyBookingConfirmed(bookingId);
```

5.11 Exception Handling

Purpose: Prevent runtime errors from stopping processes.
Example: Wrap DML in try/catch:

try {

update booking;

} catch(DmlException e){

System.debug('Error updating booking: ' + e.getMessage());

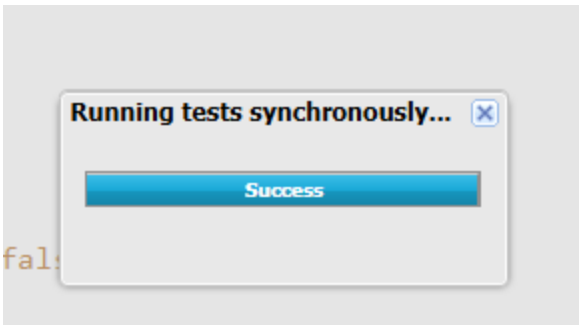
}

```
public class BookingAsyncHelper {

    // Sends a notification or logs asynchronously when booking is confirmed
    @future
    public static void notifyBookingConfirmed(Id bookingId){
        try {
            Booking__c b = [SELECT Id, Name, Customer__c FROM Booking__c WHERE Id = :bookingId LIMIT 1];

            // Example: Log a message (can be replaced with real email or notification logic)
            System.debug('Booking confirmed asynchronously: ' + b.Name + ' for Customer Id: ' + b.Customer__c);

            // Optional: send email to Venue Manager
            User venueManager = [SELECT Id, Email FROM User WHERE Profile.Name = 'Venue Manager' LIMIT 1];
            Messaging.SingleEmailMessage mail = new Messaging.SingleEmailMessage();
            mail.setToAddresses(new String[]{venueManager.Email});
            mail.setSubject('Booking Confirmed: ' + b.Name);
            mail.setPlainTextBody('Booking ' + b.Name + ' has been confirmed. Please review if needed.');
            Messaging.sendEmail(new Messaging.SingleEmailMessage[]{mail});

        } catch(Exception e){
            System.debug('Error in async notifyBookingConfirmed: ' + e.getMessage());
        }
    }
}
```

5.12 Test Classes

Purpose: Validate logic, ensure code coverage, and make deployments possible.
Example:

- Create sample Venue, Contact, Booking, Booking Payment
- Insert records → Verify conflict detection, payment status, and notifications
- Use System.assertEquals() to validate expected outcomes



5.13 Asynchronous Processing

Purpose: Improve performance for operations that can run later.
Types:

- **Future Methods** → Lightweight async tasks
- **Queueable Apex** → Chainable async tasks
- **Batch Apex** → Large volume data processing
- **Scheduled Apex** → Run at specific intervals