Phase 5: Apex Programming (Developer)

Project Title – "Finance Management System – Salesforce CRM"

Classes & Objects

Purpose of the Classes

- **ContributionHandler**: Enforces validation and automates follow-up tasks when contributions are created/updated.

Code Coverage: None | API Version: 64

```apex
public class ContributionHandler {
    public static void beforeInsert(List<Contribution__c> newCons) {
        for(Contribution__c con : newCons){
            if(con.Amount__c == null || con.Amount__c <= 0){
                con.addError('Amount must be greater than 0');
            }
        }
    }

    public static void beforeUpdate(List<Contribution__c> newCons, Map<Id, Contribution__c> oldMap) {
        for(Contribution__c con : newCons){
            Contribution__c oldRec = oldMap.get(con.Id);
            if(oldRec != null && oldRec.Status__c == 'Paid' && con.Status__c != 'Paid'){
                con.addError('Cannot change status once marked as Paid');
            }
        }
    }

    public static void afterInsert(List<Contribution__c> newCons) {
        List<Task> tasks = new List<Task>();
        for(Contribution__c con : newCons){
            Task t = new Task(
                Subject = 'Follow up with member',
                WhatId = con.Id,
                ActivityDate = Date.today().addDays(2)
```

```apex
    public static void afterInsert(List<Contribution__c> newCons) {
        List<Task> tasks = new List<Task>();
        for(Contribution__c con : newCons){
            Task t = new Task(
                Subject = 'Follow up with member',
                WhatId = con.Id,
                ActivityDate = Date.today().addDays(2)
            );
            tasks.add(t);
        }
        if(!tasks.isEmpty()){
            insert tasks;
        }
    }
}
```

- **AuctionHandler**: Ensures that an auction cannot be closed without selecting a winner.

```
Code Coverage: None  API Version: 64
1  public class AuctionHandler {
2      public static void beforeUpdate(List<Auction__c> newList, Map<Id, Auction__c> oldMap) {
3          for(Auction__c auc : newList){
4              Auction__c oldRec = oldMap.get(auc.Id);
5              if(auc.Status__c == 'Closed' && String.isBlank(auc.Winner__c)){
6                  auc.addError('Winner must be selected before closing an auction');
7              }
8          }
9      }
10 }
11
```

- **NotificationService**: Provides asynchronous reminders using future methods.

```
Code Coverage: None  API Version: 64
1  public class NotificationService {
2      @future
3      public static void sendReminder(Id memberId, String message){
4          // In real integration: perform callout to SMS/Email provider
5          System.debug('Reminder for Member: ' + memberId + ' -> ' + message);
6      }
7  }
8
```

- **OverdueBatch**: Marks pending contributions as overdue when their due date has passed.

```
Code Coverage: None  API Version: 64
1  global class OverdueBatch implements Database.Batchable<sObject> {
2      global Database.QueryLocator start(Database.BatchableContext bc) {
3          return Database.getQueryLocator(
4              [SELECT Id, Status__c, Due_Date__c FROM Contribution__c WHERE Status__c = 'Pending' AND Due_Date__c < :Date.today()]
5          );
6      }
7      global void execute(Database.BatchableContext bc, List<Contribution__c> scope) {
8          for(Contribution__c con : scope){
9              con.Status__c = 'Overdue';
10         }
11         if(!scope.isEmpty()){
12             update scope;
13         }
14     }
15     global void finish(Database.BatchableContext bc) {
16         System.debug('OverdueBatch finished');
17     }
18 }
19
```

Business Logic Implemented

- Contributions with Amount__c <= 0 are blocked.
- Contributions once marked as **Paid** cannot be changed back.
- A Task is auto-created when a new Contribution is inserted.
- Auctions require a Winner before being closed.
- Notifications can be sent asynchronously.

- Batch Apex updates overdue Contributions daily in bulk.

Trigger Design Pattern

Purpose

- Centralize trigger logic inside handler classes.
- Keep triggers lightweight and bulk-safe.

Implemented Triggers

1. **ContributionTrigger** → Before Insert, Before Update, After Insert.
2. **AuctionTrigger** → Before Update.

Business Logic

- **ContributionTrigger** validates Amount, prevents invalid status changes, and creates follow-up tasks.
- **AuctionTrigger** validates that Winner must be filled when status is Closed.

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

- **Before Insert/Update**: Validates Contribution data.
- **After Insert**: Creates Task records linked to new Contributions.
- **Before Update (Auction)**: Prevents closing auction without winner.

Code Coverage: None | API Version: 64

```
trigger AuctionTrigger on Auction__c (before update) {
    if(Trigger.isBefore && Trigger.isUpdate){
        AuctionHandler.beforeUpdate(Trigger.new, Trigger.oldMap);
    }
}
```

Code Coverage: None | API Version: 64 | Go To

```
trigger ContributionTrigger on Contribution__c (before insert, before update, after insert) {
    if(Trigger.isBefore && Trigger.isInsert){
        ContributionHandler.beforeInsert(Trigger.new);
    }
    if(Trigger.isBefore && Trigger.isUpdate){
        ContributionHandler.beforeUpdate(Trigger.new, Trigger.oldMap);
    }
    if(Trigger.isAfter && Trigger.isInsert){
        ContributionHandler.afterInsert(Trigger.new);
    }
}
```

SOQL & SOSL

SOQL Example

- Query used in **OverdueBatch**:
- SELECT Id, Status__c, Due_Date__c
- FROM Contribution__c
- WHERE Status__c = 'Pending' AND Due_Date__c < TODAY
- Purpose: Finds all pending contributions past their due date.

SOSL

- Not implemented in this project (out of scope).
- Could be used in future to search across Members & Contributions by KYC ID or Name.

```
Code Coverage: None  API Version: 64                                                                    Go To
1  global class OverdueBatch implements Database.Batchable<sObject> {
2      global Database.QueryLocator start(Database.BatchableContext bc) {
3          return Database.getQueryLocator(
4              [SELECT Id, Status__c, Due_Date__c FROM Contribution__c WHERE Status__c = 'Pending' AND Due_Date__c < :Date.today()]
5          );
6      }
7      global void execute(Database.BatchableContext bc, List<Contribution__c> scope) {
8          for(Contribution__c con : scope){
9              con.Status__c = 'Overdue';
10         }
11         if(!scope.isEmpty()){
12             update scope;
13         }
14     }
15     global void finish(Database.BatchableContext bc) {
16         System.debug('OverdueBatch finished');
17     }
18 }
```

Collections: List, Set, Map

- **List**: Used for processing multiple Contribution__c and Auction__c records.
- **Map**: Used in triggers (Trigger.oldMap) to compare old and new values.
- **Set**: Can be used in future for storing unique Member IDs to avoid duplicate processing.

Control Statements

- **For Loops**: Process bulk Contributions and Auctions.
- **If-Else Conditions**: Validate Amount, Status, Auction closure rules.
- Ensures code handles bulk inserts/updates correctly.

Asynchronous Processing

Batch Apex – OverdueBatch

- Runs across all Contributions.
- Updates Status__c = 'Overdue' for expired contributions.
- Can handle thousands of records safely.

Future Methods – NotificationService

- Used to send async reminders to Members.
- Example: NotificationService.sendReminder(memberId, 'Payment Reminder').

Scheduled Apex

- Not implemented, but OverdueBatch could be scheduled daily in production.

Exception Handling

- Basic error handling with addError() in triggers.
- Try-catch in test methods to validate expected errors.
- System.debug logs used to trace batch execution.

Test Classes

- **ContributionHandlerTest** covers Contribution Trigger and Handler logic.
- **AuctionHandlerTest** validates Auction closing logic.
- Tests ensure validation errors are raised correctly.
- Achieved **78% overall coverage** (above Salesforce 75% requirement).

Overall Code Coverage

Class	Percent	Lines
Overall	**78%**	
AuctionHandler	100%	5/5
AuctionTrigger	100%	2/2
ContributionHandler	100%	19/19
ContributionTrigger	100%	6/6
NotificationService	100%	1/1
OverdueBatch	0%	0/9