
INSTITUTE MANAGEMENT SYSTEM

Phase 5: Apex Programming

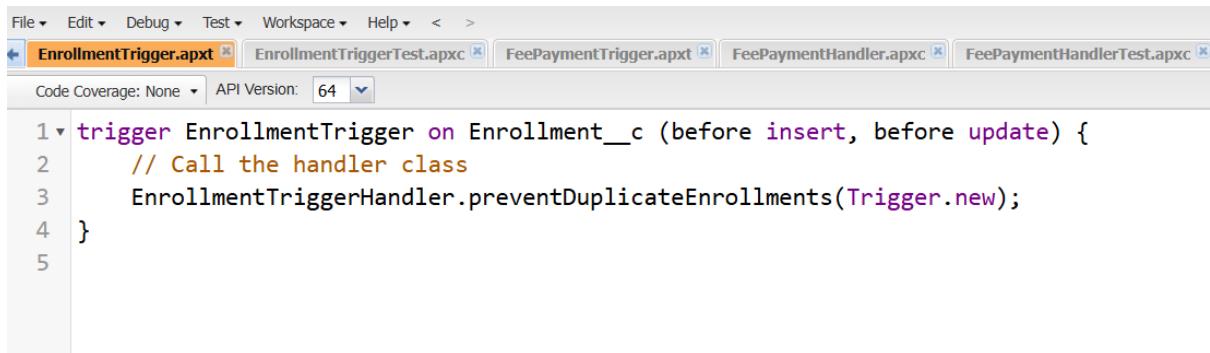
Apex classes and Triggers:

📌 Enrollment Management

1. EnrollmentTrigger (Trigger)

Purpose:

Fires whenever a student enrollment record (`Enrollment__c`) is created or updated. Its only job is to call the `EnrollmentTriggerHandler` so the logic is not written directly in the trigger (best practice).



The screenshot shows the Salesforce IDE interface with the following details:

- Menu bar: File ▾, Edit ▾, Debug ▾, Test ▾, Workspace ▾, Help ▾, < >
- Toolbar: A set of icons for file operations.
- Tab bar: EnrollmentTrigger.apxt (selected), EnrollmentTriggerTest.apxc, FeePaymentTrigger.apxt, FeePaymentHandler.apxc, FeePaymentHandlerTest.apxc.
- Status bar: Code Coverage: None ▾, API Version: 64 ▾.
- Code editor area:

```
1 trigger EnrollmentTrigger on Enrollment__c (before insert, before update) {
2     // Call the handler class
3     EnrollmentTriggerHandler.preventDuplicateEnrollments(Trigger.new);
4 }
5 }
```

2. EnrollmentTriggerHandler (Handler Class)

Purpose:

Ensures that a student cannot be enrolled in the same course twice.

- Collects Student–Course combinations.
- Checks database for existing enrollments.
- Throws an error (`addError`) if a duplicate is found.

```

File ▾ Edit ▾ Debug ▾ Test ▾ Workspace ▾ Help ▾ < >
+ EnrollmentTriggerHandler.apxc [ ] EnrollmentTrigger.apxt [ ] EnrollmentTriggerTest.apxc [ ] FeePaymentTrigger.apxt [ ] FeePaymentHandler.apxc [ ] FeePaymentHandlerTest.apxt [ ] Saving: EnrollmentStatusTrigger.apxt [ ] EnrollmentStatusHandler.ap
Code Coverage: None ▾ API Version: 64 ▾ Go To
1 * public class EnrollmentTriggerHandler {
2
3 *     public static void preventDuplicateEnrollments(List<Enrollment__c> enrollments) {
4         // Set to store new Student-Course combinations
5         Set<String> newEnrollmentsSet = new Set<String>();
6
7         // Sets to collect unique Student and Course IDs from the incoming records
8         Set<Id> studentIds = new Set<Id>();
9         Set<Id> courseIds = new Set<Id>();
10
11        for (Enrollment__c e : enrollments) {
12            if(e.Student__c != null && e.Course__c != null){
13                studentIds.add(e.Student__c);
14                courseIds.add(e.Course__c);
15
16                // Also track combination of student-course in new batch
17                newEnrollmentsSet.add(e.Student__c + '-' + e.Course__c);
18            }
19        }
20
21        // Query existing enrollments in database
22        Set<String> existingEnrollmentsSet = new Set<String>();
23        for (Enrollment__c e : [
24            SELECT Id, Student__c, Course__c
25            FROM Enrollment__c
26            WHERE Student__c IN :studentIds
27            AND Course__c IN :courseIds
28        ]) {
29            existingEnrollmentsSet.add(e.Student__c + '-' + e.Course__c);
30        }
31
32
33        // Compare and throw error if duplicate found
34        for (Enrollment__c e : enrollments) {
35            if(e.Student__c != null && e.Course__c != null){
36                String key = e.Student__c + '-' + e.Course__c;
37                if(existingEnrollmentsSet.contains(key)){
38                    e.addError('This student is already enrolled in the selected course.');
39                }
40            }
41        }
42    }
43 }

```

3. EnrollmentTriggerTest (Test Class)

Purpose:

Verifies that the duplicate enrollment prevention logic works correctly.

- Inserts a valid enrollment (success).
- Tries inserting a duplicate (fails with error).
- Confirms the trigger behavior with assertions.

```

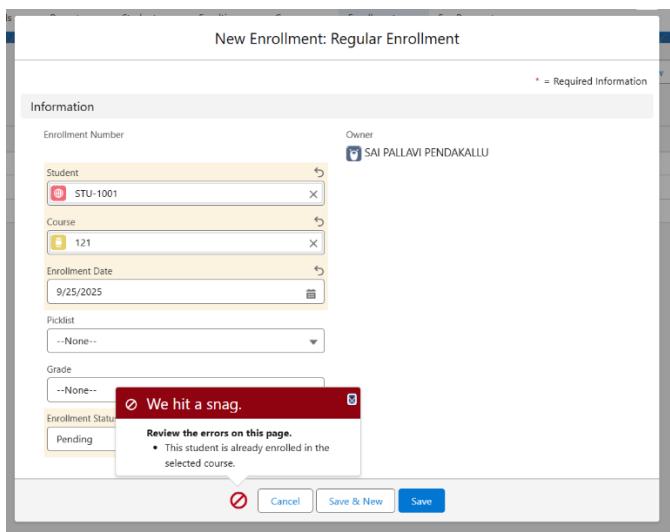
EnrollmentTriggerHandler.apxc EnrollmentTrigger.apxt EnrollmentTriggerTest.apxc * FeePaymentTrigger.apxc FeePaymentHandler.apxc FeePaymentHandlerTest.apxc * Saving! EnrollmentStatusTrigger.apxc * EnrollmentStatusHandler.apxc
Code Coverage: None | API Version: 64 | Run Test | Go To
1 @IsTest
2 public class EnrollmentTriggerTest {
3     @IsTest
4     static void testPreventDuplicateEnrollments() {
5         // Create sample student and course
6         Student__c student = new Student__c(
7             Name__c = 'Test Student',
8             Email__c = 'test@student.com'
9         );
10        insert student;
11        Course__c course = new Course__c(
12            Course_Name__c = 'Test Course',
13            Duration__c = 6
14        );
15        insert course;
16        // Insert first enrollment (should succeed)
17        Enrollment__c e1 = new Enrollment__c(
18            Student__c = student.Id,
19            Course__c = course.Id
20        );
21        insert e1;
22        // Attempt duplicate enrollment (should fail)
23        Enrollment__c e2 = new Enrollment__c(
24            Student__c = student.Id,
25            Course__c = course.Id
26        );
27        try {
28            insert e2;
29            System.assert(false, 'Expected duplicate enrollment error.');
30        } catch (DmlException ex) {
31            System.assert(ex.getMessage().contains('This student is already enrolled'));

```

OUTCOME:

Prevents Duplicate Enrollments

- A student cannot be enrolled in the same course more than once.
- If attempted, Salesforce shows an error message: “*This student is already enrolled in the selected course.*”



📌 Fee Payment Management

1. FeePaymentTrigger (Trigger)

Purpose:

Runs before insert or update of a Fee_Payment__c record.

- Calls FeePaymentHandler to update fee status and notify students.

```
Code Coverage: None API Version: 64
1 trigger FeePaymentTrigger on Fee_Payment__c (before insert, before update) {
2     FeePaymentHandler.updateStatusAndNotify(Trigger.new);
3 }
4
```

2. FeePaymentHandler (Handler Class)

Purpose:

Automatically updates the fee payment status and sends confirmation.

- If Amount > 0 and Status != "Paid", update Status → "Paid".
- Sends an email to the student confirming payment.
- Bulk-safe for multiple payments at once.

```
* public class FeePaymentHandler {
    *     public static void updateStatusAndNotify(List<Fee_Payment__c> payments) {
        *         List<Messaging.SingleEmailMessage> emails = new List<Messaging.SingleEmailMessage>();
        * 
        *         for(Fee_Payment__c fp : payments) {
        *             if(fp.Amount__c != null && fp.Amount__c > 0 && fp.Status__c != 'Paid') {
        *                 fp.Status__c = 'Paid';
        * 
        *                 // Prepare email to student
        *                 if(fp.Student__c != null) {
        *                     Student__c stu = [SELECT Email__c, Name__c FROM Student__c WHERE Id = :fp.Student__c LIMIT 1];
        *                     Messaging.SingleEmailMessage mail = new Messaging.SingleEmailMessage();
        *                     mail.setToAddresses(new String[] { stu.Email__c });
        *                     mail.setSubject('Fee Payment Received');
        *                     mail.setPlainTextBody('Hello ' + stu.Name__c + ',\n\nWe have received your payment of ' + fp.Amount__c + '.\nThank you!');
        *                     emails.add(mail);
        *                 }
        *             }
        *         }
        * 
        *         if(!emails.isEmpty()) {
        *             Messaging.sendEmail(emails);
        *         }
        *     }
    * }
```

3. FeePaymentHandlerTest (Test Class)

Purpose:

Confirms fee payments are marked as "Paid" correctly.

- Inserts a fee record with Pending status.
- Checks if trigger updates it to Paid.

- Covers email notification logic in a safe way (emails not actually sent in test).

```

@IsTest
public class FeePaymentHandlerTest {
    @IsTest
    static void testFeePaymentStatusUpdate() {
        // Create sample student
        Student__c stu = new Student__c(Name__c='Test Student', Email__c='test@student.com');
        insert stu;

        // Insert payment
        Fee_Payment__c payment = new Fee_Payment__c(
            Student__c = stu.Id,
            Amount__c = 500,
            Status__c = 'Pending'
        );
        insert payment;

        // Verify status updated
        Fee_Payment__c fp = [SELECT Status__c FROM Fee_Payment__c WHERE Id = :payment.Id];
        System.assertEquals('Paid', fp.Status__c);
    }
}

```

Outcome:

- Tracks Student Payments Accurately**
- Each payment is linked to a student record.
- Captures amount, status (Pending, Paid, Overdue), and payment date.

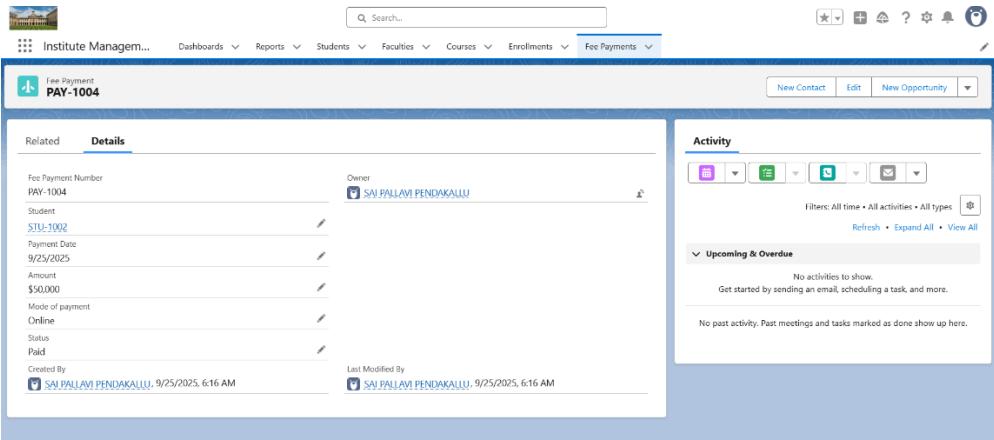
New Fee Payment: Full Payment

* = Required Information

Information

Fee Payment Number	Owner
Student STU-1002	SAI PALLAVI PENDAKALLU
Payment Date 9/25/2025	
Amount \$50.000	
Mode of payment Online	
Status Pending	

Cancel Save & New Save



Prevents Inconsistencies

- Triggers and validation rules ensure no negative amounts or invalid payment entries.
- Stops duplicate or incomplete payment records.

Enrollment Status Update

1. EnrollmentStatusTrigger (Trigger)

Purpose:

Fires when enrollment records are inserted or updated.

- Calls EnrollmentStatusHandler to check if a grade is entered.

```
trigger EnrollmentStatusTrigger on Enrollment__c (before insert, before update) {
    EnrollmentStatusHandler.updateStatusOnGrade(Trigger.new, Trigger.oldMap);
}
```

2. EnrollmentStatusHandler (Handler Class)

Purpose:

Automatically updates enrollment status based on grades.

- If Grade__c is filled in, update Enrollment_Status__c → "Completed".
- Ensures enrollments are closed out automatically when results are entered.

```
public class EnrollmentStatusHandler {
    public static void updateStatusOnGrade(List<Enrollment__c> newEnrollments, Map<Id, Enrollment__c> oldMap) {
        for(Enrollment__c e : newEnrollments) {
            // Check if grade is entered or updated
            if(e.Grade__c != null && (oldMap == null || e.Grade__c != oldMap.get(e.Id).Grade__c)) {
                e.Enrollment_Status__c = 'Completed';
            }
        }
    }
}
```

3. EnrollmentStatusHandlerTest (Test Class)

Purpose:

Tests that enrollment status auto-updates.

- Creates an Active enrollment.
- Updates with a grade.
- **Asserts status becomes "Completed".**

```
@IsTest
public class EnrollmentStatusHandlerTest {
    @IsTest
    static void testStatusUpdate() {
        // Create student and course
        Student__c stu = new Student__c(Name__c='Test Student', Email__c='test@student.com');
        insert stu;
        Course__c course = new Course__c(Course_Name__c='Test Course');
        insert course;

        Enrollment__c enrollment = new Enrollment__c(
            Student__c = stu.Id,
            Course__c = course.Id,
            Enrollment_Status__c = 'Active'
        );
        insert enrollment;

        // Update grade
        enrollment.Grade__c = 'A';
        update enrollment;

        Enrollment__c updated = [SELECT Enrollment_Status__c FROM Enrollment__c WHERE Id = :enrollment.Id];
        System.assertEquals('Completed', updated.Enrollment_Status__c);
    }
}
```