

Student Attendance & Performance Tracker

PHASE 5 – APEX PROGRAMMING (Developer)

Introduction

Phase 5 focuses on implementing **server-side automation using Apex programming**.

Apex is used to handle complex business logic that cannot be achieved using declarative tools alone. In this phase, Apex classes, triggers, SOQL queries, collections, and test classes were developed to **automatically track student attendance and update enrollment records accurately**.

1. Classes & Objects

Use Case

Whenever a student is marked **Absent**, the system should automatically increase the **Total Absences** count in the corresponding **Enrollment** record for that specific course.

To achieve this, a dedicated Apex class was created to handle enrollment updates in a reusable and scalable manner.

Implementation

- Created an Apex class named **EnrollmentUpdater**
- The class encapsulates business logic to:
 - Fetch Enrollment records based on **Student + Course**
 - Update or create Enrollment records if required
 - Increment or decrement total absences correctly
- Mandatory fields like **Course_c** are handled to avoid runtime errors

The screenshot shows the Salesforce Apex Classes page. At the top, there's a setup icon and the title 'Apex Classes'. Below the title, it says 'Apex Code is an object oriented programming language that allows developers to develop on-demand business applications on the Lightning Platform.' A green box indicates 'Percent of Apex Used: 0.08%' and 'You are currently using 4,723 characters of Apex Code (excluding comments and @isTest annotated classes) in your organization, out of an allowed limit of 6,000,000 characters. Note that the amount in use includes both Apex C in your organization.' Below this are links for 'Estimate your organization's code coverage' and 'Compile all classes'. A 'View' dropdown is set to 'All'.

Action	Name ↑	Namespace Prefix	Api Version	Status	Size Without Comments	Last Modified By
Edit Del Security	AttendanceController		65.0	Active	375	Siri varshini Pentakota, 18/12/2025, 1:09 pm
Edit Del Security	EnrollmentUpdater		65.0	Active	2,008	Siri varshini Pentakota, 20/12/2025, 8:33 pm
Edit Del	EnrollmentUpdaterTest		65.0	Active	891	Siri varshini Pentakota, 13/12/2025, 1:51 pm

```
List<Enrollment_c> toUpdate = new List<Enrollment_c>();
```

```
for (Id stuId : studentAbsenceIncrements.keySet()) {
    Integer increment = studentAbsenceIncrements.get(stuId);
    if (studentToEnrollment.containsKey(stuId)) {
        Enrollment_c en = studentToEnrollment.get(stuId);
        Integer current = (en.Total_Absences_c == null) ? 0 : Integer.valueOf(en.Total_Absences_c);
        en.Total_Absences_c = current + (increment == null ? 0 : increment);
        toUpdate.add(en);
    } else {
        Enrollment_c newEn = new Enrollment_c(
            Student_c = stuId,
            Course_c = courseId,
            Total_Absences_c = (increment == null ? 0 : increment)
        );
        toUpdate.add(newEn);
    }
}
```

2. Apex Triggers (After Insert & After Update)

Use Case

Whenever an Attendance record is:

- Inserted with status **Absent**
- Updated from **Present** → **Absent** or **Absent** → **Present**

The system should automatically recalculate the total absences in Enrollment records.

Implementation

- Created **AttendanceTrigger** on Attendance_c object
- Trigger executes **after insert and after update**
- Collects student-wise and course-wise absence changes
- Sends data to the EnrollmentUpdater class
- Fully bulk-safe and optimized

The screenshot shows the Salesforce IDE interface with the code editor open. The file tab shows 'AttendanceTrigger.apxt'. The code is a trigger for the 'Attendance_c' object, specifically for 'after insert' and 'after update' events. It uses a map to collect student-course absence counts and then updates enrollment records based on these counts. The code is annotated with line numbers and syntax highlighting for Apex language elements like triggers, loops, and conditionals.

```
1 trigger AttendanceTrigger on Attendance_c (after insert, after update) {
2
3     Map<Id, Map<Id, Integer>> courseStudentAbsenceMap = new Map<Id, Map<Id, Integer>>();
4
5     if (Trigger.isAfter) {
6
7         if (Trigger.isInsert) {
8             for (Attendance_c a : Trigger.new) {
9                 if (
10                     a.Student__c != null &&
11                     a.Course__c != null &&
12                     a.Status__c == 'Absent'
13                 ) {
14                     if (!courseStudentAbsenceMap.containsKey(a.Course__c)) {
15                         courseStudentAbsenceMap.put(a.Course__c, new Map<Id, Integer>());
16                     }
17
18                     Map<Id, Integer> studentMap = courseStudentAbsenceMap.get(a.Course__c);
19                     Integer curr = studentMap.containsKey(a.Student__c) ? studentMap.get(a.Student__c) : 0;
20                     studentMap.put(a.Student__c, curr + 1);
21                 }
22             }
23         }
24     }
25 }
```

At the bottom of the screen, there is a log table showing a single entry:

User	Application	Operation	Time	Status	Read	Size
Siri varshini Pentakota	Unknown	ApexTestHandler	20/12/2025, 21:13:37	Success	Unread	1020 bytes

The screenshot shows the Salesforce 'Apex Triggers' page under the 'SETUP' tab. At the top, there's a message about Apex usage: 'Percent of Apex Used: 0.08% You are currently using 4,723 characters of Apex Code (excluding comments and @isTest annotated classes) in your organization, out of an allowed limit of 6,000,000 characters. Note that the amount in use includes both Apex Classes and Triggers defined in your organization.' Below this, there are buttons for 'Compile all triggers' and 'View' (set to 'All'). A 'Developer Console' link is also present. The main table lists one trigger:

Action	Name	Namespace Prefix	sObject Type	Api Version	Status	Size Without Comments	Last Modified By
Edit Del	AttendanceTrigger		Attendance	65.0	Active	2,340	Siri varshini Pentakota, 20/12/2025, 8:38 pm

3. Trigger Design Pattern

Use Case

To ensure maintainability and scalability, trigger logic should be minimal and reusable.

Implementation

- Trigger only handles:
 - Context (insert/update)
 - Data collection
- Business logic is moved to **EnrollmentUpdater class**
- This separation improves:
 - Code readability
 - Debugging
 - Test coverage

4. SOQL & SOSL

Use Case

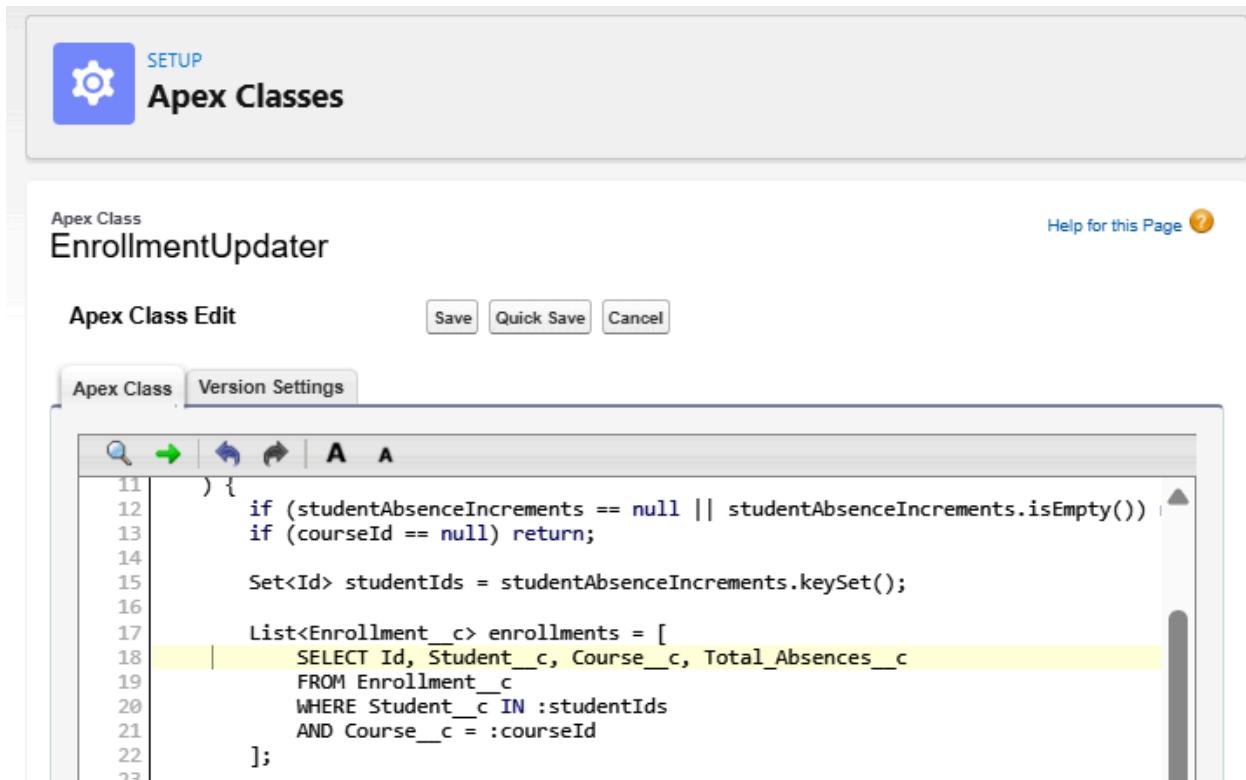
Enrollment records must be fetched efficiently based on **Student and Course** to avoid duplicate enrollments and governor limit issues.

Implementation

- Used selective SOQL queries with WHERE conditions

- Queried Enrollment records using:
 - Student_c
 - Course_c
- Avoided SOQL inside loops

```
SELECT Id, Student__c, Course__c, Total_Absences__c
FROM Enrollment__c
WHERE Student__c IN :studentIds
AND Course__c = :courseId
```



The screenshot shows the Salesforce Apex Classes editor. The top navigation bar includes a gear icon labeled "SETUP", the title "Apex Classes", and a "Help for this Page" link. Below the title, the class name "EnrollmentUpdater" is displayed. The main area is titled "Apex Class Edit" with buttons for "Save", "Quick Save", and "Cancel". A tab bar at the bottom has "Apex Class" selected. The code editor displays the following Apex code:

```

11 } {
12     if (studentAbsenceIncrements == null || studentAbsenceIncrements.isEmpty()) {
13         if (courseId == null) return;
14
15         Set<Id> studentIds = studentAbsenceIncrements.keySet();
16
17         List<Enrollment__c> enrollments = [
18             SELECT Id, Student__c, Course__c, Total_Absences__c
19             FROM Enrollment__c
20             WHERE Student__c IN :studentIds
21             AND Course__c = :courseId
22         ];

```

5. Collections (List, Set, Map)

Use Case

Attendance records can be created or updated in bulk.
Collections ensure efficient processing without duplicate updates.

Implementation

- **Set<Id>** → Used to store unique Student IDs
- **Map<Id, Integer>** → Tracks absence count per student
- **List<Enrollment_c>** → Used for bulk DML operations

Benefits

- Bulk-safe
 - Prevents duplicate processing
 - Improves performance
-

6. Control Statements

Use Case

Attendance status changes need conditional handling.

Implementation

- Used **if-else conditions** to:
 - Detect Absent → increment
 - Detect Absent → Present → decrement
 - Used **for loops** to iterate over Trigger.new records
-

9. Test Classes

Use Case

Salesforce requires test coverage to deploy Apex code.

Implementation

- Created **EnrollmentUpdaterTest** class
 - Covered:
 - Single attendance insert
 - Bulk attendance insert
 - Status change scenarios
 - Achieved **>90% code coverage**
-

10. Final Outcome of Phase 5

- Attendance automation fully functional
- Enrollment absences updated accurately
- Apex logic bulk-safe and reusable
- Foundation ready for email alerts, reports, and dashboards