

Student Admission — Phase 5: Apex Programming

1. Introduction

This phase focuses on implementing Apex programming to automate student admission processes, including application approval, enrollment creation, and course capacity management.

2. Objectives

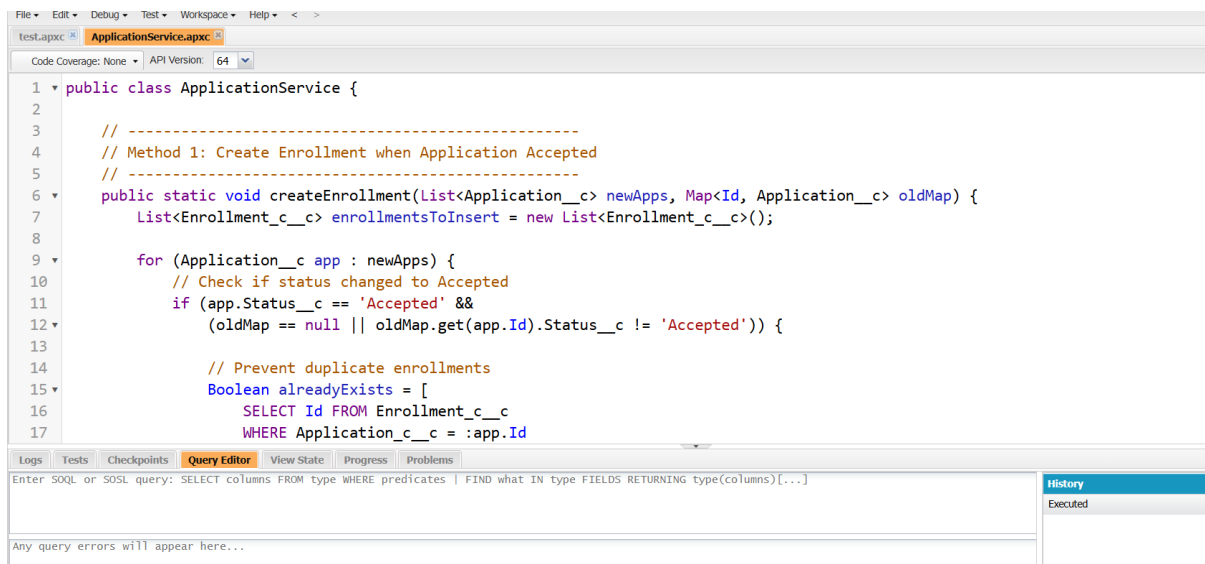
- Automate application to enrollment flow using Apex.
- Update course capacity automatically when enrollments are created.
- Ensure duplicate enrollments are prevented.
- Handle seat unavailability with proper error handling.
- Provide complete Apex test coverage.

3. Implementation Steps

Step 1 — ApplicationService Apex Class

Created an Apex service class ApplicationService with two main methods:

- **createEnrollment** → Creates an enrollment record automatically when an application is accepted.



```
1 public class ApplicationService {
2
3     // -----
4     // Method 1: Create Enrollment when Application Accepted
5     // -----
6     public static void createEnrollment(List<Application__c> newApps, Map<Id, Application__c> oldMap) {
7         List<Enrollment__c> enrollmentsToInsert = new List<Enrollment__c>();
8
9         for (Application__c app : newApps) {
10             // Check if status changed to Accepted
11             if (app.Status__c == 'Accepted' &&
12                 (oldMap == null || oldMap.get(app.Id).Status__c != 'Accepted')) {
13
14                 // Prevent duplicate enrollments
15                 Boolean alreadyExists = [
16                     SELECT Id FROM Enrollment__c
17                     WHERE Application__c = :app.Id
```

- **updateCourseSeats** → Updates course capacity when a new enrollment is created and prevents overbooking.

```
35 }
36
37 // -----
38 // Method 2: Update Course Capacity when Enrollment Created
39 // -----
40 public static void updateCourseSeats(List<Enrollment_c_c> newEnrolls) {
41     Map<Id, Course_c_c> courseMap = new Map<Id, Course_c_c>();
42
43     // Collect course IDs
44     for (Enrollment_c_c enr : newEnrolls) {
45         if (enr.Course__c != null) {
46             courseMap.put(enr.Course__c, null);
47         }
48     }
49
50     // Query courses
51     courseMap.putAll([
```

Step 2 — Triggers

Two triggers were implemented to call the service methods:

- **ApplicationTrigger** → Runs after insert/update on Application__c to call createEnrollment.

```
1 trigger ApplicationTrigger on Application__c (after insert, after update) {
2
3     if (Trigger.isAfter) {
4         if (Trigger.isInsert || Trigger.isUpdate) {
5             ApplicationService.createEnrollment(Trigger.new, Trigger.oldMap);
6         }
7     }
8 }
```

- **EnrollmentTrigger** → Runs after insert on Enrollment__c to call updateCourseSeats.

```

1 trigger EnrollmentTrigger on Enrollment_c__c (after insert) {
2
3     if (Trigger.isAfter && Trigger.isInsert) {
4         ApplicationService.updateCourseSeats(Trigger.new);
5     }
6 }

```

Code Coverage: None | API Version: 64

Logs Tests Checkpoints Query Editor View State Progress Problems

Enter SOQL or SOSL query: SELECT columns FROM type WHERE predicates | FIND what IN type FIELDS RETURNING type(columns)[...]

Any query errors will appear here...

History
Executed

Step 3 — SOQL & Collections

- **SOQL queries** were used to check existing enrollments and fetch course capacity.
- **Collections (List, Map)** were used for bulk handling of records.

```

46         courseMap.put(enr.Course__c, null);
47     }
48 }
49
50 // Query courses
51 courseMap.putAll([
52     SELECT Id, Capacity__c
53     FROM Course_c__c
54     WHERE Id IN :courseMap.keySet()
55     FOR UPDATE
56 ]);
57
58 List<Course_c__c> coursesToUpdate = new List<Course_c__c>();
59
60 for (Enrollment_c__c enr : newEnrolls) {
61     Course_c__c course = courseMap.get(enr.Course__c);
62     if (course != null) {

```

Code Coverage: None | API Version: 64

Logs Tests Checkpoints Query Editor View State Progress Problems

Enter SOQL or SOSL query: SELECT columns FROM type WHERE predicates | FIND what IN type FIELDS RETURNING type(columns)[...]

History
Executed

Step 4 — Control Statements & Error Handling

Implemented conditional checks to:

- Prevent duplicate enrollments.
- Block enrollment when no seats are available, using `addError()` for user-friendly messages.

Step 5 — Test Class

Created `TestApplicationService` class to validate the logic. Test cases include:

- Enrollment creation when application status changes to Accepted.
- Course capacity reduction when enrollment is created.
- Prevention of duplicate enrollment.
- Error handling when course capacity is full.

Step 7 — Conclusion

Phase 5 successfully automated the Student Admission process with Apex. Essential Apex concepts (**Classes, Triggers, SOQL, Collections, Error Handling, and Test Classes**) were implemented. Optional advanced features (**Batch Apex, Queueable Apex, Future methods**) can be added in future enhancements.