

Phase 5 Completion Report – Apex Programming (Developer)

1. Classes & Objects

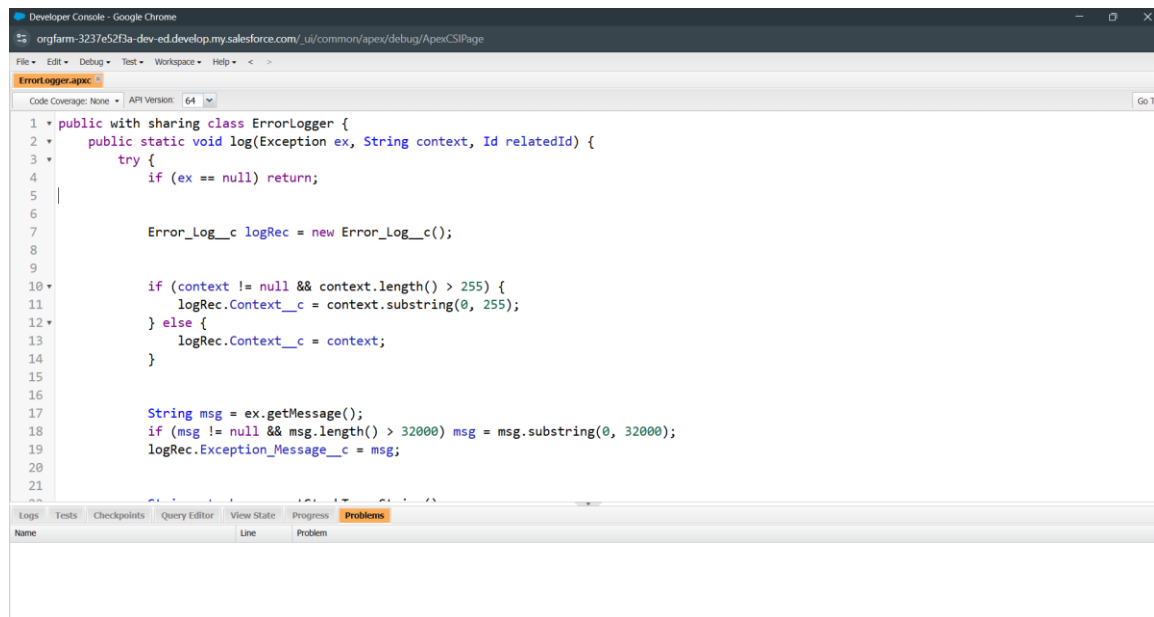
Custom Apex classes were created to handle specific business logic such as error logging, grade aggregation, performance recalculation, and guardian notifications.

Steps to Create Apex Class (applies to all classes):

1. Developer Console → File → New → Apex Class.
2. Enter Class Name → Click OK.
3. Write logic as per business requirement.
4. Save and test with Execute Anonymous or test classes.

Classes Implemented:

- **ErrorLogger** – Logs exceptions to Error_Log__c for debugging and auditing.

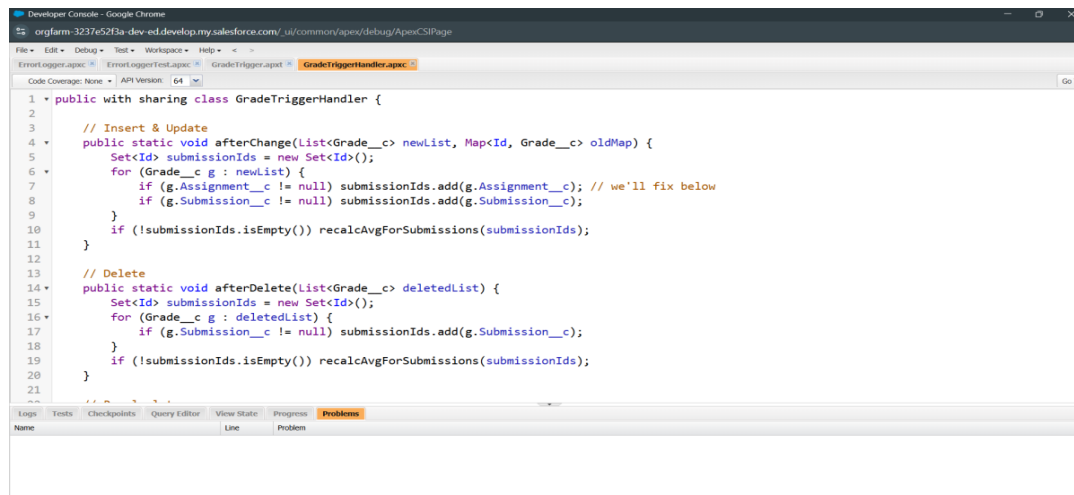


The screenshot shows the Salesforce Developer Console with the Apex class `ErrorLogger` open. The class is a public with sharing class that implements a static method `log` to log exceptions. The code is as follows:

```
1 public with sharing class ErrorLogger {
2     public static void log(Exception ex, String context, Id relatedId) {
3         try {
4             if (ex == null) return;
5
6             Error_Log__c logRec = new Error_Log__c();
7
8             if (context != null && context.length() > 255) {
9                 logRec.Context__c = context.substring(0, 255);
10            } else {
11                logRec.Context__c = context;
12            }
13
14            String msg = ex.getMessage();
15            if (msg != null && msg.length() > 32000) msg = msg.substring(0, 32000);
16            logRec.Exception_Message__c = msg;
17
18            insert logRec;
19        } catch (Exception e) {
20            // Handle logging error
21        }
22    }
23 }
```

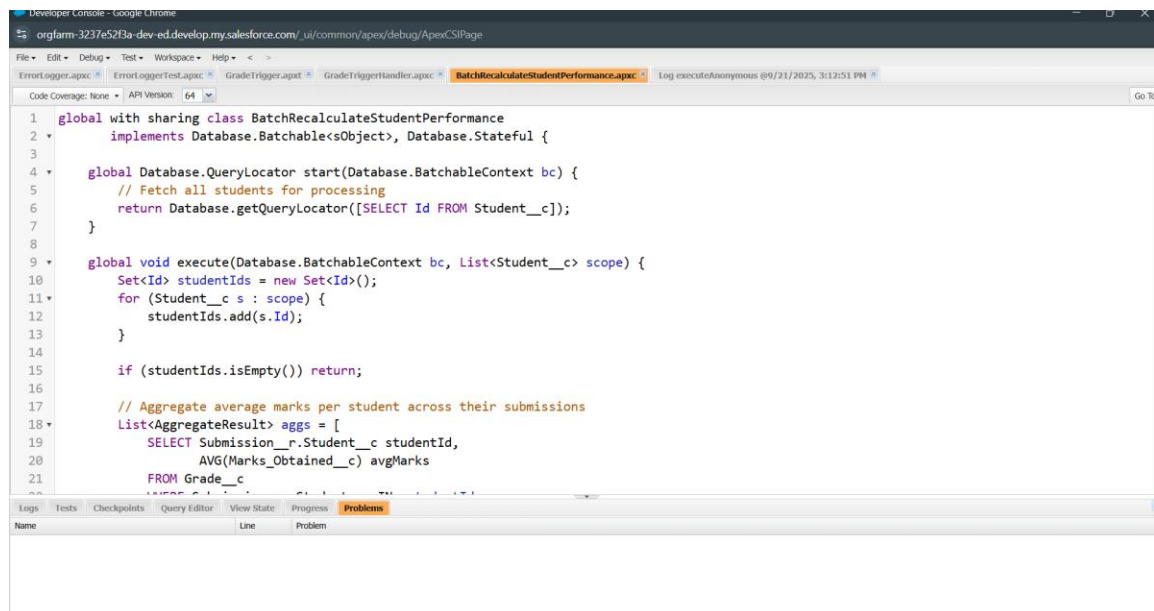
The interface at the bottom of the console shows tabs for Logs, Tests, Checkpoints, Query Editor, View State, Progress, and Problems. The Problems tab is currently selected, showing a table with columns for Name, Line, and Problem.

- **GradeTriggerHandler** – Recalculates student average marks after Grade changes.



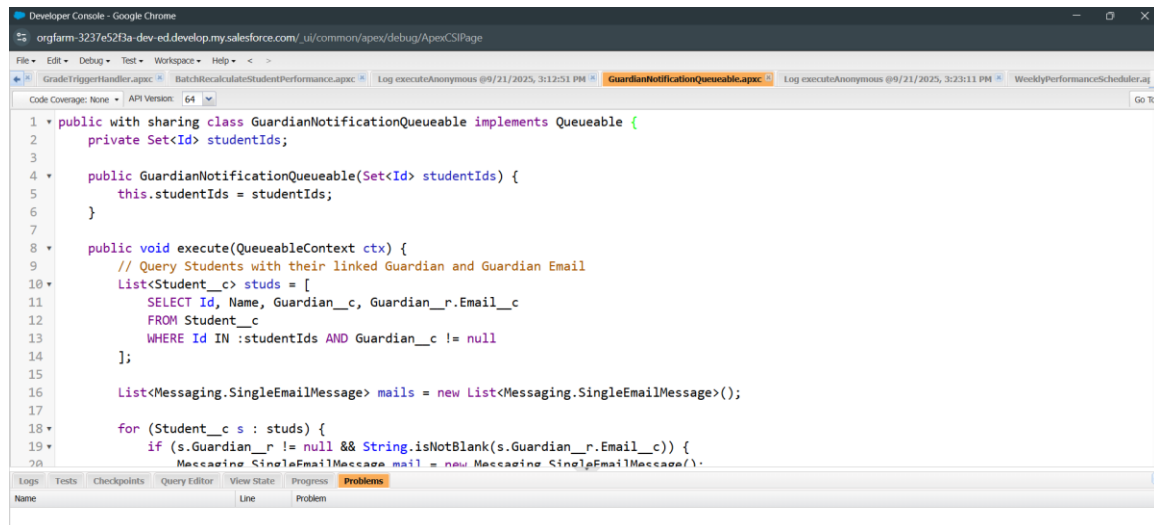
```
1 public with sharing class GradeTriggerHandler {
2
3     // Insert & Update
4     public static void afterChange(List<Grade__c> newList, Map<Id, Grade__c> oldMap) {
5         Set<Id> submissionIds = new Set<Id>();
6         for (Grade__c g : newList) {
7             if (g.Assignment__c != null) submissionIds.add(g.Assignment__c); // we'll fix below
8             if (g.Submission__c != null) submissionIds.add(g.Submission__c);
9         }
10        if (!submissionIds.isEmpty()) recalcAvgForSubmissions(submissionIds);
11    }
12
13    // Delete
14    public static void afterDelete(List<Grade__c> deletedList) {
15        Set<Id> submissionIds = new Set<Id>();
16        for (Grade__c g : deletedList) {
17            if (g.Submission__c != null) submissionIds.add(g.Submission__c);
18        }
19        if (!submissionIds.isEmpty()) recalcAvgForSubmissions(submissionIds);
20    }
21 }
```

- **BatchRecalculateStudentPerformance** – Batch Apex to recalculate all students' average marks.



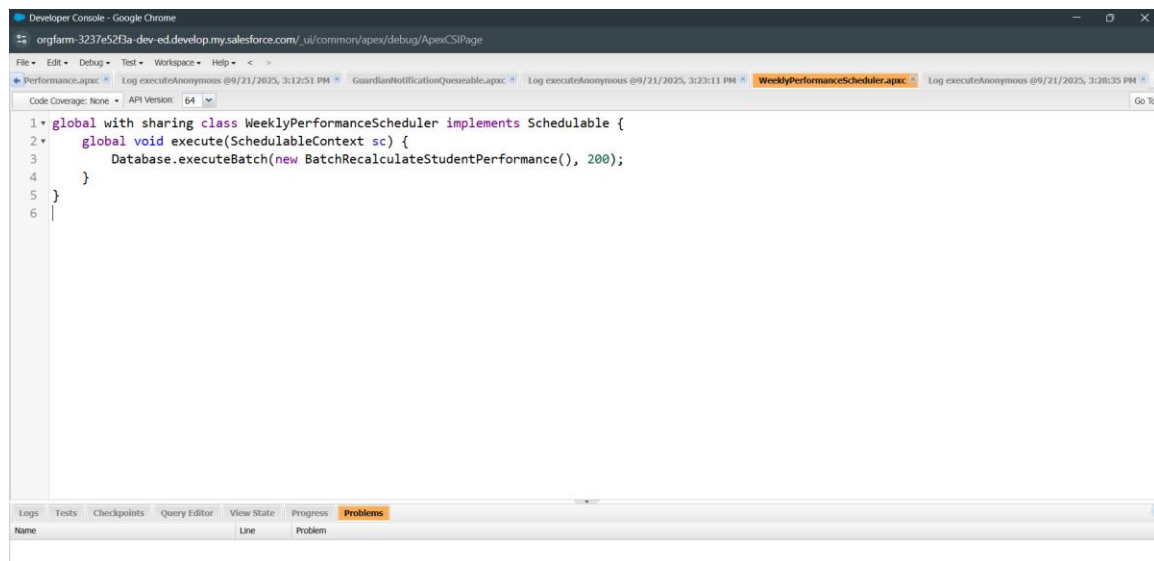
```
1 global with sharing class BatchRecalculateStudentPerformance
2     implements Database.Batchable<Object>, Database.Stateful {
3
4     global Database.QueryLocator start(Database.BatchableContext bc) {
5         // Fetch all students for processing
6         return Database.getQueryLocator([SELECT Id FROM Student__c]);
7     }
8
9     global void execute(Database.BatchableContext bc, List<Student__c> scope) {
10        Set<Id> studentIds = new Set<Id>();
11        for (Student__c s : scope) {
12            studentIds.add(s.Id);
13        }
14
15        if (studentIds.isEmpty()) return;
16
17        // Aggregate average marks per student across their submissions
18        List<AggregateResult> aggs = [
19            SELECT Submission__r.Student__c studentId,
20                AVG(Marks_Obtained__c) avgMarks
21            FROM Grade__c
22            WHERE Submission__c IN :studentIds
23            GROUP BY Student__c
24        ];
25    }
```

- **GuardianNotificationQueueable** – Queueable Apex to send email notifications to guardians.



```
1 public with sharing class GuardianNotificationQueueable implements Queueable {
2     private Set<Id> studentIds;
3
4     public GuardianNotificationQueueable(Set<Id> studentIds) {
5         this.studentIds = studentIds;
6     }
7
8     public void execute(QueueableContext ctx) {
9         // Query Students with their linked Guardian and Guardian Email
10        List<Student__c> studs = [
11            SELECT Id, Name, Guardian__c, Guardian_r.Email__c
12            FROM Student__c
13            WHERE Id IN :studentIds AND Guardian__c != null
14        ];
15
16        List<Messaging.SingleEmailMessage> mails = new List<Messaging.SingleEmailMessage>();
17
18        for (Student__c s : studs) {
19            if (s.Guardian__r != null && String.isNotBlank(s.Guardian_r.Email__c)) {
20                Messaging.SingleEmailMessage mail = new Messaging.SingleEmailMessage();
21            }
22        }
23    }
24 }
```

- **WeeklyPerformanceScheduler** – Scheduled Apex to run the batch weekly.



```
1 global with sharing class WeeklyPerformanceScheduler implements Schedulable {
2     global void execute(SchedulableContext sc) {
3         Database.executeBatch(new BatchRecalculateStudentPerformance(), 200);
4     }
5 }
6
```

2. Apex Triggers

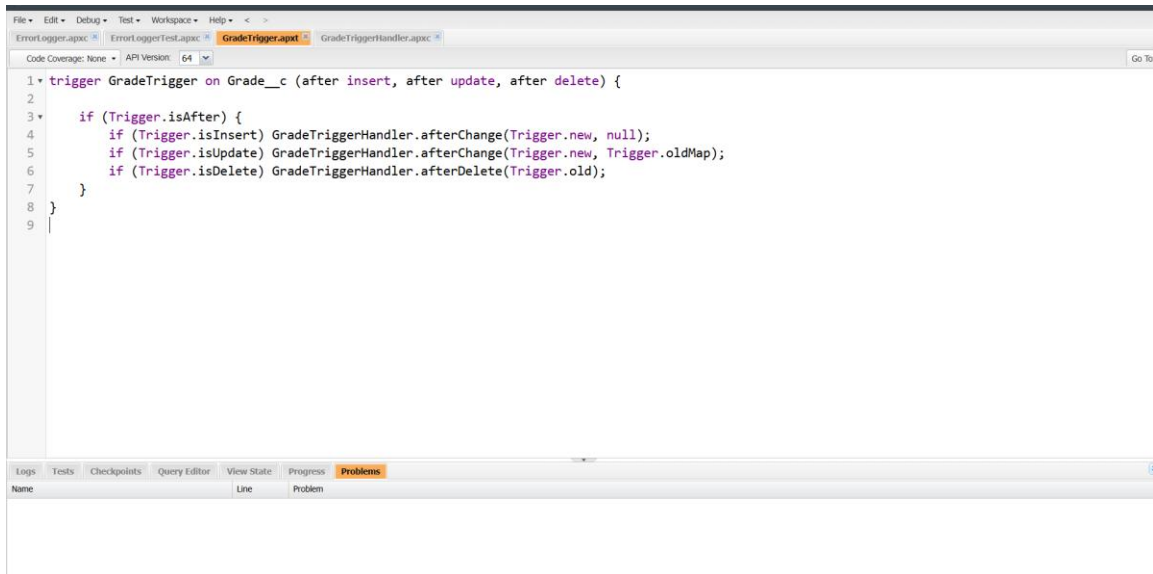
Triggers automate actions after DML events on key objects like Grade__c.

Steps to Create Trigger:

1. Developer Console → File → New → Apex Trigger.
2. Enter Trigger Name → Select Object.
3. Choose events (before/after insert/update/delete).
4. Call handler class methods.
5. Save.

Triggers Implemented:

- **GradeTrigger** – Fires after insert, after update, after delete on Grade__c to recalc averages.
- Calls GradeTriggerHandler.afterChange() and GradeTriggerHandler.afterDelete().



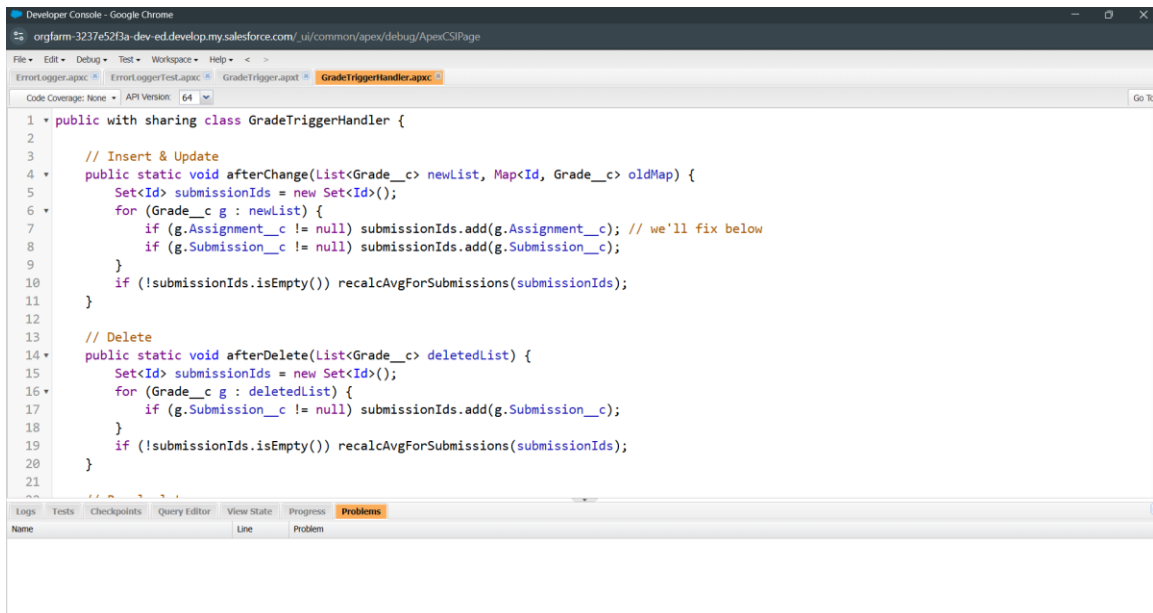
```
1 trigger GradeTrigger on Grade__c (after insert, after update, after delete) {
2
3     if (Trigger.isAfter) {
4         if (Trigger.isInsert) GradeTriggerHandler.afterChange(Trigger.new, null);
5         if (Trigger.isUpdate) GradeTriggerHandler.afterChange(Trigger.new, Trigger.oldMap);
6         if (Trigger.isDelete) GradeTriggerHandler.afterDelete(Trigger.old);
7     }
8 }
9
```

3. Trigger Handler Classes

Handler classes separate logic from triggers (Trigger Design Pattern).

GradeTriggerHandler:

- Recalculates average marks for students when grades are inserted, updated, or deleted.
- Uses try-catch with ErrorLogger.log() to safely log exceptions.



```
1 public with sharing class GradeTriggerHandler {
2
3     // Insert & Update
4     public static void afterChange(List<Grade__c> newList, Map<Id, Grade__c> oldMap) {
5         Set<Id> submissionIds = new Set<Id>();
6         for (Grade__c g : newList) {
7             if (g.Assignment__c != null) submissionIds.add(g.Assignment__c); // we'll fix below
8             if (g.Submission__c != null) submissionIds.add(g.Submission__c);
9         }
10        if (!submissionIds.isEmpty()) recalAvgForSubmissions(submissionIds);
11    }
12
13    // Delete
14    public static void afterDelete(List<Grade__c> deletedList) {
15        Set<Id> submissionIds = new Set<Id>();
16        for (Grade__c g : deletedList) {
17            if (g.Submission__c != null) submissionIds.add(g.Submission__c);
18        }
19        if (!submissionIds.isEmpty()) recalAvgForSubmissions(submissionIds);
20    }
21 }
```

Steps:

1. Collect affected Submission__c IDs from grades.
2. Map submissions to students.
3. Aggregate grades and calculate averages per student.
4. Update Student__c records with Average_Marks__c.

4. Batch Apex

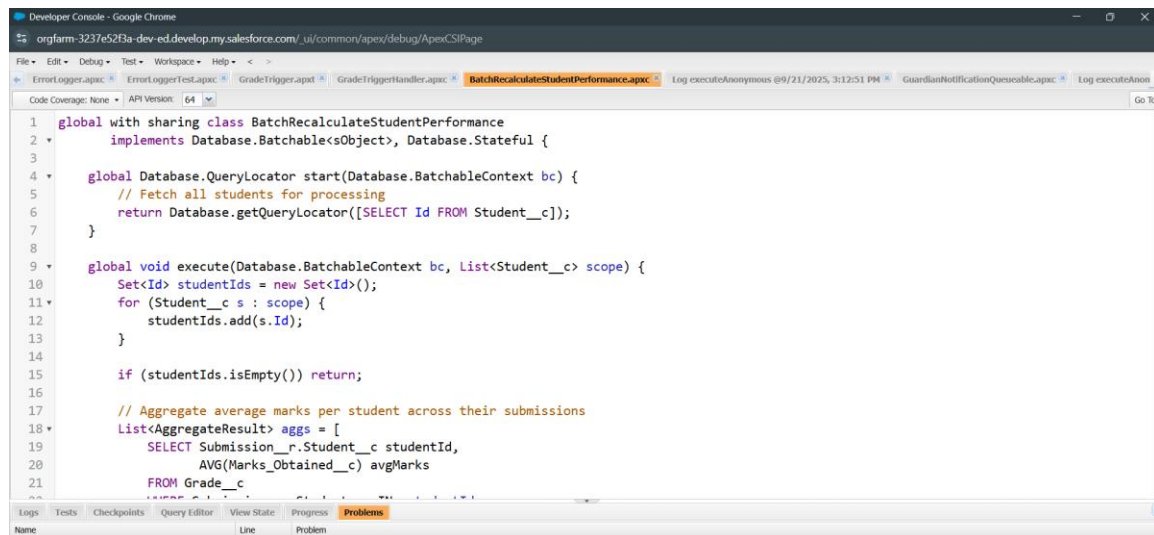
Batch Apex handles large datasets asynchronously.

BatchRecalculateStudentPerformance:

- Recalculates average marks for all students.
- Implements Database.Batchable<sObject> and Database.Stateful.

Steps:

- start() queries all Student__c records.
- execute() calculates average marks for each student in batch scope.
- Updates Student__c records.
- finish() logs completion (can enqueue further actions).



```
1 global with sharing class BatchRecalculateStudentPerformance
2 implements Database.Batchable<sObject>, Database.Stateful {
3
4     global Database.QueryLocator start(Database.BatchableContext bc) {
5         // Fetch all students for processing
6         return Database.getQueryLocator([SELECT Id FROM Student__c]);
7     }
8
9     global void execute(Database.BatchableContext bc, List<Student__c> scope) {
10         Set<Id> studentIds = new Set<Id>();
11         for (Student__c s : scope) {
12             studentIds.add(s.Id);
13         }
14
15         if (studentIds.isEmpty()) return;
16
17         // Aggregate average marks per student across their submissions
18         List<AggregateResult> aggs = [
19             SELECT Submission__r.Student__c studentId,
20                 AVG(Marks_Obtained__c) avgMarks
21             FROM Grade__c
```

Process Flow Example:

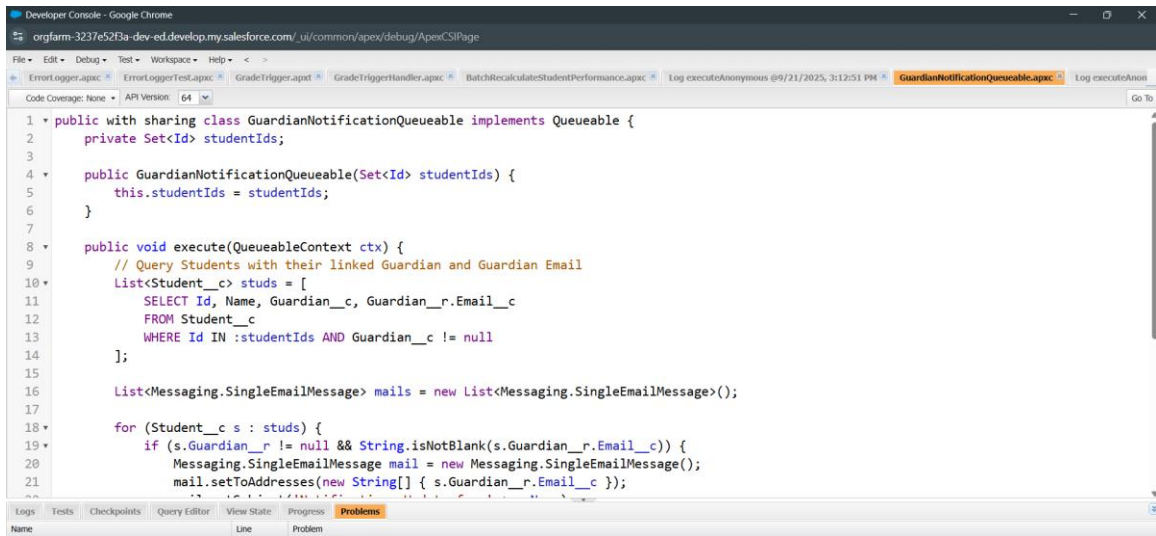
Developer Console → File → New → Apex Class → Write Batch Logic → Save → Execute via Execute Anonymous or Scheduled Apex.

5. Queueable Apex

Queueable Apex processes asynchronous tasks in a chainable, flexible manner.

GuardianNotificationQueueable:

- Sends email notifications to guardians.



```
1 public with sharing class GuardianNotificationQueueable implements Queueable {
2     private Set<Id> studentIds;
3
4     public GuardianNotificationQueueable(Set<Id> studentIds) {
5         this.studentIds = studentIds;
6     }
7
8     public void execute(QueueableContext ctx) {
9         // Query Students with their linked Guardian and Guardian Email
10        List<Student__c> studs = [
11            SELECT Id, Name, Guardian__c, Guardian__r.Email__c
12            FROM Student__c
13            WHERE Id IN :studentIds AND Guardian__c != null
14        ];
15
16        List<Messaging.SingleEmailMessage> mails = new List<Messaging.SingleEmailMessage>();
17
18        for (Student__c s : studs) {
19            if (s.Guardian__r != null && String.isNotBlank(s.Guardian__r.Email__c)) {
20                Messaging.SingleEmailMessage mail = new Messaging.SingleEmailMessage();
21                mail.setToAddresses(new String[] { s.Guardian__r.Email__c });
22            }
23        }
24    }
25 }
```

Steps:

1. Accepts student IDs in constructor.
2. Queries students with linked guardians.
3. Builds email messages.
4. Sends via Messaging.sendEmail().

Process Flow Example:

Developer Console → File → New → Apex Class → Implement Queueable → Save → Enqueue using System.enqueueJob(new ClassName(params)).

6. Scheduled Apex

Scheduled Apex automates batch processing on a time-based schedule.

WeeklyPerformanceScheduler:

- Runs the BatchRecalculateStudentPerformance weekly.

Steps:

1. Developer Console → File → New → Apex Class.
2. Implement Schedulable interface.
3. In execute(), call Database.executeBatch(new BatchRecalculateStudentPerformance(), 200).
4. Save and schedule via Setup → Apex Classes → Schedule Apex.

The screenshot shows the Salesforce Developer Console with the 'WeeklyPerformanceScheduler.apex' file open. The code defines a global class that implements the Schedulable interface. It has a single method 'execute' that calls 'Database.executeBatch' with a new 'BatchRecalculateStudentPerformance()' object and a batch size of 200.

```
1 global with sharing class WeeklyPerformanceScheduler implements Schedulable {
2     global void execute(SchedulableContext sc) {
3         Database.executeBatch(new BatchRecalculateStudentPerformance(), 200);
4     }
5 }
6
```

7. Exception Handling

Implemented via ErrorLogger class:

- Wraps risky operations in try-catch blocks.
- Logs exceptions to Error_Log__c to prevent system failures and allow debugging.

8. Test Classes

- ErrorLoggerTest – Validates that exceptions are correctly logged.

The screenshot shows the Salesforce Developer Console with the 'ErrorLoggerTest.apex' file open. The code defines a test class 'ErrorLoggerTest' with a static test method 'testLogCreatesRecord'. This method starts a test, forces an exception by dividing 1 by 0, catches the 'System.MathException', logs it using 'ErrorLogger.log', and then queries the 'Error_Log__c' table to verify the log entry.

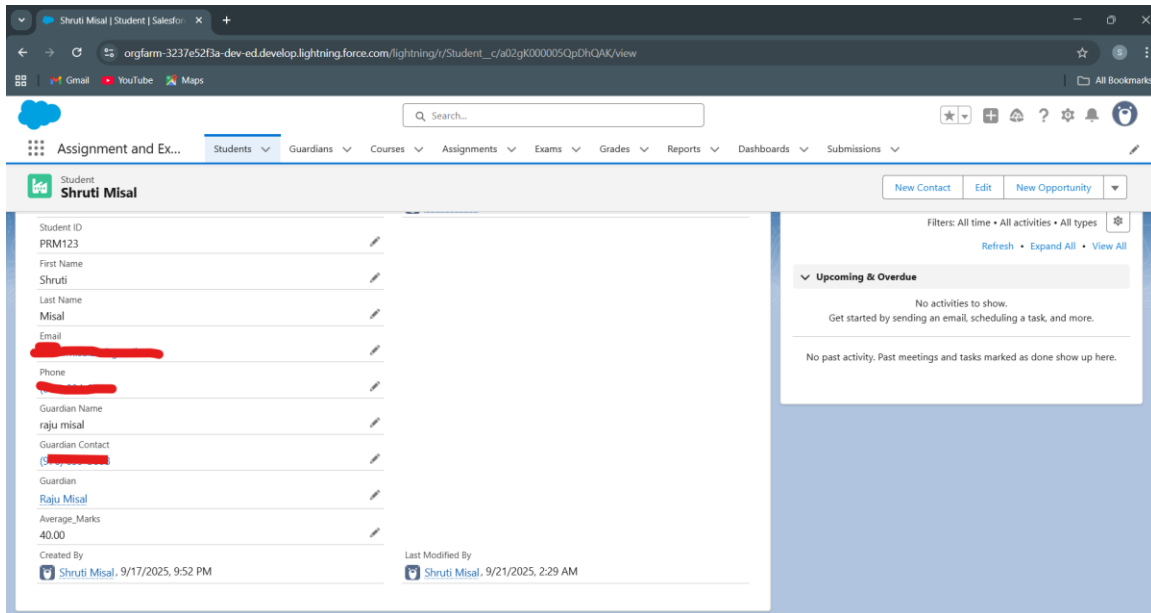
```
1 @isTest
2 private class ErrorLoggerTest {
3
4     @isTest
5     static void testLogCreatesRecord() {
6         Test.startTest();
7         try {
8             // Force an exception
9             Integer i = 1 / 0; // This will throw a System.MathException
10        } catch (Exception ex) {
11            // Call your logger with the caught exception
12            ErrorLogger.log(ex, 'UnitTestContext', null);
13        }
14        Test.stopTest();
15
16        // Query the inserted error log
17        List<Error_Log__c> logs = [
18            SELECT Id, Context__c, Exception_Message__c
19            FROM Error_Log__c
20            WHERE Context__c = 'UnitTestContext'
21            ORDER BY CreatedDate DESC
22        ];
23    }
24 }
```

Below the code editor, the 'Tests' tab shows the test results. Two test runs are listed, both of which passed. The 'Overall Code Coverage' table shows that the 'ErrorLogger' class has a 77% code coverage across 14 lines.

Status	Test Run	Enqueued Time	Duration	Failures	Total
✓	TestRun @ 11:27:47 am			0	1
✓	TestRun @ 11:31:24 am			0	1

Overall Code Coverage		
Class	Percent	Lines
Overall	77%	
ErrorLogger	77%	14/18

- GradeTriggerHandlerTest – (if created) would verify average mark recalculations.



Steps to Create Test Classes:

1. Developer Console → File → New → Apex Class.
2. Annotate with @isTest.
3. Insert test data → Perform DML → Assert expected outcomes.

9. Asynchronous Processing

Batch, Queueable, and Scheduled Apex ensure heavy or repeated calculations (attendance, grades) run asynchronously.

- Reduces governor limits issues.
- Improves performance for large datasets.