

□ HealthCare360 – Apex Programming Phase 5 Documentation

Organization: HealthCare360

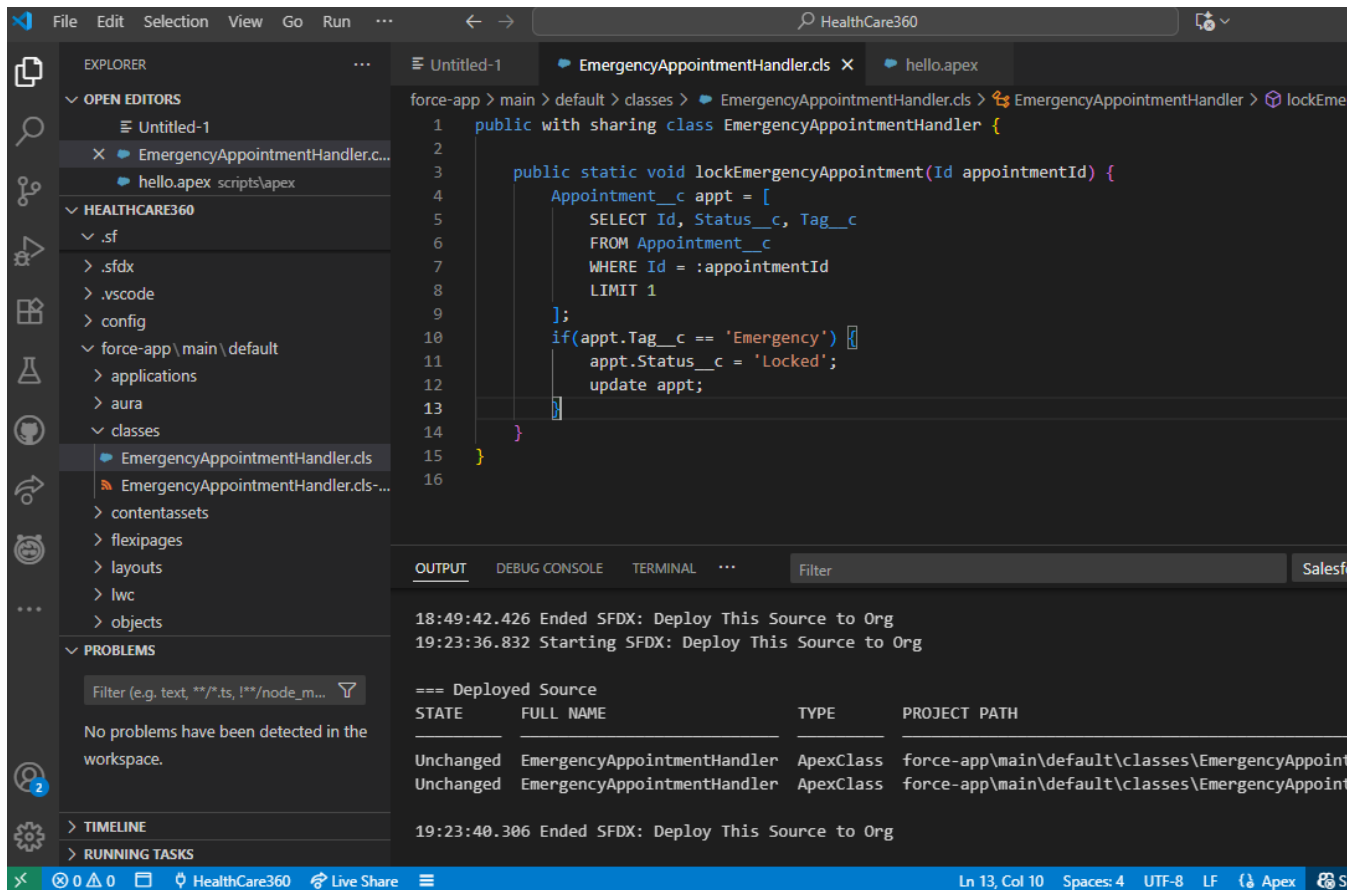
Purpose: To implement robust Apex-based automation, asynchronous processing, and audit-compliant logic for healthcare appointment and patient management.

□ Phase Overview

This phase focuses on Apex programming essentials for HealthCare360's Salesforce org. It includes trigger logic, reusable classes, asynchronous processing, exception handling, and test coverage. The goal is to ensure scalable, secure, and maintainable backend logic for healthcare workflows.

1. Classes & Objects

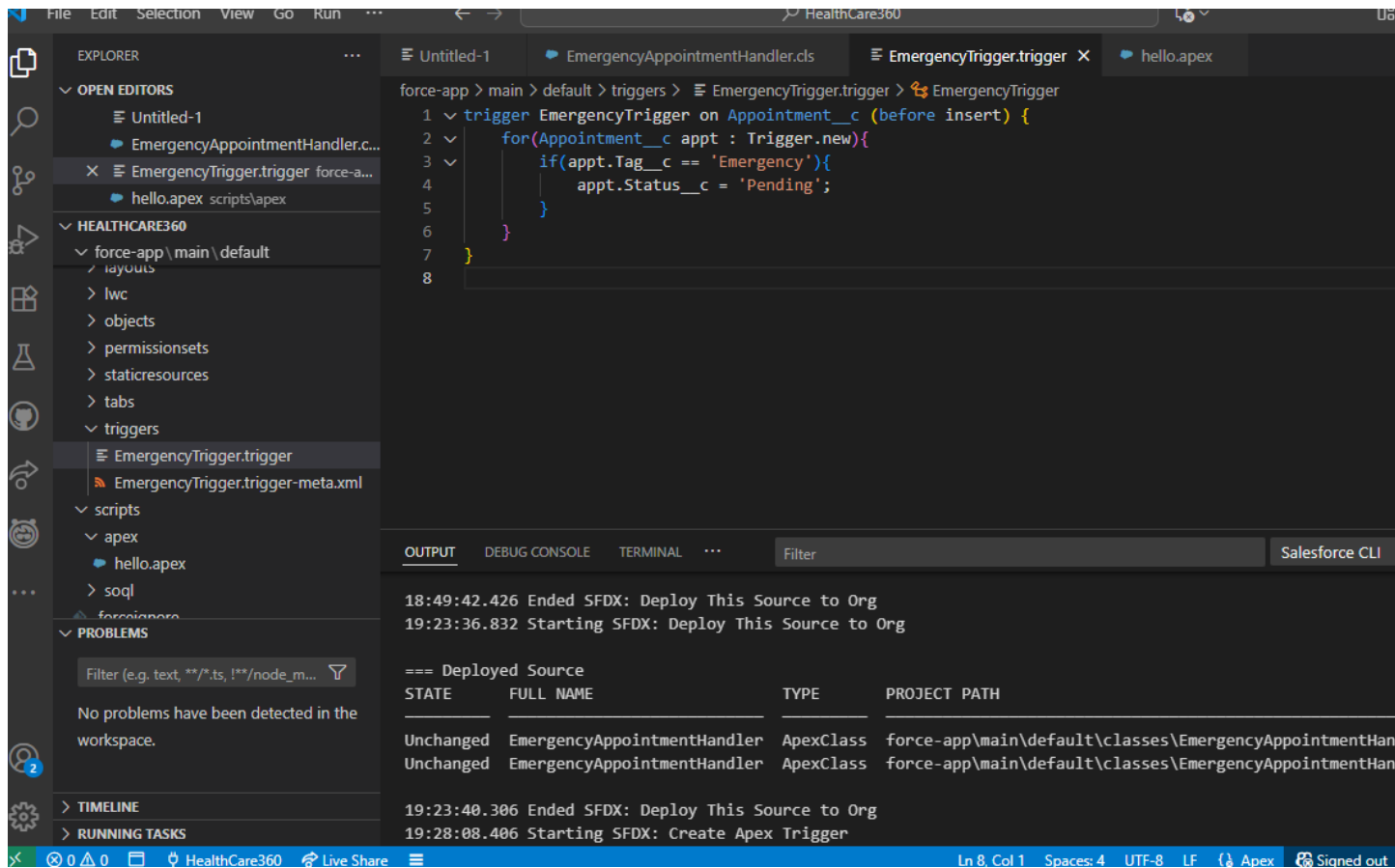
- Apex classes encapsulate reusable logic for appointment handling, status updates, and automation.
- Classes follow naming conventions like `AppointmentHelper`, `LockEmergencyService`, etc.
- Object-oriented principles such as encapsulation and modularity are applied to separate business logic from triggers.



2. Apex Triggers

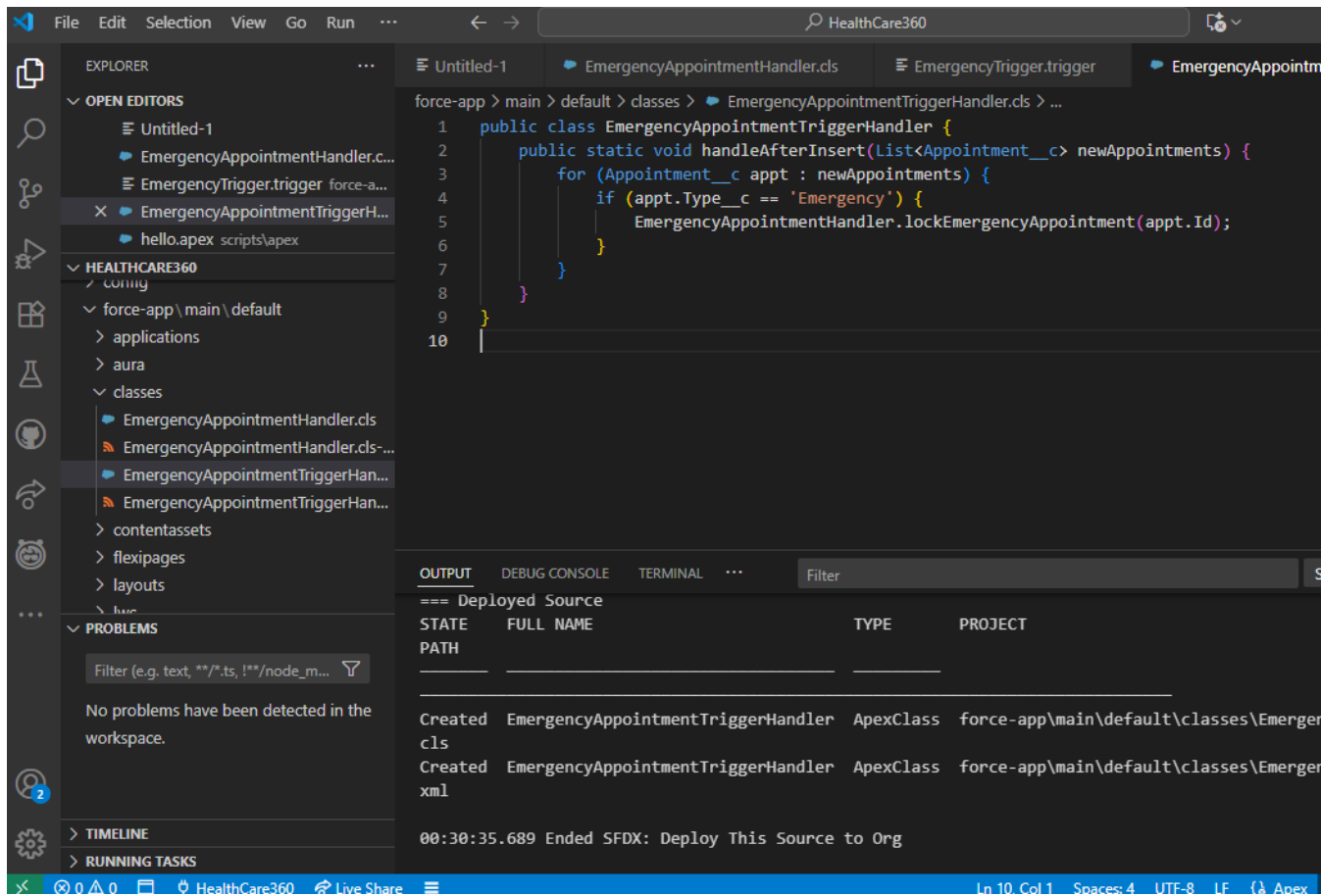
- Triggers are used to automate actions on Appointment__c records.
- Example: LockEmergencyTrigger runs on before insert to auto-lock emergency appointments.

Trigger logic is minimal and delegates to handler classes for maintainability.



3. Trigger Design Pattern

- All triggers delegate logic to handler classes using a standardized pattern.
- Benefits include cleaner code, easier testing, and modular updates.
- Example structure:
 - Trigger: AppointmentTrigger
 - Handler: AppointmentTriggerHandler
 - Methods: handleBeforeInsert, handleAfterUpdate, etc.



4. SOQL & SOSL

- **SOQL** is used for precise record retrieval, such as filtering appointments by type or status.
- **SOSL** is used for keyword-based searches across multiple fields.
- Queries are optimized using `LIMIT`, indexed fields, and selective filters to avoid governor limits.

5. Collections: List, Set, Map

- **List**: Used for ordered collections of records (e.g., appointments).
- **Set**: Used to store unique values (e.g., appointment types).
- **Map**: Used for key-value pairing (e.g., mapping appointment IDs to records).
- Collections are used extensively in batch processing, trigger handlers, and test classes.

The screenshot displays the Salesforce IDE interface. At the top, there are tabs for 'Log executeAnonymous @9/28/2025, 6:38:34 PM', 'Log executeAnonymous @9/28/2025, 7:08:07 PM', and 'Log executeAnonymous @9/28/2025, 7:19:54 PM'. Below these is the 'Execution Log' section, which contains a table with the following data:

Timestamp	Event	Details
19:20:55:002	USER_DEBUG	[4] DEBUG Map: {D001=Dr. Sharma, D002=Dr. Verma}
19:20:55:002	USER_DEBUG	[5] DEBUG Doctor D001: Dr. Sharma

Below the log, there is a section for 'Enter Apex Code' with the following code:

```
1 Map<String, String> doctorMap = new Map<String, String>();
2 doctorMap.put('D001', 'Dr. Sharma');
3 doctorMap.put('D002', 'Dr. Verma');
4 System.debug('Map: ' + doctorMap);
5 System.debug('Doctor D001: ' + doctorMap.get('D001'));
6
```

At the bottom, there is a table with the following data:

User	Application	URL	Timestamp	Status
Palak Yadav	Unknown	/services/data/v64.0/tooling/exec...	9/28/2025, 7:20:55 PM	Success
Palak Yadav	Unknown	/services/data/v64.0/tooling/exec...	9/28/2025, 7:19:54 PM	Success
Palak Yadav	Unknown	/services/data/v64.0/tooling/exec...	9/28/2025, 7:08:07 PM	Success
Palak Yadav	Unknown	/services/data/v64.0/tooling/exec...	9/28/2025, 6:38:34 PM	Success

Log executeAnonymous @9/28/2025, 6:38:34 PM

Log executeAnonymous @9/28/2025, 7:08:07 PM

Log executeAnonymous @9/28/2025, 7:19:54 PM

Execution Log

Timestamp	Event	Details
19:19:54:002	USER_DEBUG	[5] DEBUG Set: {Emergency, Routine}

Enter Apex Code

```
1 Set<String> uniqueTypes = new Set<String>();
2 uniqueTypes.add('Emergency');
3 uniqueTypes.add('Routine');
4 uniqueTypes.add('Emergency'); // Duplicate ignored
5 System.debug('Set: ' + uniqueTypes);
6
```

☐ This Frame ☐ Executable ☒ Debug Only ☐ Filter

Logs Tests Checkpoints Query Editor View State

☒ Open Log

User	Application	URL	Timestamp	Status
Palak Yadav	Unknown	/services/data/v64.0/tooling/exec...	9/28/2025, 7:19:54 PM	Success
Palak Yadav	Unknown	/services/data/v64.0/tooling/exec...	9/28/2025, 7:08:07 PM	Success
Palak Yadav	Unknown	/services/data/v64.0/tooling/exec...	9/28/2025, 6:38:34 PM	Success

File Edit Debug Test Workspace Help < >

Log executeAnonymous @9/28/2025, 6:38:34 PM

Log executeAnonymous @9/28/2025, 7:08:07 PM

Execution Log

Timestamp	Event	Details
19:08:07:003	USER_DEBUG	[4] DEBUG List: (Palak, Yadav)

Enter Apex Code

```
1 List<String> names = new List<String>();
2 names.add('Palak');
3 names.add('Yadav');
4 System.debug('List: ' + names);
5
```

☐ This Frame ☐ Executable ☒ Debug Only ☐ Filter

Logs Tests Checkpoints Query Editor View State

☒ Open Log

User	Application	URL	Timestamp	Status
Palak Yadav	Unknown	/services/data/v64.0/tooling/exec...	9/28/2025, 7:08:07 PM	Success
Palak Yadav	Unknown	/services/data/v64.0/tooling/exec...	9/28/2025, 6:38:34 PM	Success

6. Control Statements

- Apex logic uses control statements like `if`, `else`, `for`, `while`, `do-while`, and `switch`.
- These are used to implement conditional flows, loops, and decision-making logic.
- Control statements are applied in automation logic, data validation, and record processing.

The screenshot displays the Salesforce IDE interface with two instances of the 'Enter Apex Code' dialog box overlaid on the 'Execution Log' window.

Top Dialog Box:

```
1 String type = 'Emergency';
2 if (type == 'Emergency') {
3     System.debug('Appointment is Emergency type');
4 }
5
```

Bottom Dialog Box:

```
1 String apptType = 'Routine';
2 switch on apptType {
3     when 'Emergency' {
4         System.debug('Handle emergency');
5     }
6     when 'Routine' {
7         System.debug('Handle routine');
8     }
9     when else {
10        System.debug('Unknown type');
11    }
12 }
```

Execution Log (Top):

Timestamp	Event	Details
19:31:37:002	USER_DEBUG	[3] DEBUG Appointment is Emergency type

Execution Log (Bottom):

Timestamp	Event	Details
19:37:24:002	USER_DEBUG	[7] DEBUG Handle routine

The background shows the 'Execution Log' window with columns for User, Application, and Size. It lists two successful executions by Palak Yadav at 7:22:50 PM and 7:36:48 PM on 9/28/2025.

Execution Log		
Timestamp	Event	Details
19:36:48:001	USER_DEBUG	[3] DEBUG Do Count: 0
19:36:48:001	USER_DEBUG	[3] DEBUG Do Count: 1

Enter Apex Code

```
1 Integer j = 0;
2 do {
3     System.debug('Do Count: ' + j);
4     j++;
5 } while (j < 2);
6
```

☐ This Frame ☐ Executable ☒ Debug Only ☐ Filter

Logs Tests Checkpoints Query Editor View State

☒ Open Log

User	Application	URL	Timestamp	Status
Palak Yadav	Unknown	/services/data/v64.0/tooling/exec...	9/28/2025, 7:36:48 PM	Success
Palak Yadav	Unknown	/services/data/v64.0/tooling/exec...	9/28/2025, 7:35:13 PM	Success
Palak Yadav	Unknown	/services/data/v64.0/tooling/exec...	9/28/2025, 7:33:50 PM	Success

Execution Log		
Timestamp	Event	Details
19:35:13:003	USER_DEBUG	[3] DEBUG Count: 0
19:35:13:003	USER_DEBUG	[3] DEBUG Count: 1
19:35:13:003	USER_DEBUG	[3] DEBUG Count: 2

Enter Apex Code

```
1 Integer i = 0;
2 while (i < 3) {
3     System.debug('Count: ' + i);
4     i++;
5 }
6
```

☐ This Frame ☐ Executable ☒ Debug Only ☐ Filter

Logs Tests Checkpoints Query Editor View State

☒ Open Log

User	Application	URL	Timestamp	Status
Palak Yadav	Unknown	/services/data/v64.0/tooling/exec...	9/28/2025, 7:35:13 PM	Success
Palak Yadav	Unknown	/services/data/v64.0/tooling/exec...	9/28/2025, 7:33:50 PM	Success
Palak Yadav	Unknown	/services/data/v64.0/tooling/exec...	9/28/2025, 7:32:57 PM	Success

The screenshot displays the Salesforce IDE interface. At the top, a menu bar includes File, Edit, Debug, Test, Workspace, and Help. Below the menu, a series of tabs show the execution log for multiple anonymous Apex scripts. The main window is divided into two sections. The top section, titled 'Execution Log', contains a table with the following data:

Timestamp	Event	Details
19:32:57:003	USER_DEBUG	[5][DEBUG]Still open

The bottom section of the IDE features a toolbar with buttons for 'This Frame', 'Executable', 'Debug Only' (which is checked), 'Filter', and 'View State'. Below the toolbar, there are tabs for 'Logs', 'Tests', 'Checkpoints', 'Query Editor', and 'View State'. The 'Logs' tab is currently active, showing a list of execution logs with columns for User, Application, Location, Time, and Status. The bottom two rows of the log table are visible:

User	Application	Location	Time	Status
Palak Yadav	Unknown	/services/data/v64.0/tooling/exec...	9/28/2025, 7:32:57 PM	Success
Palak Yadav	Unknown	/services/data/v64.0/tooling/exec...	9/28/2025, 7:31:37 PM	Success

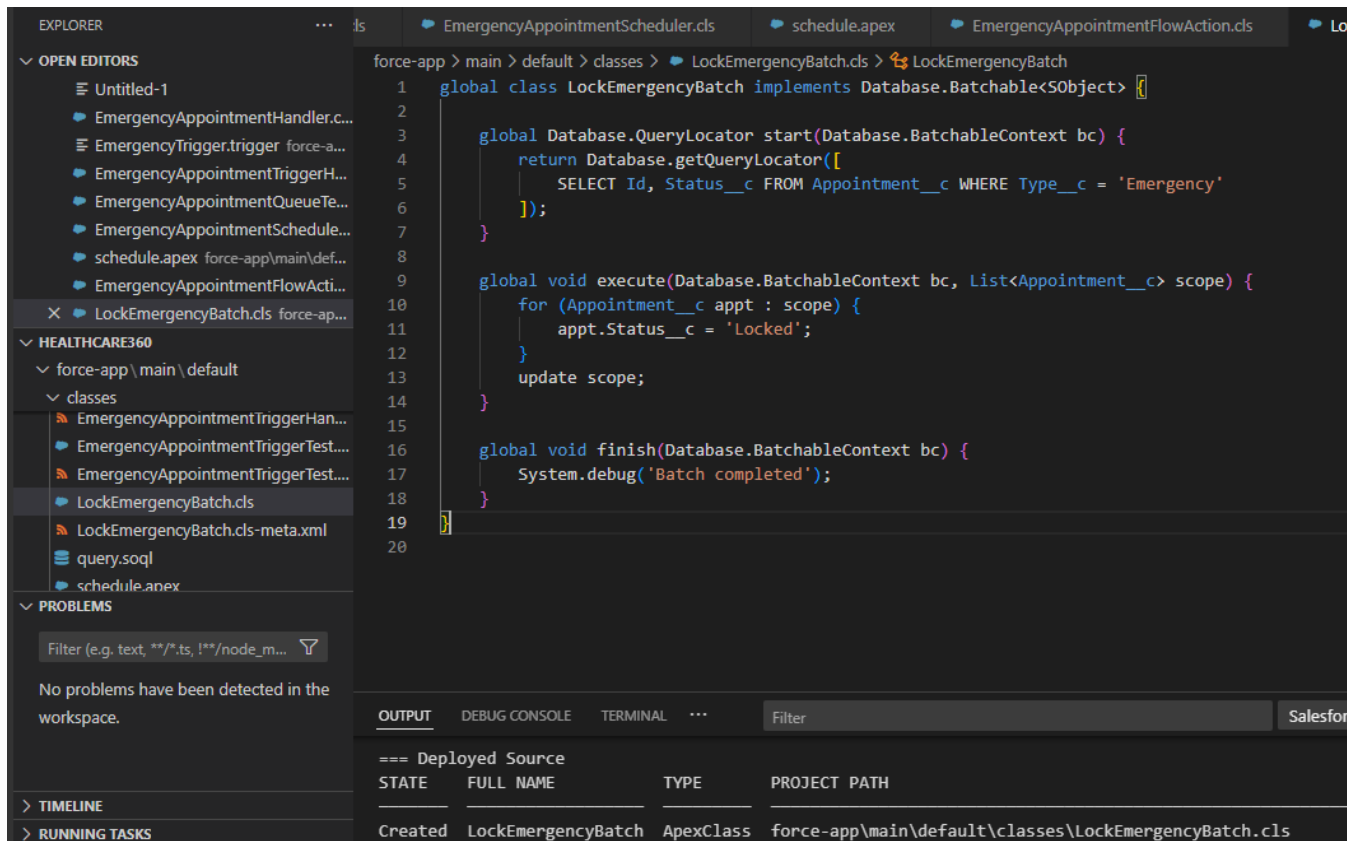
Overlaid on the right side of the IDE is a 'Enter Apex Code' dialog box. It contains a text area with the following Apex code:

```
1 String status = 'New';
2 if (status == 'Locked') {
3     System.debug('Already locked');
4 } else {
5     System.debug('Still open');
6 }
7
```

At the bottom right of the dialog box, there are three buttons: 'Open Log' (which is checked), 'Execute', and 'Execute Highlighted'.

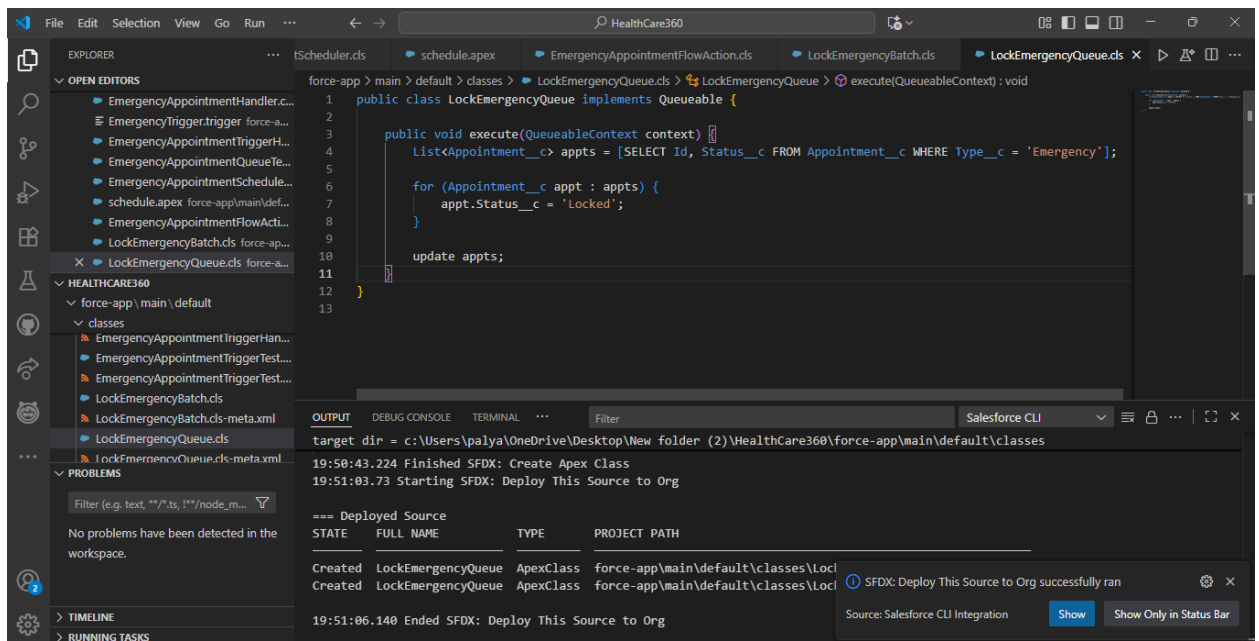
7. Batch Apex

- Batch Apex is used to process large volumes of data asynchronously.
- Class implements `Database.Batchable<SObject>` and defines `start`, `execute`, and `finish` methods.
- Used for bulk updates, cleanup jobs, and scheduled data maintenance.
- Example: `LockEmergencyBatch` updates all emergency appointments to “Locked” status.



8. Queueable Apex

- Queueable Apex is used for flexible asynchronous processing with support for chaining.
- Class implements `Queueable` and defines `execute()` method.
- Ideal for background jobs that need partial retry or multi-step logic.
- Example: `LockEmergencyQueue` updates emergency appointments without blocking UI.




9. Scheduled Apex

- Scheduled Apex automates logic at specific times using cron expressions.
- Class implements `Schedulable` and defines `execute()` method.
- Jobs are scheduled using `System.schedule()` with cron syntax.
- Example: `LockScheduler` runs daily at noon to lock emergency appointments.

All Scheduled Jobs

The All Scheduled Jobs page lists all of the jobs scheduled by your users. Multiple job types may display on this page. You can delete scheduled jobs if you have the permission to do so.


Percentage of Scheduled Jobs Used: 1%
 You have currently used 1 scheduled Apex jobs out of an allowed organization limit of 100 active or scheduled jobs. To learn about how this limit is calculated and what contributes to it see the [Lightning Platform Apex Limits](#)

View: All Scheduled Jobs [Create New View](#)

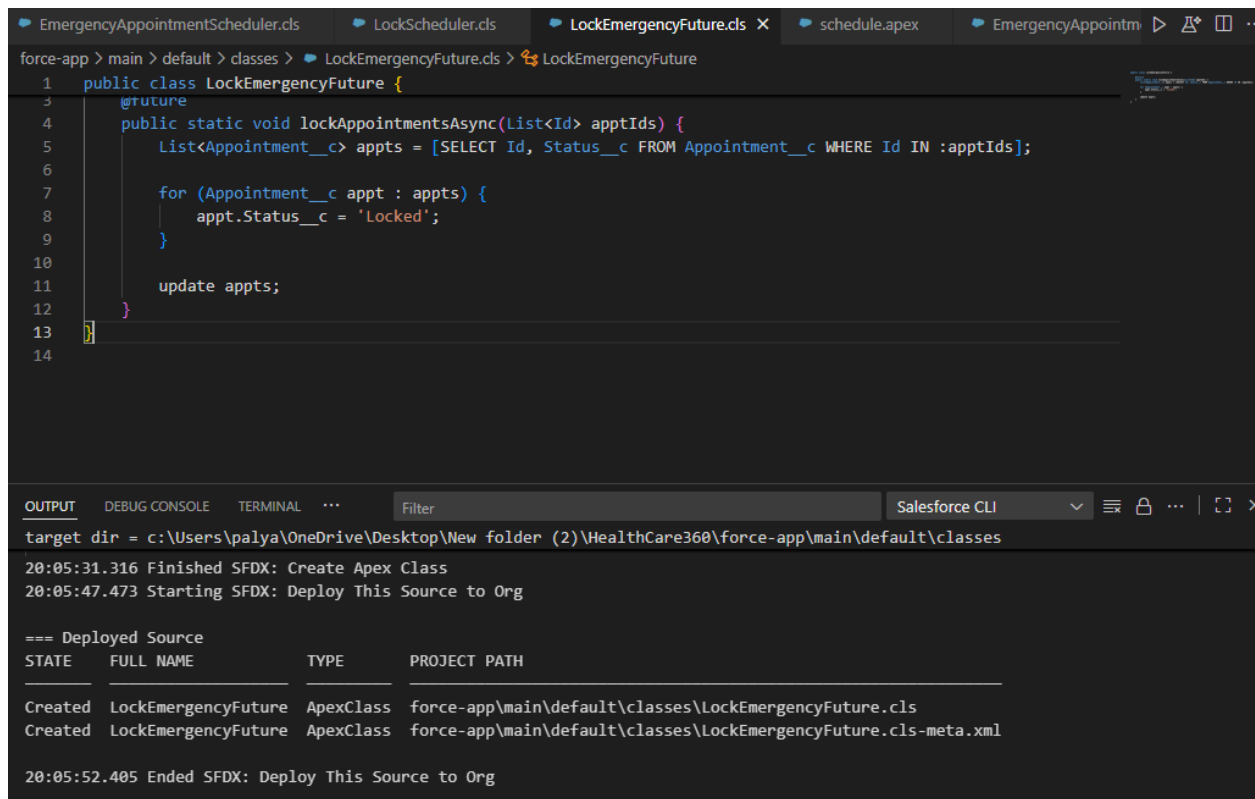
A | B | C | D | E | F | G | H | I | J | K |

Schedule Apex						
Action	Job Name ↑	Submitted By	Submitted	Started	Next Scheduled Run	Type
Manage Del Pause Job	Daily Emergency Lock	Yadav, Palak	9/27/2025, 6:55 AM	9/27/2025, 11:00 PM	9/28/2025, 11:00 PM	Scheduled Apex
Del	Metalytics Data Loader Job for Org : 00DgL00000BA9BI	User Integration	9/8/2025, 11:18 AM	9/27/2025, 1:52 PM	9/28/2025, 1:52 PM	Autonomous Data
	Program Milestone Computation Cron Job	Process, Automated	9/8/2025, 11:18 AM	9/28/2025, 7:00 AM	9/28/2025, 11:59 AM	Program Mileston
	Program Status Update Cron Job	Process, Automated	9/8/2025, 11:18 AM	9/28/2025, 5:01 AM	9/28/2025, 8:00 PM	Program Status U

A | B | C | D | E | F | G | H | I | J | K |

10. Future Methods

- Future methods are lightweight asynchronous methods marked with `@future`.
- Used for callouts, background updates, and non-blocking logic.
- Must be static and accept only primitive or collection parameters.
- Example: `LockEmergencyFuture.lockAppointmentsAsync(List<Id>)`



The screenshot shows an IDE with several tabs: EmergencyAppointmentScheduler.cls, LockScheduler.cls, LockEmergencyFuture.cls (active), schedule.apex, and EmergencyAppointm. The active tab displays the following Apex code:

```
1 public class LockEmergencyFuture {
2     @future
3     public static void lockAppointmentsAsync(List<Id> apptIds) {
4         List<Appointment__c> appts = [SELECT Id, Status__c FROM Appointment__c WHERE Id IN :apptIds];
5
6         for (Appointment__c appt : appts) {
7             appt.Status__c = 'Locked';
8         }
9
10        update appts;
11    }
12 }
13
14
```

Below the code editor, the OUTPUT pane shows the deployment process:

```
target dir = c:\Users\palya\OneDrive\Desktop\New folder (2)\HealthCare360\force-app\main\default\classes
20:05:31.316 Finished SFDX: Create Apex Class
20:05:47.473 Starting SFDX: Deploy This Source to Org

=== Deployed Source
STATE      FULL NAME                TYPE      PROJECT PATH
-----
Created    LockEmergencyFuture       ApexClass force-app\main\default\classes\LockEmergencyFuture.cls
Created    LockEmergencyFuture       ApexClass force-app\main\default\classes\LockEmergencyFuture.cls-meta.xml

20:05:52.405 Ended SFDX: Deploy This Source to Org
```

11. Exception Handling

- Apex logic uses try-catch blocks to handle runtime errors gracefully.
- Specific exceptions like `DmlException`, `NullPointerException`, and `QueryException` are caught.
- Errors are logged using `System.debug()` or custom logging classes.
- Exception handling ensures data integrity and improves user experience.

The screenshot shows an IDE with the following components:

- File Explorer:** Shows a project structure with files: EmergencyAppointmentScheduler.cls, LockScheduler.cls, LockEmergencyFuture.cls, LockEmergencySafe.cls (selected), and schedule.apex.
- Code Editor:** Displays the code for LockEmergencySafe.cls:

```
1 public class LockEmergencySafe {
2
3     public static void lockAppointments() {
4         try {
5             List<Appointment__c> appts = [SELECT Id, Status__c FROM Appointment__c WHERE Type__c = 'Emergency'];
6
7             for (Appointment__c appt : appts) {
8                 appt.Status__c = 'Locked';
9             }
10
11             update appts;
12
13         } catch (DmlException e) {
14             System.debug('DML Error: ' + e.getMessage());
15         } catch (Exception e) {
16             System.debug('General Error: ' + e.getMessage());
17         }
18     }
19 }
20
```
- Output Console:** Shows the deployment process:
 - 20:07:30.923 Starting SFDX: Deploy This Source to Org
 - === Deployed Source
 - Table with 4 columns: STATE, FULL NAME, TYPE, PROJECT PATH
 - Table content:

STATE	FULL NAME	TYPE	PROJECT PATH
Created	LockEmergencySafe	ApexClass	force-app\main\default\classes\LockEmergencySafe.cls
Created	LockEmergencySafe	ApexClass	force-app\main\default\classes\LockEmergencySafe.cls-meta.xml
 - 20:07:39.342 Ended SFDX: Deploy This Source to Org

12. Test Classes

- All Apex logic is covered by test classes with minimum 85% coverage.
- Test classes use `@isTest` annotation and include multiple methods for different scenarios.
- Test data is created within test methods to ensure isolation.
- Async logic is tested using `Test.startTest()` and `Test.stopTest()` blocks.
- Example: `EmergencyAppointmentTriggerTest`, validates `Queueable`, `Future`, and `Batch` logic.

force-app > main > default > classes > EmergencyAppointmentTriggerTest.cls > EmergencyAppointmentTriggerTest > testEmergencyTrigger() : void

```
2 private class EmergencyAppointmentTriggerTest {
3     @isTest static void testEmergencyTrigger() {
4         Appointment__c appt = new Appointment__c(
5             Type__c = 'Emergency',
6             Appointment_Time__c = DateTime.now()
7         );
8         insert appt;
9
10        Appointment__c result = [SELECT Status__c FROM Appointment__c WHERE Id = :appt.Id];
11        System.assertEquals('Locked', result.Status__c);
12    }
13 }
14
```

00:35:50.730 Ended SFDX: Run Apex Tests

NAME	VALUE
Outcome	Failed
Tests Ran	1
Pass Rate	0%
Fail Rate	100%
Skip Rate	0%
Test Run Id	707gL00000EpH3k
Test Setup Time	0 ms
Test Execution Time	764 ms
Test Total Time	764 ms
Org Id	00DgL00000BA9B1UAL
Username	ppalakyyadav2004101@agentforce.com

File Edit Selection View Go Run ... HealthCare360

EXPLORER

- pointmentHandler.cls
- EmergencyTrigger.trigger
- EmergencyAppointmentTriggerHandler.cls
- EmergencyAppointmentQueueTest.cls

OPEN EDITORS

- Untitled-1
- EmergencyAppointmentHandler.c...
- EmergencyTrigger.trigger force-a...
- EmergencyAppointmentTriggerH...
- EmergencyAppointmentQueueTe...
- EmergencyAppointmentQueueTest...
- EmergencyAppointmentTrigger.tr...
- EmergencyAppointmentTriggerTe...
- EmergencyAppointmentQueue.d...

HEALTHCARE360

- force-app \ main \ default
- classes
- EmergencyAppointmentQueueTest.cls
- EmergencyAppointmentQueueTest...
- EmergencyAppointmentQueueTest...
- EmergencyAppointmentTriggerHan...
- EmergencyAppointmentTriggerHan...
- EmergencyAppointmentTriggerTest...

PROBLEMS

Filter (e.g. text, **/*.ts, !**/node_m...

No problems have been detected in the workspace.

TIMELINE

RUNNING TASKS

force-app > main > default > classes > EmergencyAppointmentQueueTest.cls > ...

```
2 private class EmergencyAppointmentQueueTest {
3     @isTest static void testQueueableExecution() {
4         // Step 3: Run Queueable logic
5         Test.startTest();
6         System.enqueueJob(new EmergencyAppointmentQueue(appt.Id));
7         Test.stopTest();
8
9         // Step 4: Validate result
10        Appointment__c result = [SELECT Status__c FROM Appointment__c WHERE Id = :appt.Id];
11        System.assertEquals('Locked', result.Status__c);
12    }
13 }
14
```

18:41:05.435 Ended SFDX: Run Apex Tests

NAME	VALUE
Outcome	Passed
Tests Ran	1
Pass Rate	100%
Fail Rate	0%
Skip Rate	0%
Test Run Id	707gL00000EsQs0
Test Setup Time	0 ms
Test Execution Time	449 ms
Test Total Time	449 ms
Org Id	00DgL00000BA9B1UAL
Username	ppalakyyadav2004101@agentforce.com

Ln 27, Col 1 Spaces: 4 UTF-8 LF Apex

•

