# □ 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.

---

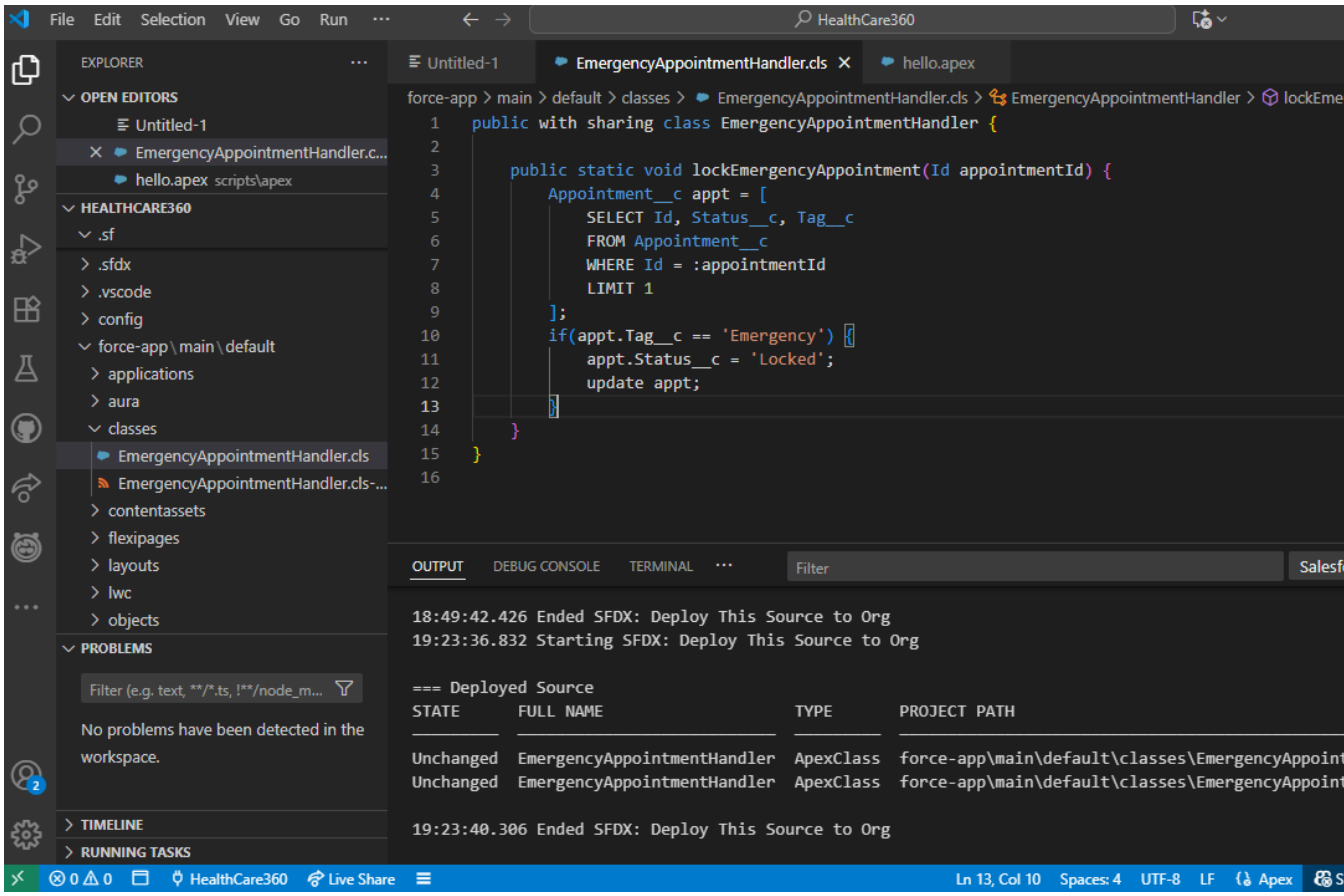
## □ Phase Overview

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.

- 

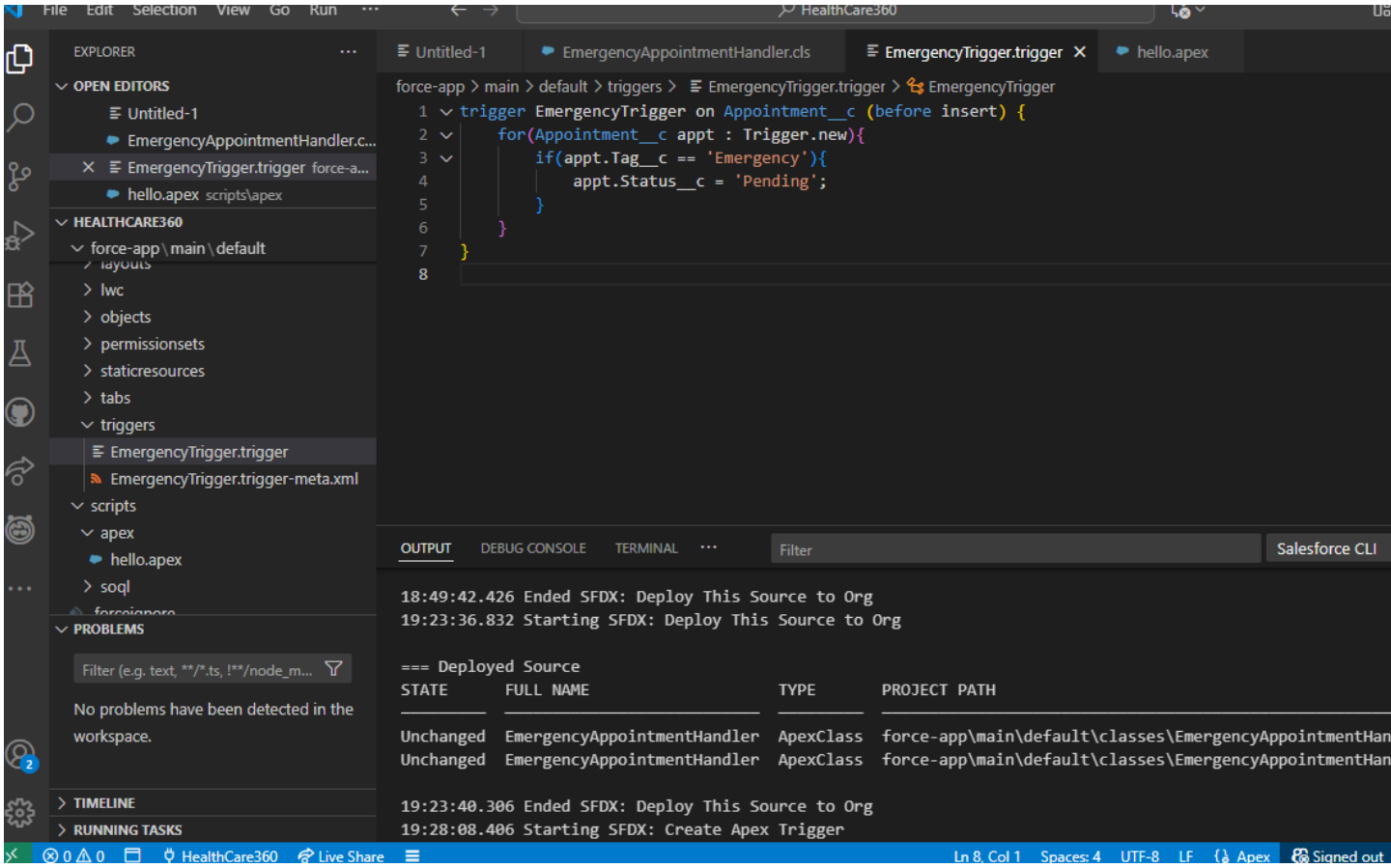
-

---

## 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.

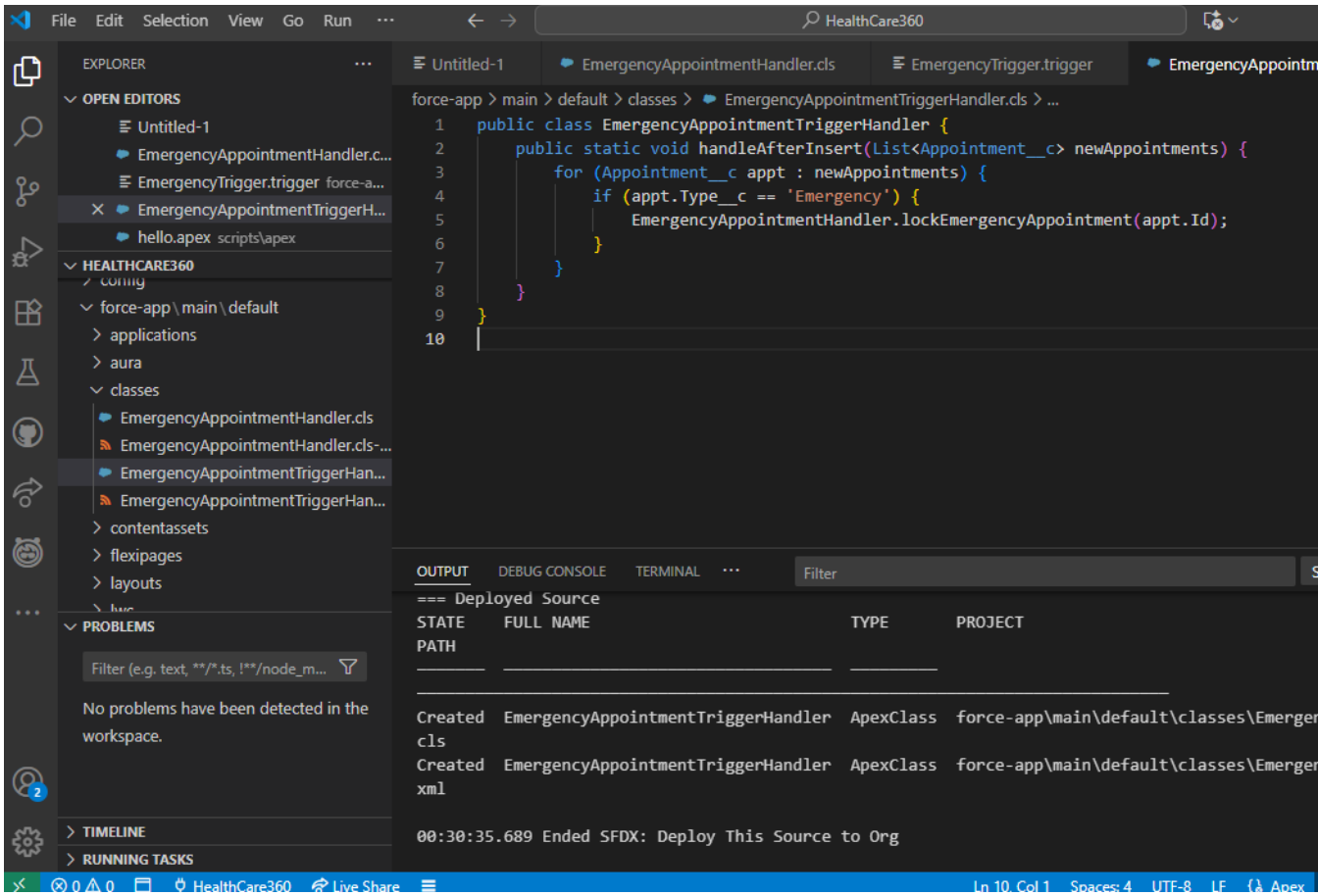



- 

---

# 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.

- 

  - 

---

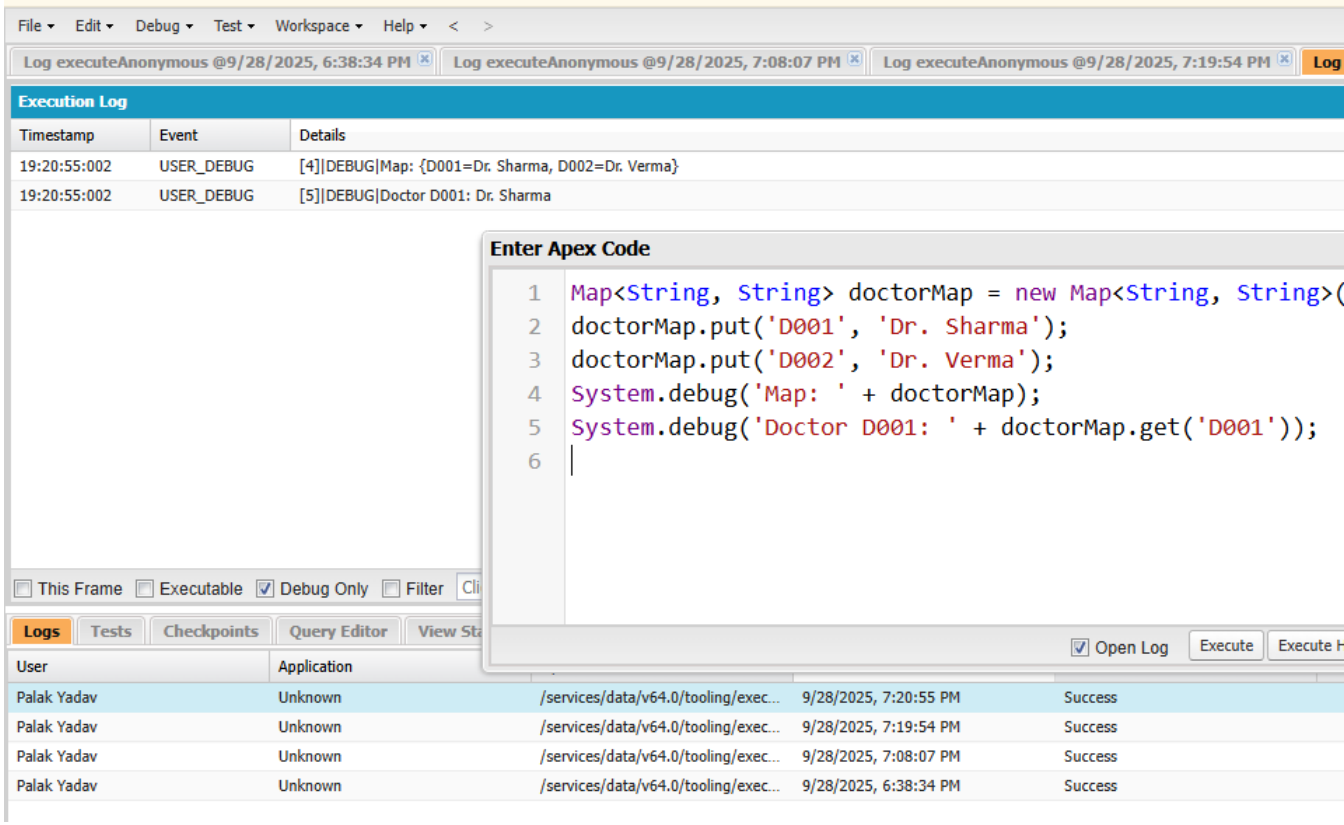
# 4. SOQL & SOSL

- **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.



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**

```
Set<String> uniqueTypes = new Set<String>();
uniqueTypes.add('Emergency');
uniqueTypes.add('Routine');
uniqueTypes.add('Emergency'); // Duplicate ignored
System.debug('Set: ' + uniqueTypes);
```

This Frame | Executable | Debug Only | Filter | Open Log | Execute | Execute High

Logs | Tests | Checkpoints | Query Editor | View Sta

| User | Application | | | |
|---|---|---|---|---|
| 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**

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

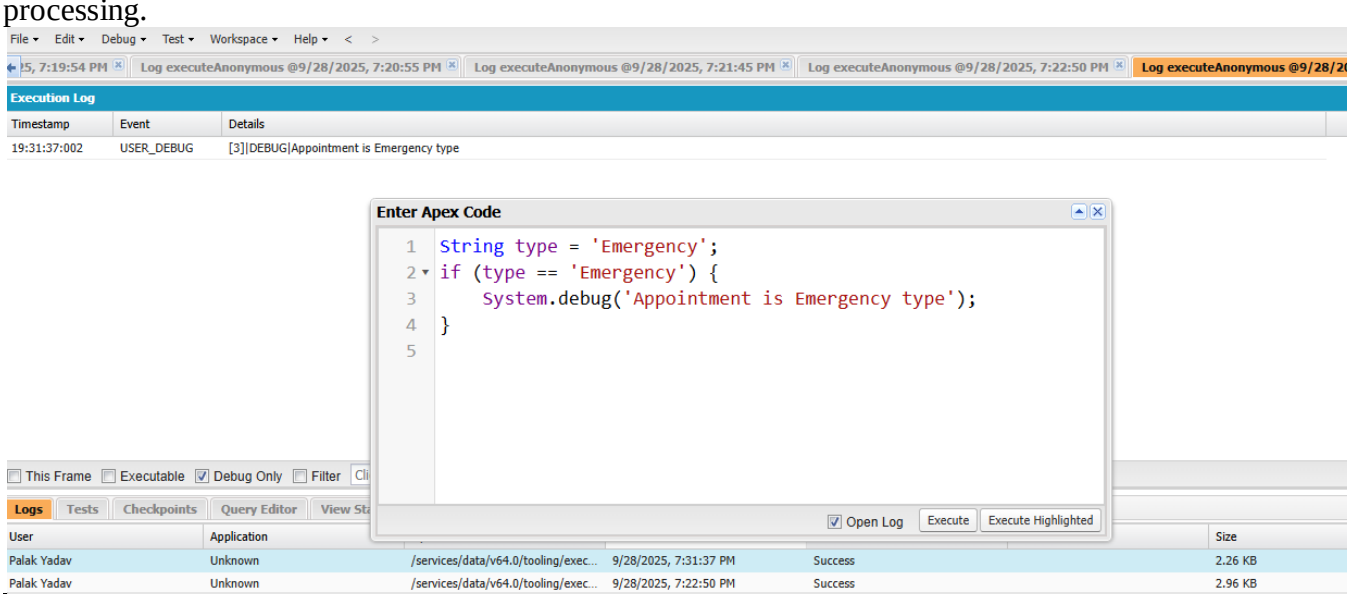
This Frame | Executable | Debug Only | Filter | Open Log | Execute | Execute High

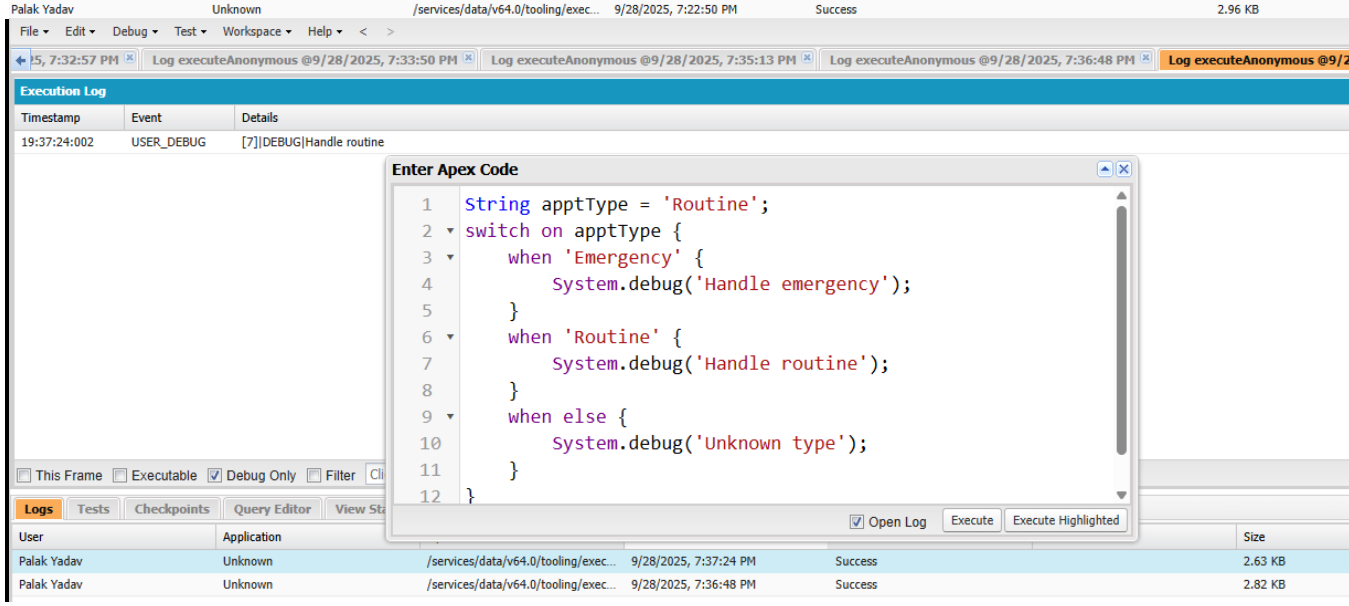
Logs | Tests | Checkpoints | Query Editor | View Sta

| User | Application | | | |
|---|---|---|---|---|
| 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.



File Edit Debug Test Workspace Help

25, 7:31:37 PM | Log executeAnonymous @9/28/2025, 7:32:57 PM | Log executeAnonymous @9/28/2025, 7:33:50 PM | Log executeAnonymous @9/28/2025, 7:35:13 P

**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**

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

Open Log | Execute | Execute Highlighted

This Frame | Executable | Debug Only | Filter | Cli

Logs | Tests | Checkpoints | Query Editor | View Sta

| User | Application | | | |
|---|---|---|---|---|
| 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 |

File Edit Debug Test Workspace Help

25, 7:22:50 PM | Log executeAnonymous @9/28/2025, 7:31:37 PM | Log executeAnonymous @9/28/2025, 7:32:57 PM | Log executeAnonymous @9/28/2025, 7:33:50 PM | Log e

**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**

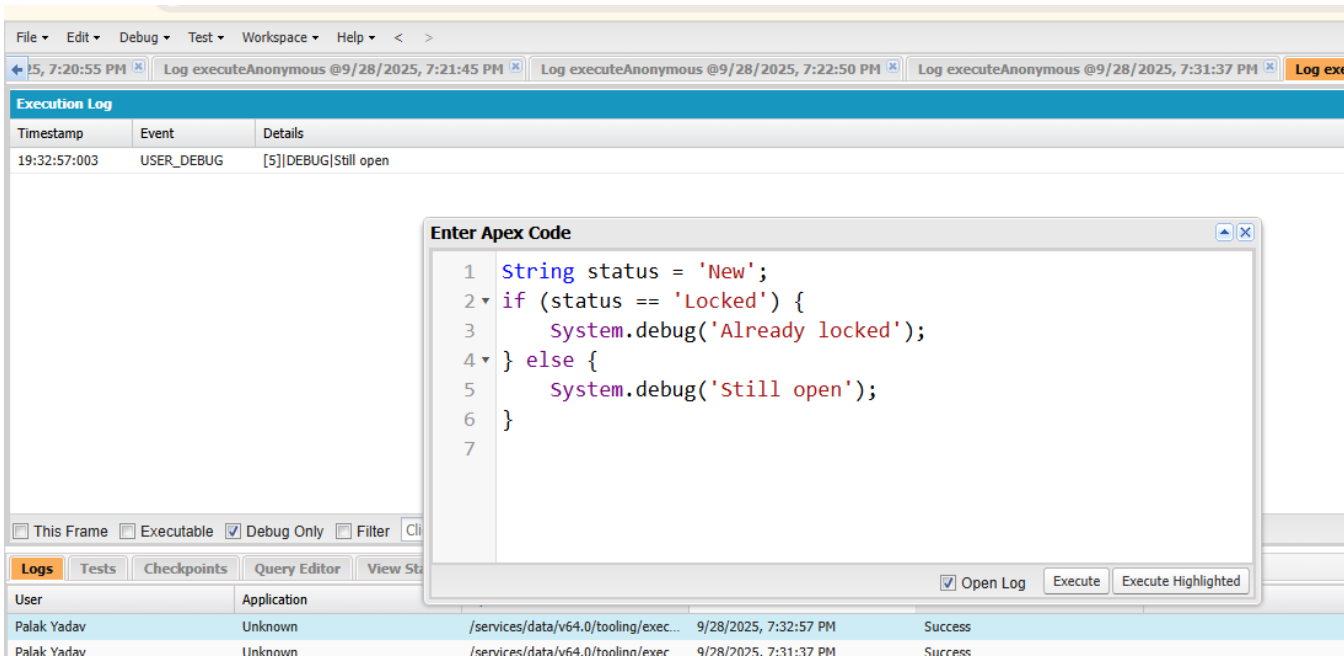
```
Integer i = 0;
while (i < 3) {
    System.debug('Count: ' + i);
    i++;
}
```

Open Log | Execute | Execute Highlighted

This Frame | Executable | Debug Only | Filter | Cli

Logs | Tests | Checkpoints | Query Editor | View Sta

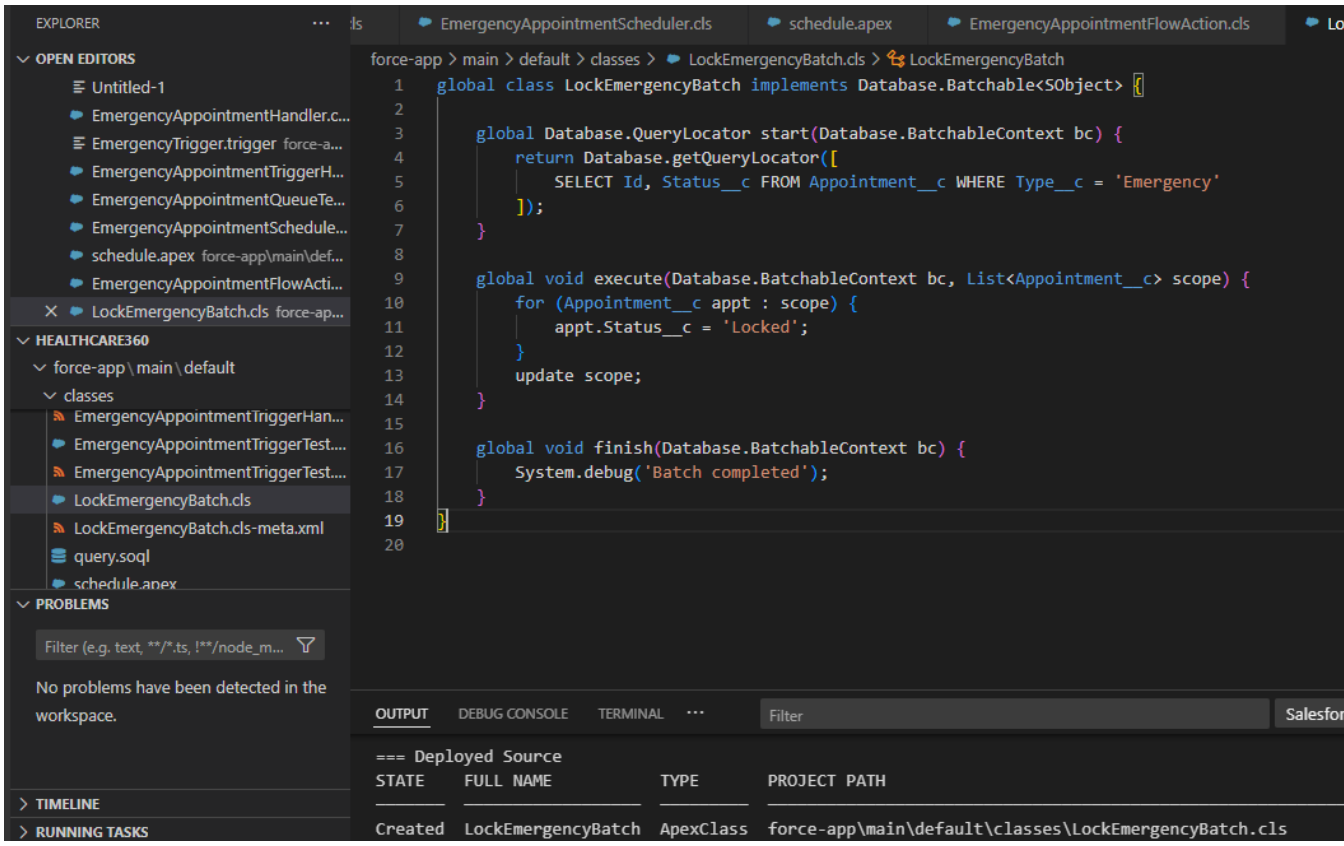
| User | Application | | | |
|---|---|---|---|---|
| Palak Yadav | Unknown | /services/data/v64.0/tooling/exec... | 9/28/2025, 7:35:13 PM | Success |
| Palak Yadav | Unknown | /services/data/v64.0/tooling/exec... | 9/28/2025, 7:33:50 PM | Success |
| Palak Yadav | Unknown | /services/data/v64.0/tooling/exec...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




---

# 7. Batch Apex

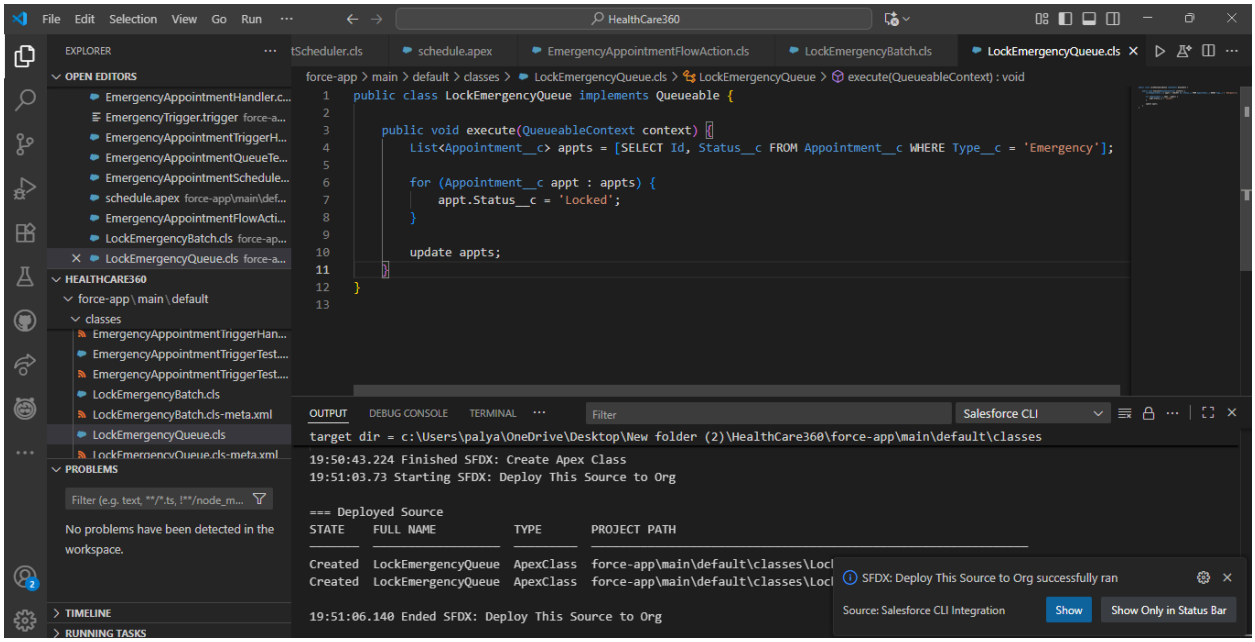
- Batch Apex is used to process large volumes of data asynchronously.
- Class implements `Database.Batchable<SObject>` and defines `start`, `execute`, and `finish` methods.
- Used for bulk updates, cleanup jobs, and scheduled data maintenance.
- Example: `LockEmergencyBatch` updates all emergency appointments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

- 


---

# 8. Queueable Apex

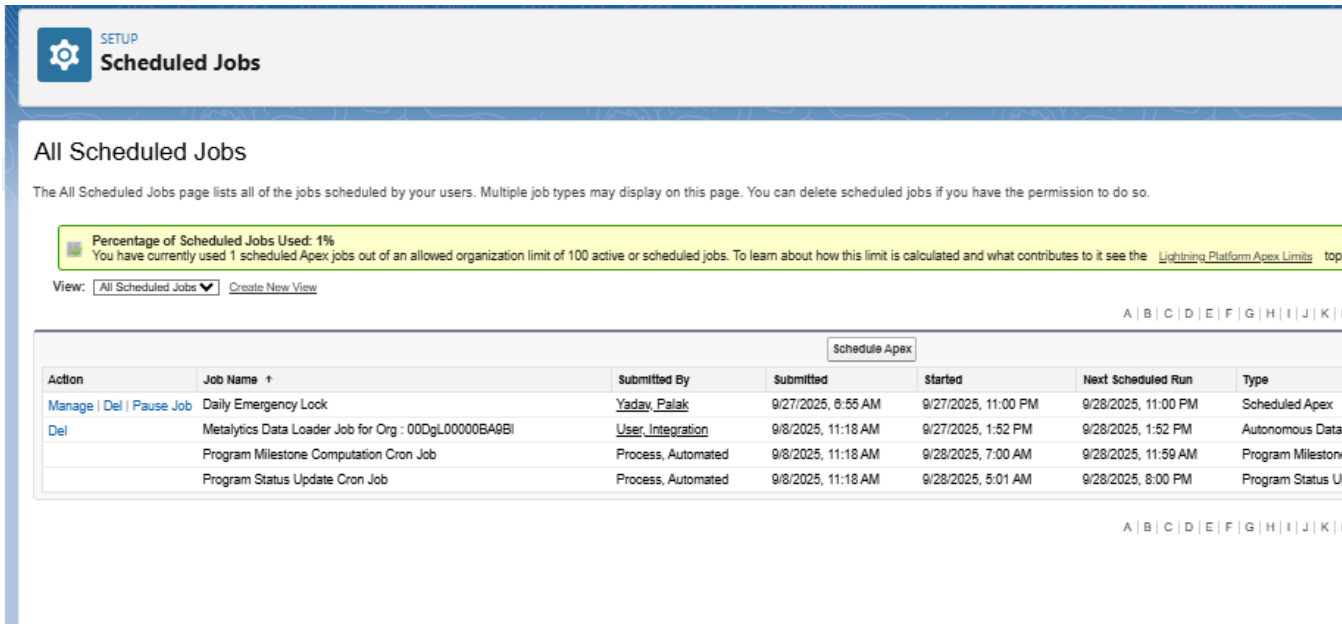
- Queueable Apex is used for flexible asynchronous processing with support for chaining.
- Class implements `Queueable` and defines `execute()` method.
- Ideal for background jobs that need partial retry or multi-step logic.
- Example: `LockEmergencyQueue` updates emergency appointments without blocking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




---

# 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>)`

EmergencyAppointmentScheduler.cls | LockScheduler.cls | LockEmergencyFuture.cls | schedule.apex | EmergencyAppointm

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

```
public class LockEmergencyFuture {
    @future
    public static void lockAppointmentsAsync(List<Id> apptIds) {
        List<Appointment__c> appts = [SELECT Id, Status__c FROM Appointment__c WHERE Id IN :apptIds];

        for (Appointment__c appt : appts) {
            appt.Status__c = 'Locked';
        }

        update appts;
    }
}
```

OUTPUT | DEBUG CONSOLE | TERMINAL | Filter | Salesforce CLI

```
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.

EmergencyAppointmentScheduler.cls | LockScheduler.cls | LockEmergencyFuture.cls | LockEmergencySafe.cls | schedule.ape

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

```apex
public class LockEmergencySafe {

    public static void lockAppointments() {
        try {
            List<Appointment__c> appts = [SELECT Id, Status__c FROM Appointment__c WHERE Type__c = 'Emergency']

            for (Appointment__c appt : appts) {
                appt.Status__c = 'Locked';
            }

            update appts;

        } catch (DmlException e) {
            System.debug('DML Error: ' + e.getMessage());
        } catch (Exception e) {
            System.debug('General Error: ' + e.getMessage());
        }
    }
}
```

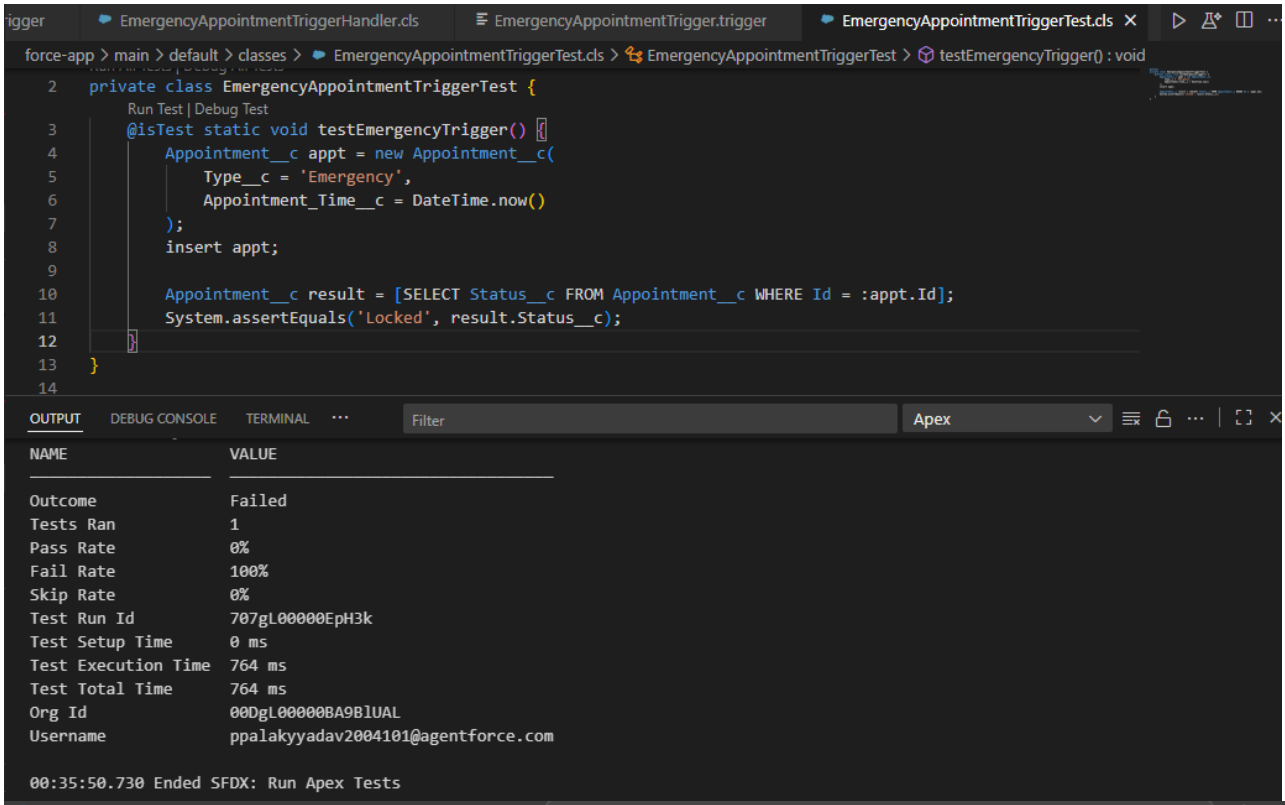
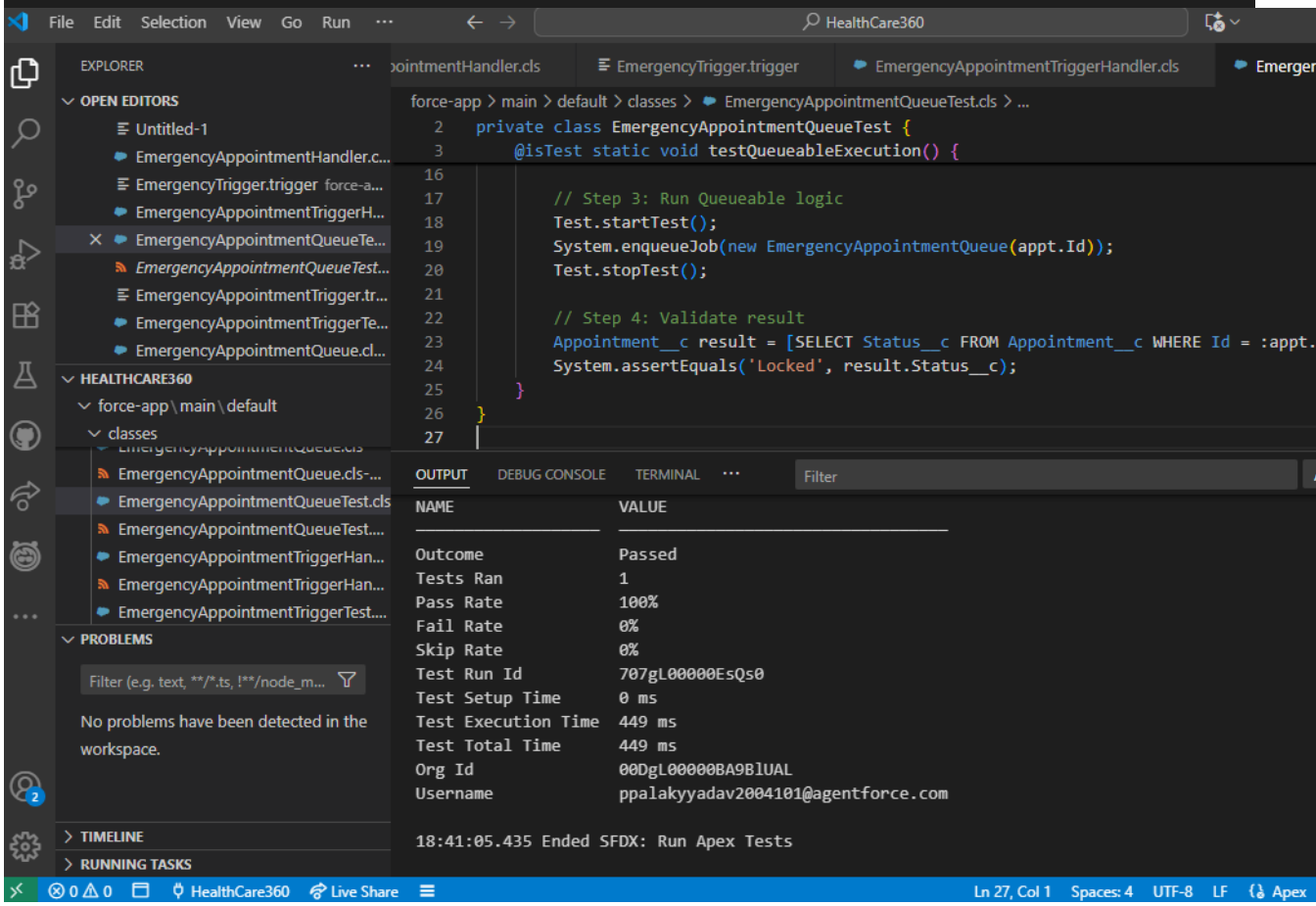
OUTPUT | DEBUG CONSOLE | TERMINAL | Filter | Salesforce CLI

```
20:07:30.923 Starting SFDX: Deploy This Source to Org

=== Deployed Source
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.

- 
EmergencyAppointmentTriggerHandler.cls
EmergencyAppointmentTrigger.trigger
EmergencyAppointmentTriggerTest.cls
force-app > main > default > classes > EmergencyAppointmentTriggerTest.cls > EmergencyAppointmentTriggerTest > testEmergencyTrigger() : void
private class EmergencyAppointmentTriggerTest {
Run Test | Debug Test
@isTest static void testEmergencyTrigger() {
Appointment__c appt = new Appointment__c(
Type__c = 'Emergency',
Appointment_Time__c = DateTime.now()
);
insert appt;
Appointment__c result = [SELECT Status__c FROM Appointment__c WHERE Id = :appt.Id];
System.assertEquals('Locked', result.Status__c);
}
}
OUTPUT
DEBUG CONSOLE
TERMINAL
Filter
Apex
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 00DgL00000BA9BlUAL
Username ppalakyyadav2004101@agentforce.com
00:35:50.730 Ended SFDX: Run Apex Tests
- 
File Edit Selection View Go Run
HealthCare360
EXPLORER
OPEN EDITORS
Untitled-1
EmergencyAppointmentHandler.c...
EmergencyTrigger.trigger force-a...
EmergencyAppointmentTriggerH...
EmergencyAppointmentQueueTe...
EmergencyAppointmentQueueTest...
EmergencyAppointmentTrigger.tr...
EmergencyAppointmentTriggerTe...
EmergencyAppointmentQueue.cl...
HEALTHCARE360
force-app\main\default
classes
EmergencyAppointmentQueue.cls-...
EmergencyAppointmentQueueTest.cls
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 > ...
private class EmergencyAppointmentQueueTest {
@isTest static void testQueueableExecution() {
// Step 3: Run Queueable logic
Test.startTest();
System.enqueueJob(new EmergencyAppointmentQueue(appt.Id));
Test.stopTest();
// Step 4: Validate result
Appointment__c result = [SELECT Status__c FROM Appointment__c WHERE Id = :appt.
System.assertEquals('Locked', result.Status__c);
}
}
OUTPUT
DEBUG CONSOLE
TERMINAL
Filter
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 00DgL00000BA9BlUAL
Username ppalakyyadav2004101@agentforce.com
18:41:05.435 Ended SFDX: Run Apex Tests
HealthCare360
Live Share
Ln 27, Col 1
Spaces: 4
UTF-8
LF
Apex

•