

# Phase 5 - Apex Programming

- PROJECT: DEEPCODE CRM: INTELLIGENT SALESFORCE INTEGRATION PROJECT

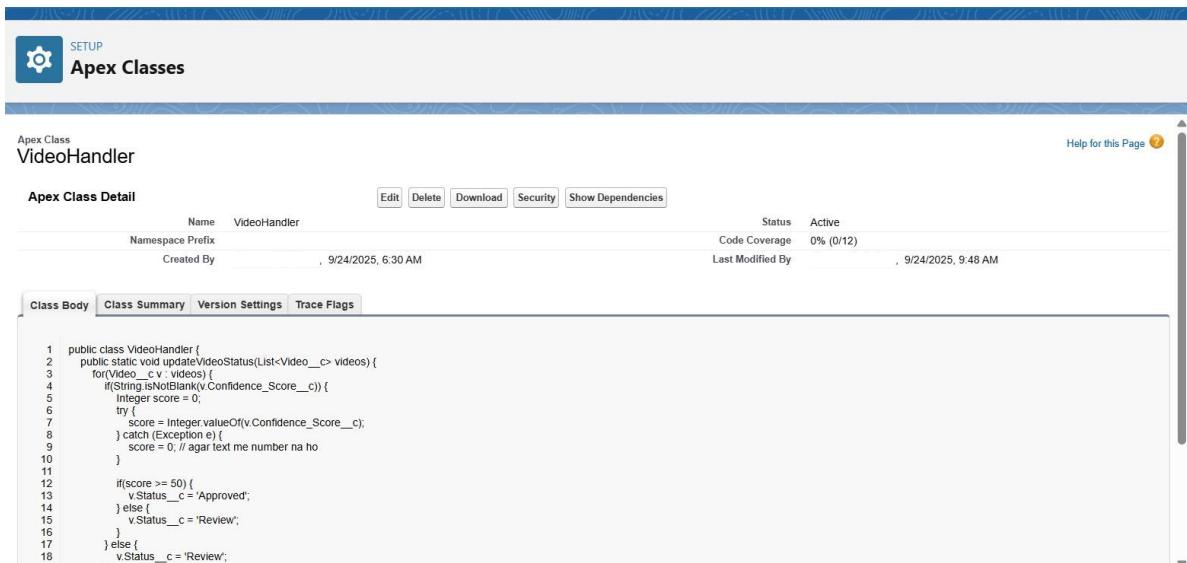
## ■ 1. Apex Classes & Objects

**Definition:** Apex Classes are templates that define **custom business logic**. Objects are Salesforce database tables, and fields are columns.

### Key Points:

- Classes contain **methods** (functions) and **variables**
- Encapsulate logic → reusable across triggers, Visualforce, Lightning

### Video Handler:



The screenshot shows the Apex Class Detail page for the class 'VideoHandler'. The page has a header with 'SETUP' and 'Apex Classes'. Below the header, it says 'Apex Class VideoHandler'. There are tabs for 'Edit', 'Delete', 'Download', 'Security', and 'Show Dependencies'. The 'Status' is 'Active' and 'Code Coverage' is '0% (0/12)'. The 'Created By' and 'Last Modified By' sections show the same user with the date '9/24/2025, 6:30 AM' and '9/24/2025, 9:48 AM'. At the bottom, there are tabs for 'Class Body', 'Class Summary', 'Version Settings', and 'Trace Flags'. The 'Class Body' tab is selected, showing the following Apex code:

```
1 public class VideoHandler {  
2     public static void updateVideoStatus(List<Video__c> videos) {  
3         for(Video__c v : videos) {  
4             if(String.isNotBlank(v.Confidence_Score__c)) {  
5                 Integer score = 0;  
6                 try {  
7                     score = Integer.valueOf(v.Confidence_Score__c);  
8                 } catch (Exception e) {  
9                     score = 0; // agar text me number na ho  
10                }  
11                if(score >= 50) {  
12                    v.Status__c = 'Approved';  
13                } else {  
14                    v.Status__c = 'Review';  
15                }  
16            } else {  
17                v.Status__c = 'Review';  
18            }  
19        }  
20    }  
21}
```

## Alert Handler:

The screenshot shows the Salesforce Apex Classes page. At the top, there's a navigation bar with a gear icon labeled "SETUP" and a "Apex Classes" button. Below the bar, the page title is "AlertHandler". On the right, there's a "Help for this" link. The main area is titled "Apex Class Detail" for "AlertHandler". It shows basic information: Name (AlertHandler), Namespace Prefix (empty), Status (Active), Created By (empty), Code Coverage (0% / 0/10), Last Modified By (empty), and Last Modified (empty). Below this, there are tabs for "Class Body", "Class Summary", "Version Settings", and "Trace Flags". The "Class Body" tab is selected, displaying the following Apex code:

```
1 public class AlertHandler {  
2     // Method to send alert based on type  
3     public static void processAlerts(List<Alert__c> alerts) {  
4         for(Alert__c a : alerts) {  
5             if(a.Alert_Type__c == 'System') {  
6                 a.Status__c = 'Pending Review';  
7             } else if(a.Alert_Type__c == 'Video') {  
8                 a.Status__c = 'Check Video';  
9             } else {  
10                 a.Status__c = 'User Action Required';  
11             }  
12         }  
13         update alerts;  
14     }  
15     // Fetch alerts by type  
16     public static List<Alert__c> getAlertsByType(String alertType) {  
17         return [SELECT Id, Name, Alert_Type__c FROM Alert__c WHERE Alert_Type__c = :alertType];  
18     }  
19 }  
20 }
```

## All Apex Classes :

The screenshot shows the "All Apex Classes" page. At the top, there's a navigation bar with a gear icon labeled "SETUP" and a "Apex Classes" button. A green message box displays "Percent of Apex Used: 0.03%" and notes that 1,881 characters of Apex code are in use out of a limit of 6,000,000. Below the message, there are buttons for "Estimate your organization's code coverage" and "Compile all classes". The "View:" dropdown is set to "All" and has a "Create New View" option. The main area is a table with columns: Action, Name, Namespace Prefix, Api Version, Status, Size Without Comments, Last Modified By, and Has Trace Flags. The table contains three rows:

Action	Name	Namespace Prefix	Api Version	Status	Size Without Comments	Last Modified By	Has Trace Flags
Edit   Del   Security	AlertHandler		64.0	Active	617	, 9/24/2025, 9:35 AM	<input type="checkbox"/>
Edit   Del   Security	UtilityClass		64.0	Active	481	, 9/24/2025, 9:37 AM	<input type="checkbox"/>
Edit   Del   Security	VideoHandler		64.0	Active	666	, 9/24/2025, 9:48 AM	<input type="checkbox"/>

**Tip:** Always use bulk-safe logic – process multiple records using lists.

## ■ 2. Apex Triggers

**Definition:** Triggers automate logic before or after record events (insert, update, delete).

**Trigger Design Pattern:**

- Trigger calls Handler Class
- Keeps logic centralized & maintainable
- Supports bulkification

### Video Trigger

```
trigger VideoTrigger on Video_c (before insert, before update) {  
    VideoHandler.updateVideoStatus(Trigger.new);  
}
```

The screenshot shows the Apex Trigger Detail page for a trigger named "VideoTrigger". The page includes the trigger code, its details (Name: VideoTrigger, sObject Type: Video, Status: Active), and a tab bar with "Apex Trigger" selected.

Name	Code Coverage	sObject Type
VideoTrigger	0% (0/1)	Video
Created By	, 9/24/2025, 9:39 AM	Status
Namespace Prefix		Last Modified By
		, 9/24/2025, 9:41 AM

Trigger Code:

```
trigger VideoTrigger on Video_c (before insert, before update) {  
    VideoHandler.updateVideoStatus(Trigger.new);  
}
```

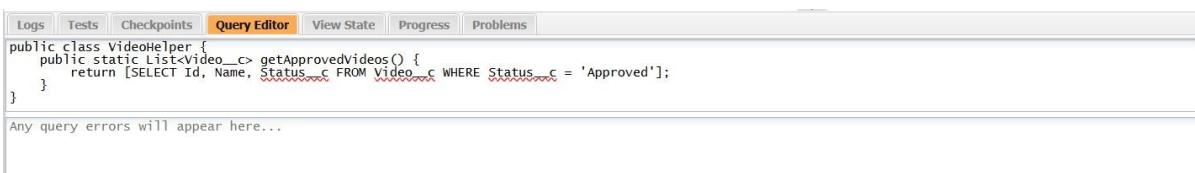
■ Use **before triggers** to update fields before saving, **after triggers** for related objects.

### ■ 3. SOQL & SOSL

**SOQL (Salesforce Object Query Language):**

- Fetch records from a **single object**
- Can filter, order, and aggregate

```
List<Video_c> approvedVideos = [SELECT Id, Name, Status_c  
                                FROM Video_c  
                                WHERE Status_c = 'Approved'];
```



```
Logs Tests Checkpoints Query Editor View State Progress Problems  
public class VideoHelper {  
    public static List<Video_c> getApprovedVideos() {  
        return [SELECT Id, Name, Status_c FROM Video_c WHERE Status_c = 'Approved'];  
    }  
}  
Any query errors will appear here...
```

**SOSL (Salesforce Object Search Language):**

- Search **text across multiple objects/fields**
- Returns a **list of lists of sObjects**

```
List<List<sObject>> results = [FIND 'Tutorial*' IN ALL FIELDS  
                                RETURNING Video_c(Id, Name, Title_c)];  
List<Video_c> videosFound = (List<Video_c>)results[0];
```

■ **Tip:** Use SOSL for quick text search, SOQL for specific field filtering.

---

### ■ 4. Collections: List, Set, Map

1. **List** – Ordered, allows duplicates
2. **Set** – Unique elements, unordered
3. **Map** – Key-Value pairs, keys unique

**Example:**

```
List<Video_c> videos = [SELECT Id, Name FROM Video_c];  
Set<Id> videoIds = new Set<Id>();  
Map<Id, Video_c> videoMap = new Map<Id, Video_c>(videos);
```

■ Use collections for **bulk-safe triggers** and to avoid **SOQL inside loops**.

---

## ■ 5. Batch Apex

**Purpose:** Process large datasets asynchronously in chunks (batches).

**Structure:**

1. start → Fetch records
2. execute → Process batch
3. finish → Post-processing

### Video Batch Update

```
global class VideoBatchUpdate implements Database.Batchable<SObject> {

    // 1 Start - Query records
    global Database.QueryLocator start(Database.BatchableContext BC) {
        return Database.getQueryLocator(
            'SELECT Id, Confidence_Score__c, Status__c FROM Video__c'
        );
    }

    // 2 Execute - Process batch
    global void execute(Database.BatchableContext BC, List<Video__c> scope) {
        for(Video__c v : scope) {
            Integer score = 0;
            try {
                score = Integer.valueOf(v.Confidence_Score__c);
            } catch(Exception e){
                score = 0;
            }
            v.Status__c = (score >= 50) ? 'Approved' : 'Review';
        }
    }
}
```

Tests Checkpoints Query Editor View State Progress Problems

**Run:**

```
Database.executeBatch(new VideoBatchUpdate(), 200);
```

■ **Tip:** Batch Apex is essential for 50,000+ records.

---

## ■ 6. Asynchronous Apex

- **Future Methods:** Run in background

```
@future
```

```
public static void sendEmailAsync(String email) { ... }
```

- **Queueable Apex:** Supports chaining jobs

```
public class QueueableExample implements Queueable { ... }
```

- **Scheduled Apex:** Cron-like scheduling

```
global class ScheduledJob implements Schedulable { ... }
```

■ Use async apex for **heavy processing** without hitting governor limits.

---

## ■ 7. Exception Handling

- **Use try-catch to prevent runtime errors**



```
public class VideoExceptionHandler {
    public static void updateVideoStatus(List<Video__c> videos) {
        for(Video__c v : videos){
            try {
                Integer score = Integer.valueOf(v.Confidence_Score__c);
                v.Status__c = (score >= 50) ? 'Approved' : 'Review';
            } catch(Exception e){
                System.debug('Error processing Video: ' + v.Name + ' - ' + e.getMessage());
                v.Status__c = 'Error';
            }
        }
        update videos;
    }
}
```

The screenshot shows the Salesforce Apex code editor. At the top, there is a trigger definition:

```
trigger VideoTriggerException on Video__c (before insert, before update) {  
    VideoExceptionHandler.updateVideoStatus(Trigger.new);  
}
```

Below it is a test method:

```
Enter Apex Code  
1 List<Video__c> testVideos = [SELECT Id, Name, Confidence_Score__c, Status__c FROM Video__c];  
2 VideoExceptionHandler.updateVideoStatus(testVideos);  
3
```

At the bottom of the editor window, there are three buttons: "Open Log", "Execute", and "Execute Highlighted".

- Always log errors for debugging.

## 8. Test Classes

Purpose: Validate logic, ensure **code coverage ≥75%**

### Video Handler Test

```
@IsTest  
public class VideoHandlerTest {  
    static testMethod void testUpdateVideoStatus() {  
        Video__c v1 = new Video__c(Name='V1', Confidence_Score__c='60');  
        Video__c v2 = new Video__c(Name='V2', Confidence_Score__c='40');  
  
        Test.startTest();  
        insert new List<Video__c>{v1, v2};  
        Test.stopTest();  
  
        System.assertEquals('Approved', [SELECT Status__c FROM Video__c WHERE Id=:v1.Id].Status__c);  
        System.assertEquals('Review', [SELECT Status__c FROM Video__c WHERE Id=:v2.Id].Status__c);  
    }  
}
```

Other Test Classes: Alert Helper Test, Utility Class Test (same format)

## Overall Code Coverage

Class	Percent	Lines
<b>Overall</b>	<b>94%</b>	17/17
VideoHander	100%	17/17
AlertHander	77%	14/18
UtilityClass	<b>100%</b>	15/15

---

### G. Best Practices

- Always **bulkify triggers**
- Avoid SOQL/DML inside loops
- Use **Handler classes** for logic
- Write **robust test classes** with positive/negative scenarios
- Use **Async Apex** for large or delayed processing