

Phase 5: Apex Programming (Developer)

Apex is Salesforce's object-oriented programming language that allows developers to write complex business logic and automate processes that go beyond declarative tools (like Flows or Process Builder). It runs on the Salesforce platform and helps implement customized, scalable, and efficient solutions.

1 Apex Classes & Utility Classes

Purpose:

- Create reusable code to handle business logic.
- Automate notifications, calculations, or custom logic.

Use Cases:

- Create a utility class to calculate recycled product cost.
- Build classes for automated notifications or custom business logic.

2 Apex Triggers

Purpose:

- Automatically execute Apex code when records are created, updated, deleted, or undeleted.

Trigger Types:

Type	Purpose	Example Use Case
Before Insert/Update	Validate or modify records before saving	Ensure total waste collected is correct before insert
After Insert/Update/Delete	Execute logic after records are saved	Notify recycling center when order status changes

The screenshot shows the Salesforce Developer Console interface. The top navigation bar includes links for Recently Viewed, Spam, Student, No Change in Payment, Developer Console, Salesorg setup, Replaxt-Innovation, and a plus sign for new tabs. Below the navigation is a toolbar with icons for back, forward, search, and refresh.

The main area displays two Apex classes:

```

1  public class EnrollmentPaymentHandler {
2
3    public static void handleBeforeInsert(List<Enrollment__c> newEnrollments){
4      for(Enrollment__c e : newEnrollments){
5        if(e.Payment_Status__c == 'Paid'){
6          e.Status_Message__c = 'Your payment is successfully received!';
7        } else if(e.Payment_Status__c == 'Pending'){
8          e.Status_Message__c = 'Your payment is pending. Please complete it soon.';
9        } else if(e.Payment_Status__c == 'Failed'){
10          e.Status_Message__c = 'Your payment has failed. Please try again.';
11        } else if(e.Payment_Status__c == 'Refunded'){
12          e.Status_Message__c = 'Your payment has been refunded successfully.';
13        }
14      }
15    }
16
17    public static void handleBeforeUpdate(List<Enrollment__c> updatedEnrollments, Map<Id, Enrollment__c> oldMap){
18      for(Enrollment__c e : updatedEnrollments){
19        Enrollment__c oldE = oldMap.get(e.getId());
20        if(oldE != null && oldE.Payment_Status__c != e.Payment_Status__c){
21          e.Status_Message__c = 'Status has been updated';
22        }
23      }
24    }
}

```

Below the code editor is a progress bar showing deployment status for various runs, with a total duration of 400 ms. The progress bar includes columns for Request ID, Nice Order, Description, Status, Start, End, Duration (ms), Handler, Err, Ajax, Error, and Delay.

ReqId	Nice Order	Description	Status	Start	End	Duration (ms)	Handler	Err	Ajax	Error	Delay
12	0	Getting members of ApexClassMember for containerId=1dgc5000000ME17AAG	Finished	8:13:54	8:13:54	400					
11	-2	Getting deployment for id=1dgc500000002NKA2	Finished	8:13:53	8:13:54	301					
10	-1	Creating deployment for containerId=1dgc5000000ME17AAG Save=false runTests=false	Finished	8:13:52	8:13:52	260					
9	-1	Creating or Updating containerMember for containerId=1dgc5000000ME17AAG	Finished	8:13:51	8:13:52	235					
8	-1	Creating deployment for containerId=1dgc5000000ME17AAG Save=false runTests=false	Group Failed	8:13:40	8:13:40	0					
7	-1	Creating or Updating containerMember for containerId=1dgc5000000ME17AAG	Failed	8:13:39	8:13:40	213					

At the bottom of the screen, there is a system status bar showing the date (19-10-2025), time (10:02 PM), and weather information (26°C, Mostly clear). A taskbar with various application icons is also visible.

3 Trigger Design Pattern

Purpose:

- Maintain clean, reusable, and scalable trigger logic.

Best Practices:

- Only one trigger per object.
- Delegate logic to handler classes.
- Ensure triggers are bulkified (handle multiple records efficiently).

Use Case:

- Maintain organized logic for Plastic Waste automation through a central handler class.

4 SOQL & SOSL

Purpose:

- SOQL: Fetch data from one or more Salesforce objects.
- SOSL: Search data across multiple objects and fields.

Use Cases:

- Retrieve all orders placed by a specific customer (SOQL).
- Search for a keyword in multiple objects (SOSL).

5 Collections

Purpose:

- Store and manage multiple data values efficiently.

Collection Description	Example Use Case
List	Ordered collection of elements List of Orders or Waste records
Set	Unordered, unique elements Unique Waste Types
Map	Key-value pairs Map of Customer ID → Order List

6 Control Statements

Purpose:

- Execute code conditionally or repeatedly.

Common Statements:

- Conditional: if, else, switch
 - Loops: for, while, do-while
 - Others: break, continue

7 Asynchronous Processing

Purpose:

- Perform background tasks that don't block the main execution thread.

Types & Use Cases:

Type	Purpose	Example Use Case
Batch Apex	Process large datasets asynchronously	Update thousands of recycled product records nightly
Queueable Apex	Run async operations with more complex logic	Send confirmation emails after order approval
Scheduled Apex	Run Apex code at defined times	Generate weekly recycled product reports

Type	Purpose	Example Use Case
Future Methods	Execute code asynchronously, useful for callouts	Notify customers about shipment updates

8 Exception Handling

Purpose:

- Catch and handle errors gracefully to prevent transaction failures.

Use Cases:

- Handle record insert/update exceptions.
- Log errors for admin review.

9 Test Classes

Purpose:

- Validate Apex code and ensure proper functionality before deployment.
- Salesforce requires at least 75% code coverage for production deployment.

Use Cases:

- Test triggers, classes, and async logic.
- Simulate user actions and validate outcomes.