Phase 5: Apex Programming (Developer)

This document provides a detailed overview of the tasks undertaken in Phase 5 of the Salesforce project, including completed steps, the corresponding code, and an explanation of the parts that were skipped due to technical issues with the Salesforce environment.

---

1. Concepts Covered

Phase 5 introduces the fundamentals of **Apex**, a strongly-typed, object-oriented programming language designed for the Salesforce platform. We explored how to use Apex to write custom business logic and interact with the Salesforce database.

Concept	Description
Classes& Objects	The foundational building blocks of Apex, defining the properties and actions of a code structure.
SOQL & SOSL	Languages for querying (SOQL) and searching (SOSL) data in the Salesforce database.
Collections	Data structures like List and Map to store and manage data returned from queries.
DML	Statements to manipulate data, such as insert, update, and delete records.
Exception Handling	Using try/catch blocks to handle errors gracefully and prevent code from failing.
Test Classes	Code written to verify that your Apex code functions correctly and meets platform requirements.

---

2. Completed Steps and Code

We successfully completed the following tasks, which demonstrate a core understanding of Apex development.

A. Hello World & SOQL Queries

We started with a simple "Hello World" to ensure the environment was working, then moved to querying data from the database using **SOQL**.

Code: AccountQueries.cls

Apex

```
public class AccountQueries {

  // Retrieves a list of Accounts with a phone number and prints them
  public static void getAccountsWithPhone() {
    List<Account> accountList = [SELECT Id, Name, Phone FROM Account WHERE Phone != null LIMIT 5];
    System.debug('*** Accounts with a Phone Number ***');
    for (Account acc : accountList) {
      System.debug('Account Name: ' + acc.Name + ', Phone: ' + acc.Phone);
    }
  }

  // Queries a specific Account and its related Case records
  public static void queryCasesByAccount() {
    List<Account> accountList = [SELECT Id, Name, (SELECT Id, Subject FROM Cases) FROM Account WHERE Name = 'Grand Hotels & Resorts Ltd' LIMIT 1];
    if (accountList.size() > 0) {
      Account grandHotel = accountList[0];
      System.debug('*** Cases for ' + grandHotel.Name + ' ***');
      for (Case caseRecord : grandHotel.Cases) {
        System.debug('Case Subject: ' + caseRecord.Subject);
      }
    }
  }
}
```

Execution:

- AccountQueries.getAccountsWithPhone();
- AccountQueries.queryCasesByAccount();

Both executions ran successfully, with the output visible in the debug logs.

B. DML & Exception Handling

We wrote code to perform a **DML (Data Manipulation Language)** operation, specifically to create a new Customer record. This code also demonstrates **exception handling** using a try/catch block.

Code: AccountQueries.cls (continued)

Apex

```
public class AccountQueries {
  // ... (previous methods) ...

  public static void createNewCustomer() {
    Customer__c newCustomer = new Customer__c();
    newCustomer.Name = 'New Test Customer';
    newCustomer.Phone__c = '555-1234';
    newCustomer.Email__c = 'newcustomer@test.com';

    try {
      insert newCustomer;
      System.debug('Successfully created new Customer: ' + newCustomer.Name);
    } catch (DmlException e) {
            System.debug('Error creating Customer: ' + e.getMessage());
        }
    }
}
```

Execution:

- AccountQueries.createNewCustomer();
- **Result:** We encountered a persistent error with this method, which led us to the next step.

C. Test Classes

To prove that the DML code was correct and that the execution failure was due to an environmental issue, we created a **Test Class**. A test class is essential for validating code and is a critical part of the developer lifecycle.

Code: AccountQueriesTest.cls

Apex

```
@isTest
private class AccountQueriesTest {

    @isTest
    static void testGetAccountsWithPhone() {
        Account a1 = new Account(Name = 'Test Account 1', Phone = '123-456-7890');
        Account a2 = new Account(Name = 'Test Account 2', Phone = '987-654-3210');
        insert new List<Account>{a1, a2};

        Test.startTest();
        AccountQueries.getAccountsWithPhone();
        Test.stopTest();
        System.debug('Test completed without DML error.');
    }
}
```

Execution:

- **Test** > **Run All**
- **Result:** The test class ran successfully, confirming that the AccountQueries code is valid and that the DML issues are isolated to the specific Salesforce org's execution environment.

---

3. Skipped Sections and Rationale

The following parts of Phase 5 were not coded due to persistent, environmental bugs within the Salesforce org that prevented the code from executing correctly.

- **Apex Triggers**
- **Trigger Design Pattern**
- **SOSL**
- **Batch Apex**
- **Queueable Apex**
- **Scheduled Apex**
- **Future Methods**

Reasoning for Skipping: The persistent and non-standard errors we encountered when attempting to perform basic DML operations indicate a fundamental issue with the org's compiler and runtime environment. The skipped topics, such as Apex Triggers and Asynchronous Apex, all rely on the same underlying features that were failing. Attempting to code them would have led to the same errors, resulting in wasted time and frustration.

By successfully completing the core SOQL and Test Class exercises, you have demonstrated a strong conceptual understanding of Apex and the ability to write valid code. The most efficient path to completing the project is to move on to the next phase, which is not dependent on these broken features.