Apex Triggers

Get Started with apex Triggers

Create an Apex trigger

Create an Apex trigger that sets an account's Shipping Postal Code to match the Billing Postal Code if the Match Billing Address option is selected. Fire the trigger before inserting an account or updating an account.

After the completion of the pre-work i.e., To add a checkbox field to the account object, let's create an apex trigger that completes our basic need.

Trigger class: AccountAddressTrigger.apxt

```
trigger AccountAddressTrigger on Account (before insert,before update) {
    for(Account account:Trigger.New){
        if(account.Match_Billing_Address__c == True){
            account.ShippingPostalCode = account.BillingPostalCode;
        }
    }
}
```

Apex Triggers

Bulk apex Triggers

Create a Bulk Apex trigger

Create a bulkified Apex trigger that adds a follow-up task to an opportunity if its stage is Closed Won. Fire the Apex trigger after inserting or updating an opportunity.

Trigger class: ClosedOpportunityTrigger.apxt

```
trigger ClosedOpportunityTrigger on Opportunity (after insert, after update) {
    List<Task> tasklist = new List<Task>();
    for(Opportunity opp: Trigger.New){
        if(opp.StageName == 'Closed won'){
            tasklist.add(new Task(Subject = 'Follow Up Test Task', WhatId = opp.Id));
        }
    }
    if(tasklist.size()>0){
        insert tasklist;
    }
}
```

Apex Testing

Get Started with Apex Unit Tests

Create a Unit Test for a Simple Apex Class

Create and install a simple Apex class to test if a date is within a proper range, and if not, returns a date that occurs at the end of the month within the range. You'll copy the code for the class from GitHub. Then write unit tests that achieve 100% code coverage.

Apex class: VerifyDate.apxc

```
public class VerifyDate {



        //method to handle potential checks against two dates

        public static Date CheckDates(Date date1, Date date2) {

                //if date2 is within the next 30 days of date1, use date2.  Otherwise use the end of the month

                if(DateWithin30Days(date1,date2)) {

                        return date2;

                } else {

                        return SetEndOfMonthDate(date1);

                }

        }
```

```
    //method to check if date2 is within the next 30 days of date1
    @TestVisible private static Boolean DateWithin30Days(Date date1, Date date2) {
        //check for date2 being in the past
    if( date2 < date1) { return false; }
    //check that date2 is within (>=) 30 days of date1
    Date date30Days = date1.addDays(30); //create a date 30 days away from date1
        if( date2 >= date30Days ) { return false; }
        else { return true; }
    }

    //method to return the end of the month of a given date
    @TestVisible private static Date SetEndOfMonthDate(Date date1) {
        Integer totalDays = Date.daysInMonth(date1.year(), date1.month());
        Date lastDay = Date.newInstance(date1.year(), date1.month(), totalDays);
        return lastDay;
    }

}
```

Apex Testing

Test Apex Triggers Unit

Create a Unit Test for a Simple Apex Trigger

Create and install a simple Apex trigger which blocks inserts and updates to any contact with a last name of 'INVALIDNAME'. You'll copy the code for the class from GitHub. Then write unit tests that achieve 100% code coverage.

Apex class: RestrictContactByName.apxc

```
trigger RestrictContactByName on Contact (before insert, before update) {

	//check contacts prior to insert or update for invalid data

	For (Contact c : Trigger.New) {

		if(c.LastName == 'INVALIDNAME') {	//invalidname is invalid

			c.AddError('The Last Name "'+c.LastName+'" is not allowed for DML');

		}

	}

}
```

Apex Testing

Create Test Data for Apex Tests

Create a Contact Test Factory

Create an Apex class that returns a list of contacts based on two incoming parameters: the number of contacts to generate and the last name. Do not insert the generated contact records into the database.

Apex class: RandomContactFactory.apxc

```
public class RandomContactFactory {

    public static List<Contact> generateRandomContacts(Integer numcnt, string lastname){

        List<Contact> contacts = new List<contact>();

        for(Integer i = 0;i < numcnt;i++){

            Contact cnt = new Contact(FirstName = 'Test'+i, Lastname = lastname);

            Contacts.add(cnt);

        }

        return contacts;

            }

}
```

Asynchronous apex

Use Future Methods

Create an Apex class that uses the @future annotation to update Account records.

Create an Apex class with a future method that accepts a List of Account IDs and updates a custom field on the Account object with the number of contacts associated to the Account. Write unit tests that achieve 100% code coverage for the class. Every hands-on challenge in this module asks you to create a test class.

let's complete the pre-work of creating a field on the Account object.

Apex class: AccountProcessor.apxc

```
public class AccountProcessor {
   @future
   public static void countContacts(List<Id> accountIds){
      List<Account> accList = [Select Id,Number_Of_Contacts__c, (Select Id from
Contacts) from Account where Id in : accountIds ];
      For(Account acc : accList){
         acc.Number_Of_Contacts__c = acc.Contacts.size();
      }
      update accList;
   }
}
```

`Apex test class:AccountProcessorTest.apxc`

```
@isTest
public class AccountProcessorTest {
   public static testmethod void testAccountProcessor(){
      Account a = new Account();
      a.Name = 'Test Account';
      insert a;
      Contact con = new Contact();
      con.FirstName = 'Binary';
      con.LastName = 'programming';
      con.AccountId = a.Id;
      insert con;
      List<Id> accListId = new List<Id>();
      accListId.add(a.Id);
      Test.startTest();
      AccountProcessor.countContacts(accListId);
      Test.stopTest();
      Account acc = [select Number_Of_Contacts__c from Account where Id =: a.Id];
      system.assertEquals(Integer.valueOf(acc.Number_Of_Contacts__c), 1);
   }
}
```

Asynchronous apex

Use Batch Apex Unit

Create an Apex class that uses Batch Apex to update Lead records.

Create an Apex class that implements the Database.Batchable interface to update all Lead records in the org with a specific LeadSource.

Apex class: LeadProcessor.apxc

```
global class LeadProcessor implements    Database.Batchable<Sobject>
{
   global Database.QueryLocator start(Database.BatchableContext bc)
   {
      return Database.getQueryLocator([Select LeadSource From Lead ]);
   }

   global void execute(Database.BatchableContext bc, List<Lead> scope)
   {
         for (Lead Leads : scope)
         {
            Leads.LeadSource = 'Dreamforce';
         }
      update scope;
   }

   global void finish(Database.BatchableContext bc){   }
}
```

Apex test class: LeadProcessorTest.apxc

```
@isTest
public class LeadProcessorTest
{
    static testMethod void testMethod1()
    {
        List<Lead> lstLead = new List<Lead>();
        for(Integer i=0 ;i <200;i++)
        {
            Lead led = new Lead();
            led.FirstName ='FirstName';
            led.LastName ='LastName'+i;
            led.Company ='demo'+i;
            lstLead.add(led);
        }
        insert lstLead;
        Test.startTest();

            LeadProcessor obj = new LeadProcessor();
            DataBase.executeBatch(obj);
        Test.stopTest();
    }
}
```

Asynchronous apex

Control Processes with Queueable Apex

Create a Queueable Apex class that inserts Contacts for Accounts.

Create a Queueable Apex class that inserts the same Contact for each Account for a specific state.

`Apex class:` AddPrimaryContact.apxc

```
public class AddPrimaryContact implements Queueable
{
   private Contact c;
   private String state;
   public  AddPrimaryContact(Contact c, String state)
   {
      this.c = c;
      this.state = state;
   }
   public void execute(QueueableContext context)
   {
       List<Account> ListAccount = [SELECT ID, Name ,(Select id,FirstName,LastName
from contacts ) FROM ACCOUNT WHERE BillingState = :state LIMIT 200];
       List<Contact> lstContact = new List<Contact>();
       for (Account acc:ListAccount)
       {
            Contact cont = c.clone(false,false,false,false);
            cont.AccountId =  acc.id;
            lstContact.add( cont );
       }
       if(lstContact.size() >0 )
       {
          insert lstContact;
       }
```

```
  }

}
```

Apex test class: `AddPrimaryContactTest.apxc`

```
@isTest
public class AddPrimaryContactTest
{
     @isTest static void TestList()
     {
         List<Account> Teste = new List <Account>();
         for(Integer i=0;i<50;i++)
         {
             Teste.add(new Account(BillingState = 'CA', name =
'Test'+i));
         }
         for(Integer j=0;j<50;j++)
         {
             Teste.add(new Account(BillingState = 'NY', name =
'Test'+j));
         }
         insert Teste;

         Contact co = new Contact();
         co.FirstName='demo';
         co.LastName ='demo';
         insert co;
         String state = 'CA';
          AddPrimaryContact apc = new AddPrimaryContact(co,
state);
          Test.startTest();
            System.enqueueJob(apc);
          Test.stopTest();
      }
 }
```

Asynchronous apex

Schedule Jobs Using the Apex Scheduler

Create an Apex class that uses Scheduled Apex to update Lead records.

Create an Apex class that implements the Schedulable interface to update Lead records with a specific LeadSource. (This is very similar to what you did for Batch Apex.)

`Apex class:` DailyLeadProcessor.apxc

```
global class DailyLeadProcessor implements Schedulable{
   global void execute(SchedulableContext ctx){
      List<Lead> leads = [SELECT Id, LeadSource FROM Lead WHERE LeadSource = ''];
      if(leads.size() > 0){
         List<Lead> newLeads = new List<Lead>();
         for(Lead lead : leads){
            lead.LeadSource = 'DreamForce';
            newLeads.add(lead);
         }
         update newLeads;
      }
   }
}
```

Apex test class: DailyLeadProcessorTest.apxc

```
@isTest
private class DailyLeadProcessorTest{
   //Seconds Minutes Hours Day_of_month Month Day_of_week optional_year
   public static String CRON_EXP = '0 0 0 2 6 ? 2022';
   static testmethod void testScheduledJob(){
      List<Lead> leads = new List<Lead>();
      for(Integer i = 0; i < 200; i++){
         Lead lead = new Lead(LastName = 'Test ' + i, LeadSource = '', Company = 'Test
Company ' + i, Status = 'Open - Not Contacted');
```

```
            leads.add(lead);
        }
        insert leads;
        Test.startTest();
        // Schedule the test job
        String jobId = System.schedule('Update LeadSource to DreamForce', CRON_EXP,
new DailyLeadProcessor());
        // Stopping the test will run the job synchronously
        Test.stopTest();
    }
}
```

Apex Integration Services

Apex REST Callouts Unit

Create an Apex class that calls a REST endpoint and write a test class.

Create an Apex class that calls a REST endpoint to return the name of an animal, write unit tests that achieve 100% code coverage for the class using a mock response, and run your Apex tests.

Prework: Be sure the Remote Sites from the first unit are set up.

Apex class: AnimalLocator.apxc

```
public class AnimalLocator

{



  public static String getAnimalNameById(Integer id)

  {

      Http http = new Http();

      HttpRequest request = new HttpRequest();

      request.setEndpoint('https://th-apex-http-callout.herokuapp.com/animals/'+id);

      request.setMethod('GET');

      HttpResponse response = http.send(request);
```

```
        String strResp = '';
        system.debug('******response '+response.getStatusCode());
        system.debug('******response '+response.getBody());
      // If the request is successful, parse the JSON response.
      if (response.getStatusCode() == 200)
      {
         // Deserializes the JSON string into collections of primitive data types.
        Map<String, Object> results = (Map<String, Object>)
JSON.deserializeUntyped(response.getBody());
         // Cast the values in the 'animals' key as a list
        Map<string,object> animals = (map<string,object>) results.get('animal');
         System.debug('Received the following animals:' + animals );
         strResp = string.valueof(animals.get('name'));
         System.debug('strResp >>>>>>' + strResp );
      }
      return strResp ;
   }
}
```

Apex test class: AnimalLocatorTest.apxc

```
@isTest
private class AnimalLocatorTest{
    @isTest static  void AnimalLocatorMock1() {
        Test.SetMock(HttpCallOutMock.class, new AnimalLocatorMock());
        string result=AnimalLocator.getAnimalNameById(3);
        string expectedResult='chicken';
        System.assertEquals(result, expectedResult);
    }
}
```

Apex Integration Services

Apex SOAP Callouts Unit

Generate an Apex class using WSDL2Apex and write a test class.

Generate an Apex class using WSDL2Apex for a SOAP web service, write unit tests that achieve 100% code coverage for the class using a mock response, and run your Apex tests.

Prework: Be sure the Remote Sites from the first unit are set up.

Apex class: ParkLocator.apxc

```
public class ParkLocator {
    public static String[] country(String country){
        ParkService.ParksImplPort parks = new ParkService.ParksImplPort();
        String[] parksname = parks.byCountry(country);
        return parksname;
    }
}
```

Apex test class: ParkLocatorTest.apxc

```
@isTest
private class ParkLocatorTest{
    @isTest
    static void testParkLocator() {
        Test.setMock(WebServiceMock.class, new ParkServiceMock());
        String[] arrayOfParks = ParkLocator.country('India');
        System.assertEquals('Park1', arrayOfParks[0]);
    }
}
```

Apex Integration Services

Apex Web Services Unit

Create an Apex REST service that returns an account and its contacts.

Create an Apex REST class that is accessible at /Accounts/<Account_ID>/contacts. The service will return the account's ID and name plus the ID and name of all contacts associated with the account. Write unit tests that achieve 100% code coverage for the class and run your Apex tests.

Prework: Be sure the Remote Sites from the first unit are set up.

Apex class: AccountManager.apxc

```
@RestResource(urlMapping='/Accounts/*/contacts')

global with sharing class AccountManager{

    @HttpGet

    global static Account getAccount(){

        RestRequest req = RestContext.request;

        String accId = req.requestURI.substringBetween('Accounts/', '/contacts');

        Account acc = [SELECT Id, Name, (SELECT Id, Name FROM Contacts)
```

```
                FROM Account WHERE Id = :accId];
        return acc;
    }
}
```

Apex test class: AccountManagerTest.apxc

```
@IsTest
private class AccountManagerTest{
    @isTest static void testAccountManager(){
        Id recordId = getTestAccountId();
        // Set up a test request
        RestRequest request = new RestRequest();
        request.requestUri =
            'https://ap5.salesforce.com/services/apexrest/Accounts/'+ recordId +'/contacts';
        request.httpMethod = 'GET';
        RestContext.request = request;
```

```
    // Call the method to test
    Account  acc = AccountManager.getAccount();
    // Verify results
    System.assert(acc != null);
}
private static Id getTestAccountId(){
    Account acc = new Account(Name = 'TestAcc2');
    Insert acc;
    Contact con = new Contact(LastName = 'TestCont2', AccountId = acc.Id);
    Insert con;
    return acc.Id;
```