Apex Specialist Superbadge

1. Automate record creation using Apex Triggers:

MaintenanceRequestHelper.apxc

```
public with sharing class MaintenanceRequestHelper {

  public static void updateworkOrders(List<Case> updWorkOrders, Map<Id,Case>
nonUpdCaseMap) {

    // TODO: Complete the method to update workorders
      Set<Id> Ids = new Set<Id>();

    for (Case c : updWorkOrders){
      if (nonUpdCaseMap.get(c.Id).Status != 'Closed' && c.Status == 'Closed'){
        if (c.Type == 'Repair' || c.Type == 'Routine Maintenance'){
          Ids.add(c.Id);
        }
      }
    }

    if (!Ids.isEmpty()){
      List<Case> newCase = new List<Case>();
      Map<Id,Case> closedCasesM = new Map<Id,Case>([SELECT Id, Vehicle__c,
Equipment__c, Equipment__r.Maintenance_Cycle__c,(SELECT Id,Equipment__c,Quantity__c
FROM Equipment_Maintenance_Items__r)
                                    FROM Case WHERE Id IN :Ids]);
      Map<Id,Decimal> maintenanceCycles = new Map<ID,Decimal>();
      AggregateResult[] results = [SELECT Maintenance_Request__c,
MIN(Equipment__r.Maintenance_Cycle__c)cycle FROM Equipment_Maintenance_Item__c
WHERE Maintenance_Request__c IN :Ids GROUP BY Maintenance_Request__c];

    for (AggregateResult ar : results){
      maintenanceCycles.put((Id) ar.get('Maintenance_Request__c'), (Decimal) ar.get('cycle'));
    }

      for(Case cc : closedCasesM.values()){
        Case nc = new Case (
```

```
            ParentId = cc.Id,
          Status = 'New',
            Subject = 'Routine Maintenance',
            Type = 'Routine Maintenance',
            Vehicle__c = cc.Vehicle__c,
            Equipment__c =cc.Equipment__c,
            Origin = 'Web',
            Date_Reported__c = Date.Today()

          );

          If (maintenanceCycles.containskey(cc.Id)){
            nc.Date_Due__c = Date.today().addDays((Integer) maintenanceCycles.get(cc.Id));
          }

          newCase.add(nc);
        }

        insert newCase;

        List<Equipment_Maintenance_Item__c> clonedWPs = new
List<Equipment_Maintenance_Item__c>();
        for (Case nc : newCase){
          for (Equipment_Maintenance_Item__c wp :
closedCasesM.get(nc.ParentId).Equipment_Maintenance_Items__r){
            Equipment_Maintenance_Item__c wpClone = wp.clone();
            wpClone.Maintenance_Request__c = nc.Id;
            ClonedWPs.add(wpClone);

          }
        }
        insert ClonedWPs;
      }
  }
}
```

MaitenanceRequest.apxt

```
trigger MaintenanceRequest on Case (before update, after update) {
  // ToDo: Call MaintenanceRequestHelper.updateWorkOrders
```

```
    if(Trigger.isUpdate && Trigger.isAfter){

      MaintenanceRequestHelper.updateWorkOrders(Trigger.New, Trigger.OldMap);
}
}
```

2. Synchronize Salesforce data with an external system using asynchronous REST callouts:

WarehouseCalloutService.apxc

```
public with sharing class WarehouseCalloutService {
   private static final String WAREHOUSE_URL = 'https://th-superbadge-
apex.herokuapp.com/equipment';
   @future(callout=true)
   public static void runWarehouseEquipSync(){
      Http ht=new Http();
      HttpRequest req=new HttpRequest();
      req.setEndpoint(WAREHOUSE_URL);
      req.setMethod('GET');
      HttpResponse res=ht.send(req);
      List<Product2> li=new List<Product2>();
      if(res.getStatusCode()==200){
         List<object> jsonRes=(List<Object>)JSON.deserializeUntyped(res.getBody());
         System.debug(res.getBody());
         for(Object obj:jsonRes){
            Map<String,Object> mp=(Map<String,Object>)obj;
            Product2 equip=new Product2();
            equip.ProductCode=(String)mp.get('_id');
            equip.Name=(String)mp.get('name');
            equip.Replacement_Part__c=(Boolean)mp.get('replacement');
            equip.Current_Inventory__c=(Double)mp.get('quantity');
            equip.Maintenance_Cycle__c=(Integer)mp.get('maintenanceperiod');
            equip.Lifespan_Months__c=(Integer)mp.get('lifespan');
            equip.Cost__c=(Integer)mp.get('cost');
```

```
            equip.Warehouse_SKU__c=(String)mp.get('sku');
            li.add(equip);
        }
        if(li.size()>0){
            upsert li;
            System.debug('Equipment Synched');
        }
    }
  }
  public static void execute(QueueableContext con){
    runWarehouseEquipSync();
  }
}
```

3. Schedule synchronization using Apex Code:

WarehouseSyncShedule.apxc

```
public with sharing class WarehouseSyncSchedule implements Schedulable{
  public static void execute(SchedulableContext con){
    WarehouseCalloutService.runWarehouseEquipSync();
  }
}
```

4. Test automation logic to confirm Apex trigger side effects:

MaintenanceRequestHelperTest.apxc

```
@isTest
public with sharing class MaintenanceRequestHelperTest {
  private static Vehicle__c createVeh(){
    Vehicle__c v=new Vehicle__c(name='Super Car');
    return v;
```

```
    }
    private static Product2 createEquip(){
        Product2 eq=new Product2(Name='Equipment
1',Lifespan_Months__c=10,Maintenance_Cycle__c=10,Replacement_Part__c=true);
        return eq;
    }
    private static Case createMaintenanceReq(Id vId, Id eqId){
        case c = new
case(Type='Repair',Status='New',Origin='Web',Subject='Test',Equipment__c=eqId,Vehicle
__c=vId);
        return c;
    }
    private static Equipment_Maintenance_Item__c createPart(Id eqId,Id reqId){
        Equipment_Maintenance_Item__c part = new
Equipment_Maintenance_Item__c(Equipment__c = eqId,Maintenance_Request__c =
reqId);
        return part;
    }
    @istest
    private static void testMaintenanceReqPos(){
        Vehicle__c veh = createVeh();
        insert veh;
        Id vehicleId = veh.Id;

        Product2 eq = createEquip();
        insert eq;
        Id equipmentId = eq.Id;

        case ca = createMaintenanceReq(vehicleId,equipmentId);
        insert ca;

        Equipment_Maintenance_Item__c work = createPart(equipmentId,ca.Id);
        insert work;
        test.startTest();
        ca.status = 'Closed';
        update ca;
        test.stopTest();
```

```
        Case newC = [Select Id, subject, type, Equipment__c, Date_Reported__c, Vehicle__c,
Date_Due__c from case where status =:'New'];
        System.assert(work != null);
        System.assert(newC.Subject != null);
        System.assertEquals(newC.Type, 'Routine Maintenance');
        System.assertEquals(newC.Equipment__c, equipmentId);
        System.assertEquals(newC.Vehicle__c, vehicleId);
        System.assertEquals(newC.Date_Reported__c, system.today());

    }
    @istest
    private static void testMaintenanceReqNeg(){
        Vehicle__C veh = createVeh();
        insert veh;
        id vehicleId = veh.Id;

        product2 equip = createEquip();
        insert equip;
        Id equipmentId = equip.Id;

        case empty = createMaintenanceReq(vehicleId,equipmentId);
        insert empty;

        Equipment_Maintenance_Item__c work = createPart(equipmentId, empty.Id);
        insert work;
        test.startTest();
        empty.Status = 'Working';
        update empty;
        test.stopTest();
        list<Case> cases = [select Id from case];

        Equipment_Maintenance_Item__c part = [select Id from
Equipment_Maintenance_Item__c where Maintenance_Request__c = :empty.Id];

        system.assert(work != null);
        system.assert(cases.size() == 1);
```

```
    }
    @istest
    private static void testMaintenanceReqBulk(){
        List<Vehicle__C> vehList = new List<Vehicle__C>();
        List<Product2> equipList = new List<Product2>();
        List<Equipment_Maintenance_Item__c> workList = new
List<Equipment_Maintenance_Item__c>();
        List<Case> reqList = new List<Case>();
        List<Id> oldReqIds = new List<Id>();

        for(Integer i = 0; i < 300; i++){
          vehList.add(createVeh());
           equipList.add(createEquip());
        }
        insert vehList;
        insert equipList;
         for(Integer i = 0; i < 300; i++){
            reqList.add(createMaintenanceReq(vehList.get(i).Id, equipList.get(i).Id));
        }
        insert reqList;
        for(Integer i = 0; i < 300; i++){
            workList.add(createPart(equipList.get(i).Id, reqList.get(i).Id));
        }
        insert workList;
        test.startTest();
        for(case c : reqList){
            c.Status = 'Closed';
            oldReqIds.add(c.Id);
        }
        update reqList;
        test.stopTest();
        List<Case> li = [select Id from Case where Status =:'New'];
        List<Equipment_Maintenance_Item__c> parts = [select Id from
Equipment_Maintenance_Item__c where Maintenance_Request__c in: oldReqIds];
        System.assert(li.size() == 300);
}
}
```

5. Test integration logic using callout mocks:

WarehouseCalloutServiceTest.apxc

```
@isTest
private class WarehouseCalloutServiceTest {
@isTest
   static void testWareHouseCallout(){
      Test.startTest();
      Test.setMock(HTTPCalloutMock.class,new WarehouseCalloutServiceMock());
      WarehouseCalloutService.runWarehouseEquipSync();
      Test.stopTest();
      System.assertEquals(2,[Select count() from Product2]);
   }
}
```

WarehouseCalloutServiceMock.apxc

```
@isTest
public class WarehouseCalloutServiceMock implements HttpCalloutMock{
   // implement http mock callout
   private String resJson = '[' +
'{"_id":"55d66226726b611100aaf741","replacement":false,"quantity":5,"name":"Generator
1000 kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku":"100003"},' +

'{"_id":"55d66226726b611100aaf743","replacement":true,"quantity":143,"name":"Fuse
20A","maintenanceperiod":0,"lifespan":0,"cost":22,"sku":"100005"}' +']';

   public HttpResponse respond(HTTPRequest req){
      System.assertEquals('https://th-superbadge-
apex.herokuapp.com/equipment',req.getEndpoint());
      System.assertEquals('GET', req.getMethod());
      HttpResponse res=new HttpResponse();
```

```
        res.setHeader('Content-Type','applcation/json');
        res.setBody(resJson);
        res.setStatusCode(200);
        return res;
    }

}
```

6. Test scheduling logic to confirm actions get queued:

WarehouseSyncScheduleTest.apxc

```
@isTest
public with sharing class WarehouseSyncScheduleTest {
    @isTest
    static void WarehouseschedTest(){
        String schedTime = '00 00 01 * * ?';
        Test.startTest();
        Test.setMock(HttpCalloutMock.class, new WarehouseCalloutServiceMock());
        String jobId=System.schedule('Scheduling time test', schedTime, new
WarehouseSyncSchedule());
        Test.stopTest();
        CronTrigger ct=[Select Id FROM CronTrigger where NextFireTime > today];
        System.assertEquals(jobId, ct.Id,'Schedule');
}
}
```

Apex Modules

1. Apex Triggers Module:

Get Started with Apex Triggers

AccountAddressTrigger.apxt

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

Bulk Apex Triggers

ClosedOpportunityTrigger.apxt

```
trigger ClosedOpportunityTrigger on Opportunity (after insert, after update) {
   List<Task> newtsk = new List<Task>();
   if(trigger.IsAfter && (trigger.IsInsert || trigger.IsUpdate)){
   for(Opportunity op:Trigger.New){
      if(op.StageName == 'Closed Won'){
         Task tsk = new Task();
          tsk.Subject = 'Follow Up Test Task';
          tsk.WhatId = op.id;
          newtsk.add(tsk);
          }
          }
   }
   if(newtsk.size()>0){
      insert newtsk;
   }
```

```
}
```

2. Apex Testing Module:

Get Started with Apex Unit Tests

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);
                }
        }
```

TestVerifyDate.apxt

```
@isTest
public class TestVerifyDate {
   @isTest static void PosTestCase(){
      Date res=VerifyDate.CheckDates(Date.parse('05/02/2022'),Date.parse('05/20/22'));
      System.assertEquals(Date.parse('05/20/22'),res);
   }
   @isTest static void NegTestCase(){
      Date
res=VerifyDate.CheckDates(Date.parse('05/02/2022'),Date.parse('06/22/2022'));

   }
}
```

Test Apex Triggers

RestrictContactByName.apxt

```
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');
		}
	}
}
```

TestRestrictContactByName.apxc

```
@isTest
public class TestRestrictContactByName {
   @isTest
   public static void test(){
      Contact c=new Contact();
      c.LastName='INVALIDNAME';
      Database.SaveResult r=Database.insert(c,false);
         System.assertEquals('The Last Name "INVALIDNAME" is not allowed for
DML',r.getErrors()[0].getMessage());
   }
}
```

Create Test Data for Apex Tests

RandomContactFactory.apxc

```
public class RandomContactFactory {
```

```
    public static List<Contact> generateRandomContacts(Integer no,String lname){
        List<Contact> li=new List<Contact>();
        for(Integer i=1;i<=no;i++)
        {
            Contact c=new Contact(FirstName='Test '+i,LastName=lname);
            li.add(c);
        }
        return li;
    }
}
```

3. Asynchronous Apex Module:

Use Future Methods

AccountProcessor.apxc

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

AccountProcessorTest.apxc

```
@isTest
public class AccountProcessorTest {
    public static testmethod void testAcc(){
        Account acc=new Account(Name='Test Account');
        insert acc;
        Contact c=new Contact(FirstName='Sai',LastName='Harshitha',AccountId=acc.Id);
```

```
        insert c;
        List<Id> li=new List<Id>();
       li.add(acc.Id);
        Test.startTest();
        AccountProcessor.countContacts(li);
        Test.stopTest();
        Account a=[Select Number_Of_Contacts__c from Account where Id =:acc.Id];
        System.assertEquals(Integer.valueOf(a.Number_Of_Contacts__c),1);
    }
}
```

Use Batch Apex

LeadProcessor.apxc

```
global class LeadProcessor implements Database.Batchable<sObject> {
    global Integer c=0;
    global DataBase.QueryLocator start(Database.BatchableContext bc){
     return DataBase.getQueryLocator([Select Id,LeadSource from Lead]);
    }
    global void execute(Database.BatchableContext bc,List<Lead> li){
        //List<Lead> t=new List<Lead>();
        for(Lead l:li){
            l.LeadSource='Dreamforce';
           // t.add(l);
            c+=1;
        }
        update li;
    }
    global void finish(Database.BatchableContext bc){
        System.debug('count = '+c);
    }
    }
```

LeadProcessorTest.apxc

```
@isTest
public class LeadProcessorTest {
@isTest
    public static  void testLead(){
      List<Lead> li=new List<Lead>();
      for(Integer i=0;i<200;i++){
         Lead l=new Lead();
         l.LastName='name'+i;
         l.Company='Company';
         l.Status='Random Status';
         li.add(l);
      }
      insert li;
      Test.startTest();
        LeadProcessor ob=new LeadProcessor();
         Id batchId=DataBase.executeBatch(ob,200);
         Test.stopTest();
      System.assertEquals(200,[select count() from Lead where
LeadSource='Dreamforce']);
          }
      }
```

Control Processes with Queueable Apex

AddPrimaryContact.apxc

```
public class AddPrimaryContact implements Queueable{

   private Contact con;
   private String state;
   public AddPrimaryContact(Contact con,String state){
      this.con=con;
      this.state=state;
   }
```

```
   public void execute(QueueableContext context){
      List<Account> li=[Select Id from Account where BillingState=:state limit 200];
      List<Contact> contacts=new List<Contact>();
      for(Account a:li){
         Contact contactClone = con.clone();
         contactclone.AccountId=a.Id;
         contacts.add(contactClone);
}
      insert contacts;
   }
}
```

AddPrimaryContactTest.apxc

```
@isTest
private class AddPrimaryContactTest {
  @isTest
   public static void testQueueable(){
      List<Account> a=new List<Account>();
      for(Integer i=0;i<100;i++){
         Account acc=new Account(Name='Test Account');
         if(i<50){
            acc.BillingState='NY';
         }
         else
         {
            acc.BillingState='CA';
         }
         a.add(acc);
      }
      insert a;
      Contact c=new Contact(FirstName='fname',LastName='lname');
      insert c;

      Test.startTest();
      Id jobId =System.enqueueJob(new AddPrimaryContact(c,'CA'));
```

```
        Test.stopTest();
       List<Contact> li=[Select id from Contact where Contact.Account.BillingState='CA'];
        System.assertEquals(50, li.size() ,'ERROR: Incorrect number of contact records
found');
    }
}
```

Schedule Jobs Using the Apex Scheduler

DailyLeadProcessor.apxc

```
public class DailyLeadProcessor implements Schedulable{
   public void execute(SchedulableContext conte){
      List<Lead> leadstoupdate=new List<Lead>();
      List<Lead> leads=[Select Id,LeadSource from Lead where LeadSource='' Limit 200];
      for(Lead l:leads){
         l.LeadSource='Dreamforce';
         leadstoupdate.add(l);
      }
      update leads;
   }
}
```

DailyLeadProcessorTest.apxc

```
@isTest

private class DailyLeadProcessorTest {
private static String CRON_EXP='0 0 1 * * ?';
   @isTest
   public static void testSchedulable(){
      List<Lead> leads=new List<Lead>();
      for(Integer i = 0; i < 200; i++){
         Lead lead = new Lead(LastName = 'DreamForce' + i, LeadSource = '', Company =
'TestCompany ' + i, Status = 'Open - Not Contacted');
         leads.add(lead);
```

```
        }

        insert leads;
        Test.startTest();
        String jobId=System.schedule('Process Leads',CRON_EXP,new
DailyLeadProcessor());
        Test.stopTest();


       List<Lead> li= [Select Id,LeadSource from Lead where LeadSource='Dreamforce'];
        System.assertEquals(200,li.size(),'ERROR: At least 1 record not updated correctly');
       List<CronTrigger> cts=[Select Id,TimesTriggered,NextFireTime from CronTrigger
where Id=:jobId];
        System.debug('Next Fire Time '+cts[0].NextFireTime);
    }
}
```

4. Apex Integration Services Module:

Apex REST Callouts

AnimalLocator.apxc

```
public class AnimalLocator {
    public static String getAnimalNameById(Integer no){
        Http http=new Http();
        HttpRequest req=new HttpRequest();
        req.setEndPoint('https://th-apex-http-callout.herokuapp.com/animals/'+no);
        req.setMethod('GET');
        HttpResponse res=http.send(req);
             Map<String,Object>
result=(Map<String,Object>)JSON.deserializeUntyped(res.getBody());
        Map<String,Object> animal=(Map<String,Object>)result.get('animal');
           System.debug('name: '+string.valueOf(animal.get('name')));
        return string.valueOf(animal.get('name'));

    }
```

```
}
```

AnimalLocatorTest.apxc

```
@isTest
private class AnimalLocatorTest {
@isTest
   static void animalLocatorTest1(){
      Test.setMock(HttpCalloutMock.class,new AnimalLocatorMock());
      String actual=AnimalLocator.getAnimalNameById(1);
      String exp='moose';
      System.assertEquals(actual,exp);
   }
}
```

Apex SOAP Callouts

Generated a class using this using this WSDL file

ParkService.apxc

```
//Generated by wsdl2apex

public class ParkService {
   public class byCountryResponse {
      public String[] return_x;
      private String[] return_x_type_info = new
String[]{'return','http://parks.services/',null,'0','-1','false'};
      private String[] apex_schema_type_info = new
String[]{'http://parks.services/','false','false'};
      private String[] field_order_type_info = new String[]{'return_x'};
   }
   public class byCountry {
      public String arg0;
      private String[] arg0_type_info = new
String[]{'arg0','http://parks.services/',null,'0','1','false'};
      private String[] apex_schema_type_info = new
String[]{'http://parks.services/','false','false'};
      private String[] field_order_type_info = new String[]{'arg0'};
```

```
    }
        public class ParksImplPort {
        public String endpoint_x = 'https://th-apex-soap-
service.herokuapp.com/service/parks';
        public Map<String,String> inputHttpHeaders_x;
        public Map<String,String> outputHttpHeaders_x;
        public String clientCertName_x;
        public String clientCert_x;
        public String clientCertPasswd_x;
        public Integer timeout_x;
        private String[] ns_map_type_info = new String[]{'http://parks.services/',
'ParkService'};
        public String[] byCountry(String arg0) {
            ParkService.byCountry request_x = new ParkService.byCountry();
            request_x.arg0 = arg0;
            ParkService.byCountryResponse response_x;
            Map<String, ParkService.byCountryResponse> response_map_x = new
Map<String, ParkService.byCountryResponse>();
            response_map_x.put('response_x', response_x);
            WebServiceCallout.invoke(
              this,
              request_x,
              response_map_x,
              new String[]{endpoint_x,
              '',
              'http://parks.services/',
              'byCountry',
              'http://parks.services/',
              'byCountryResponse',
              'ParkService.byCountryResponse'}
            );
            response_x = response_map_x.get('response_x');
            return response_x.return_x;
        }
    }
}
```

ParkLocator.apxc

```
public class ParkLocator{
   public static List<String> country(String country){
      ParkService.ParksImplPort park=new ParkService.ParksImplPort();
      return park.byCountry(country);
   }
}
```

ParkLocatorTest.apxc

```
@isTest
public class ParkLocatorTest{
@isTest
   static void testCallout(){
      Test.setMock(WebServiceMock.class,new ParkServiceMock());
      String country='United States';

      List<String> li=new List<String>{'Yosemite','Sequoia','Carter Lake'};
      System.assertEquals(li,ParkLocator.country(country));
   }
}
```

Apex Web Services

AccountManager.apxc

```
@RestResource(urlMapping='/Accounts/*/contacts')
global with sharing class AccountManager {
   @HttpGet
   global static Account getAccount(){
      RestRequest request=RestContext.request;
      String accId=request.requestURI.substringBetween('Accounts/','/contacts');
      Account res=[Select ID,Name,(Select ID,FirstName,LastName from Contacts)
                  from Account where ID =:accId];
     return res;
   }
```

```
}
```

AccountManagerTest.apxc

```
@isTest
private class AccountManagerTest {
@isTest
   static void testGetAccount(){
      Account acc=new Account(Name='TestAccount');
      insert acc;
      Contact con=new Contact(AccountId=acc.Id,Firstname='Test',LastName='Test');
      insert con;
      RestRequest request=new RestRequest();

request.requestURI='https://yourInstance.salesforce.com/services/apexrest/Accounts/'
+acc.id+'/contacts';
      request.httpMethod='GET';
      RestContext.request=request;
      Account myAcc=AccountManager.getAccount();
      System.assert(myAcc!=null);
      System.assertEquals('TestAccount', myAcc.Name);

   }
}
```

5. Apex Basics & Database Module:

Get Started with Apex

StringArrayTest.apxc

```
public class StringArrayTest {
   public static List<String> generateStringArray(Integer no){
      List<String> li=new List<String>();
      for(Integer j=0;j<no;j++){
         li.add('Test '+j);
```

```
    }
    return li;
  }
}
```

Manipulate Records with DML

AccountHandler.apxc

```
public class AccountHandler {
  public static Account insertNewAccount(String name){
    try{
      Account a=new Account(Name=name);
      insert a;
      return a;
    }
    catch(DMLException e){
      return null;
    }
  }

}
```

Write SOQL Queries

ContactSearch.apxc

```
public class ContactSearch {
  public static List<Contact> searchForContacts(String lname,String
mailingPostalCode){
    List<Contact> li=[Select Id,Name from Contact where LastName=:lname and
MailingPostalCode=:mailingPostalCode];
    return li;
  }

}
```

Write SOSL Queries

ContactAndLeadSearch.apxc

```
public class ContactAndLeadSearch {
   public static List<List< sObject>> searchContactsAndLeads(String lname){
      List<List< sObject>> li=[Find:lname in all fields returning
Contact(Name),Lead(Name)];
      return li;
   }
}
```