**Apex Specialist SuperBadge**

**1.Use Future Methods**

**Q. 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.**

**Output:**

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;

}

}

**Test Class**

@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);

}

}

**2.Use Batch Apex**

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

* **Create an Apex class:**
  + **Name: LeadProcessor**
  + **Interface: Database.Batchable**
  + **Use a QueryLocator in the start method to collect all Lead records in the org**
  + **The execute method must update all Lead records in the org with the LeadSource value of Dreamforce**
* **Create an Apex test class:**
  + **Name: LeadProcessorTest**
  + **In the test class, insert 200 Lead records, execute the LeadProcessor Batch class and test that all Lead records were updated correctly**
  + **The unit tests must cover all lines of code included in the LeadProcessor class, resulting in 100% code coverage**

**Output:**

global class LeadProcessor implements

Database.Batchable<sObject> {

global Database.QueryLocator start(Database.BatchableContext bc) {

return Database.getQueryLocator(

'SELECT ID from Lead'

);

}

global void execute(Database.BatchableContext bc, List<Lead> scope){

// process each batch of records

List<Lead> leads = new List<Lead>();

for (Lead lead : scope) {

lead.LeadSource = 'Dreamforce';

leads.add(lead);

}

update leads;

}

global void finish(Database.BatchableContext bc){

}

}

**Test Class**

@isTest

private class LeadProcessorTest {

@testSetup

static void setup() {

List<Lead> leads = new List<Lead>();

// insert 10 accounts

for (Integer i=0;i<200;i++) {

leads.add(new Lead(Lastname='Lead '+i , Company='Test Co'));

}

insert leads;

}

static testmethod void test() {

Test.startTest();

LeadProcessor myLeads = new LeadProcessor();

Id batchId = Database.executeBatch(myLeads);

Test.stopTest();

// after the testing stops, assert records were updated properly

System.assertEquals(200, [select count() from Lead where LeadSource = 'Dreamforce']);

}

}

**3. Control Processes with Queueable Apex**

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

**Output:**

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;

}

}

}

**Test Class**

@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();

}

}

**4.Schedule Jobs Using the Apex Scheduler**

**Q. 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.)**

**Output:**

global class DailyLeadProcessor implements Schedulable {

global void execute(SchedulableContext ctx) {

List<lead> leadstoupdate = new List<lead>();

List<Lead> leads = [SELECT Id

FROM Lead

WHERE LeadSource = Null Limit 200

];

for(Lead l:leads){

l.LeadSource = 'DreamForce';

leadstoupdate.add(l);

}

update leadstoupdate;

}

}

Test Class

@isTest

private class DailyLeadProcessorTest {

// Dummy CRON expression: midnight on March 15.

// Because this is a test, job executes

// immediately after Test.stopTest().

public static String CRON\_EXP = '0 0 0 15 3 ? 2022';

static testmethod void testScheduledJob() {

// Create some out of date Opportunity records

List<Lead> leads = new List<lead>();

for (Integer i=0; i<200; i++) {

Lead l = new Lead(

FirstName = 'First ' + i,

LastName = 'LastName',

Company = 'The Inc'

);

leads.add(l);

}

insert leads;

Test.startTest();

// Schedule the test job

String jobId = System.schedule('ScheduledApexTest',

CRON\_EXP,

new DailyLeadProcessor());

Test.stopTest();

// Now that the scheduled job has executed,

// check that we have 200 Leads with dreamforce

List<Lead> Checkleads = new List<Lead>();

Checkleads = [SELECT Id

FROM Lead

WHERE LeadSource='Dreamforce' and Company = 'The Inc'];

System.assertEquals(200,

Checkleads.size(),

'Leads were not created');

}

}