Asynchronous Apex

(1) Asynchronous Processing Basics.

MCQ-based Quiz Challenge:

(1) What is a key benefit of Asynchronous Processing?

(A) Higher governor and execution limits.
(B) Unlimited number of external callouts.
(C) Override organization-wide sharing defaults.
(D) Enabling turbo mode for transactional processing.

(2) Batch Apex is typically the best type of asynchronous processing when you want to:

(A) Make a callout to a web service when a user updates a record.
(B) Schedule a task to run on a weekly basis.
(C) Update all records in your org.
(D) Send a rickroll email to a contact.

(2) 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.

- Create a field on the Account object:
  - Label: `Number Of Contacts`
  - Name: `Number_Of_Contacts`
  - Type: **Number**
  - This field will hold the total number of Contacts for the Account
- Create an Apex class:
  - Name: `AccountProcessor`
  - Method name: `countContacts`
  - The method must accept a List of Account IDs
  - The method must use the @future annotation
  - The method counts the number of Contact records associated to each Account ID passed to the method and updates the 'Number_Of_Contacts__c' field with this value
- Create an Apex test class:
  - Name: `AccountProcessorTest`
  - The unit tests must cover all lines of code included in the **AccountProcessor** class, resulting in 100% code coverage.
- Before verifying this challenge, run your test class at least once using the Developer Console Run All feature

Steps taken to complete this challenge:

(1) After launching a Salesforce Playground or an org, click on "Gear" icon on top right-side corner and then click on Setup. Then click on Object Manager Tab.
(2) Once Object Manager Tab is opened, click on Account Object and then go to Fields & Relationships and then click on New.

(3) Select datatype as Number, then fill Field Label as “Number Of Contacts” and Field Name as “Number_Of_Contacts”, then let rest all the data be as it is and then Click on Next, Next and Save buttons respectively.

(4) Then again, click on gear icon and click “Developer Console” and create two new Apex classes named as “AccountProcessor” and “AccountProcessorTest”.

(5) Code needs to be written in AccountProcessor.apxc file:

```
public class AccountProcessor {
    @future
    public static void countContacts(List<Id> accountIds){
        List<Account> accounts = [Select Id, Name from Account Where Id IN :
accountIds];
        List<Account> updatedAccounts = new List<Account>();
        for(Account account : accounts){
          account.Number_of_Contacts__c = [Select count() from Contact Where
AccountId =: account.Id];
            System.debug('No Of Contacts = ' + account.Number_of_Contacts__c);
            updatedAccounts.add(account);
        }
        update updatedAccounts;
    }
}
```

(6) Code needs to be written in AccountProcessorTest.apxc file:

```
@isTest
public class AccountProcessorTest {
    @isTest
    public static void testNoOfContacts(){
        Account a = new Account();
        a.Name = 'Test Account';
        Insert a;
        Contact c = new Contact();
        c.FirstName = 'Bob';
        c.LastName =  'Willie';
        c.AccountId = a.Id;
        Contact c2 = new Contact();
        c2.FirstName = 'Tom';
        c2.LastName = 'Cruise';
        c2.AccountId = a.Id;
        List<Id> acctIds = new List<Id>();
        acctIds.add(a.Id);
        Test.startTest();
        AccountProcessor.countContacts(acctIds);
        Test.stopTest();
    }
}
```

(7) Save all the Apex code files and click on Test and then, click on “Run All” and then click on “Check Challenge”, the task will be then successfully completed and submitted.

(3) Use Batch Apex.

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.

- 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
- Before verifying this challenge, run your test class at least once using the Developer Console Run All feature

Steps taken to complete this challenge:

(1) After launching a Salesforce Playground or an org, click on “Gear” icon on top-right side corner and then click on “Developer Console”.

(2) After coming inside the Developer Console, click on File > New > Apex Class and create two new apex class files named as “LeadProcessor” and “LeadProcessorTest”.

(3) Code needs to be written in LeadProcessor.apxc file:

```
public class LeadProcessor implements Database.Batchable<sObject> {
    public Database.QueryLocator start(Database.BatchableContext bc) {
      // collect the batches of records or objects to be passed to execute
       return Database.getQueryLocator([Select LeadSource From Lead ]);
    }
    public void execute(Database.BatchableContext bc, List<Lead> leads){
       // process each batch of records
         for (Lead Lead : leads) {
            lead.LeadSource = 'Dreamforce';
         }
      update leads;
    }
    public void finish(Database.BatchableContext bc){
     }
}
```

(4) Code needs to be written in LeadProcessorTest.apxc file:

```
@isTest
public class LeadProcessorTest {
      @testSetup
```

```
    static void setup() {
        List<Lead> leads = new List<Lead>();
        for(Integer counter=0 ;counter <200;counter++){
            Lead lead = new Lead();
            lead.FirstName ='FirstName';
            lead.LastName ='LastName'+counter;
            lead.Company ='demo'+counter;
            leads.add(lead);
        }
        insert leads;
    }
    @isTest static void test() {
        Test.startTest();
        LeadProcessor leadProcessor = new LeadProcessor();
        Id batchId = Database.executeBatch(leadProcessor);
        Test.stopTest();
    }
}
```

(5) Save both of the Apex class files and click on Test and then, click on “Run All” and then click on “Check Challenge”, the task will be then successfully completed and submitted.

(4) 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.

- Create an Apex class:
  - Name: `AddPrimaryContact`
  - Interface: `Queueable`
  - Create a constructor for the class that accepts as its first argument a Contact sObject and a second argument as a string for the State abbreviation
  - The `execute` method must query for a maximum of 200 Accounts with the `BillingState` specified by the State abbreviation passed into the constructor and insert the Contact sObject record associated to each Account. Look at the sObject `clone()` method.
- Create an Apex test class:
  - Name: `AddPrimaryContactTest`
  - In the test class, insert 50 Account records for `BillingState NY` and 50 Account records for `BillingState CA`
  - Create an instance of the `AddPrimaryContact` class, enqueue the job, and assert that a Contact record was inserted for each of the 50 Accounts with the `BillingState` of `CA`
  - The unit tests must cover all lines of code included in the **AddPrimaryContact** class, resulting in 100% code coverage
- Before verifying this challenge, run your test class at least once using the Developer Console Run All feature

Steps taken to complete this challenge:

(1) After launching a Salesforce Playground or an org, click on “Gear” icon on top-right side corner and then click on “Developer Console”.

(2) After coming inside the Developer Console, click on File > New > Apex Class and create two new apex class files named as “AddPrimaryContact” and “AddPrimaryContactTest”.

(3) Code needs to be written in AddPrimaryContact.apxc file:

```
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;
       }
   }
}
```

(4) Code needs to be written in AddPrimaryContactTest.apxc file:

```
@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();
    }
}
```

(5) Save both of the Apex class files and click on Test and then, click on “Run All” and then click on “Check Challenge”, the task will be then successfully completed and submitted.

(5) 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.)

- Create an Apex class:
  - Name: `DailyLeadProcessor`
  - Interface: `Schedulable`
  - The execute method must find the first 200 Lead records with a blank LeadSource field and update them with the LeadSource value of `Dreamforce`
- Create an Apex test class:
  - Name: `DailyLeadProcessorTest`
  - In the test class, insert 200 Lead records, schedule the DailyLeadProcessor class to run and test that all Lead records were updated correctly
  - The unit tests must cover all lines of code included in the **DailyLeadProcessor** class, resulting in 100% code coverage.
- Before verifying this challenge, run your test class at least once using the Developer Console Run All feature

Steps taken to complete this challenge:

(1) After launching a Salesforce Playground or an org, click on “Gear” icon on top-right side corner and then click on “Developer Console”.

(2) After coming inside the Developer Console, click on File > New > Apex Class and create two new apex class files named as “DailyLeadProcessor” and “DailyLeadProcessorTest”.

(3) Code needs to be written in DailyLeadProcessor.apxc file:

```
public class DailyLeadProcessor implements Schedulable  {
   Public void execute(SchedulableContext SC){
```

```
        List<Lead> LeadObj=[SELECT Id from Lead where LeadSource=null limit
    200];
         for(Lead l:LeadObj){
            l.LeadSource='Dreamforce';
            update l;
         }
      }
    }
```

(4) Code needs to be written in DailyLeadProcessorTest.apxc file:

```
    public class DailyLeadProcessor implements Schedulable  {
       Public void execute(SchedulableContext SC){
         List<Lead> LeadObj=[SELECT Id from Lead where LeadSource=null limit
    200];
         for(Lead l:LeadObj){
            l.LeadSource='Dreamforce';
            update l;
         }
      }
    }
```

(5) Save both of the Apex class files and click on Test and then, click on “Run All” and then click on “Check Challenge”, the task will be then successfully completed and submitted.

(6) Monitor Asynchronous Apex.

MCQ-based Quiz Challenge:

(1) What type of jobs do not show up in the Apex Flex Queue?

(A) Future Method Jobs.
(B) Batch Apex Jobs.
(C) Queueable Apex Jobs.
(D) Scheduled Apex Jobs.

(2) Which statement is true regarding the Flex Queue?

(A) You can submit up to 200 batch jobs for execution.
(B) Jobs are processed first-in first-out.
(C) Jobs can only be scheduled during Taco Tuesdays.
(D) Jobs are executed depending upon user license.