APEX SPECIALIST - SUPERBADGE

PREREQUISITES MODULES

APEX TRIGGERS

Get Started with Apex Triggers

FILENAME : AccountAddressTrigger

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

EXPLAINATION :

so here we have to used for loop for finding the account and set the shippingPostalCode to BillingPostalCode this has to be done before the insertion and update.

Bulk Apex Triggers

FILENAME : ClosedOpportunityTrigger

```
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;
   }

}
```

EXPLAINATION :

Here, we created a new list and we used the for loop to find where StageName is Closed Won and also subject is 'follow up test task' that will be added in the tasklist and when the list(tasklist) is greater than zero then insert the list(tasklist).

APEX TESTING

Get started with Apex Unit Test

FILENAME : VerifyDate

```
public class VerifyDate {
   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;
        }

}
```

FILENAME : TestVerifyDate

```
@isTest
public class TestVerifyDate {

   @isTest static void Test_CheckDates_case1(){
      Date D = VerifyDate.CheckDates(date.parse('01/01/2020'),
date.parse('01/05/2020'));
```

```
        System.assertEquals(date.parse('01/05/2020'), D);

    }

    @isTest static void Test_CheckDates_case2(){
        Date D = VerifyDate.CheckDates(date.parse('01/01/2020'),
date.parse('05/05/2020'));
        System.assertEquals(date.parse('01/31/2020'), D);

    }

    @isTest static void Test_DateWithin30Days_case1(){
        Boolean flag =
VerifyDate.DateWithin30Days(date.parse('01/01/2020'),
date.parse('12/30/2020'));
        System.assertEquals(false, flag);
    }

    @isTest static void Test_DateWithin30Days_case2(){
        Boolean flag =
VerifyDate.DateWithin30Days(date.parse('01/01/2020'),
date.parse('02/02/2020'));
        System.assertEquals(false, flag);
    }

    @isTest static void Test_DateWithin30Days_case3(){
        Boolean flag =
```

```
VerifyDate.DateWithin30Days(date.parse('01/01/2020'),
date.parse('01/15/2020'));
        System.assertEquals(true, flag);
    }

    @isTest static void Test_SetEndOfMonthDate(){
        Date returndate =
VerifyDate.SetEndOfMonthDate(date.parse('01/01/2020'));
    }

}
```

EXPLAINATION :

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.

Test Apex Triggers

FILENAME : **RestrictContactByName**

```
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');
            }

        }

}
```

FILENAME : TestRestrictContactByName

```
@isTest
public class TestRestrictContactByName {

    @isTest static void Test_insertupdateContact(){
        Contact cnt = new Contact();
        cnt.LastName = 'INVALIDNAME';

        Test.startTest();
        Database.SaveResult result = Database.insert(cnt, false);
        Test.stopTest();

        System.assert(!result.isSuccess());
```

```
        System.assert(result.getErrors().size() > 0);
        System.assertEquals('The Last Name "INVALIDNAME" is not
allowed for DML',result.getErrors()[0].getMessage());
    }
}
```

EXPLAINATION :

Here, which blocks inserts and updates to any contact with a last name of 'INVALIDNAME'.

Create Test Data For Apex Tests

FILENAME : RandomContactFactory

```
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;

    }
}
```

EXPLAINATION :

Here, we will use for loop for finding the contacts and creating a new contact and returns a list of contacts based on two incoming parameters: the number of contacts to generate and the last name.

ASYNCHRONOUS APEX

Use Future Methods

FILENAME : AccountProcessor

```
public class AccountProcessor {
```

```
    @future
    public static void countContacts(List<Id> accountIds){

        List<Account> accountsToUpdate = new
List<Account>();

        List<Account> accounts = [Select Id, Name, (Select Id
from Contacts) from Account Where Id in :accountIds];

        For(Account acc:accounts){
            List<Contact> contactList = acc.Contacts;
            acc.Number_Of_Contacts__c = contactList.size();
            accountsToUpdate.add(acc);
        }
        update accountsToUpdate;



    }
}
```

FILENAME : AccountProcessorTest

```
@IsTest
private class AccountProcessorTest {
```

```
    @IsTest
    private static void testCountContacts(){
        Account newAccount = new Account(Name='Test
Account');
        insert newAccount;

        Contact newContact1 = new
Contact(FirstName='John',LastName='Doe',AccountId =
newAccount.Id);
        insert newContact1;

        Contact newContact2 = new
Contact(FirstName='Jane',LastName='Doe',AccountId =
newAccount.Id);
        insert newContact2;

        List<Id> accountIds = new List<Id>();
        accountIds.add(newAccount.Id);

        Test.startTest();
        AccountProcessor.countContacts(accountIds);
        Test.stopTest();
    }
```

EXPLAINATION :

we used 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.

}

Use Batch Apex Unit

FILENAME : LeadProcessor

```
global class LeadProcessor implements
Database.Batchable<sObject> {
   global Integer count = 0;
   global Database.QueryLocator
start(Database.BatchableContext bc){
       return Database.getQueryLocator('SELECT ID,
LeadSource FROM Lead');    }
   global void execute (Database.BatchableContext bc,
List<Lead> L_list){
       List<lead> L_list_new = new List<lead>();
       for(lead L:L_list) {
```

```
            L.leadsource = 'Dreamforce';
            L_list_new.add(L);
            count += 1;
        }
        update L_list_new;
    }
```

FILENAME : LeadProcessorTest

```
@isTest
public class LeadProcessorTest {

    @isTest
    public static void testit(){
        List<lead> L_list = 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';
            L_list.add(L);
        }
        insert L_list;
```

```
        Test.startTest();
        LeadProcessor lp = new LeadProcessor();
        Id batchId = Database.executeBatch(lp);
        Test.stopTest();
    }

}
```

EXPLAINATION :

We use implements the Database.Batchable interface to update all Lead records in the org with a specific LeadSource.

Control Processes with Queueable Apex

FILENAME : AddPrimaryContact

```
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> accounts = [Select Id, Name,
(Select FirstName, LastName, Id from contacts)
```

```
from Account where BillingState = :state Limit
200];
        List<Contact> primaryContacts = new
List<Contact>();

        for(Account acc:accounts){
            Contact c = con.clone();
            c.AccountId = acc.Id;
            primaryContacts.add(c);
        }

        if(primaryContacts.size() > 0){
            insert primaryContacts;

        }

    }


}
```

FILENAME : AddPrimaryContactTest

```
@isTest
public class AddPrimaryContactTest {

    static testmethod void testQueueable(){
        List<Account> testAccounts = new
List<Account>();
        for(Integer i=0;i<50;i++){
            testAccounts.add(new
Account(Name='Account '+i,BillingState='CA'));
        }
        for(Integer j=0;j<50;j++){
            testAccounts.add(new
Account(Name='Account '+j,BillingState='NY'));
        }
        insert testAccounts;
```

```
        Contact testContact = new
Contact(FirstName = 'John', LastName = 'Doe');
        insert testContact;

        AddPrimaryContact addit = new
addPrimaryContact(testContact, 'CA');

        Test.startTest();
        system.enqueueJob(addit);
        Test.stopTest();

        System.assertEquals(50,[Select count() from
Contact where accountId in (Select Id from
Account where BillingState='CA')]);
    }
}
```

EXPLAINATION :

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

Schedule Jobs Using the Apex Scheduler

FILENAME : DailyLeadProcessor

```
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;
    }

}
```

FILENAME : DailyLeadProcessorTest

```
@isTest
private class DailyLeadProcessorTest {

    public static String CRON_EXP = '0 0 0 15 3 ? 2022';
    static testmethod void testScheduledJob(){
        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();

        String jobId =
System.schedule('ScheduledApexTest',CRON_EXP,new
DailyLeadProcessor());
        Test.stopTest();

        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');
    }

}
```

EXPLAINATION :

Here the code that implements the Schedulable interface to update Lead records with a specific LeadSource. The Apex Scheduler lets you delay execution so that you can run Apex classes at a specified time. This is ideal for daily or weekly

maintenance tasks using Batch Apex. To take advantage of the scheduler, write an Apex class that implements the Schedulable interface, and then schedule it for execution on a specific schedule.

APEX INTEGRATION SERVICES

Apex REST Callouts

FILENAME : AnimalLocator

```
public class AnimalLocator {
    public static String getAnimalNameById(Integer x){
        Http http = new Http();
        HttpRequest request = new HttpRequest();
        request.setEndpoint('https://th-apex-http-
callout.herokuapp.com/animals' + x);
        request.setMethod('GET');
        Map<String, Object> animal= new Map<String,
```

```
Object>();
        HttpResponse response = http.send(request);
            if(response.getStatusCode() == 200) {
        Map<String, Object> results = (Map<String, Object>)
JSON.deserializeUntyped(response.getBody());
        List<Object> animals = (List<Object>)
results.get('animals');
            }
return (String)animal.get('name');
    }

}
```

FILENAME : AnimalLocatorTest

```
@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);
    }
```

}

EXPLAINATION :

To test your callouts, use mock callouts by either implementing an interface or using static resources. In this example, we use static resources and a mock interface later on. The static resource contains the response body to return. Again, when using a mock callout, the request isn't sent to the endpoint. Instead, the Apex runtime knows to look up the response specified in the static resource and return it instead. The Test.setMock method informs the runtime that mock callouts are used in the test method. Let's see mock callouts in action. First, we create a static resource containing a JSON-formatted string to use for the GET request.

Apex SOAP Callouts

FILENAME : ParkServices

```
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;
        }
```

```
    }
}
```

FILENAME : ParkLocator

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

FILENAME : ParkLocatorTest

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

```
        List <String> result = ParkLocator.country (x);
        string resultstring = string.join (result,',');
        System.assertEquals ('USA', resultstring);
    }
}
```

EXPLAINATION :

Here, we used the code by using WSDL2Apex for a SOAP web service. Class must have a country method that uses the ParkService class Method must return an array of available park names for a particular country passed to the web service (such as Germany, India, Japan, and United States).

Apex Web Services

FILENAME : AccountManager

```
@RestResource (urlMapping = '/Account/*/contacts')
global with sharing class AccountManager
{
    @HttpGet
```

```
    global static Account getAccount ()
    {
        RestRequest request = RestContext.request;
        String accountId =
request.requestURI.substringBetween('Accounts/','/contact
s');
        Account result =  [SELECT Id, Name, (SELECT Id,
Name FROM Contacts) FROM
Account WHERE Id = :accountId Limit 1];
        return result;
    }
}
```

FILENAME : AccountManagerTest

```
@isTest
private class AccountManagerTest
{
    @isTest static void testGetContactsByAccountId ()
    {
        Id recordId = createTestRecord ();
        RestRequest request = new RestRequest ();
        request.requestUri =
'https://yourInstance.salesforce.com/services/apexrest/Ac
```

```
counts/' + recordId +
'/contacts';
        request.httpMethod = 'GET';
        RestContext.request = request;
        Account thisAccount = AccountManager.getAccount();
        System.assert (thisAccount != null);
        System.assertEquals ('Test Record',
thisAccount.Name);
    }
    static Id createTestRecord ()
    {
        Account accountTest = new Account (
            Name = 'Test Record');
        insert accountTest;
        Contact contactTest = new Contact (
            FirstName='John',
            LastName='Doe',
            AccountId =accountTest.Id);
        insert contactTest;
        return accountTest.Id;
    }
}
```

EXPLAINATION :

Here the code which gives 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. Method must be annotated with @HttpGet and return an Account object. Method must return the ID and Name for the requested record and all associated contacts with their ID and Name.Unit tests must be in a separate Apex class called AccountManagerTest. Unit tests must cover all lines of code included in the AccountManager class.