Apex Triggers -

1) Create an Apex Trigger -

```
trigger AccountAddressTrigger on Account (before insert, before update) {

    for(Account a : Trigger.new){
        If (a.Match_Billing_Address__c == true) {
            a.ShippingPostalCode = a.BillingPostalCode;
        }
    }

}

```

2) Bulk Apex Triggers -

```
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

        }
       }
    if(taskList.size()>0){
        insert taskList;
    }
}
```

Apex Triggers -

3)Get Started with Apex Unit Test -

```

@isTest
public class TestVerifyDate {

    @isTest static void test1(){
        Date d = VerifyDate.CheckDates(Date.parse('01/01/2020'),Date.parse('01/003/2020'));
        System.assertEquals(Date.parse('01/03/2020'), d);
     }

    @isTest static void test2(){
        Date d = VerifyDate.checkDates(Date.parse('01/01/2020'),Date.parse('03/03/2020'));
        System.assertEquals(Date.parse('01/31/2020'), d);
    }
}
```

4) Test Apex Triggers -

```
@isTest
public class TestRestrictContactByName {

    @isTest
    public static void testContact(){
        Contact ct = new Contact();
        ct.LastName = 'INVALIDNAME';
        Database.SaveResult res =
Database.insert(ct,false);
        System.assertEquals('The Lasr Name



    }

}
```

5) Create Test Data For Apex Test -

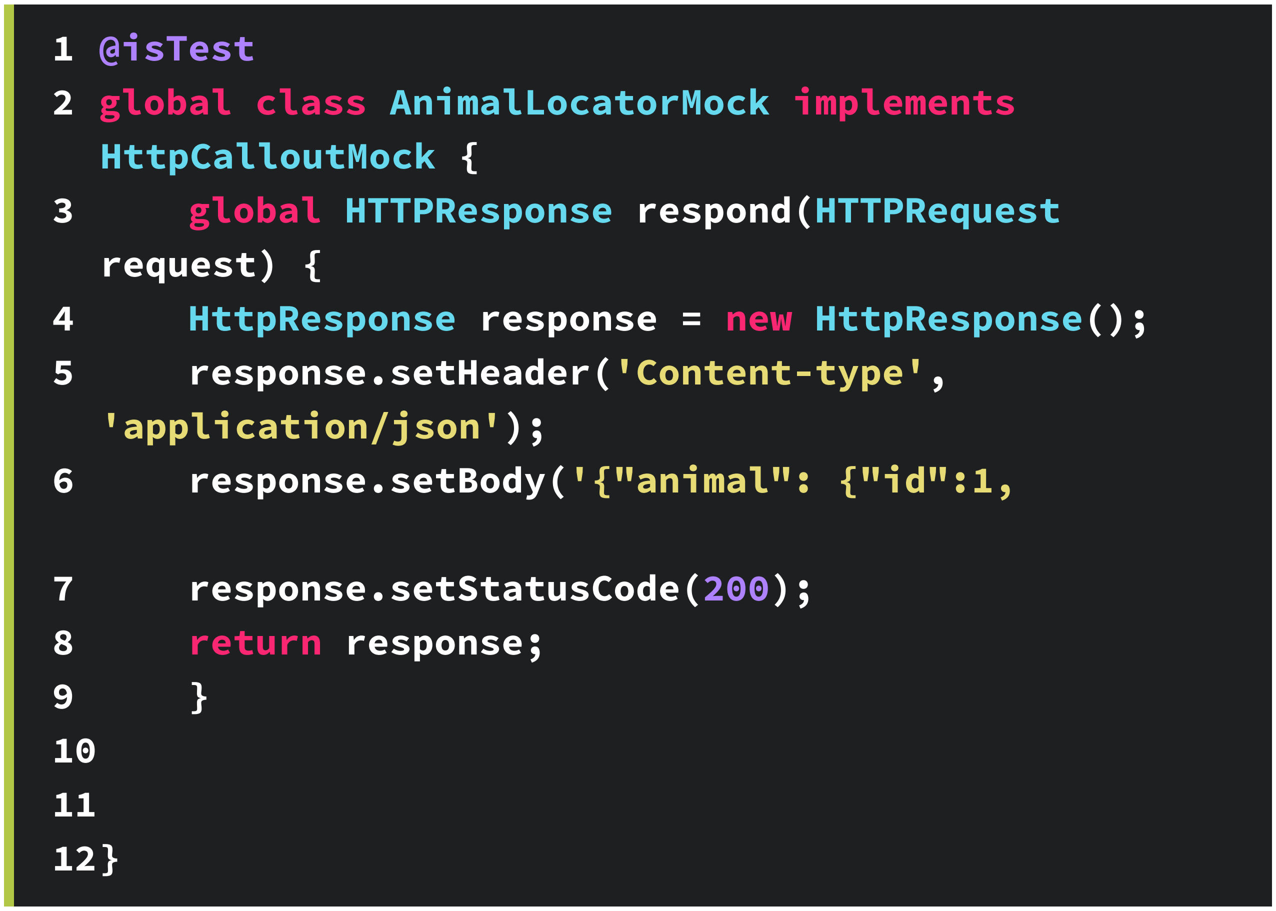
```
public class RandomContactFactory {

    public static List<Contact> generateRandomContacts(Integer num, String lastName){
        List<Contact> contactList = new List<Contact>();
        for(Integer i = 1;i<=num;i++){
            Contact ct = new Contact(FirstName = 'Test '+i, LastName =lastName);
            contactList.add(ct);

        }
         return contactList;
    }

}
```

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

1) Create Apex Class For Animal Locator Test :

```
public class AnimalLocator {

    public static String getAnimalNameById(Integer Id) {
        Http http = new Http();
        HttpRequest request = new HttpRequest();
        request.setEndpoint(' https://th-apex-http-

        request.setMethod('GET');
        HttpResponse response = http.send(request);
        String strResp = '';
        system.debug('******response ' +
response.getStatusCode());
        system.debug('******response ' +
response.getBody());
        if (response.getStatusCode() == 200)
        {
            Map<String, Object> results = (Map<String,
Object>) JSON.deserializeUntyped(response.getBody());
            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;
    }

}
```

2) Animal Locator Mock Test:



```
@isTest
global class AnimalLocatorMock implements HttpCalloutMock {
    global HTTPResponse respond(HTTPRequest request) {
    HttpResponse response = new HttpResponse();
    response.setHeader('Content-type', 'application/json');
    response.setBody('{"animal": {"id":1,

    response.setStatusCode(200);
    return response;
    }


}
```

3) Animal Locator Test:

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

}
```

Apex Class Park Locator Test:

1) Apex Class Park Locator :

```
public class ParkLocator {
    public static string[] country(String country) {
        parkService.parksImplPort park = new parkService.parksImplPort();
        return park.byCountry(country);
    }
}
```

2) **Apex Class For Park Locator Test :**

```
@isTest
private class ParkLocatorTest {
    @isTest static void testCallout() {
        // This causes a fake response to be generated
        Test.setMock(WebServiceMock.class, new ParkServiceMock());
        // Call the method that invokes a callout
        //Double x = 1.0;
        //Double result = AwesomeCalculator.add(x, y);

        String country = 'Germany';
        String[] result = ParkLocator.Country(country);


        // Verify that a fake result is returned
        System.assertEquals(new List<String>{'Hamburg Wadden Sea National Park', 'Hainich National Park', 'Bavarian Forest National

    }
}
```

2) Apex Class For Park Locator MockTest :

```
@isTest
global class ParkServiceMock implements WebServiceMock {
   global void doInvoke(
           Object stub,
           Object request,
           Map<String, Object> response,
           String endpoint,
           String soapAction,
           String requestName,
           String responseNS,
           String responseName,
           String responseType) {
        // start - specify the response you want to send
        parkService.byCountryResponse response_x = new parkService.byCountryResponse();
        response_x.return_x = new List<String>{'Hamburg

'Bavarian Forest National Park'};

        //calculatorServices.doAddResponse response_x = new calculatorServices.doAddResponse();
        //response_x.return_x = 3.0;
        // end
        response.put('response_x', response_x);
   }
}
```

Apex Web Services :

1) Apex Class Account Manager :

```

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


    @HttpGet
    global static account getAccount() {

        RestRequest request = RestContext.request;

        String accountId = request.requestURI.substring(request.requestURI.las

          request.requestURI.lastIndexOf('/'));
        List<Account> a = [select id, name, (select id, name from contacts) from account where id = :accountId];
        List<contact> co = [select id, name from contact where account.id = :accountId];
        system.debug('** a[0]= '+ a[0]);
        return a[0];

    }

}
```

2) Apex Class Account Manager Test :

```
@istest
public class AccountManagerTest {
@istest static void testGetContactsByAccountId() {
Id recordId = createTestRecord();
// Set up a test request
RestRequest request = new RestRequest();
request.requestUri =
'https://yourInstance.salesforce.com/services/apexrest/Accounts/

request.httpMethod = 'GET';
RestContext.request = request;

Account thisAccount = AccountManager.getAccount();
System.assert(thisAccount!= null);
System.assertEquals('Test record', thisAccount.Name);
}

// Helper method
static Id createTestRecord() {

// Create test record
Account accountTest = new Account(
Name='Test record');
insert accountTest;
Contact contactTest = new Contact(
FirstName='John',
LastName='Doe',
AccountId=accountTest.Id
);
return accountTest.Id;
}
}
```

- **Visual Force :**

1) Display Image :

```
<apex:page showHeader="false">
    <apex:image
url="https://developer.salesforce.com/files/salesforce-developer-

</apex:page>
```

2) Display User Info :

```
<apex:page >
    {! $User.FirstName}
</apex:page>
```

3) Contact View :

```
<apex:page standardController="Contact">
    <apex:pageBlockSection>
        First Name : {! Contact.FirstName}
        Last Name: {! Contact.LastName}
        Owner Email : {! Contact.Owner.Email}
    </apex:pageBlockSection>
</apex:page>
```

4) **Opp View :**

```
<apex:page standardController="Opportunity">
    <apex:outputField value ="{! Opportunity.Name}"/>
    <apex:outputField value ="{! Opportunity.Amount}"/>
    <apex:outputField value ="{! Opportunity.CloseDate}"/>
    <apex:outputField value ="{! Opportunity.Account.Name}"/>
</apex:page>
```

5) **Create Contact :**

```
<apex:page standardController="Contact">
    <apex:form>
    <apex:pageBlockSection>
        <apex:inputField value ="{! Contact.FirstName}"/>
        <apex:inputField value ="{! Contact.LastName}"/>
        <apex:inputField value ="{! Contact.Email}"/>
        </apex:pageBlockSection>
        <apex:commandButton action="{! save}" value
="Save"/>
    </apex:form>
</apex:page>
```

6) Account List :

```

<apex:page standardController="Account" recordSetVar="accounts">
    <apex:form>
    <apex:pageBlock>
        <apex:repeat value="{!Accounts}" id="acccount_list"
rendered="true" var="a">

        <li>
            <apex:outputLink value="/{!a.id}"/>
            <apex:outputText value="{!a.name}"/>

       </li>


        </apex:repeat>


        </apex:pageBlock>


    </apex:form>
</apex:page>
```

7) Show Image :

```
<apex:page >
    <apex:image url="{! URLFOR($Resource.vfimagetest,

</apex:page>
```

8) New case List Controller Apex Class :

```
public class NewCaseListController {
    public List<Case> getNewCases(){
        List<Case> filterList = [Select ID, CaseNumber from Case where status ='New'];
        return filterList;
    }
}
```

9) New Case List Visual Force Page :

```
<apex:page controller="NewCaseListController">
    <apex:repeat var="case" value="{!newCases}">
        <apex:outputLink value="/{!case.ID}">
        <apex:outputText value="{!case.CaseNumber}"></apex:outputText>
        </apex:outputLink>
    </apex:repeat>
</apex:page>
```

10) Contact Form :

```
<apex:page >
    Hello
</apex:page>
```

11) Contact Form :

```
<apex:page standardController="Contact">

          <head>
                    <meta charset="utf-8" />
      <meta name="viewport" content="width=device-width, initial-

      <title>Quick Start: Visualforce</title>
      <!-- Import the Design System style sheet -->
      <apex:slds />

         </head>
         <body>

                  <apex:form>
      <apex:pageBlock title="New Contact">
        <!--Buttons -->
         <apex:pageBlockButtons>
            <apex:commandButton action="{!save}" value="Save"/>
         </apex:pageBlockButtons>
         <!--Input form -->
         <apex:pageBlockSection columns="1">
         <apex:inputField value="{!Contact.Firstname}"/>
         <apex:inputField value="{!Contact.Lastname}"/>
         <apex:inputField value="{!Contact.Email}"/>
        </apex:pageBlockSection>
      </apex:pageBlock>
      </apex:form>

         </body>

</apex:page>
```

Asynchronous Methods :

1) Account Processor Apex Class :

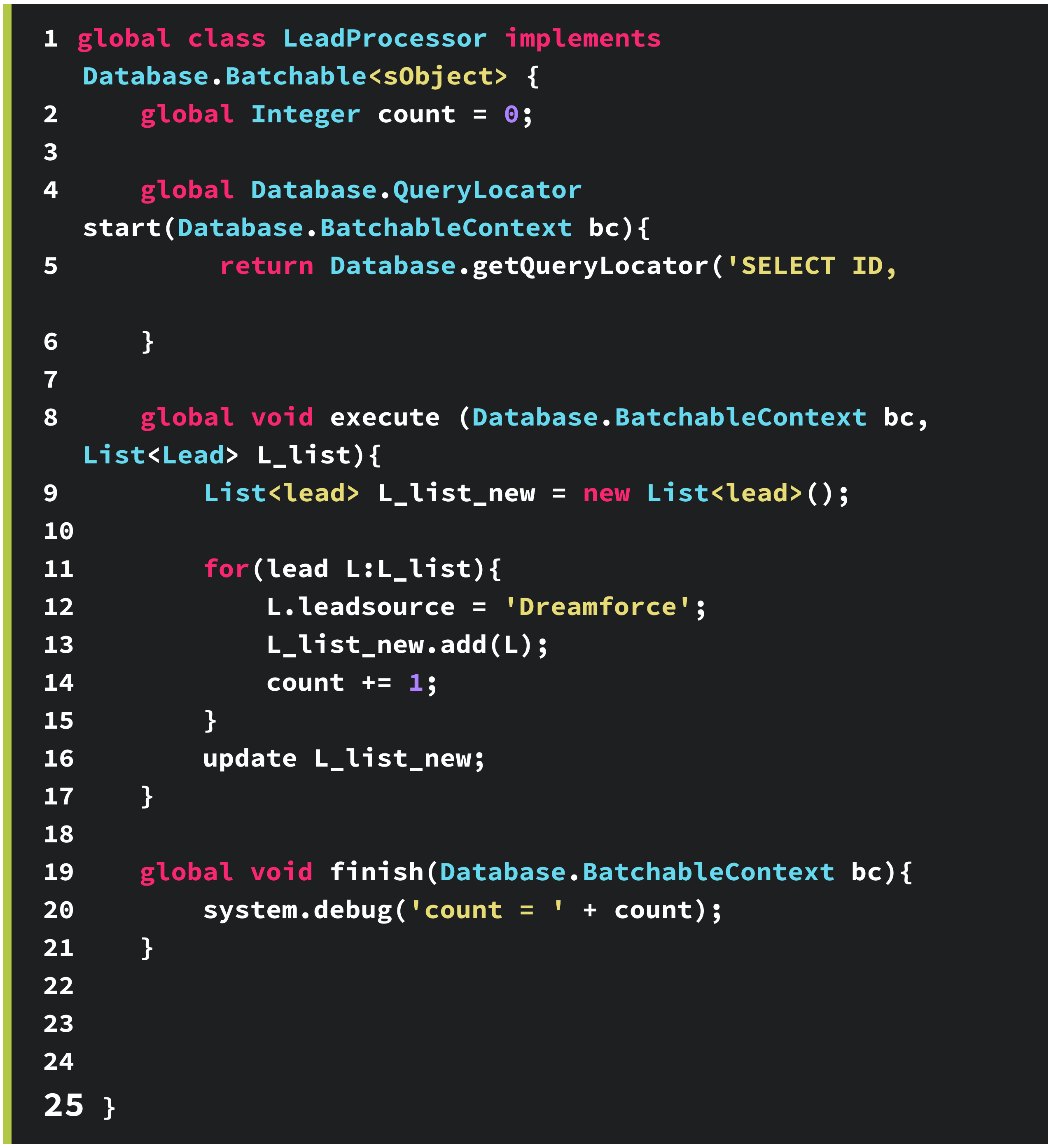
```
public class AccountProcessor {

    @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;
    }
}
```

2)Account Processor Apex Class Test :

```
@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);
    }

}
```

Use Batch Apex :

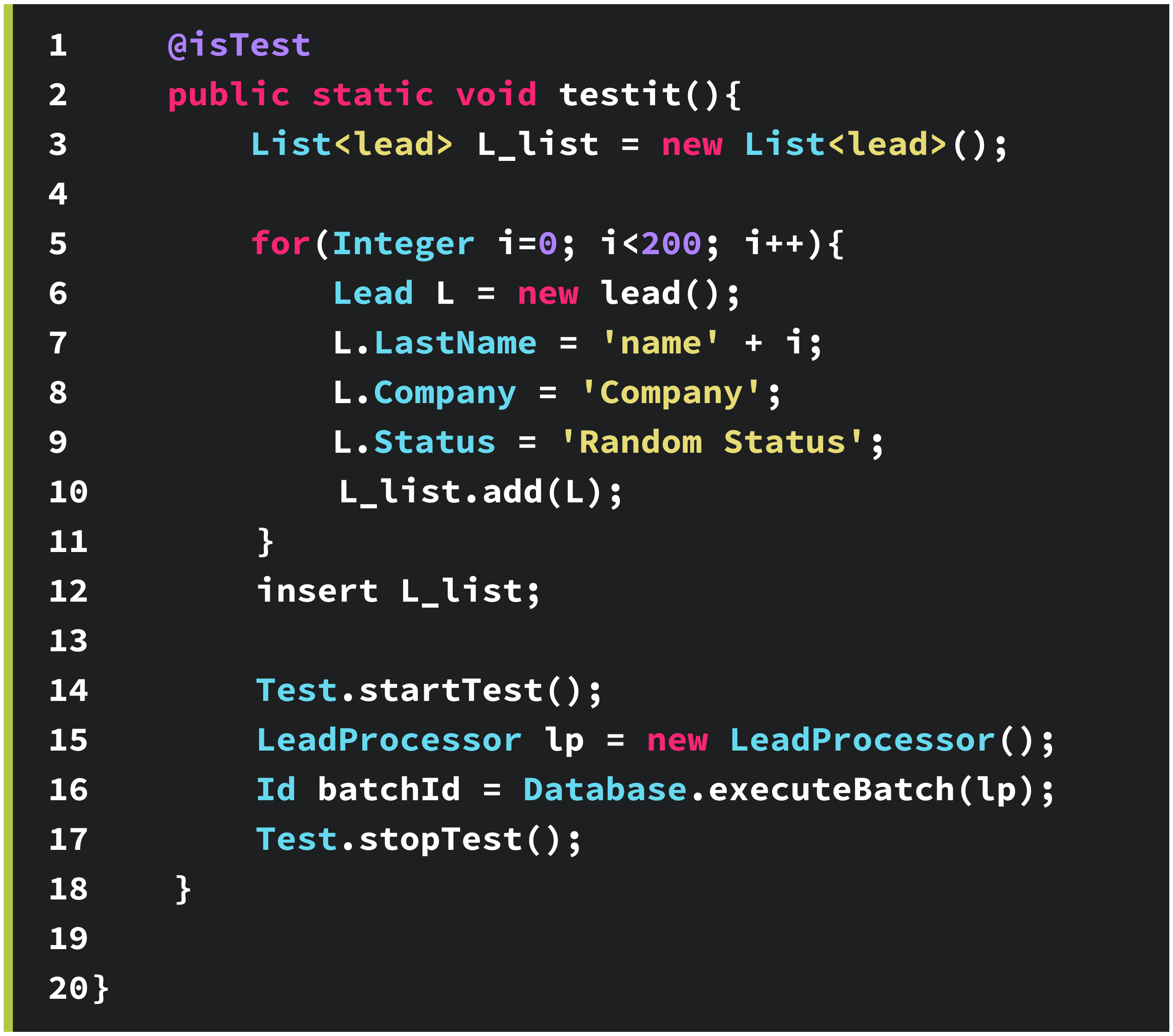
1) LeadProcessor Apex Class :



```
global class LeadProcessor implements
Database.Batchable<sObject> {
    global Integer count = 0;

    global Database.QueryLocator
start(Database.BatchableContext bc){
         return Database.getQueryLocator('SELECT ID,

    }

    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;
    }

    global void finish(Database.BatchableContext bc){
        system.debug('count = ' + count);
    }



}
```

2) LeadProcessor Apex Class Test :

@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();
    }

}
```

Control Processes With Queueable Apex :

1) AddPrimaryContact Apex Class :

```
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;
        }
    }
}
```

2) AddPrimaryContact Apex Class Test :

```
@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

        }
        for(Integer j=0;j<50;j++){
            testAccounts.add(new Account(Name='Account

        }
        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')]);
    }

}
```

Schedule Jobs Using The Apex Scheduler :

1) DailyLeadProcessor Apex Class :

```
global class DailyLeadProcessor implements Schedulable {
 global void execute(SchedulableContext ctx) {
        List<Lead> lList = [Select Id, LeadSource from Lead where LeadSource = null];

        if(!lList.isEmpty()) {
   for(Lead l: lList) {
    l.LeadSource = 'Dreamforce';
   }
   update lList;
  }
    }

}
```

2) DailyLeadProcessor Apex Class Test :

```
@isTest
public 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 -

            leads.add(lead);
        }

        insert leads;

        Test.startTest();
        // Schedule the test job
        String jobId = System.schedule('Update LeadSource to


        // Stopping the test will run the job synchronously
        Test.stopTest();
    }

}
```

APEX SPECIALIST SUPERBADGE :

1) Automate Record Creation :

Apex Class :

```
public with sharing class MaintenanceRequestHelper {
    public static void updateworkOrders(List<Case> updWorkOrders, Map<Id,Case> nonUpdCaseMap) {
        Set<Id> validIds = 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'){
                    validIds.add(c.Id);


                }
            }
        }

        if (!validIds.isEmpty()){
            List<Case> newCases = 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 :validIds]);
            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 :ValidIds 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

                    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));
                } else {
                    nc.Date_Due__c = Date.today().addDays((Integer) cc.Equipment__r.maintenance_Cycle__c);
                }

                newCases.add(nc);
            }

           insert newCases;

           List<Equipment_Maintenance_Item__c> clonedWPs = new List<Equipment_Maintenance_Item__c>();
           for (Case nc : newCases){
                for (Equipment_Maintenance_Item__c wp : closedCasesM.get(nc.ParentId).Equipment_Maintenance

                    Equipment_Maintenance_Item__c wpClone = wp.clone();
                    wpClone.Maintenance_Request__c = nc.Id;
                    ClonedWPs.add(wpClone);

                }
            }
            insert ClonedWPs;
        }
    }
}
```

Apex Trigger :

```
trigger MaintenanceRequest on Case (before update,
after update) {

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

MaintenanceRequestHelper.updateWorkOrders(Trigger.N

    }

}
```

2) Synchronize Salesforce Data With An External System :

```
public with sharing class WarehouseCalloutService implements Queueable {
    private static final String WAREHOUSE_URL = 'https://th-superbadge-

    //class that makes a REST callout to an external warehouse system to get a list of equipment that needs to be updated.
    //The callout's JSON response returns the equipment records that you upsert in Salesforce.

    @future(callout=true)
    public static void runWarehouseEquipmentSync(){
        Http http = new Http();
        HttpRequest request = new HttpRequest();

        request.setEndpoint(WAREHOUSE_URL);
        request.setMethod('GET');
        HttpResponse response = http.send(request);

        List<Product2> warehouseEq = new List<Product2>();

        if (response.getStatusCode() == 200){
            List<Object> jsonResponse = (List<Object>)JSON.deserializeUntyped(response.getB

            System.debug(response.getBody());

            //class maps the following fields: replacement part (always true), cost, current inventory, lifespan, maintenance cycle, and warehouse SKU
            //warehouse SKU will be external ID for identifying which equipment records to update within Salesforce
            for (Object eq : jsonResponse){
                Map<String,Object> mapJson = (Map<String,Object>)eq;
                Product2 myEq = new Product2();
                myEq.Replacement_Part__c = (Boolean) mapJson.get('replacement');
                myEq.Name = (String) mapJson.get('name');
                myEq.Maintenance_Cycle__c = (Integer) mapJson.get('maintenanceperiod');
                myEq.Lifespan_Months__c = (Integer) mapJson.get('lifespan');
                myEq.Cost__c = (Integer) mapJson.get('cost');
                myEq.Warehouse_SKU__c = (String) mapJson.get('sku');
                myEq.Current_Inventory__c = (Double) mapJson.get('quantity');
                myEq.ProductCode = (String) mapJson.get('_id');
                warehouseEq.add(myEq);
            }

```

```
            if (warehouseEq.size() > 0){
                upsert warehouseEq;
                System.debug('Your equipment was

            }
        }
    }

    public static void execute (QueueableContext
context){
        runWarehouseEquipmentSync();
    }

}
```

3) Schedule Synchronization :

```

global with sharing class WarehouseSyncSchedule implements
Schedulable{
    global void execute(SchedulableContext ctx){
        System.enqueueJob(new WarehouseCalloutService());
    }
}
```

4) Test Automation Logic :

Apex test :

```
@istest
public with sharing class MaintenanceRequestHelperTest {

    private static final string STATUS_NEW = 'New';
    private static final string WORKING = 'Working';
    private static final string CLOSED = 'Closed';
    private static final string REPAIR = 'Repair';
    private static final string REQUEST_ORIGIN = 'Web';
    private static final string REQUEST_TYPE = 'Routine

    private static final string REQUEST_SUBJECT = 'Testing


    PRIVATE STATIC Vehicle__c createVehicle(){
        Vehicle__c Vehicle = new Vehicle__C(name = 'SuperTruck');
        return Vehicle;
    }

    PRIVATE STATIC Product2 createEq(){
        product2 equipment = new product2(name = 'SuperEquipment',
                                          lifespan_months__C = 10,
                                          maintenance_cycle__C = 10,
                                          replacement_part__c = true);
        return equipment;
    }

    PRIVATE STATIC Case createMaintenanceRequest(id vehicleId, id equipmentId){
        case cs = new case(Type=REPAIR,
                          Status=STATUS_NEW,
                          Origin=REQUEST_ORIGIN,
                          Subject=REQUEST_SUBJECT,
                          Equipment__c=equipmentId,
                          Vehicle__c=vehicleId);
        return cs;
    }

    PRIVATE STATIC Equipment_Maintenance_Item__c createWorkPart(id equipmentId,id requestId){
        Equipment_Maintenance_Item__c wp = new Equipment_Maintenance_Item__c(Equipment__c = equipmentId,
        Maintenance_Request__c = requestId);
        return wp;
    }


    @istest
    private static void testMaintenanceRequestPositive(){
        Vehicle__c vehicle = createVehicle();
        insert vehicle;
        id vehicleId = vehicle.Id;

        Product2 equipment = createEq();
        insert equipment;
        id equipmentId = equipment.Id;

        case somethingToUpdate = createMaintenanceRequest(vehicleId,equipmentId);
        insert somethingToUpdate;

        Equipment_Maintenance_Item__c workP = createWorkPart(equipmentId,somethingToUpdate.id);
        insert workP;

        test.startTest();
        somethingToUpdate.status = CLOSED;
        update somethingToUpdate;
        test.stopTest();

        Case newReq = [Select id, subject, type, Equipment__c, Date_Reported__c, Vehicle__c, Date_Due__c
```

```
                      from case
                      where status =:STATUS_NEW];

        Equipment_Maintenance_Item__c workPart = [select id
                                                 from Equipment_Maintenance_Item__c
                                                 where Maintenance_Request__c =:newReq.Id];

        system.assert(workPart != null);
        system.assert(newReq.Subject != null);
        system.assertEquals(newReq.Type, REQUEST_TYPE);
        SYSTEM.assertEquals(newReq.Equipment__c, equipmentId);
        SYSTEM.assertEquals(newReq.Vehicle__c, vehicleId);
        SYSTEM.assertEquals(newReq.Date_Reported__c, system.today());
    }

    @istest
    private static void testMaintenanceRequestNegative(){
        Vehicle__C vehicle = createVehicle();
        insert vehicle;
        id vehicleId = vehicle.Id;

        product2 equipment = createEq();
        insert equipment;
        id equipmentId = equipment.Id;

        case emptyReq = createMaintenanceRequest(vehicleId,equipmentId);
        insert emptyReq;

        Equipment_Maintenance_Item__c workP = createWorkPart(equipmentId, emptyReq.Id);
        insert workP;

        test.startTest();
        emptyReq.Status = WORKING;
        update emptyReq;
        test.stopTest();

        list<case> allRequest = [select id
                                 from case];

        Equipment_Maintenance_Item__c workPart = [select id
                                                  from Equipment_Maintenance_Item__c
                                                  where Maintenance_Request__c = :emptyReq.Id];

        system.assert(workPart != null);
        system.assert(allRequest.size() == 1);
    }

    @istest
    private static void testMaintenanceRequestBulk(){
        list<Vehicle__C> vehicleList = new list<Vehicle__C>();
        list<Product2> equipmentList = new list<Product2>();
        list<Equipment_Maintenance_Item__c> workPartList = new list<Equipment_Maintenance_Item__c>();
        list<case> requestList = new list<case>();
        list<id> oldRequestIds = new list<id>();

        for(integer i = 0; i < 300; i++){
           vehicleList.add(createVehicle());
            equipmentList.add(createEq());
        }
        insert vehicleList;
        insert equipmentList;

        for(integer i = 0; i < 300; i++){
            requestList.add(createMaintenanceRequest(vehicleList.get(i).id, equipmentList.get(i).id));
        }
        insert requestList;

        for(integer i = 0; i < 300; i++){
            workPartList.add(createWorkPart(equipmentList.get(i).id,
requestList.get(i).id));
        }
        insert workPartList;

        test.startTest();
        for(case req : requestList){
            req.Status = CLOSED;
            oldRequestIds.add(req.Id);
        }
        update requestList;
        test.stopTest();

        list<case> allRequests = [select id
                                 from case
                                 where status =: STATUS_NEW];

        list<Equipment_Maintenance_Item__c> workParts = [select id
                                                         from Equipment_Maintenance_Item__c
                                                         where Maintenance_Request__c in: oldRequestIds];

        system.assert(allRequests.size() == 300);
    }
}
```

5) Test Callout Logic :

WarehouseCalloutServiceTest :

```
@isTest

private class WarehouseCalloutServiceTest {
@isTest
static void testWareHouseCallout(){
Test.startTest();
// implement mock callout test here
Test.setMock(HTTPCalloutMock.class, new WarehouseCalloutServiceMock());
WarehouseCalloutService.runWarehouseEquipmentSync();
WarehouseCalloutService apc = new WarehouseCalloutService();
System.enqueueJob(apc);
Test.stopTest();
System.assertEquals(1, [SELECT count() FROM Product2]);
}
}
```

WarehouseCalloutServiceMockTest :

```apex
@isTest
global class WarehouseCalloutServiceMock implements HttpCalloutMock {
    // implement http mock callout
    global static HttpResponse respond(HttpRequest request) {

        HttpResponse response = new HttpResponse();
        response.setHeader('Content-Type', 'application/json');
        response.setBody('[{"_id":"55d66226726b611100aaf74"

        response.setStatusCode(200);

        return response;
    }
}
```

6) Scheduling Logic :

WarehouseSyncScheduleTest :

```
@isTest
public class WarehouseSyncScheduleTest {

    @isTest static void WarehousescheduleTest(){
        String scheduleTime = '00 00 01 * * ?';
        Test.startTest();
        Test.setMock(HttpCalloutMock.class, new WarehouseCalloutServiceMock());
        String jobID=System.schedule('Warehouse Time To Schedule

        Test.stopTest();
        //Contains schedule information for a scheduled job. CronTrigger is similar to a cron job on UNIX systems.
        // This object is available in API version 17.0 and later.
        CronTrigger a=[SELECT Id FROM CronTrigger where NextFireTime > today];
        System.assertEquals(jobID, a.Id,'Schedule ');


    }
}
```