Apex Triggers:

AccountAddressTrigger.apxt:

```
trigger AccountAddressTrigger on Account (before insert, before update) {
    if(Trigger.isInsert){
        for(Account a:Trigger.new){
            IF(a.Match_Billing_Address__c == True && a.BillingPostalCode!=Null){
                a.ShippingPostalCode = a.BillingPostalCode;
            }
        }
    }
    else if(Trigger.isUpdate){
        for(Account a:Trigger.new){
            IF(a.Match_Billing_Address__c == True){
                a.ShippingPostalCode = a.BillingPostalCode;
            }
        }
    }
}
```

ClosedOpportunityTrigger.apxt:

```
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;
    }
}
```

AccountDeletion.apxt:

```
trigger AccountDeletion on Account (before insert)
{
    for (Account a : [SELECT Id FROM Account
                      WHERE Id IN (SELECT AccountId
FROM Opportunity) AND
                      Id IN :Trigger.old]) {
        Trigger.oldMap.get(a.Id).addError(
            'Cannot delete account with related
opportunities.');
    }
}
```

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');
          }

     }
}
```

MaintenanceRequest.apxt:

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

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

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

    }

}
```

Bulk Apex Triggers:

MyTriggerNotBulk.apxt:

```
trigger MyTriggerNotBulk on Account(before insert) {
    Account a = Trigger.New[0];
    a.Description = 'New description';
}
```

MyTriggerBulk.apxt:

```
trigger MyTriggerBulk on Account(before insert) {
    for(Account a : Trigger.New) {
        a.Description = 'New description';
    }
}
```

SoqlTriggerNotBulk.apxt:

```
trigger SoqlTriggerNotBulk on Account(after update) {
    for(Account a : Trigger.New) {
        // Get child records for each account
        // Inefficient SOQL query as it runs once for
each account!
        Opportunity[] opps = [SELECT Id,Name,CloseDate
                              FROM Opportunity WHERE
AccountId=:a.Id];

        // Do some other processing
    }
```

```
}
```

AddRelatedRecord.apxt:

```
trigger AddRelatedRecord on Account(after insert,
after update) {
    List<Opportunity> oppList = new
List<Opportunity>();

    // Add an opportunity for each account if it
doesn't already have one.
    // Iterate over accounts that are in this
trigger but that don't have opportunities.
    for (Account a : [SELECT Id,Name FROM Account
                      WHERE Id IN :Trigger.New AND
                      Id NOT IN (SELECT AccountId
FROM Opportunity)]) {
        // Add a default opportunity for this
account
        oppList.add(new Opportunity(Name=a.Name +
' Opportunity',

StageName='Prospecting',

CloseDate=System.today().addMonths(1),

AccountId=a.Id));
    }

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

Test Apex Triggers:

AccountDeletion.apxt:

```
trigger AccountDeletion on Account (before delete) {
```

```
    // Prevent the deletion of accounts if they have
related opportunities.
    for (Account a : [SELECT Id FROM Account
                      WHERE Id IN (SELECT AccountId FROM
Opportunity) AND
                      Id IN :Trigger.old]) {
        Trigger.oldMap.get(a.Id).addError(
            'Cannot delete account with related
opportunities.');
    }
}
```

TestAccountDeletion.apxc:

```
@isTest
private class TestAccountDeletion {
    @isTest static void
TestDeleteAccountWithOneOpportunity() {
        // Test data setup
        // Create an account with an opportunity, and
then try to delete it
        Account acct = new Account(Name='Test
Account');
        insert acct;
        Opportunity opp = new
Opportunity(Name=acct.Name + ' Opportunity',

StageName='Prospecting',

CloseDate=System.today().addMonths(1),

AccountId=acct.Id);
        insert opp;
        // Perform test
        Test.startTest();
        Database.DeleteResult result =
Database.delete(acct, false);
        Test.stopTest();
        // Verify
        // In this case the deletion should have been
stopped by the trigger,
        // so verify that we got back an error.
```

```
        System.assert(!result.isSuccess());
        System.assert(result.getErrors().size() > 0);
        System.assertEquals('Cannot delete account with
related opportunities.',

result.getErrors()[0].getMessage());
    }
}
```

Create Test Data for Apex Tests:

TestDataFactory.apxc:

```
@isTest
public class TestDataFactory {
   public static List<Account> createAccountsWithOpps(Integer
numAccts, Integer numOppsPerAcct) {
      List<Account> accts = new List<Account>();
      for(Integer i=0;i<numAccts;i++) {
         Account a = new Account(Name='TestAccount' + i);
         accts.add(a);
      }
      insert accts;
      List<Opportunity> opps = new List<Opportunity>();
      for (Integer j=0;j<numAccts;j++) {
         Account acct = accts[j];
         // For each account just inserted, add opportunities
         for (Integer k=0;k<numOppsPerAcct;k++) {
            opps.add(new Opportunity(Name=acct.Name + ' Opportunity ' +
k,
                           StageName='Prospecting',
                           CloseDate=System.today().addMonths(1),
                           AccountId=acct.Id));
         }
      }
      // Insert all opportunities for all accounts.
      insert opps;
```

```
        return accts;
    }
}
```

TestAccountDeletion.apxc:

```
@isTest
private class TestAccountDeletion {
    @isTest static void TestDeleteAccountWithOneOpportunity() {
        // Test data setup
        // Create one account with one opportunity by calling a utility method
        Account[] accts = TestDataFactory.createAccountsWithOpps(1,1);
        // Perform test
        Test.startTest();
        Database.DeleteResult result = Database.delete(accts[0], false);
        Test.stopTest();
        // Verify that the deletion should have been stopped by the trigger,
        // so check that we got back an error.
        System.assert(!result.isSuccess());
        System.assert(result.getErrors().size() > 0);
        System.assertEquals('Cannot delete account with related
opportunities.',
                            result.getErrors()[0].getMessage());
    }
    @isTest static void TestDeleteAccountWithNoOpportunities() {
        // Test data setup
        // Create one account with no opportunities by calling a utility method
        Account[] accts = TestDataFactory.createAccountsWithOpps(1,0);
        // Perform test
        Test.startTest();
        Database.DeleteResult result = Database.delete(accts[0], false);
        Test.stopTest();
        // Verify that the deletion was successful
        System.assert(result.isSuccess());
    }
    @isTest static void TestDeleteBulkAccountsWithOneOpportunity() {
        // Test data setup
        // Create accounts with one opportunity each by calling a utility
method
        Account[] accts = TestDataFactory.createAccountsWithOpps(200,1);
        // Perform test
```

```
        Test.startTest();
        Database.DeleteResult[] results = Database.delete(accts, false);
        Test.stopTest();
        // Verify for each record.
        // In this case the deletion should have been stopped by the trigger,
        // so check that we got back an error.
        for(Database.DeleteResult dr : results) {
            System.assert(!dr.isSuccess());
            System.assert(dr.getErrors().size() > 0);
            System.assertEquals('Cannot delete account with related
opportunities.',
                                dr.getErrors()[0].getMessage());
        }
    }
    @isTest static void TestDeleteBulkAccountsWithNoOpportunities() {
        // Test data setup
        // Create accounts with no opportunities by calling a utility method
        Account[] accts = TestDataFactory.createAccountsWithOpps(200,0);
        // Perform test
        Test.startTest();
        Database.DeleteResult[] results = Database.delete(accts, false);
        Test.stopTest();
        // For each record, verify that the deletion was successful
        for(Database.DeleteResult dr : results) {
            System.assert(dr.isSuccess());
        }
    }
}
```

Apex Classes:

Use Future Methods:

SomeClass.apxc:

```
public class SomeClass {
  @future
  public static void someFutureMethod(List<Id> recordIds) {
    List<Account> accounts = [Select Id, Name from Account Where Id IN
:recordIds];
    // process account records to do awesome stuff
  }
}
```

Sample Callout Code:

SMSUtils.apxc:

```
public class SMSUtils {
    // Call async from triggers, etc, where callouts are not permitted.
    @future(callout=true)
    public static void sendSMSAsync(String fromNbr, String toNbr, String m)
{
        String results = sendSMS(fromNbr, toNbr, m);
        System.debug(results);
    }
    // Call from controllers, etc, for immediate processing
    public static String sendSMS(String fromNbr, String toNbr, String m) {
        // Calling 'send' will result in a callout
        String results = SmsMessage.send(fromNbr, toNbr, m);
        insert new SMS_Log__c(to__c=toNbr, from__c=fromNbr,
msg__c=results);
        return results;
    }
}
```

Test Classes:

SMSCalloutMock.apxc:

```
@isTest
public class SMSCalloutMock implements HttpCalloutMock {
    public HttpResponse respond(HttpRequest req) {
        // Create a fake response
        HttpResponse res = new HttpResponse();
        res.setHeader('Content-Type', 'application/json');
        res.setBody('{"status":"success"}');
        res.setStatusCode(200);
        return res;
    }
}
```

Test_SMSUtils.apxc:

```
@IsTest
private class Test_SMSUtils {
  @IsTest
  private static void testSendSms() {
    Test.setMock(HttpCalloutMock.class, new SMSCalloutMock());
    Test.startTest();
      SMSUtils.sendSMSAsync('111', '222', 'Greetings!');
    Test.stopTest();
    // runs callout and check results
    List<SMS_Log__c> logs = [select msg__c from SMS_Log__c];
    System.assertEquals(1, logs.size());
    System.assertEquals('success', logs[0].msg__c);
  }
}
```

Use Batch Apex:

UpdateContactAddresses.apxc:

```
public class UpdateContactAddresses implements
    Database.Batchable<sObject>, Database.Stateful {
    // instance member to retain state across transactions
    public Integer recordsProcessed = 0;
    public Database.QueryLocator start(Database.BatchableContext bc) {
        return Database.getQueryLocator(
            'SELECT ID, BillingStreet, BillingCity, BillingState, ' +
            'BillingPostalCode, (SELECT ID, MailingStreet, MailingCity, ' +
            'MailingState, MailingPostalCode FROM Contacts) FROM Account '
+
            'Where BillingCountry = \'USA\''
        );
    }
    public void execute(Database.BatchableContext bc, List<Account>
scope){
        // process each batch of records
        List<Contact> contacts = new List<Contact>();
        for (Account account : scope) {
            for (Contact contact : account.contacts) {
                contact.MailingStreet = account.BillingStreet;
                contact.MailingCity = account.BillingCity;
                contact.MailingState = account.BillingState;
                contact.MailingPostalCode = account.BillingPostalCode;
                // add contact to list to be updated
                contacts.add(contact);
                // increment the instance member counter
                recordsProcessed = recordsProcessed + 1;
            }
        }
        update contacts;
    }
    public void finish(Database.BatchableContext bc){
        System.debug(recordsProcessed + ' records processed. Shazam!');
        AsyncApexJob job = [SELECT Id, Status, NumberOfErrors,
            JobItemsProcessed,
            TotalJobItems, CreatedBy.Email
            FROM AsyncApexJob
            WHERE Id = :bc.getJobId()];
        // call some utility to send email
        EmailUtils.sendMessage(job, recordsProcessed);
```

```
    }
}
```

Testing Batch Apex:

UpdateContactAddressesTest.apxc:

```
@isTest
private class UpdateContactAddressesTest {
    @testSetup
    static void setup() {
        List<Account> accounts = new List<Account>();
        List<Contact> contacts = new List<Contact>();
        // insert 10 accounts
        for (Integer i=0;i<10;i++) {
            accounts.add(new Account(name='Account '+i,
                billingcity='New York', billingcountry='USA'));
        }
        insert accounts;
        // find the account just inserted. add contact for each
        for (Account account : [select id from account]) {
            contacts.add(new Contact(firstname='first',
                lastname='last', accountId=account.id));
        }
        insert contacts;
    }
    @isTest static void test() {
        Test.startTest();
        UpdateContactAddresses uca = new UpdateContactAddresses();
        Id batchId = Database.executeBatch(uca);
        Test.stopTest();
        // after the testing stops, assert records were updated properly
        System.assertEquals(10, [select count() from contact where
MailingCity = 'New York']);
    }
}
```

Control Processes with Queueable

Apex:

UpdateParentAccount.apxc:

```
public class UpdateParentAccount implements Queueable {
   private List<Account> accounts;
   private ID parent;
   public UpdateParentAccount(List<Account> records, ID id) {
      this.accounts = records;
      this.parent = id;
   }
   public void execute(QueueableContext context) {
      for (Account account : accounts) {
        account.parentId = parent;
        // perform other processing or callout
      }
      update accounts;
   }
}
```

Testing Queueable Apex:

UpdateParentAccountTest.apxc:

```
@isTest
public class UpdateParentAccountTest {
   @testSetup
   static void setup() {
      List<Account> accounts = new List<Account>();
      // add a parent account
      accounts.add(new Account(name='Parent'));
      // add 100 child accounts
      for (Integer i = 0; i < 100; i++) {
         accounts.add(new Account(
            name='Test Account'+i
         ));
      }
      insert accounts;
```

```
    }
    static testmethod void testQueueable() {
        // query for test data to pass to queueable class
        Id parentId = [select id from account where name = 'Parent'][0].Id;
        List<Account> accounts = [select id, name from account where name
like 'Test Account%'];
        // Create our Queueable instance
        UpdateParentAccount updater = new
UpdateParentAccount(accounts, parentId);
        // startTest/stopTest block to force async processes to run
        Test.startTest();
        System.enqueueJob(updater);
        Test.stopTest();
        // Validate the job ran. Check if record have correct parentId now
        System.assertEquals(100, [select count() from account where
parentId = :parentId]);
    }
}
```

Testing Scheduled Apex:

RemindOppyOwnersTest.apxc:

```
@isTest
private class RemindOppyOwnersTest {
    // 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<Opportunity> opptys = new List<Opportunity>();
        Date closeDate = Date.today().addDays(-7);
        for (Integer i=0; i<10; i++) {
            Opportunity o = new Opportunity(
                Name = 'Opportunity ' + i,
                CloseDate = closeDate,
                StageName = 'Prospecting'
```

```
            );
            opptys.add(o);
        }
        insert opptys;
        // Get the IDs of the opportunities we just inserted
        Map<Id, Opportunity> opptyMap = new Map<Id, Opportunity>(opptys);
        List<Id> opptyIds = new List<Id>(opptyMap.keySet());
        Test.startTest();
        // Schedule the test job
        String jobId = System.schedule('ScheduledApexTest',
            CRON_EXP,
            new RemindOpptyOwners());
        // Verify the scheduled job has not run yet.
        List<Task> lt = [SELECT Id
            FROM Task
            WHERE WhatId IN :opptyIds];
        System.assertEquals(0, lt.size(), 'Tasks exist before job has run');
        // Stopping the test will run the job synchronously
        Test.stopTest();
        // Now that the scheduled job has executed,
        // check that our tasks were created
        lt = [SELECT Id
            FROM Task
            WHERE WhatId IN :opptyIds];
        System.assertEquals(opptyIds.size(),
            lt.size(),
            'Tasks were not created');
    }
}
```

CreateDefaultData.apxc:

```
public with sharing class CreateDefaultData{
    Static Final String TYPE_ROUTINE_MAINTENANCE = 'Routine
Maintenance';
    //gets value from custom metadata How_We_Roll_Settings__mdt to
know if Default data was created
    @AuraEnabled
    public static Boolean isDataCreated() {
        How_We_Roll_Settings__c          customSetting =
How_We_Roll_Settings__c.getOrgDefaults();
```

```
        return customSetting.Is_Data_Created__c;
    }

    //creates Default Data for How We Roll application
    @AuraEnabled
    public static void createDefaultData(){
        List<Vehicle__c> vehicles = createVehicles();
        List<Product2> equipment = createEquipment();
        List<Case> maintenanceRequest =
createMaintenanceRequest(vehicles);
        List<Equipment_Maintenance_Item__c> joinRecords =
createJoinRecords(equipment, maintenanceRequest);

        updateCustomSetting(true);
    }



    public static void updateCustomSetting(Boolean isDataCreated){
        How_We_Roll_Settings__c          customSetting =
How_We_Roll_Settings__c.getOrgDefaults();
        customSetting.Is_Data_Created__c = isDataCreated;
        upsert customSetting;
    }

    public static List<Vehicle__c> createVehicles(){
        List<Vehicle__c> vehicles = new List<Vehicle__c>();
        vehicles.add(new Vehicle__c(Name = 'Toy Hauler RV',
Air_Conditioner__c = true, Bathrooms__c = 1, Bedrooms__c = 1, Model__c
= 'Toy Hauler RV'));
        vehicles.add(new Vehicle__c(Name = 'Travel Trailer RV',
Air_Conditioner__c = true, Bathrooms__c = 2, Bedrooms__c = 2, Model__c
= 'Travel Trailer RV'));
        vehicles.add(new Vehicle__c(Name = 'Teardrop Camper',
Air_Conditioner__c = true, Bathrooms__c = 1, Bedrooms__c = 1, Model__c
= 'Teardrop Camper'));
        vehicles.add(new Vehicle__c(Name = 'Pop-Up Camper',
Air_Conditioner__c = true, Bathrooms__c = 1, Bedrooms__c = 1, Model__c
= 'Pop-Up Camper'));
        insert vehicles;
        return vehicles;
```

```
    }

    public static List<Product2> createEquipment(){
        List<Product2> equipments = new List<Product2>();
        equipments.add(new Product2(Warehouse_SKU__c =
'55d66226726b611100aaf741',name = 'Generator 1000 kW',
Replacement_Part__c = true,Cost__c = 100 ,Maintenance_Cycle__c =
100));
        equipments.add(new Product2(name = 'Fuse
20B',Replacement_Part__c = true,Cost__c = 1000, Maintenance_Cycle__c =
30  ));
        equipments.add(new Product2(name = 'Breaker
13C',Replacement_Part__c = true,Cost__c = 100  , Maintenance_Cycle__c =
15));
        equipments.add(new Product2(name = 'UPS 20
VA',Replacement_Part__c = true,Cost__c = 200  , Maintenance_Cycle__c =
60));
        insert equipments;
        return equipments;

    }

    public static List<Case> createMaintenanceRequest(List<Vehicle__c>
vehicles){
        List<Case> maintenanceRequests = new List<Case>();
        maintenanceRequests.add(new Case(Vehicle__c = vehicles.get(1).Id,
Type = TYPE_ROUTINE_MAINTENANCE, Date_Reported__c =
Date.today()));
        maintenanceRequests.add(new Case(Vehicle__c = vehicles.get(2).Id,
Type = TYPE_ROUTINE_MAINTENANCE, Date_Reported__c =
Date.today()));
        insert maintenanceRequests;
        return maintenanceRequests;
    }

    public static List<Equipment_Maintenance_Item__c>
createJoinRecords(List<Product2> equipment, List<Case>
maintenanceRequest){
        List<Equipment_Maintenance_Item__c> joinRecords = new
List<Equipment_Maintenance_Item__c>();
```

```
        joinRecords.add(new
Equipment_Maintenance_Item__c(Equipment__c = equipment.get(0).Id,
Maintenance_Request__c = maintenanceRequest.get(0).Id));
        joinRecords.add(new
Equipment_Maintenance_Item__c(Equipment__c = equipment.get(1).Id,
Maintenance_Request__c = maintenanceRequest.get(0).Id));
        joinRecords.add(new
Equipment_Maintenance_Item__c(Equipment__c = equipment.get(2).Id,
Maintenance_Request__c = maintenanceRequest.get(0).Id));
        joinRecords.add(new
Equipment_Maintenance_Item__c(Equipment__c = equipment.get(0).Id,
Maintenance_Request__c = maintenanceRequest.get(1).Id));
        joinRecords.add(new
Equipment_Maintenance_Item__c(Equipment__c = equipment.get(1).Id,
Maintenance_Request__c = maintenanceRequest.get(1).Id));
        joinRecords.add(new
Equipment_Maintenance_Item__c(Equipment__c = equipment.get(2).Id,
Maintenance_Request__c = maintenanceRequest.get(1).Id));
        insert joinRecords;
        return joinRecords;

    }
}
```

CreateDefaultDataTest.apxc:

```
@isTest
private class CreateDefaultDataTest {
    @isTest
    static void createData_test(){
        Test.startTest();
        CreateDefaultData.createDefaultData();
        List<Vehicle__c> vehicles = [SELECT Id FROM Vehicle__c];
        List<Product2> equipment = [SELECT Id FROM Product2];
        List<Case> maintenanceRequest = [SELECT Id FROM Case];
        List<Equipment_Maintenance_Item__c> joinRecords = [SELECT Id
FROM Equipment_Maintenance_Item__c];

        System.assertEquals(4, vehicles.size(), 'There should have been 4
vehicles created');
```

```
        System.assertEquals(4, equipment.size(), 'There should have been 4
equipment created');
        System.assertEquals(2, maintenanceRequest.size(), 'There should
have been 2 maintenance request created');
        System.assertEquals(6, joinRecords.size(), 'There should have been 6
equipment maintenance items created');

    }

    @isTest
    static void updateCustomSetting_test(){
        How_We_Roll_Settings__c          customSetting =
How_We_Roll_Settings__c.getOrgDefaults();
        customSetting.Is_Data_Created__c = false;
        upsert customSetting;

        System.assertEquals(false, CreateDefaultData.isDataCreated(), 'The
custom setting How_We_Roll_Settings__c.Is_Data_Created__c should be
false');

        customSetting.Is_Data_Created__c = true;
        upsert customSetting;

        System.assertEquals(true, CreateDefaultData.isDataCreated(), 'The
custom setting How_We_Roll_Settings__c.Is_Data_Created__c should be
true');

    }
}
```

MaintenanceRequestHelperTest.apxc:

```
@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 Maintenance';
    private static final string REQUEST_SUBJECT = 'Testing subject';

    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);
    }
}
```

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 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.getBody());
            System.debug(response.getBody());

            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 = (Decimal) mapJson.get('lifespan');
                myEq.Warehouse_SKU__c = (String) mapJson.get('sku');
                myEq.Current_Inventory__c = (Double) mapJson.get('quantity');
                warehouseEq.add(myEq);
            }
```

```
            if (warehouseEq.size() > 0){
                upsert warehouseEq;
                System.debug('Your equipment was synced with the warehouse
one');
                System.debug(warehouseEq);
            }

        }
    }
}
```

WarehouseCalloutServiceMock.apxc:

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

        System.assertEquals('https://th-superbadge-
apex.herokuapp.com/equipment', request.getEndpoint());
        System.assertEquals('GET', request.getMethod());

        // Create a fake response
        HttpResponse response = new HttpResponse();
        response.setHeader('Content-Type', 'application/json');

response.setBody('[{"_id":"55d66226726b611100aaf741","replacement":fal
se,"quantity":5,"name":"Generator 1000
kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku":"100003"}]');
        response.setStatusCode(200);
        return response;
    }
}
```

WarehouseCalloutServiceTest.apxc:

```
@isTest
private class WarehouseCalloutServiceTest {
    @isTest
```

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

WarehouseSyncSchedule.apxc:

```
global class WarehouseSyncSchedule implements Schedulable {
    global void execute(SchedulableContext ctx) {

        WarehouseCalloutService.runWarehouseEquipmentSync();
    }
}
```

WarehouseSyncScheduleTest.apxc:

```
@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 to Test',
scheduleTime, new WarehouseSyncSchedule());
        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 ');


    }
}
```

MaintenanceRequestHelper.apxc:

```
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 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));
            }

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

            }
        }
        insert ClonedWPs;
    }
  }
}
```

DailyLeadProcessor.apxc:

```
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;
   }
}
```

DailyLeadProcessorTest.apxc:

```
@isTest
private class DailyLeadProcessorTest {
          public static String CRON_EXP = '0 0 0 15 3 ? 2022';
   static testmethod void testSchedulableJob()
   {
      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');
   }
}
```

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> 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;
      }
   }
}
```

AddPrimaryContactTest.apxc:

```
@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')]);
    }
}
```

VerifyDate.apxc:

```
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;
        }

}
```

TestVerifyDate.apxc:

```
@isTest
private class TestVerifyDate {
        @isTest static void Test_CheckDates_casel()
   {
      Date D =
VerifyDate.CheckDates(date.parse('01/01/2022'),date.parse('01/01/2022')
);
      System.assertEquals(date.parse('01/05/2022'),D);
   }
   @isTest static void Test_CheckDates_case2()
   {
      Date D =
VerifyDate.CheckDates(date.parse('01/01/2022'),date.parse('05/05/2022')
);
      System.assertEquals(date.parse('01/31/2022'),D);
   }
   @isTest static void Test_DateWithin30Days_case1()
   {
      Boolean flag =
VerifyDate.DateWithin30Days(date.parse('01/01/2022'),date.parse('12/30
/2021'));
      System.assertEquals(false,flag);
   }
   @isTest static void Test_DateWithin30Days_case2()
   {
      Boolean flag =
VerifyDate.DateWithin30Days(date.parse('01/01/2022'),date.parse('02/02
```

```
/2021'));
        System.assertEquals(false,flag);
    }
    @isTest static void Test_DateWithin30Days_case3()
    {
        Boolean flag =
VerifyDate.DateWithin30Days(date.parse('01/01/2022'),date.parse('01/15
/2022'));
        System.assertEquals(true,flag);
    }
    @isTest static void Test_SetEndOfMonthDate()
    {
        Date returndate =
VerifyDate.setEndOfMonthDate(date.parse('01/01/2022'));
    }
}
```

LeadProcessor.apxc:

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

LeadProcessorTest.apxc:

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

ParkLocator.apxc:

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

ParkLocatorTest.apxc:

```
@isTest
private class ParkLocatorTest{
   @isTest
   static void testParkLocator() {
      Test.setMock(WebServiceMock.class, new ParkServiceMock());
```

```
        String[] arrayOfParks = ParkLocator.country('India');

        System.assertEquals('Park1', arrayOfParks[0]);
    }
}
```

AnimalLocator.apxc:

```
public class AnimalLocator {
            public static String getAnimalNameById(Integer animalId) {
        String animalName;
        Http http = new Http();
        HttpRequest request = new HttpRequest();
        request.setEndpoint('https://th-apex-http-
callout.herokuapp.com/animals/'+animalId);
        request.setMethod('GET');
        HttpResponse response = http.send(request);
        // If the request is successful, parse the JSON response.
        if(response.getStatusCode() == 200) {
            Map<String,Object> r = (Map<String ,Object>)
                JSON.deserializeUntyped(response.getBody());
            Map<String,Object> animal = (Map<String,Object>)r.get('animal');
            animalName = string.valueOf(animal.get('name'));

        }
        return animalName;
    }
}
```

AnimalLocatorMock.apxc:

```
@isTest
global class AnimalLocatorMock implements HttpCalloutMock {
    // Implement this interface method
    global HTTPResponse respond(HTTPRequest request) {
        // Create a fake response
        HttpResponse response = new HttpResponse();
        response.setHeader('Content-Type', 'application/json');
        response.setBody('{"animal":{"id":0,"name":"","eats":"","says":""}}');
        response.setStatusCode(200);
        return response;
    }
```

```
}
```

AccountProcessor.apxc:

```
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;
   }
}
```

AccountProcessorTest.apxc:

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

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;
      }
    }
}
```

AsyncParkService.apxc:

```
//Generated by wsdl2apex

public class AsyncParkService {
    public class byCountryResponseFuture extends
System.WebServiceCalloutFuture {
      public String[] getValue() {
        ParkService.byCountryResponse response =
(ParkService.byCountryResponse)System.WebServiceCallout.endInvoke(t
his);
        return response.return_x;
      }
    }
    public class AsyncParksImplPort {
      public String endpoint_x = 'https://th-apex-soap-
service.herokuapp.com/service/parks';
      public Map<String,String> inputHttpHeaders_x;
      public String clientCertName_x;
      public Integer timeout_x;
      private String[] ns_map_type_info = new
```

```
String[]{'http://parks.services/', 'ParkService'};
        public AsyncParkService.byCountryResponseFuture
beginByCountry(System.Continuation continuation,String arg0) {
            ParkService.byCountry request_x = new ParkService.byCountry();
            request_x.arg0 = arg0;
            return (AsyncParkService.byCountryResponseFuture)
System.WebServiceCallout.beginInvoke(
              this,
              request_x,
              AsyncParkService.byCountryResponseFuture.class,
              continuation,
              new String[]{endpoint_x,
              '',
              'http://parks.services/',
              'byCountry',
              'http://parks.services/',
              'byCountryResponse',
              'ParkService.byCountryResponse'}
            );
        }
    }
}
```