**Task - 9** : Getting started with Apex and Apex triggers using the Salesforce Trailhead Platform.

**DESCRIPTION**:

Apex is a strongly typed, object-oriented programming language that allows developers to execute flow and transaction control statements on the Salesforce platform. It is a development platform for building software as a service (SaaS) applications on top of Salesforce.com's customer relationship management (CRM) functionality. It allows developers to access Salesforce.com's back-end database and client-server interfaces to create third-party SaaS applications. Apex applications are usually hosted and run directly from Salesforce.com's servers. No installation on a user's local PC is necessary because of this hosting.

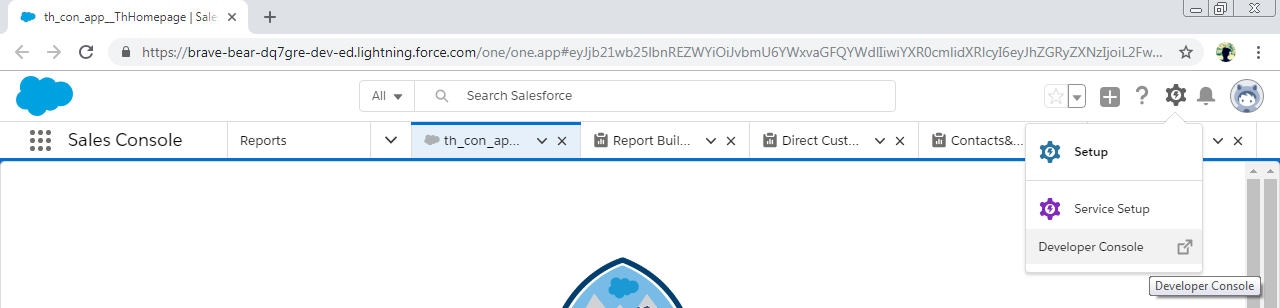
**a)AIM** : To create an Apex class that updates the Old Account Records.

**PROCEDURE** :

**i)First we need to Login to the trailhead Platform and follow the below steps**:

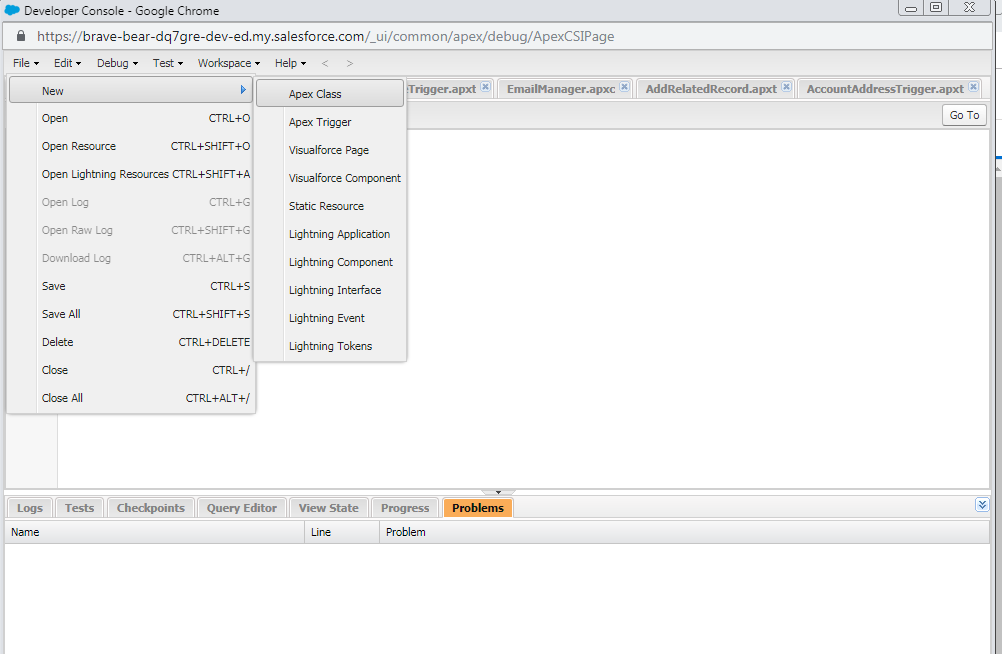
1. After logging into the trailhead, launch your Trailhead Playground by clicking **Launch** under the hands on challenge at the bottom.

2. Click setting.png at the top right and select Developer Console.



3. From the **File** menu, select **New -> Apex Class**.

The Developer console looks as follows :



4. For the class name, enter **OlderAccountsUtility** and then click **OK**

**Now we get the Code in the console as follows .**

*public class OlderAccountsUtility {*

*}*

After the above process click verify and move to further section.

# ii) Now we Add a Method to that Class

# 1.In the body of the OlderAccountsUtility class (the information between the curly brackets), copy and paste the following method.

*public static void updateOlderAccounts() {*

*// Get the 5 oldest accounts*

*Account[] oldAccounts = [SELECT Id, Description FROM Account ORDER BY CreatedDate ASC LIMIT 5];*

*// loop through them and update the Description field*

*for (Account acct : oldAccounts) {*

*acct.Description = 'Heritage Account';*

*}*

*// save the change you made*

*update oldAccounts;*

*}*

2.Now click File and save it .

Click verify to check the process and move to next section.

The above process first sorts Accounts by the date that they were created on. It then grabs the five oldest records. It uses the SOQL query language (line 3) to do the querying and sorting. It then iterates through each Account record to update the Description field. Finally, it updates the Account records by using the Apex Data Manipulation Language (DML).

**iii) After creating a method its time to test ,here we invoke and test the code.**

Apex code is tested as follows :

1 .In the Developer Console, select Debug -> Open Execute Anonymous Window.

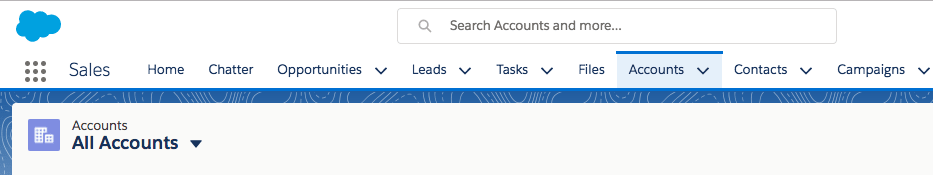
2. In the Enter Apex Code window, enter the following:

**OlderAccountsUtility.updateOlderAccounts();**

3. Now at the bottom right click Execute and verify the process.

**iv)Finally we verify the Updated Accounts as follows**

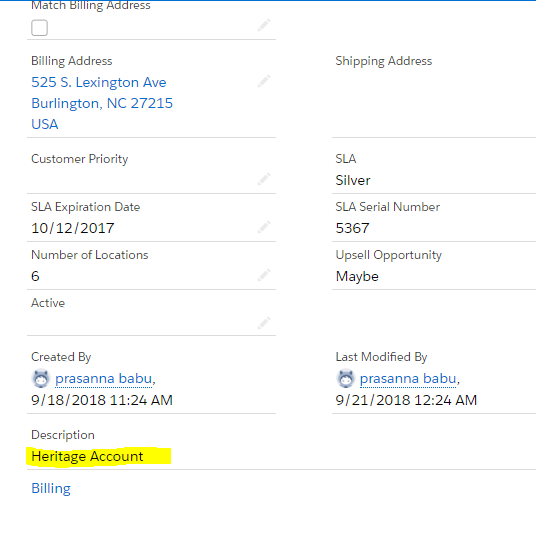
1. In your Trailhead Playground, click se.pngand select **Sales** under All Apps.
2. Click the **Accounts** tab.
3. Click **Recently Viewed** to show the list of views, and then select **All Accounts**.



1. Click see.pngand select **Select Fields to Display**. Move **Last Modified Date** to the Visible Fields column, and then click **Save**.
2. Click one of the five most recently modified account records (use the Last Modified Date to find these accounts).
3. Click **Details**

.

1. Look for the Description field. The value should be: **Heritage Account**

  
 **Result:** Thu**s we** created an Apex class that which updates the Old Account Records successfully.

**b) AIM :** To Create an Apex trigger that which is used in calling a Class method.

**DESCRIPTION :** Apex triggers enable you to perform custom actions before or after events to records in Salesforce, such as insertions, updates, or deletions. Just like database systems support triggers, Apex provides trigger support for managing records. Use triggers to perform tasks that can’t be done by using the point-and-click tools in the Salesforce user interface. For example, if validating a field value or updating a field on a record, use validation rules and workflow rules instead. Triggers can be defined for top-level standard objects, such as Account or Contact, custom objects, and some standard child objects.

**PROCEDURE :**

In this task we create a trigger that which helps in calling the class method .

**Step 1** : Launch Trailhead -> go to setup -> open Developer Console.

**Step 2**: Click on File -> New -> Apex Trigger.

**Step 3**: Enter the Trigger name as Example Trigger and select the Contact Object and submit.

**Step 4**: Now we get the skeletal structure of the trigger ,replace the default code with this code :

trigger ExampleTrigger on Contact (after insert, after delete) {

***if (Trigger.isInsert) {***

***Integer recordCount = Trigger.New.size();***

***// Call a utility method from another class***

***EmailManager.sendMail('prasannababu025@gmail.com', 'Trailhead Trigger Tutorial',***

***recordCount + ' contact(s) were inserted.');***

***}***

***else if (Trigger.isDelete) {***

***// Process after delete***

***}***

***}***

**Step 5**: Save and test the trigger .

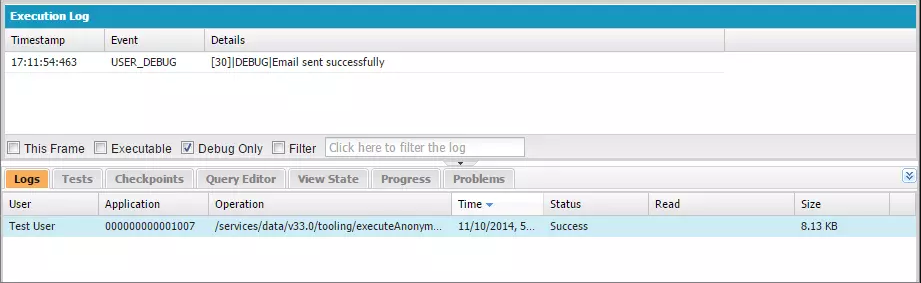
**Step 6:** To test the trigger

**Debug -> Open Execute Anonymous Window and write the following code and Click Execute.**

***Contact c = new Contact(LastName='Test Contact');***

***insert c;***

You can check whether the trigger was fired or not by checking the debug log



Now you will get a mail to your mail id if the trigger is fired.

But here it gives an error because we didn’t create any class called EmailManager

So first lets create EmailManager Class

Open Developer console -> File -> New -> Apex class.

Enter the name as EmailManager and click ok

Replace the Default code with the following code:

*public class EmailManager {*

*// Public method*

*public void sendMail(String address, String subject, String body) {*

*// Create an email message object*

*Messaging.SingleEmailMessage mail = new Messaging.SingleEmailMessage();*

*String[] toAddresses = new String[] {address};*

*mail.setToAddresses(toAddresses);*

*mail.setSubject(subject);*

*mail.setPlainTextBody(body);*

*// Pass this email message to the built-in sendEmail method*

*// of the Messaging class*

*Messaging.SendEmailResult[] results = Messaging.sendEmail(*

*new Messaging.SingleEmailMessage[] { mail });*

*// Call a helper method to inspect the returned results*

*inspectResults(results);*

*}*

*// Helper method*

*private static Boolean inspectResults(Messaging.SendEmailResult[] results) {*

*Boolean sendResult = true;*

*// sendEmail returns an array of result objects.*

*// Iterate through the list to inspect results.*

*// In this class, the methods send only one email,*

*// so we should have only one result.*

*for (Messaging.SendEmailResult res : results) {*

*if (res.isSuccess()) {*

*System.debug('Email sent successfully');*

*}*

*else {*

*sendResult = false;*

*System.debug('The following errors occurred: ' + res.getErrors());*

*}*

*}*

*return sendResult;*

*}*

*}*

Save and Debug the class

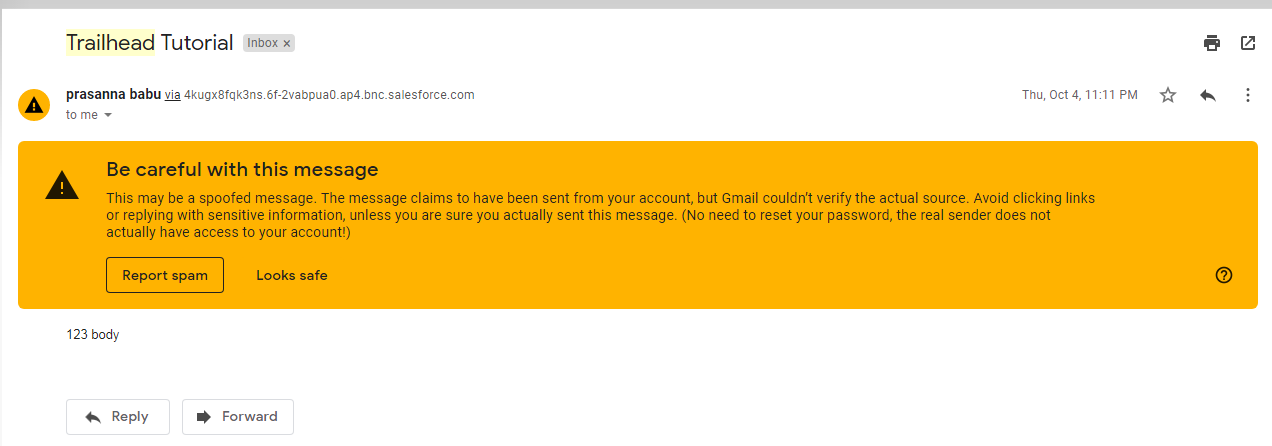
Click Debug -> Open Execute Anonymous Window

Enter the following

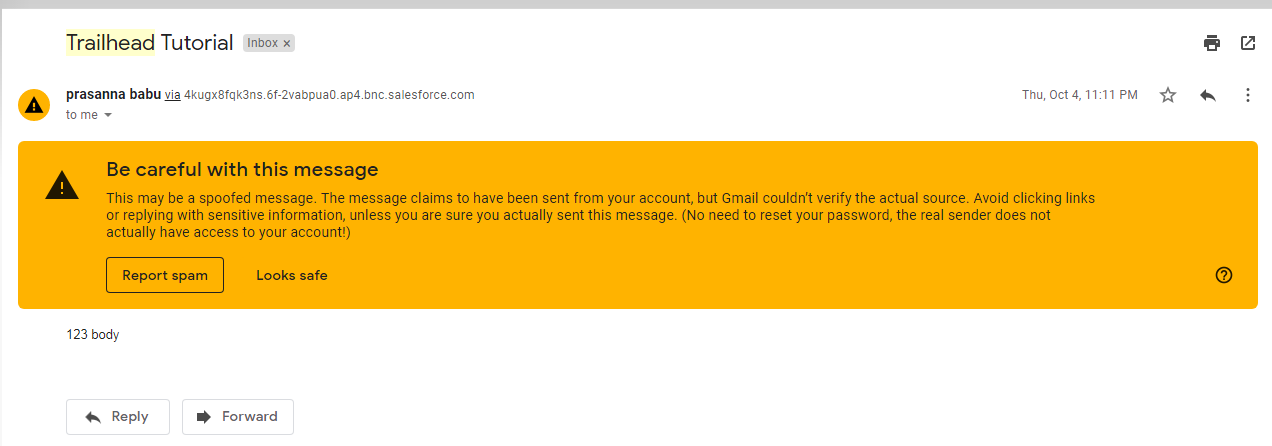
*EmailManager em = new EmailManager();*

*em.sendMail('prasannababu025@gmail.com', 'Trailhead Tutorial', '123 body');*

Click Execute .

Now you will get the email as follows*:*

If you re execute the first trigger you will get an email as follows :

**

**Result :**Thus we created a trigger successfully using Trailhead