Liberty Security Summarizer

Jacob Bengtson

Winter 2014

Executive Summary

Liberty Security is a security sales and monitoring company that is based out of Edmonton, Alberta Canada. Nathan Baldry, one of the two owners, married my wife's cousin 7 years ago. Before they were married he was my roommate for a year.

One of the secretaries at Liberty is in charge of creating a summary document of all the sales and cancellations that happened over the past month. She gets these numbers from their CRM system which outputs a csv file for sales and cancellations of the indicated period.

The purpose of this project is to automate the summarizing process so that what was once a 2 to 3 hour task can be accomplished in a matter of seconds. My project allows the user to select the appropriate csv files from their directory. It then proceeds to summarize the information and put into the desired types of tables that our find on the summary worksheet.

Implementation

I used a lot of global variables for this project because I wanted to create a bunch of different sub procedures to keep the main controller of the program clean and easy to track. Below are all the global variables I used with an explanation of what each variable is used for.

Global Variables

```
Dim filePath As String
    'the filepath of the crm output file
Dim cancelFilePath As String
    'the filepath of the cancel crm output file
Dim summaryWb As Workbook
    'the workbook where the summary will be created
Dim crmWb As Workbook
    'the workbook of the crm output
Dim cancWb As Workbook
    'the workbook of the cancel output
Dim cancWs As Worksheet
    'the workshet of the cancel worksheet
Dim dataWs As Worksheet
    'the worksheet where Lisa can add sales reps and so forth
Dim summaryWs As Worksheet
    'the worksheet where the data will actually be summarized
Dim crmWs As Worksheet
    'the worksheet that holds the crm output
Dim salesReps() As New clsSalesRep
    'array of sales reps names
Dim salesRepsRange As Range
    'range of salesReps
Dim cancenlRange As Range
    'range of cancel reasons
Dim cancel() As New clsCancel
    'array of cancel reason
Dim promRange As Range
    'range of promotions
Dim campaigns() As New clsCampaign
    'array of clsCampaign objects
Dim numHomeCancel As Integer
    'number of home accounts that have been canceled
Dim numShowCancel As Integer
    'number of show homes that have been canceled
Dim numCancelRange As Range
    'range of cancel cells on crmWs
Dim numCancel As Integer
    'total number of cancels, it is the sumb of show home and regular home cancelations
Dim periodEnd As Date
    'the last day of the period
Dim jjCancels As Integer
    'the total number of cancels for J&J Northstar accounts
Dim jjSales As Integer
    'the total number of sales for J&J Northstar accounts
Dim sumDate As String
    'this is the end date of the period that is being summarized
```

Below is the summarize sub procedure. It served as the main controller of my program. It calls several different sub procedures that will be described throughout this implementation section.

Main Controller

```
'This is the main controller for the program
Sub summarize(control As IRibbonControl)
    Dim continue As Boolean
    Dim x, y, z As Integer
        'counters to be used

    sumDate = InputBox("Please enter the end date for the period that is being summarized", "END DATE", "Format: Feb-14")

    'get the filepath of the crm sales output
    MsgBox ("Select the sales report")
    filePath = getFilePath()
    If filePath = "" Then Exit Sub

    'get the filepath of the crm cancel output
    MsgBox ("Select the cancel report")
    cancelFilePath = getFilePath()
    If filePath = "" Then Exit Sub

    'set file variables
    Set summaryWb = ActiveWorkbook
    Set dataWs = summaryWb.Sheets(1)
    Set summaryWs = summaryWb.Sheets(2)
    Set cancWb = Workbooks.Open(cancelFilePath)
    Set cancWs = cancWb.Sheets(1)
    Set crmWb = Workbooks.Open(filePath)
    Set crmWs = crmWb.Sheets(1)

    'select range of sales reps and set them to salesRepsRange
    dataWs.Parent.Activate
    dataWs.Select

    'Put sales reps in salesReps array
    Set salesRepsRange = Range(dataWs.Range("a3"), dataWs.Range("a3").End(xlDown))
    For Each cell In salesRepsRange
        ReDim Preserve salesReps(x)
        salesReps(x).name = cell.Value
        x = x + 1
    Next
    x = 0

    'select range of cancellation codes and put in array
    Set cancelRange = Range(dataWs.Range("B3"), dataWs.Range("B3").End(xlDown))
    For Each cell In cancelRange
        ReDim Preserve cancel(x)
        cancel(x).name = cell.Value
        x = x + 1
    Next
    x = 0



    'select range of promotions and put in campaigns array
    Set promoRange = Range(dataWs.Range("C3"), dataWs.Range("C3").End(xlDown))
    For Each cell In promoRange
        ReDim Preserve campaigns(x)
        campaigns(x).name = cell.Value
        x = x + 1
    Next
    x = 0

    'use functions to populate arrays of created classes with proper amounts
    continue = calcCancels
    If continue = False Then Exit Sub
    continue = calcCampaigns
    If continue = False Then Exit Sub
    continue = calcSalesReps
    If continue = False Then Exit Sub

    'activate the summary worksheet
    summaryWs.Activate

    'in this next section, i add a row to each table and then get and add the appropriate values for each row
    runningTotCAP summaryWs.Range("B47")
    fillCampaigns
    runningTotCAP summaryWs.Range("B80")
    fillSalesReps
    runningTotCAP summaryWs.Range("B98")
    fillCancellations
    singleCAP summaryWs.Range("B17")
    fillJJ
    singleCAP summaryWs.Range("B7")
    fillLibOverview
    singleCAP summaryWs.Range("B25")
    fillShowHomeAccounts
    runningTotCAP summaryWs.Range("B38")

    'close the workbooks that were being used
    crmWb.Close
    cancWb.Close
End Sub
```

One of the difficulties that I faced in creating this project was how to associate Cancellations, Sales Reps, and campaigns with the correlating amounts for each. I decided to create three different classes named clsCancel, clsSalesRep, and clsCampaign that hold a string name value as well as an integer. Each class also has a method on it that when called adds 1 to the integer value. Below is the Sales Rep class.

clsSalesRep

```
Option Explicit
Private repName As String
Private repSales As Integer

'Getter and setter for rep name
Public Property Get name() As String
    name = repName
End Property
Public Property Let name(n As String)
    repName = n
End Property

'Getter and setter for rep sales
Public Property Get sales() As Integer
    sales = repSales
End Property
Public Property Let sales(s As Integer)
    repSales = s
End Property

'Adds a sale to the total number of sales this rep has
Sub addSale()
    repSales = repSales + 1
End Sub
```

To start off I needed three things from the user: (1) the CRM output file that contains all the sales from the desired period, (2) the CRM output file that contains all the cancelations from the desired period, and (3) the end date of the sales/cancellation period. I used a simple input box to obtain the end date.

I created the following method to obtain the file path of both output csv files.

getFilePath

```
'This function gets the file path for which document you want to summarize
Function getFilePath() As String
    Dim fd As Office.FileDialog
    Set fd = Application.FileDialog(msoFileDialogFilePicker)
    If fd.Show = True Then
        ' the user chose a file
        getFilePath = fd.SelectedItems(1)
    Else
        ' the user pressed cancel
        getFilePath = ""
    End If
End Function
```

It returns the file path as a string, unless cancel is pressed then the summarize sub procedure is exited as shown in the following section of the summarize sub procedure.

```
'get the filepath of the crm sales output
MsgBox ("Select the sales report")
filePath = getFilePath()
If filePath = "" Then Exit Sub

'get the filepath of the crm cancel output
MsgBox ("Select the cancel report")
cancelFilePath = getFilePath()
If filePath = "" Then Exit Sub
```

The next step was to populate two different arrays with all the names of the three different variables we are dealing with: Sales Reps, Campaigns, and Cancellations. These can be found on the "Data Control" worksheet of the summary workbook. I made this dynamic so that the company could add or delete any one of the three variables types. Below are the lists of different types of instances for each class.

Sales Represntatives	Cancellation Reason Code	Campaigns
Al Michalchuk	Collections / Non Payment	Alarm.com Lead
Hailey Litchfield	Show Home	Brookfield Customer (Sold)
Jared Steed	Move Outside Canada	Brookfield Show Home
Justin De Rush	Move Within Canada	Customer Referral
Lisa Schwager	Deceased	Door to Door
Nathan Baldry	Competitor Switchover	DTD - Family & Friends
Pat Kickham	No Perceived Value	Employee Referral
Russell Keddie	Pull Equipment / Rip-Out	Family/Friends
Sarah Depew	Closed Business	Homeshow
Taylor Wolsey	Moved HOUSE to PTRON	Homexx Show Home
Troy Cole	Retirement Home	HOUSE - Move Outside (1) Year
D2D Accounts	J&J Northstar	HOUSE - Move Within (1) Year
J&J Northstar Accounts		HOUSE - Takeover Outside (1) Year
		HOUSE - Takeover Within (1) Year
		HouseMaster
		Insure Your Way Inc.
		InterNet Search
		J&J Northstar HOUSE Accounts
		Landmark Customers (Sold)
		Landmark Employee
		Landmark Show Home - Calgary
		Landmark Show Home - Edmonton
		Liberty Security Website
		Meadows Park Project
		PERS Customer Accounts
		Self Generated
		Twilite Music
		UrbanAge Show Home

Below is how a populated the variables. I selected the range using the xlDown method, and then using a for each loop I redimmed the array of each class and then added the name of that specific instance of the class.

```
'Put sales reps in salesReps array
Set salesRepsRange = Range(dataWs.Range("a3"), dataWs.Range("a3").End(xlDown))
For Each cell In salesRepsRange
    ReDim Preserve salesReps(x)
    salesReps(x).name = cell.Value
    x = x + 1
Next
x = 0

'select range of cancellation codes and put in array
Set cancelRange = Range(dataWs.Range("B3"), dataWs.Range("B3").End(xlDown))
For Each cell In cancelRange
    ReDim Preserve cancel(x)
    cancel(x).name = cell.Value
    x = x + 1
Next
x = 0

'select range of promotions and put in campaigns array
Set promoRange = Range(dataWs.Range("C3"), dataWs.Range("C3").End(xlDown))
For Each cell In promoRange
    ReDim Preserve campaigns(x)
    campaigns(x).name = cell.Value
    x = x + 1
Next
x = 0
```

The next task was to populate each class instance with the appropriate value from the crmWs worksheet. I created three procedures, one for each type of class. All three are virtually identical accept that the sub procedure for filling clsCancel classes with values also sums the total amount of cancels as well as show home cancels (statistic that was requested by the company). Below is this function.

calcCancels

```
'Figure out the amount of cancels as well as well as number of show homes cancelled and non show homes cancelled
Function calcCancels() As Boolean
    cancWs.Parent.Activate
    cancWs.Select
    'look up the column titled "Cancelled Reason"
    Rows("1:1").Select
    Set cell = Selection.Find(What:=UCase(Trim("CANCELLATION REASON")), After:=ActiveCell, LookIn:=xlFormulas, LookAt:=xlWhole, Sear
    Set numCancelRange = Range(cancWs.Range(cell.Address).Offset(1, 0), cancWs.Range(cell.Address).Offset(1, 0).End(xlDown))
    For Each cell In numCancelRange
        If Not Len(Trim(cell)) = 0 Then
            For Each clsCancel In cancel
                'Add one cancel to the clsCancel that matches the cell value
                If UCase(Trim(clsCancel.name)) = UCase(Trim(cell.Value)) Then clsCancel.addCancel
                'keep track of number of show homes canceled
                If UCase(Trim(clsCancel.name)) = "SHOW HOME" And UCase(Trim(cell.Value)) = UCase(Trim(clsCancel.name)) Then
                    numShowCancel = numShowCancel + 1
                End If
            Next
            'keep track of total number of cancels
            numCancel = numCancel + 1
        End If
    Next
    'figure out number of non show home cancels
    numHomeCancel = numCancel - numShowCancel
    calcCancels = True
End Function
```

Some unique aspects to this sub procedure are:

1. I used the Selection.Find method to look up the appropriate column of cancellations in the crmWs worksheet. I did this just in case the CRM created a csv file with the attributes for each sale/cancellation in a different order.
2. Throughout my code you will probably notice the use of trimming and forced capitalization to ensure that hanging spaces and random capital letters aren't throwing off my comparison logic.
3. The first if statement checks to see if the value in the current cell matches the name of the current cancellation, the second checks to see if the value in the cell is "SHOW HOME" and if the cell value and the class name match. If the second if statement is met numShowCancel has 1 added to it.
4. The calcCancels sub procedure is pulling values from the cancWs instead of the crmWs that the other two sub procedures pull values from.
5. If the row had a value in the cancellation reason column then that means there was a cancellation, so numCancel has 1 added to it.

Below are the other two functions which are very similar, accept they are doing the same process for clsSalesRep and clsCampaign.

```
'this functions fills the array campaigns() with how many times they occur
Function calcCampaigns() As Boolean
    Dim campRange As Range
    crmWs.Parent.Activate
    crmWs.Select
    'look up the column titled "campaignname"
    Rows("1:1").Select
    Set cell = Selection.Find(What:=UCase(Trim("CAMPAIGNNAME")), After:=ActiveCell, LookIn:=xlFormulas, LookAt:=xlWhole, S
    Set campRange = Range(crmWs.Range(cell.Address).Offset(1, 0), crmWs.Range(cell.Address).Offset(1, 0).End(xlDown))

    For Each cell In campRange
        If Not Len(Trim(cell)) = 0 Then
            For Each clsCampaign In campaigns
                If UCase(Trim(clsCampaign.name)) = UCase(Trim(cell.Value)) Then clsCampaign.addCampaign
            Next
        End If
    Next
    calcCampaigns = True
End Function
```

```
'this function populates the array of sales reps with how many sales they've had
Function calcSalesReps() As Boolean
    Dim repRange As Range
    crmWs.Parent.Activate
    crmWs.Select
    'look up the column titled "campaignname"
    Rows("1:1").Select
    Set cell = Selection.Find(What:=UCase(Trim("REP NAME")), After:=ActiveCell, LookIn:=xlFormulas, LookAt:=xlWhole, Searc
    Set repRange = Range(crmWs.Range(cell.Address).Offset(1, 0), crmWs.Range(cell.Address).Offset(1, 0).End(xlDown))

    For Each cell In repRange
        If Not Len(Trim(cell)) = 0 Then
            For Each clsSalesRep In salesReps
                If UCase(Trim(clsSalesRep.name)) = UCase(Trim(cell.Value)) Then clsSalesRep.addSale
            Next
        End If
    Next
    calcSalesReps = True
End Function
```

Now came the final step. For each different table on the summarization work sheet I needed to add a row either on the far right or between the running total column and the second to last column. I hard coded this logic and made two different sub procedures to copy and paste columns in these two different ways. Below are the two sub procedures.

```
'This sub procedure copies and pastes a single continuous column
Sub singleCAP(rng As Range)
    rng.End(xlToRight).Select
    Range(ActiveCell, ActiveCell.End(xlDown)).Select
    Selection.Copy
    ActiveCell.Offset(0, 1).Range("A1").Select
    ActiveSheet.Paste
    ActiveCell.Offset(0, -2).Range("A1").Select
    Application.CutCopyMode = False
    Selection.Copy
    Selection.Offset(0, 1).PasteSpecial Paste:=xlPasteFormats, Operation:=xlNone, _
        SkipBlanks:=False, Transpose:=False
    Application.CutCopyMode = False
    ActiveCell.Offset(0, 1).Value = sumDate
End Sub
```

```
'This sub procedure copies and pastes a running totals column
Sub runningTotCAP(rng As Range)
    summaryWs.Activate
    rng.End(xlToRight).Select
    ActiveCell.Offset(0, -1).Range("A1").Select
    Range(Selection, Selection.End(xlDown)).Select
    Range(Selection, Selection.Offset(0, 1)).Select
    Selection.Copy
    ActiveCell.Offset(0, 1).Range("A1").Select
    ActiveSheet.Paste
    ActiveCell.Offset(0, -2).Range("A1").Select
    Selection.Copy
    Selection.Offset(0, 1).PasteSpecial Paste:=xlPasteFormats, Operation:=xlNone, _
        SkipBlanks:=False, Transpose:=False
    Application.CutCopyMode = False
    ActiveCell.Offset(0, 1).Value = sumDate
End Sub
```

Each accept the far left top cell as a range, they then get to either the last column or the second to last column using xlToRight and offset. They copy and past the needed section and then reformat the old heading row using PasteSpecial.

I had to create these new columns and populate them with values in a certain order because of how certain tables referenced values in others. Below is the portion of the main controller that all the sub procedures were called in.

```
'in this next section, i add a row to each table and then get and add the appropriate values for each row
runningTotCAP summaryWs.Range("B47")
fillCampaigns
runningTotCAP summaryWs.Range("B80")
fillSalesReps
runningTotCAP summaryWs.Range("B98")
fillCancellations
singleCAP summaryWs.Range("B17")
fillJJ
singleCAP summaryWs.Range("B7")
fillLibOverview
singleCAP summaryWs.Range("B25")
fillShowHomeAccounts
runningTotCAP summaryWs.Range("B38")
```

Each of the sub procedure that start with “fill” are calculating the total amounts for different requirements as requested by Liberty Security. Below is an image of some of the tables that they were populating.

LIBERTY SECURITY ACCOUNTS OVERVIEW:

	Dec-14	Jan-14	Feb-14
Opening Balance	1253	1279	1319
Customer Account Cancellations	6	5	0
Show Home Cancellations	2	1	0
Sub Total of Active HOUSE Accounts	1245	1273	1319
New Active HOUSE Accounts	34	46	13
Total Active HOUSE Accuonts	1279	1319	1332

J&J NORTHSTAR ACCOUNTS OVERVIEW:

	Dec-13	Jan-14	Feb-14
Opening Balance	146	146	145
J&J Customer Account Cancellations	0	1	1
J&J New Active HOUSE Accounts	0	0	0
Total Active J&J HOUSE Accuonts	146	145	144

SHOW HOME ACCOUNTS OVERVIEW:

	Dec-14	Jan-14	Feb-14
*Artisan Homes	1	1	1
**Brookfield Homes	22	22	22
*Homexx Homes	2	3	3
**Jaya Homes	1	1	1
*Landmark Homes	42	41	41
*UrbanAge Homes	1	1	0
Total Active Show Home Accounts	69	69	68

*Free Monitoring Show Homes
**Paying Monitoring Show Homes

OVERALL HOUSE ACCOUNT OVERVIEW:

	Dec-13	Jan-14	Feb-14	Running Totals
Opening Balance	1399	1425	1464	
Total Customer Account Cancellations	8	7	1	16
Total NEW Active HOUSE Accounts	34	46	13	93
Total Active HOUSE Accounts	1425	1464	1476	

The table named “OVERALL HOUSE ACCOUNT OVERVIEW” if filled with references that calculate based off of the first two tables.

fillCampaigns, fillSalesReps, and fillCancellations all fill tables that look like the following table:

43R x 1C

Campaign	Dec-13	Jan-14	Feb-14	Running Totals
Alarm.com Lead	0	0	1	1
Brookfield Customer (Sold)	0	0	0	0
Brookfield Show Home	0	1	3	4
Customer Referral	3	4	0	7
Door to Door	5	7	5	17
DTD - Family & Friends	0	0	4	4
Employee Referral	0	1	0	1
Family/Friends	3	3	3	9
Homeshow	0	2	0	2
Homexx Show Home	0	0	1	1
HOUSE - Move Outside (1) Year	0	0	0	0
HOUSE - Move Within (1) Year	0	0	1	1
HOUSE - Takeover Outside (1) Year	0	0	2	2
HOUSE - Takeover Within (1) Year	0	0	0	0
HouseMaster	0	0	1	1
Insure Your Way Inc.	0	0	0	0
InterNet Search	4	3	4	11
J&J Northstar HOUSE Accounts	0	0	0	0
Landmark Customers (Sold)	0	5	1	6
Landmark Employee	2	1	0	3
Landmark Show Home - Calgary	0	0	0	0
Landmark Show Home - Edmonton	3	0	2	5
Liberty Security Website	0	0	0	0
Meadows Park Project	4	2	2	8
PERS Customer Accounts	1	1	0	2
Self Generated	9	16	11	36
Twilite Music	0	0	0	0
UrbanAge Show Home	0	0	0	0
Total Add HOUSE Accounts	34	46	41	121

Below is the sub procedure for filling the above table with values.

```
'This sub procedure fills the summary worksheet with all the amounts of campaigns
Sub fillCampaigns()
    Dim cancRng As Range
    summaryWb.Activate
    summaryWs.Select
    Columns(1).Select
    Set cell = Selection.Find(What:=UCase(Trim("CAMPAIGN")), After:=ActiveCell, LookIn:=xlFormulas, LookAt:=xlWhol
    Set cancRng = Range(Range(cell.Address).Offset(1, 0), Range(cell.Address).Offset(1, 0).End(xlDown).Offset(-1,
    For Each cell In cancRng
        For Each clsCampaign In campaigns
            If UCase(Trim(cell.Value)) = UCase(Trim(clsCampaign.name)) Then
                cell.End(xlToRight).Select
                ActiveCell.Offset(0, -1).Select
                ActiveCell.Value = clsCampaign.number
            End If
        Next
    Next
End Sub
```

The procedure searches in the first column of the summary worksheet for a row with the value of "CAMPAIGN." It then creates a range out of the subsequent list of campaign names. Using a for each loop, each cell checks its value against the name of one of the clsCampaigns in the campaign array. If they match, the value held in that object is added to the second to last row using a combination of xlToRight and offset. Below are the sub procedures to fill the sales rep table and cancellation reason table. They follow the same logic.

```
'sub procedure that fills all the cancellations on the summary sheet
Sub fillCancellations()
    Dim cRng As Range
    summaryWb.Activate
    summaryWs.Select
    Columns(1).Select
    Set cell = Selection.Find(What:=UCase(Trim("CANCELLATION REASON CODE")), After:=ActiveCell, LookIn:=xlForm
    Set cRng = Range(Range(cell.Address).Offset(1, 0), Range(cell.Address).Offset(1, 0).End(xlDown).Offset(-1,
    For Each cell In cRng
        For Each clsCancel In cancel
            If UCase(Trim(cell.Value)) = UCase(Trim(clsCancel.name)) Then
                cell.End(xlToRight).Select
                ActiveCell.Offset(0, -1).Select
                ActiveCell.Value = clsCancel.number
            End If
        Next
    Next
End Sub
```

```
'sub procedure that fills the sales of all the sales reps on the summary sheet
Sub fillSalesReps()
    Dim srRng As Range
    summaryWb.Activate
    summaryWs.Select
    Columns(1).Select
    Set cell = Selection.Find(What:=UCase(Trim("SALES REPRESENTATIVE NAME")), After:=ActiveCell, LookIn:=xlFor
    Set srRng = Range(Range(cell.Address).Offset(1, 0), Range(cell.Address).Offset(1, 0).End(xlDown).Offset(-1
    For Each cell In srRng
        For Each clsSalesRep In salesReps
            If UCase(Trim(cell.Value)) = UCase(Trim(clsSalesRep.name)) Then
                cell.End(xlToRight).Select
                ActiveCell.Offset(0, -1).Select
                ActiveCell.Value = clsSalesRep.sales
            End If
        Next
    Next
End Sub
```

I then had to add values to the other, more finicky tables. Filling the JJ table consisted of adding up the cancellation and sales rep classes that had “J&J” in the left three characters of their names, and then putting these totals in the table. Below is the sub procedure that does this.

```
'sub procedure that fills the summary table for J&J Northstar Accounts, also stores the val
Sub fillJJ()
    Dim jjRng As Range
    summaryWb.Activate
    summaryWs.Select
    Columns(1).Select
    Set cell = Selection.Find(What:=UCase(Trim("J&J CUSTOMER ACCOUNT CANCELLATIONS")), Afte
    cell.End(xlToRight).Select
    For Each clsCancel In cancel
        If Left(clsCancel.name, 3) = "J&J" Then
            ActiveCell.Value = clsCancel.number
            jjCancels = clsCancel.number
        End If
    Next
    For Each clsSalesRep In salesReps
        If Left(clsSalesRep.name, 3) = "J&J" Then
            ActiveCell.Offset(1, 0).Value = clsSalesRep.sales
            jjSales = clsSalesRep.sales
        End If
    Next
End Sub
```

Filling the table named “LIBERTY SECURITY ACCOUNTS OVERVIEW” was a process of looping through all the cancellations that were not J&J homes and keeping track of which ones were show homes and not show homes. I did this using if statements within a for each loop. I also had to count up all the sales that were not of the J&J variety. I accomplished this using an if statement in a for each loop as well. Below is the sub procedure that accomplished this.

```
'this sub procedure fills the liberty security accounts overview table
Sub fillLibOverview()
    Dim x As Integer
    Dim r As Range
    summaryWb.Activate
    summaryWs.Select
    Columns(1).Select
    Set cell = Selection.Find(What:=UCase(Trim("LIBERTY SECURITY ACCOUNTS OVERVIEW:")), After:=ActiveCell, LookIn:=
    cell.Offset(4, 0).Select
    ActiveCell.End(xlToRight).Select
    For Each clsCancel In cancel
        If Not Left(clsCancel.name, 3) = "J&J" And Not UCase(clsCancel.name) = "SHOW HOME" Then
            x = x + clsCancel.number
        End If
        If UCase(clsCancel.name) = "SHOW HOME" Then
            ActiveCell.Offset(1, 0).Value = clsCancel.number
        End If
    Next
    ActiveCell.Value = x
    ActiveCell.Offset(3, 0).Select
    x = 0
    For Each clsSalesRep In salesReps
        If Not Left(clsSalesRep.name, 3) = "J&J" Then
            Debug.Print clsSalesRep.name
            x = x + clsSalesRep.sales
        Else
            Debug.Print clsSalesRep.name
            x = x - clsSalesRep.sales
        End If
    Next
    ActiveCell.Value = x
End Sub
```

The last step was filling the table named “SHOW HOMES ACCOUNT OVERVIEW.” This required looping through the range of listed show home names in the left column, trimming of either one or two asterisks, taking the name before the first space and adding “SHOW HOME” to it, and then for each cell I had to loop through the array of campaigns and check to see if the cell value and the class name equaled each other. If they did I added the value to the second to last column and put it in the last column (last column was a running total). There was one trick part to this though; Landmark campaigns had two different names in the data base, Edmonton north and south. Therefore, in the row that was Landmark, if the clsCampaign left 18 characters were “LANDMARK SHOW HOME” then I added the value to an intermediate variable and at the end of the loop at added that value to the second to last column and placed the resulting value in the last column of the Landmark row. Below is the sub procedure that accomplishes this task.

```
'sub procedure that fills the table named show home accounts overview
Sub fillShowHomeAccounts()
    Dim space As Integer
    Dim x As Integer
    Dim first As String
    Dim showRng As Range
    summaryWb.Activate
    summaryWs.Select
    Columns(1).Select
    Set cell = Selection.Find(What:=UCase(Trim("SHOW HOME ACCOUNTS OVERVIEW:")), After:=ActiveCell, LookIn:=xlFormulas, Lo
    Set showRng = Range(Range(cell.Address).Offset(3, 0), Range(cell.Address).Offset(3, 0).End(xlDown).Offset(-3, 0))
    For Each cell In showRng
        space = InStr(1, cell.Value, " ")
        first = Left(cell.Value, space)
        If Left(first, 2) = "**" Then
            first = Replace(first, "**", "")
        Else
            first = Replace(first, "*", "")
        End If
        For Each clsCampaign In campaigns
            If UCase(Trim(first)) & " SHOW HOME" = UCase(clsCampaign.name) And Not "LANDMARK" = UCase(Trim(first)) Then
                cell.Select
                ActiveCell.End(xlToRight).Value = ActiveCell.End(xlToRight).Offset(0, -1).Value + clsCampaign.number
            ElseIf UCase(Trim(first)) = "LANDMARK" Then
                If Left(UCase(clsCampaign.name), 18) = "LANDMARK SHOW HOME" Then
                    x = x + clsCampaign.number
                End If
            End If
        Next
        cell.Select
        ActiveCell.End(xlToRight).Value = ActiveCell.End(xlToRight).Offset(0, -1).Value + x
        x = 0
    Next
End Sub
```

After that all that was left to do was close the workbooks that I opened and end the summarize sub procedure.

Learning Outcomes and Conceptual Difficulties

When I set off to accomplish this task, one of the first problems that I ran into was getting proper clarification from the client. I found that email was a very inefficient ways to get answers; therefore, I quickly opted for phone calls to get the answers that I needed.

One of the major conceptual difficulties that I ran into was how to associate sales, cancellation, and campaign values with the names of each of these categories. After thinking about it I figured that there must be some way to create classes like you can in Java. I did some research and found out this was possible. Creating classes to hold multiple values was one of the most useful things that I learned during the project.

A long the way of completing this project I got very comfortable working with for each loops which I had previously shied away from. I originally learned in java to do loops using counters so I used to always default to this, but for each loops are much quicker to create and easier to interact with each variable inside the loop. This was especially valuable because I was working with arrays.

One of the downsides to working with Arrays was that I had to loop through the array every time that I wanted to find a specific type of class. I thought that there must be a better way to do this, like using a map in java. It turns out there is. Looking back I should have used a collection. However, because I was already ¾ the way done by the time that I discovered collections even existed, I opted to stick with arrays and keep collections in mind for the future.

The main other problem that I faced was the scope of the project. There were many different custom types of summaries that Liberty wanted and I had to create custom sub procedures for each one to pull that exact information that they wanted. I got very good at using if statements in VBA and was as the End and offset methods.

Assistance

I was able to complete the project without substantial help from anyone.