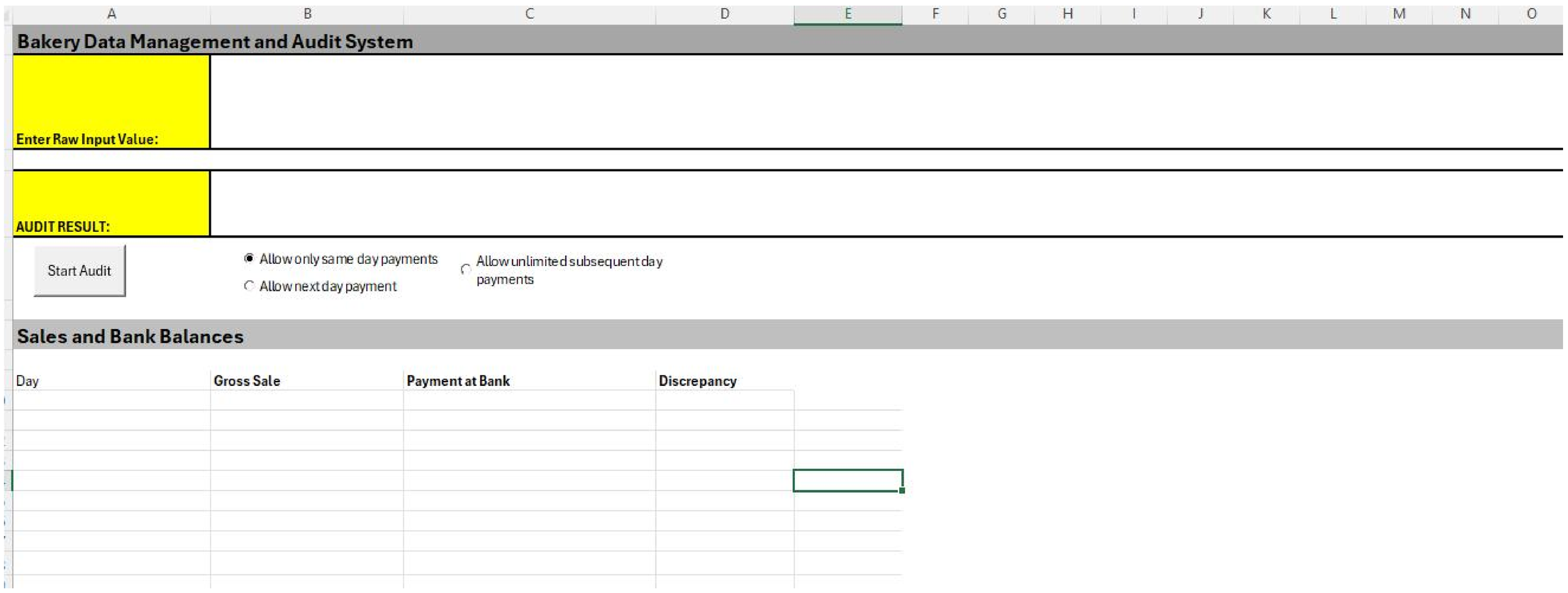
Portfolio:
VBA Learning Project

Kaan Bora Karapınar

A look at the project



The problem set that I solved with VBA :

Level 3 — Catalysts

- Da John seine Mittagessen aus seinen täglichen Einnahmen bezahlt kann, muss er das gesamte Geld nicht sofort einzahlen. Er darf den Betrag in bis zu 4 Teilen bei der Bank einzahlen. Die Einzahlungen können auch an Folgetagen erfolgen.
- Eingabe:
  - {F <Tag> <Tagesumsatz> } {B <Tag> <Bankeinzahlung> }
  - Tag ... Tag im Jahr (1...365)
  - Tagesumsatz ... positive ganze Zahl
  - Bankeinzahlung ... positive ganze Zahl (eine Bankeinzahlung kann nur für max. einen Tag verwendet werden!)
- Ausgabe:
  - {Tag }
  - Tag ... Tag an dem der Tagesumsatz nicht komplett eingezahlt wurde.
  - Die Tage müssen aufsteigend sortiert sein.
- Beispiel:
  - Eingabe:F 1 200 F 2 170 B 1 100 B 2 80 B 2 15 B 2 100 B 3 70
  - Ausgabe: 2

- Since John is allowed to use parts of his daily revenue to buy lunch, he can pay the earnings divided in up to 4 parts into the bank. John is allowed to pay the parts also on the days after he got it from his customers.
- Input:
  - {F <day> <daily sales> } {B <day> <payment at the bank> }
  - Day... Day in the year (1...365)
  - Daily sales ... Positive integer
  - Payment at the bank ... Positive integer (one payment can NOT be used for two different daily sales!)
- Output:
  - {day }
  - day ... Day where the daily sales were not completely paid into the bank account.
  - The days must be in ascending order.
- Example:
  - Input: F 1 200 F 2 170 B 1 100 B 2 80 B 2 15 B 2 100 B 3 70
  - Output: 2

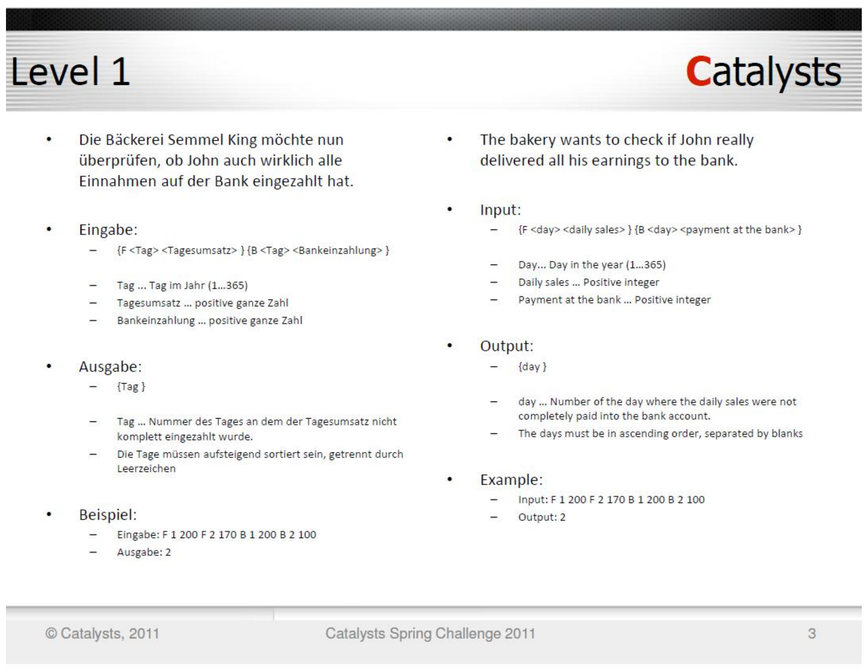
Day	Salary
1	200
2	170

Day	Payment
1	100
2	80
2	15
2	100
3	70

© Catalysts, 2011

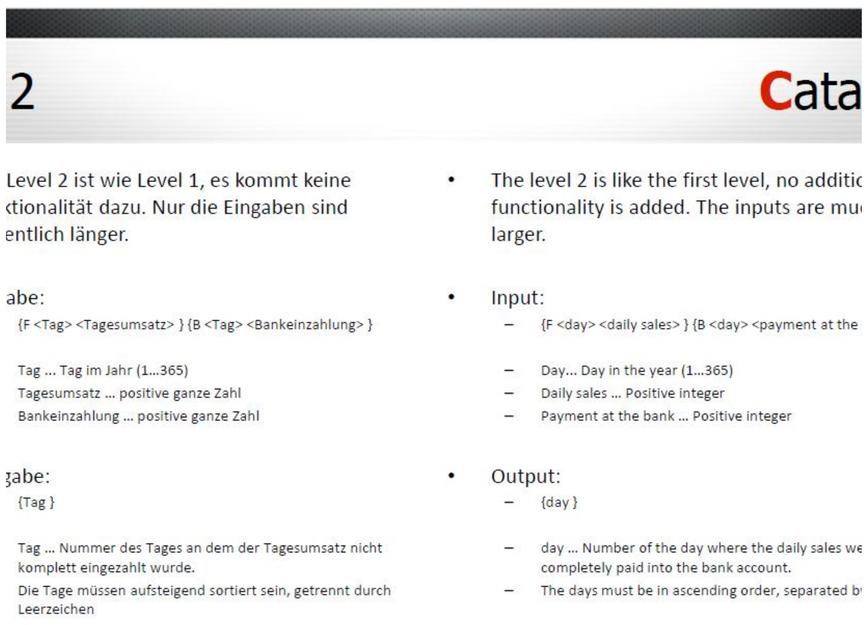
Github Repo for better code review: KaanBoraKarapinar/VBABakeryAuditScripts

Project Details

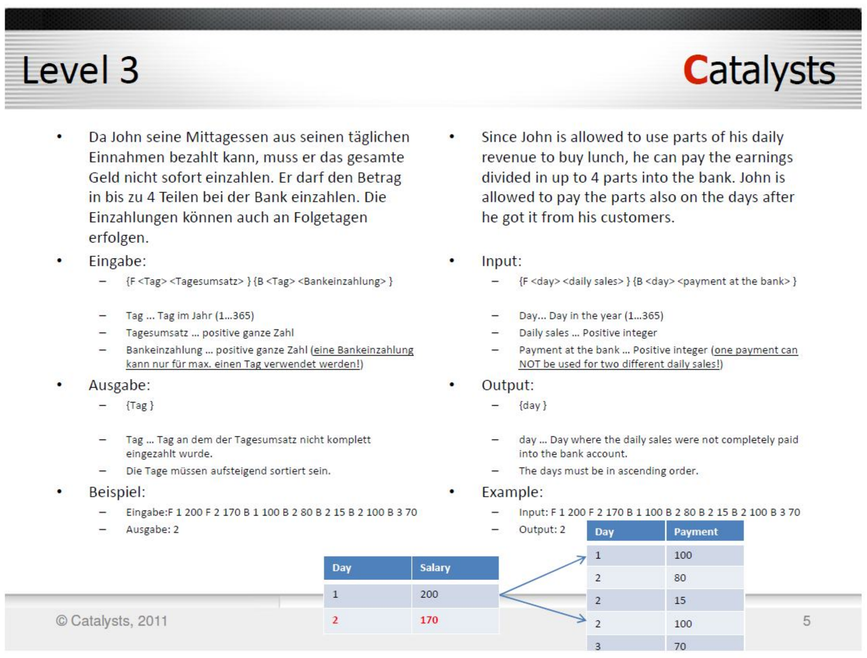


Catalyst / Cloudflight Catcoder has released several algorithmic and data management quizzes. To build my VBA skills, I selected a finance-focused quiz and implemented a solution.

My task was to identify discrepancies in gross amounts and bank payments by comparing them using manual records.



In level two, while the core logic remained the same, the dataset became significantly larger, increasing the complexity of the task.



In level three, the challenge was to identify matching dates for untagged late-day payments, ensuring accuracy in tracking delays across each day.

Source Code

Initially, each daily record contained a single clsSalesRecord and a single clsPaymentRecord. In level three, however, up to four clsPaymentRecord entries can now be assigned per day. Additionally, it's no longer assumed that all payments correspond directly to the gross sale of the payment day, adding another layer of complexity to tracking and validation.

```
Option Explicit

' Class: clsSalesRecord
' Represents the sales amount on a specific day.

Public day As Long
' The day of the sales.

Public SalesAmount As Double
' The amount of sales for the day.
```

```
Option Explicit

' Class: clsPaymentRecord
' Represents a payment made on a specific day.

Public day As Long
' The day the payment was made.

Public Payment As Double
' The amount of the payment.

Public IsUsed As Boolean
' Indicates if the payment has been used.

Public PaymentID As String
' A unique identifier for the payment.

Private Sub Class_Initialize()
    ' Initializes IsUsed to False when the class is created.
    IsUsed = False
End Sub
```

Source Code: clsDailyRecord

The clsDailyRecord class contains the primary strategy for identifying basic discrepancies, as discrepancies are initially considered on a daily basis. However, for more complex calculations, such as those in level three, the operations are handled in a separate class, clsDays, which represents and manages all daily records and their corresponding days. This structure allows for more advanced, cross-day analysis of discrepancies.

```
Option Explicit

' Class: clsDailyRecord
' Represents a daily record containing payments and sales.
' Calculates discrepancies between sales and payments.

Public day As Long
' The day number (e.g., 1, 2, 3, etc.)

Private PaymentRecord() As clsPaymentRecord
' Array of PaymentRecords for the day.

Private EffectivePaymentRecord() As clsPaymentRecord
' Array of Effective PaymentRecords used for discrepancy calculation.

Public TotalNumberOfEffectivePayments As Long
' Total number of effective payments for the day.

Public TotalEffectivePaymentAmount As Long
' Total amount of effective payments for the day.

Public SalesRecord As clsSalesRecord
' SalesRecord for the day.

Public Discrepancy As Double
' The discrepancy amount for the day.

Public IsDiscrepant As Boolean
' Indicates if there is a discrepancy on this day.

Public ThereIsSavedPayment As Boolean
' Indicates if there are any saved payments for the day.

Public TotalPaymentAmount As Double
' Total amount of payments for the day.

Private TotalNumberOfPayment As Long
' Total number of payments for the day.

Sub ToString()
    ' Displays information about the day's record.
    On Error Resume Next
    MsgBox "In day: " & day & " payment: " & PaymentRecord.Payment & " sales: " &
    SalesRecord.SalesAmount & _
           " isDiscrepant and Amount: " & IsDiscrepant & " " & Discrepancy
End Sub

Public Property Get getReadOnlyPaymentRecords() As Long
    ' Getter for PaymentRecord array (read-only).
    paymentrecords = PaymentRecord
End Property

Public Sub addToPaymentRecordsArray(value As clsPaymentRecord)
    ' Adds a PaymentRecord to the PaymentRecord array.
    ThereIsSavedPayment = True
    TotalNumberOfPayment = TotalNumberOfPayment + 1
    ReDim Preserve PaymentRecord(0 To TotalNumberOfPayment - 1)
    Set PaymentRecord(TotalNumberOfPayment - 1) = value
    TotalPaymentAmount = TotalPaymentAmount + value.Payment
End Sub

Public Sub AddEffectivePayment(effectivePayment As clsPaymentRecord)
    ' Adds a PaymentRecord to the EffectivePaymentRecord array if it hasn't been used
    yet.
    If effectivePayment.IsUsed = False Then
        TotalNumberOfEffectivePayments = TotalNumberOfEffectivePayments + 1
        Dim lastLegalIndex As Long
        lastLegalIndex = TotalNumberOfEffectivePayments - 1
        ReDim Preserve EffectivePaymentRecord(0 To lastLegalIndex)
        Set EffectivePaymentRecord(lastLegalIndex) = effectivePayment
        TotalEffectivePaymentAmount = TotalEffectivePaymentAmount + effectivePayment.
        Payment
        effectivePayment.IsUsed = True
    Else
        ' MsgBox "Used payments cannot be effective"
    End If
End Sub

Public Sub ApplyAllPaymentsOfDayAsEffective()
    ' Makes all payments of the day effective.
    Dim current As Variant
    Dim tempRecord As clsPaymentRecord
    For Each current In PaymentRecord
        Set tempRecord = current
        AddEffectivePayment tempRecord
    Next
End Sub

Public Sub ApplyAllRemainingPaymentsOfDayAsEffective()
    ' Makes all unused payments of the day effective.
    Dim current As Variant
    Dim tempRecord As clsPaymentRecord
    For Each current In PaymentRecord
        If current.IsUsed = False Then
            Set tempRecord = current
            AddEffectivePayment tempRecord
        End If
    Next
End Sub

Public Sub makeBiggestPaymentEffective()
    ' Makes the biggest unused payment effective.
    Dim tempIndex As Long
    tempIndex = IndexOfBiggestNonUsedPayment
    If tempIndex <> -1 Then
        AddEffectivePayment PaymentRecord(tempIndex)
    End If
End Sub

Public Sub CalculateDiscrepancyWithCurrentEffectivePayments()
    ' Calculates discrepancy based on current effective payments and sales record.
    If SalesRecord Is Nothing Then
        IsDiscrepant = True
        Discrepancy = -TotalEffectivePaymentAmount
        If TotalEffectivePaymentAmount = 0 Then
            IsDiscrepant = False
            Discrepancy = -TotalEffectivePaymentAmount
        End If
        Exit Sub
    ElseIf TotalEffectivePaymentAmount = 0 Then
        IsDiscrepant = True
        Discrepancy = SalesRecord.SalesAmount
        Exit Sub
    End If

    If SalesRecord.SalesAmount = TotalEffectivePaymentAmount Then
        Discrepancy = 0
        IsDiscrepant = False
    Else
        IsDiscrepant = True
        Discrepancy = SalesRecord.SalesAmount - TotalEffectivePaymentAmount
    End If
End Sub

Public Function IndexOfBiggestNonUsedPayment() As Long
    ' Returns the index of the biggest non-used payment.
    Dim i As Long
    Dim maxIndex As Long
    Dim maxValue As Double
    maxIndex = -1
    maxValue = -1

    For i = 0 To UBound(PaymentRecord)
        If Not PaymentRecord(i).IsUsed And PaymentRecord(i).Payment > maxValue Then
            maxValue = PaymentRecord(i).Payment
            maxIndex = i
        End If
    Next i

    IndexOfBiggestNonUsedPayment = maxIndex
End Function

Public Function getNumberOfNotUsedPayments() As Long
    ' Returns the number of payments that have not been used.
    Dim count As Long
    count = 0
    For Each Payment In PaymentRecord
        If Payment.IsUsed = False Then
            count = count + 1
        End If
    Next
    getNumberOfNotUsedPayments = count
End Function

Sub BasicCheckDiscrepancy()
    ' Performs a basic discrepancy check between sales and total payments.
    If SalesRecord Is Nothing And Not ThereIsSavedPayment Then
        IsDiscrepant = False
        Exit Sub
    ElseIf SalesRecord Is Nothing Then
        IsDiscrepant = True
        Discrepancy = -TotalPaymentAmount
        Exit Sub
    ElseIf Not ThereIsSavedPayment Then
        IsDiscrepant = True
        Discrepancy = SalesRecord.SalesAmount
        Exit Sub
    End If

    If SalesRecord.SalesAmount = TotalPaymentAmount Then
        IsDiscrepant = False
    Else
        IsDiscrepant = True
        Discrepancy = SalesRecord.SalesAmount - TotalPaymentAmount
    End If
End Sub

Sub ComplexCheckDiscrepancy(effectivePayment As Double)
    ' Performs a discrepancy check using effective payments.
    If SalesRecord Is Nothing And Not ThereIsSavedPayment Then
        IsDiscrepant = False
        Exit Sub
    ElseIf SalesRecord Is Nothing Then
        IsDiscrepant = True
        Discrepancy = -effectivePayment
        Exit Sub
    ElseIf Not ThereIsSavedPayment Then
        IsDiscrepant = True
        Discrepancy = SalesRecord.SalesAmount
        Exit Sub
    End If

    If SalesRecord.SalesAmount = effectivePayment Then
        IsDiscrepant = False
    Else
        IsDiscrepant = True
        Discrepancy = SalesRecord.SalesAmount - effectivePayment
    End If
End Sub

Sub CheckDiscrepancy()
    ' Empty Sub (possibly for future use).
End Sub

Private Sub Class_Initialize()
    ' Initializes arrays and counters.
    ReDim PaymentRecord(0 To 0)
    TotalNumberOfPayment = 0
    TotalPaymentAmount = 0
```

Source Code: clsDays

clsDays contains the main business logic implementation. It manages data using Scripting.Dictionary objects. The mainDict holds all clsDailyRecord objects, which encapsulate originalclsSalesRecord and clsPaymentRecord objects. The centralPaymentManagement dictionary references all created payment objects, while freePaymentMethod keeps track of unused payment methods for the day–payment assignment task introduced in Level 3.

Dictionary objects maintain data in the given order, and the data provided by CCC was chronologically ordered. **However, I implemented a QuickSort algorithm to handle cases where the data might not be ordered**. Later, I enhanced and migrated the sorting logic to HelperMethods to include additional functionality, such as reversing functions and **distinguishing between shallow and deep copies**. Despite this migration, a basic sort method remains within the class.

```
Option Explicit

' Class: DAYS
' Manages daily records, payments, and discrepancies across multiple days.

Private mainDict As Scripting.Dictionary
' Dictionary to hold clsDailyRecord objects, keyed by day.

Private centralPaymentManagement As Scripting.Dictionary
' Central management of all payments, keyed by PaymentID.

Private freePaymentMethods As Scripting.Dictionary
' Dictionary of payments that are not yet used, keyed by PaymentID.

Public Sub AddPaymentRecordToDay(day As Long, PaymentRecord As clsPaymentRecord)
    ' Adds a PaymentRecord to a day's record.
    If Not mainDict.Exists(day) Then
        Dim DRtoSave As clsDailyRecord
        Set DRtoSave = New clsDailyRecord
        DRtoSave.day = day
        DRtoSave.addToPaymentRecordsArray PaymentRecord
        mainDict.Add day, DRtoSave
        AddReferenceOfPaymentRecordToTheCentralManagement PaymentRecord
    Else
        mainDict(day).addToPaymentRecordsArray PaymentRecord
        AddReferenceOfPaymentRecordToTheCentralManagement PaymentRecord
    End If
End Sub

Public Sub AddReferenceOfPaymentRecordToTheCentralManagement(PaymentRecord As
clsPaymentRecord)
    ' Adds a PaymentRecord to central management and free payments.
    PaymentRecord.PaymentID = "D" & PaymentRecord.day & "A" & PaymentRecord.Payment
    centralPaymentManagement.Add PaymentRecord.PaymentID, PaymentRecord
    freePaymentMethods.Add PaymentRecord.PaymentID, PaymentRecord
End Sub

Public Sub AddSalesRecordToDay(day As Long, SalesRecord As clsSalesRecord)
    ' Adds a SalesRecord to a day's record.
    If Not mainDict.Exists(day) Then
        Dim DRtoSave As clsDailyRecord
        Set DRtoSave = New clsDailyRecord
        DRtoSave.day = day
        Set DRtoSave.SalesRecord = SalesRecord
        mainDict.Add day, DRtoSave
    Else
        If Not mainDict(day).SalesRecord Is Nothing Then
            MsgBox ("Wrong Raw Data: Two Sales Records for same day: " & day)
            Exit Sub
        End If
        Set mainDict(day).SalesRecord = SalesRecord
    End If
End Sub

Property Get Dictionary() As Scripting.Dictionary
    ' Getter for the main dictionary of days.
    Set Dictionary = mainDict
End Property

Public Sub CheckAllDiscrepancy()
    ' Checks discrepancies for all days.
    Dim current As Variant
    Dim currentDay As clsDailyRecord

    For Each current In mainDict.Keys
        Set currentDay = mainDict(current)

        If current = 1 Then
            currentDay.ApplyAllPaymentsOfDayAsEffective
            currentDay.CalculateDiscrepancyWithCurrentEffectivePayments
            ForceRefreshFreePaymentRecords
        Else
            If Not currentDay.SalesRecord Is Nothing Then
                currentDay.ApplyAllRemainingPaymentsOfDayAsEffective
                ForceRefreshFreePaymentRecords
            End If
        End If

        currentDay.CalculateDiscrepancyWithCurrentEffectivePayments

        If currentDay.IsDiscrepant Then
            Dim discrepancyFiller As clsPaymentRecord
            Set discrepancyFiller = FindPaymentWithPaymentAmount(currentDay.
            Discrepancy)

            If Not discrepancyFiller Is Nothing Then
                currentDay.AddEffectivePayment discrepancyFiller
                currentDay.CalculateDiscrepancyWithCurrentEffectivePayments
                ForceRefreshFreePaymentRecords
            End If
        End If
    Next
End Sub

Public Sub applyAllRemaining()
    ' Applies all remaining payments as effective.
    Dim current As Variant
    Dim currentDay As clsDailyRecord

    For Each current In mainDict.Keys
        Set currentDay = mainDict(current)
        currentDay.ApplyAllRemainingPaymentsOfDayAsEffective
    Next
End Sub

Public Sub ForceRefreshFreePaymentRecords()
    ' Refreshes the list of free payment methods.
    Set freePaymentMethods = New Scripting.Dictionary
    Dim currentID As Variant

    For Each currentID In centralPaymentManagement.Keys
        If centralPaymentManagement(currentID).IsUsed = False Then
            freePaymentMethods.Add currentID, centralPaymentManagement(currentID)
        End If
    Next
End Sub

Public Function FindPaymentWithPaymentAmount(Amount As Long) As clsPaymentRecord
    ' Finds a free payment record with a specific payment amount.
    On Error Resume Next
    Dim currentID As Variant
    Dim returnRecord As clsPaymentRecord

    For Each currentID In freePaymentMethods.Keys
        If freePaymentMethods(currentID).IsUsed = False And freePaymentMethods(
        currentID).Payment = Amount Then
            Set returnRecord = freePaymentMethods(currentID)
            Exit For
        End If
    Next

    Set FindPaymentWithPaymentAmount = returnRecord
End Function

Public Sub BasicCheckAllDiscrepancy()
    ' Performs basic discrepancy checks for all days.
    Dim currentDay As Variant

    For Each currentDay In mainDict.Keys
        mainDict(currentDay).BasicCheckDiscrepancy
    Next
End Sub

Public Sub checkAllDiscrepancyAllowOnlyNextDay()
    ' Checks discrepancies, allowing payments from the next day.
    Dim currentDay As Variant
```

Source Code: clsDays

```
    For Each currentDay In mainDict.Keys
        On Error Resume Next
        Dim previousDay As Long
        Dim effectivePaymentToday As Double
        Dim calculativePaymentToday As Double

        calculativePaymentToday = mainDict(currentDay).TotalPaymentAmount
        previousDay = currentDay - 1

        If mainDict.Exists(previousDay) Then
            If mainDict(previousDay).IsDiscrepant Then
                Dim discrepancyPreviousDay As Long
                discrepancyPreviousDay = mainDict(previousDay).Discrepancy

                If calculativePaymentToday >= discrepancyPreviousDay Then
                    mainDict(previousDay).Discrepancy = 0
                    mainDict(previousDay).IsDiscrepant = False
                    effectivePaymentToday = calculativePaymentToday -
                    discrepancyPreviousDay
                Else
                    mainDict(previousDay).Discrepancy = discrepancyPreviousDay -
                    calculativePaymentToday
                    effectivePaymentToday = 0
                End If
            Else
                effectivePaymentToday = calculativePaymentToday
            End If
        Else
            effectivePaymentToday = calculativePaymentToday
        End If

        mainDict(currentDay).ComplexCheckDiscrepancy effectivePaymentToday
    Next
End Sub

Function getAnArrayOfSortedKeys() As Long()
    ' Returns an array of sorted keys (days).
    Dim k As Variant
    Dim i0 As Long
    i0 = 0
    Dim SortedArray() As Long

    ReDim SortedArray(0 To mainDict.Count - 1)

    For Each k In mainDict.Keys()
        SortedArray(i0) = k
        i0 = i0 + 1
    Next

    quicksort SortedArray, 0, UBound(SortedArray)
    getAnArrayOfSortedKeys = SortedArray
End Function

Sub quicksort(a() As Long, left As Long, right As Long)
    ' QuickSort algorithm.
    Dim P As Long

    If left < right Then
        P = partition(a, left, right)
        quicksort a, left, P
        quicksort a, P + 1, right
    End If
End Sub

Function partition(a() As Long, left As Long, right As Long) As Long
    ' Partition function for QuickSort.
    Dim Pivot As Long
    Pivot = a(left)
    Dim pl As Long
    Dim pr As Long
    pl = left
    pr = right

    Do While pl < pr
        Do While a(pl) <= Pivot And pl < right
            pl = pl + 1
        Loop

        Do While a(pr) > Pivot And pr > left
            pr = pr - 1
        Loop

        If pl < pr Then
            swap a, pl, pr
        End If
    Loop

    swap a, left, pr
    partition = pr
End Function

Sub swap(a() As Long, i As Long, j As Long)
    ' Swaps two elements in an array.
    Dim temp As Long
    temp = a(i)
    a(i) = a(j)
    a(j) = temp
End Sub

Sub WriteData()
    ' Writes data to the worksheet starting at cell A10.
    On Error Resume Next
    Dim startIndex As Range
    Set startIndex = Range("A10")
    Dim ColumnPointer As Long
    ColumnPointer = 1
    Dim sortedKeys As Variant
    sortedKeys = Me.getAnArrayOfSortedKeys
    Dim Key As Variant

    Range("A10", "D100").Value = ""

    For Each Key In sortedKeys
        startIndex.Cells(ColumnPointer, 1).Value = mainDict(Key).day
        startIndex.Cells(ColumnPointer, 2).Value = mainDict(Key).SalesRecord.
        SalesAmount
        startIndex.Cells(ColumnPointer, 3).Value = mainDict(Key).TotalPaymentAmount
        startIndex.Cells(ColumnPointer, 4).Value = mainDict(Key).Discrepancy
        ColumnPointer = ColumnPointer + 1
    Next Key
End Sub

Sub WriteAuditResult()
    ' Writes the audit results to cell B4.
    Dim Index As Range
    Set Index = Range("B4")
    Dim Result As String
    Dim sortedKeys As Variant
    sortedKeys = Me.getAnArrayOfSortedKeys
    Dim Key As Variant

    For Each Key In sortedKeys
        If mainDict(Key).IsDiscrepant Then
            Result = Result & " " & mainDict(Key).day
        End If
    Next Key

    Index.Value = Result
End Sub

Sub ReverseArray(a() As Long)
    ' Reverses the elements of an array.
    Dim left As Long
    Dim right As Long
    left = LBound(a)
    right = UBound(a)

    Do While left < right
        swap a, left, right
        left = left + 1
        right = right - 1
    Loop
End Sub

Private Sub Class_Initialize()
    ' Initializes the dictionaries.
    Set mainDict = New Scripting.Dictionary
    Set centralPaymentManagement = New Scripting.Dictionary
    Set freePaymentMethods = New Scripting.Dictionary
End Sub

Private Sub Class_Terminate()
    ' Cleanup code if necessary.
End Sub
```

Source Code: clsDays

```
    For Each currentDay In mainDict.Keys
        On Error Resume Next
        Dim previousDay As Long
        Dim effectivePaymentToday As Double
        Dim calculativePaymentToday As Double

        calculativePaymentToday = mainDict(currentDay).TotalPaymentAmount
        previousDay = currentDay - 1

        If mainDict.Exists(previousDay) Then
            If mainDict(previousDay).IsDiscrepant Then
                Dim discrepancyPreviousDay As Long
                discrepancyPreviousDay = mainDict(previousDay).Discrepancy

                If calculativePaymentToday >= discrepancyPreviousDay Then
                    mainDict(previousDay).Discrepancy = 0
                    mainDict(previousDay).IsDiscrepant = False
                    effectivePaymentToday = calculativePaymentToday -
                    discrepancyPreviousDay
                Else
                    mainDict(previousDay).Discrepancy = discrepancyPreviousDay -
                    calculativePaymentToday
                    effectivePaymentToday = 0
                End If
            Else
                effectivePaymentToday = calculativePaymentToday
            End If
        Else
            effectivePaymentToday = calculativePaymentToday
        End If

        mainDict(currentDay).ComplexCheckDiscrepancy effectivePaymentToday
    Next
End Sub

Function getAnArrayOfSortedKeys() As Long()
    ' Returns an array of sorted keys (days).
    Dim k As Variant
    Dim i0 As Long
    i0 = 0
    Dim SortedArray() As Long

    ReDim SortedArray(0 To mainDict.Count - 1)

    For Each k In mainDict.Keys()
        SortedArray(i0) = k
        i0 = i0 + 1
    Next

    quicksort SortedArray, 0, UBound(SortedArray)
    getAnArrayOfSortedKeys = SortedArray
End Function

Sub quicksort(a() As Long, left As Long, right As Long)
    ' QuickSort algorithm.
    Dim P As Long

    If left < right Then
        P = partition(a, left, right)
        quicksort a, left, P
        quicksort a, P + 1, right
    End If
End Sub

Function partition(a() As Long, left As Long, right As Long) As Long
    ' Partition function for QuickSort.
    Dim Pivot As Long
    Pivot = a(left)
    Dim pl As Long
    Dim pr As Long
    pl = left
    pr = right

    Do While pl < pr
        Do While a(pl) <= Pivot And pl < right
            pl = pl + 1
        Loop

        Do While a(pr) > Pivot And pr > left
            pr = pr - 1
        Loop

        If pl < pr Then
            swap a, pl, pr
        End If
    Loop

    swap a, left, pr
    partition = pr
End Function

Sub swap(a() As Long, i As Long, j As Long)
    ' Swaps two elements in an array.
    Dim temp As Long
    temp = a(i)
    a(i) = a(j)
    a(j) = temp
End Sub

Sub WriteData()
    ' Writes data to the worksheet starting at cell A10.
    On Error Resume Next
    Dim startIndex As Range
    Set startIndex = Range("A10")
    Dim ColumnPointer As Long
    ColumnPointer = 1
    Dim sortedKeys As Variant
    sortedKeys = Me.getAnArrayOfSortedKeys
    Dim Key As Variant

    Range("A10", "D100").Value = ""

    For Each Key In sortedKeys
        startIndex.Cells(ColumnPointer, 1).Value = mainDict(Key).day
        startIndex.Cells(ColumnPointer, 2).Value = mainDict(Key).SalesRecord.
        SalesAmount
        startIndex.Cells(ColumnPointer, 3).Value = mainDict(Key).TotalPaymentAmount
        startIndex.Cells(ColumnPointer, 4).Value = mainDict(Key).Discrepancy
        ColumnPointer = ColumnPointer + 1
    Next Key
End Sub

Sub WriteAuditResult()
    ' Writes the audit results to cell B4.
    Dim Index As Range
    Set Index = Range("B4")
    Dim Result As String
    Dim sortedKeys As Variant
    sortedKeys = Me.getAnArrayOfSortedKeys
    Dim Key As Variant

    For Each Key In sortedKeys
        If mainDict(Key).IsDiscrepant Then
            Result = Result & " " & mainDict(Key).day
        End If
    Next Key

    Index.Value = Result
End Sub

Sub ReverseArray(a() As Long)
    ' Reverses the elements of an array.
    Dim left As Long
    Dim right As Long
    left = LBound(a)
    right = UBound(a)

    Do While left < right
        swap a, left, right
        left = left + 1
        right = right - 1
    Loop
End Sub

Private Sub Class_Initialize()
    ' Initializes the dictionaries.
    Set mainDict = New Scripting.Dictionary
    Set centralPaymentManagement = New Scripting.Dictionary
    Set freePaymentMethods = New Scripting.Dictionary
End Sub

Private Sub Class_Terminate()
    ' Cleanup code if necessary.
End Sub
```

Source Code: HelperMethods

```vb
Option Explicit

' Module: HelperMethods
' Contains helper functions for array sorting and manipulation.

Function getSortedArrayAscending(ArrayToSort() As Long) As Long
    ' Returns a sorted array in ascending order.
    Dim SortedArray() As Long
    SortedArray = CloneArray(ArrayToSort)
    quicksort SortedArray, 0, UBound(SortedArray)
    getSortedArrayAscending = SortedArray
End Function

Function getSortedArrayDescending(ArrayToSort() As Long) As Long
    ' Returns a sorted array in descending order.
    Dim SortedArray() As Long
    SortedArray = CloneArray(ArrayToSort)
    quicksort SortedArray, 0, UBound(SortedArray)
    ReverseArray SortedArray
    sortArray = SortedArray
End Function

Sub sortGivenArrayAscending(ArrayToSort() As Long)
    ' Sorts the given array in ascending order.
    quicksort ArrayToSort, 0, UBound(ArrayToSort)
End Sub

Sub sortGivenArrayDescending(ArrayToSort() As Long)
    ' Sorts the given array in descending order.
    quicksort ArrayToSort, 0, UBound(ArrayToSort)
    ReverseArray ArrayToSort
End Sub

Sub quicksort(a() As Long, left As Long, right As Long)
    ' QuickSort algorithm.
    Dim P As Long

    If left < right Then
        P = partition(a, left, right)
        quicksort a, left, P
        quicksort a, P + 1, right
    End If
End Sub

Function partition(a() As Long, left As Long, right As Long) As Long
    ' Partition function for QuickSort.
    Dim Pivot As Long
    Pivot = a(left)
    Dim pl As Long
    Dim pr As Long
    pl = left
    pr = right

    Do While pl < pr
        Do While a(pl) <= Pivot And pl < right
            pl = pl + 1
        Loop

        Do While a(pr) > Pivot And pr > left
            pr = pr - 1
        Loop

        If pl < pr Then
            swap a, pl, pr
        End If
    Loop

    swap a, left, pr
    partition = pr
End Function

Sub swap(a() As Long, i As Long, j As Long)
    ' Swaps two elements in an array.
    Dim temp As Long
    temp = a(i)
    a(i) = a(j)
    a(j) = temp
End Sub

Sub ReverseArray(a() As Long)
    ' Reverses the elements of an array.
    Dim left As Long
    Dim right As Long
    left = LBound(a)
    right = UBound(a)

    Do While left < right
        swap a, left, right
        left = left + 1
        right = right - 1
    Loop
End Sub

Function CloneArray(arr() As Long) As Long
    ' Clones an array.
    Dim newArray() As Long
    ReDim newArray(LBound(arr) To UBound(arr))
    Dim i As Long

    For i = LBound(arr) To UBound(arr)
        newArray(i) = arr(i)
    Next i

    CloneArray = newArray
End Function

```

Thank you for your attention.

Vielen Dank für Ihre Aufmerksamkeit.

I look forward to your feedback.
Ich freue mich auf ihre Rückmeldung.

Kaan Bora Karapınar