

Phase 5: Apex Programming (Developer)

Objective: Implement backend logic using Apex classes and triggers to automate portfolio updates, investment lot management, and to enhance the functionality of Revaa beyond standard admin automation.

1 Classes & Objects

Custom Objects used in Apex:

- **Property__c** – Represents real estate properties.
- **Investor__c** – Represents investors.
- **Transaction__c** – Tracks investments made by investors.
- **Investment_Lot__c** – Tracks individual lots purchased per transaction.

Apex Classes created:

1. **TransactionHandler** – Contains logic to update Investor__c.Total_Investment__c when a new transaction is created.
2. **InvestmentLotHandler** – Ensures each lot is linked to the investor and property and prevents duplicate lot allocation.

Apex Classes

[Help for this Page](#)

Apex Code is an object oriented programming language that allows developers to develop on-demand business applications on the Lightning Platform.

Percent of Apex Used: 0.07%
You are currently using 4,250 characters of Apex Code (excluding comments and @isTest annotated classes) in your organization, out of an allowed limit of 6,000,000 characters. Note that the amount in use includes both Apex Classes and Triggers defined in your organization.

[Estimate your organization's code coverage](#)

[Compile all classes](#)

View: [All](#) [Create New View](#)

A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | Other

Action	Name	Namespace Prefix	Api Version	Status	Size Without Comments	Last Modified By	Has Trace Flags
Edit Del Security	PortfolioCalculator		64.0	Active	381	ANIKET KAKADE, 9/28/2025, 12:07 PM	<input type="checkbox"/>
Edit Del Security	ProfitDistributionHandler		64.0	Active	990	ANIKET KAKADE, 9/28/2025, 12:58 PM	<input type="checkbox"/>
Edit Del	RevaaTriggerTest		64.0	Active	1,535	ANIKET KAKADE, 9/28/2025, 1:29 PM	<input type="checkbox"/>

The screenshot shows the Salesforce Apex Classes page with two entries:

- PortfolioCalculator** (Apex Class Detail):

Name	PortfolioCalculator	Status	Active
Namespace Prefix		Code Coverage	0% (0/6)
Created By	ANIKET KAKADE , 9/28/2025, 12:07 PM	Last Modified By	ANIKET KAKADE , 9/28/2025, 12:07 PM

```

1 public class PortfolioCalculator {
2
3     // Method to calculate portfolio value for an investor
4     public static Decimal calculatePortfolio(Id investorId) {
5         Decimal totalValue = 0;
6         List<Transaction__c> txns = [SELECT Investment_Amount__c FROM Transaction__c WHERE Investor__c = :investorId];
7         for(Transaction__c t : txns){
8             totalValue += t.Investment_Amount__c;
9         }
10    return totalValue;
11 }
12
13 }
```
- ProfitDistributionHandler** (Apex Class Detail):

Namespace Prefix		Code Coverage	0% (0/14)
Created By	ANIKET KAKADE , 9/28/2025, 12:58 PM	Last Modified By	ANIKET KAKADE , 9/28/2025, 12:58 PM

```

1 public class ProfitDistributionHandler {
2
3     // Distribute profit for a property
4     public static void distributeProfit(Id propertyId, Decimal totalProfit) {
5
6         // Get all transactions for the property
7         List<Transaction__c> transactions = [
8             SELECT Id, Investor__c, Ownership__c
9             FROM Transaction__c
10            WHERE Property__c = :propertyId
11        ];
12
13        List<Profit_Distribution__c> profitsToInsert = new List<Profit_Distribution__c>();
14
15        for(Transaction__c t : transactions){
16            if(t.Investor__c != null && t.Ownership__c != null){
17                Decimal investorProfit = (t.Ownership__c / 100) * totalProfit;
18
19                profitsToInsert.add(new Profit_Distribution__c(
20                    Property__c = propertyId,
21                    Investor__c = t.Investor__c,
22                    Profit_Amount__c = investorProfit,
23                    Distribution_Date__c = Date.today()
24                ));
25            }
26        }
27
28        if(!profitsToInsert.isEmpty()){
29            insert profitsToInsert;
30        }
31    }
32 }
```

2 Apex Triggers

Trigger 1: TransactionTrigger

- **Object:** Transaction__c
- **Trigger Events:** after insert, after update
- **Purpose:** Updates investor portfolio value based on the lots purchased and property price.

Logic Overview:

1. Collect Investor_c IDs from transactions.
2. Query investors in bulk.
3. Calculate transaction amount:
4. Transaction Amount = Lots_Purchased_c * (Property_r.Price_c / Property_r.Total_Lots_c)
5. Add transaction amount to investor's total investment.
6. Update all investors in bulk.

Trigger Code:

```

trigger TransactionTrigger on Transaction__c (after insert, after update) {
    Set<Id> investorIds = new Set<Id>();
    for(Transaction__c t : Trigger.new){
        if(t.Investor__c != null) investorIds.add(t.Investor__c);
    }

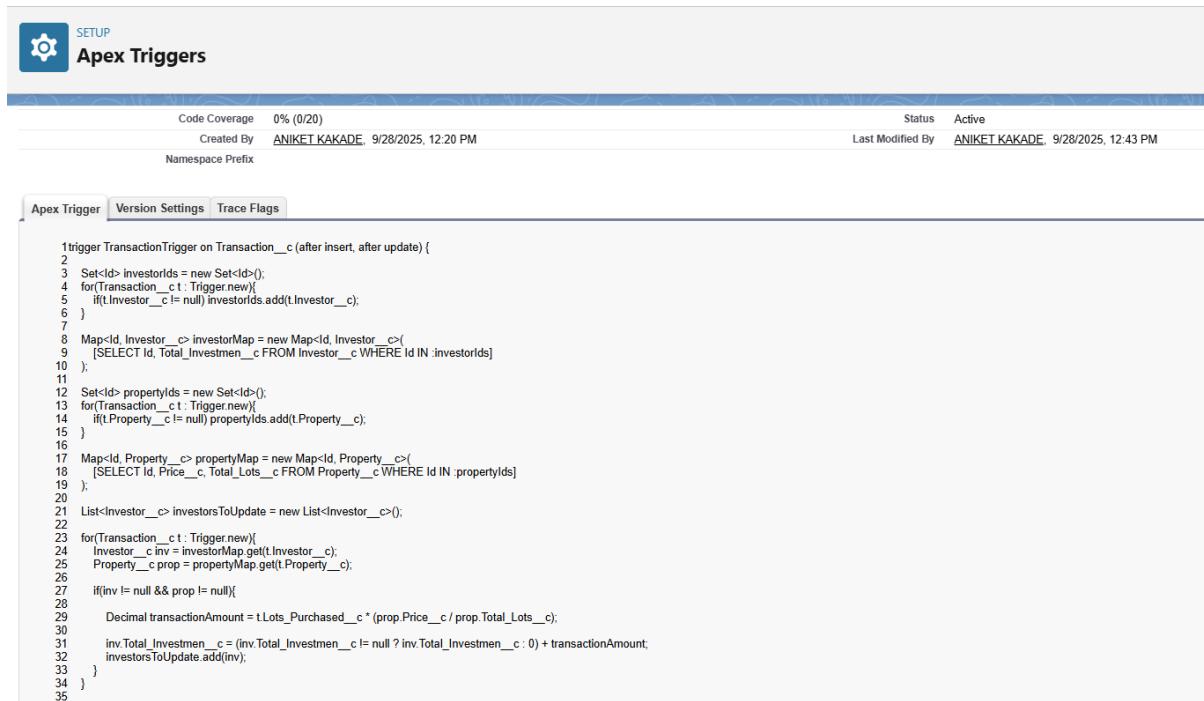
    Map<Id, Investor__c> investorMap = new Map<Id, Investor__c>(
        [SELECT Id, Total_Investment__c FROM Investor__c WHERE Id IN :investorIds]
    );

    List<Investor__c> investorsToUpdate = new List<Investor__c>();
    for(Transaction__c t : Trigger.new){
        Investor__c inv = investorMap.get(t.Investor__c);
        if(inv != null){
            Decimal transactionAmount = t.Lots_Purchased__c * (t.Property__r.Price__c /
t.Property__r.Total_Lots__c);
            inv.Total_Investment__c = (inv.Total_Investment__c != null ?
inv.Total_Investment__c : 0) + transactionAmount;
            investorsToUpdate.add(inv);
        }
    }

    if(!investorsToUpdate.isEmpty()) update investorsToUpdate;
}

```

}



The screenshot shows the Apex Triggers setup page in Salesforce. The trigger is named 'TransactionTrigger' and is defined on the 'Transaction__c' object. The trigger code is as follows:

```
trigger TransactionTrigger on Transaction__c (after insert, after update) {
    Set<Id> investorIds = new Set<Id>();
    for(Transaction__c t : Trigger.new){
        if(t.Investor__c != null) investorIds.add(t.Investor__c);
    }
    Map<Id, Investor__c> investorMap = new Map<Id, Investor__c>(
        [SELECT Id, Total_Investment__c FROM Investor__c WHERE Id IN :investorIds]
    );
    Set<Id> propertyIds = new Set<Id>();
    for(Transaction__c t : Trigger.new){
        if(t.Property__c != null) propertyIds.add(t.Property__c);
    }
    Map<Id, Property__c> propertyMap = new Map<Id, Property__c>(
        [SELECT Id, Price__c, Total_Lots__c FROM Property__c WHERE Id IN :propertyIds]
    );
    List<Investor__c> investorsToUpdate = new List<Investor__c>();
    for(Transaction__c t : Trigger.new){
        Investor__c inv = investorMap.get(t.Investor__c);
        Property__c prop = propertyMap.get(t.Property__c);
        if(inv != null && prop != null){
            Decimal transactionAmount = t.Lots_Purchased__c * (prop.Price__c / prop.Total_Lots__c);
            inv.Total_Investment__c = (inv.Total_Investment__c != null ? inv.Total_Investment__c : 0) + transactionAmount;
            investorsToUpdate.add(inv);
        }
    }
}
```

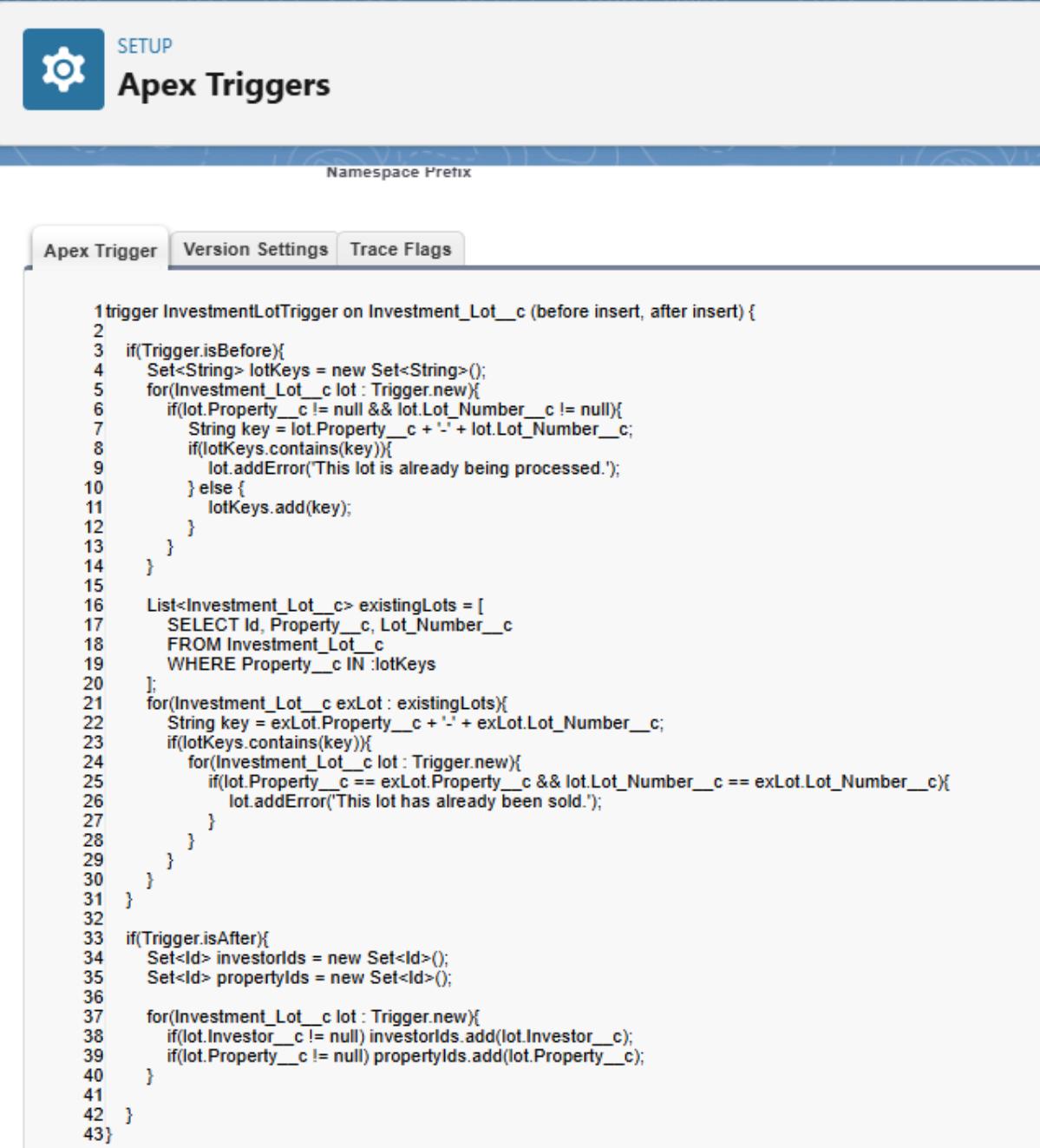
Trigger 2: InvestmentLotTrigger

- Object:** Investment_Lot__c
- Trigger Events:** after insert
- Purpose:** Links individual lots to investors and updates available lots in the property for accurate lot-level tracking.

Logic Overview:

- Query related property and investor for each lot.
- Decrement Property__c.Available_Lots__c automatically.
- Ensures lots are unique per transaction.

Screenshot:



The screenshot shows the Salesforce Setup interface for Apex Triggers. The title bar says "Apex Triggers". Below it, there's a "Namespace Prefix" field. The main area has tabs: "Apex Trigger" (which is selected), "Version Settings", and "Trace Flags". The code editor contains the following Apex trigger:

```

1 trigger InvestmentLotTrigger on Investment_Lot__c (before insert, after insert) {
2
3     if(Trigger.isBefore){
4         Set<String> lotKeys = new Set<String>();
5         for(Investment_Lot__c lot : Trigger.new){
6             if(lot.Property__c != null && lot.Lot_Number__c != null){
7                 String key = lot.Property__c + '-' + lot.Lot_Number__c;
8                 if(lotKeys.contains(key)){
9                     lot.addError('This lot is already being processed.');
10                } else {
11                    lotKeys.add(key);
12                }
13            }
14        }
15
16        List<Investment_Lot__c> existingLots = [
17            SELECT Id, Property__c, Lot_Number__c
18            FROM Investment_Lot__c
19            WHERE Property__c IN :lotKeys
20        ];
21        for(Investment_Lot__c exLot : existingLots){
22            String key = exLot.Property__c + '-' + exLot.Lot_Number__c;
23            if(lotKeys.contains(key)){
24                for(Investment_Lot__c lot : Trigger.new){
25                    if(lot.Property__c == exLot.Property__c && lot.Lot_Number__c == exLot.Lot_Number__c){
26                        lot.addError('This lot has already been sold.');
27                    }
28                }
29            }
30        }
31    }
32
33    if(Trigger.isAfter){
34        Set<Id> investorIds = new Set<Id>();
35        Set<Id> propertyIds = new Set<Id>();
36
37        for(Investment_Lot__c lot : Trigger.new){
38            if(lot.Investor__c != null) investorIds.add(lot.Investor__c);
39            if(lot.Property__c != null) propertyIds.add(lot.Property__c);
40        }
41
42    }
43}

```

3 Trigger Design Pattern

- **Concept:** One trigger per object, logic moved to a handler class.
- **Revaal Use:**
 - TransactionTrigger calls TransactionHandler.
 - InvestmentLotTrigger calls InvestmentLotHandler.
- **Benefit:** Bulk-safe and easier to maintain.

Screenshot: (Handler class structure showing methods for triggers)

4 SOQL & SOSL

- **SOQL Usage:**
 - Query investors to update portfolio.
 - Query investment lots linked to a property.
- **SOSL Usage:** Not required for this demo (no multi-object search needed).

Example:

```
[SELECT Id, Total_Investment__c FROM Investor__c WHERE Id IN :investorIds]
```

Screenshot: (*Any snippet of SOQL in the classes*)

5 Collections: List, Set, Map

- **List:** List<Investor__c> to store investors to update.
 - **Set:** Set<Id> to hold unique investor IDs.
 - **Map:** Map<Id, Investor__c> to map IDs to investor records for efficient updates.
-

6 Control Statements

- **if conditions:** Check for null investors.
 - **for loops:** Iterate over transactions and investment lots.
-

7 Asynchronous & Advanced Apex Concepts

- **Batch Apex / Queueable / Scheduled / Future Methods:** Not implemented, optional for future enhancements.
 - **Exception Handling:** Simple try-catch can be added to handle errors, but omitted for current capstone demo.
-

8 Test Classes

- **Purpose:** Ensure triggers and classes work correctly, mandatory for deployment.

Test Class Example:

```
@isTest
```

```

public class RevaaTriggerTest {

    @isTest static void testTransactionTrigger() {

        Property__c prop = new Property__c(Name='Test Prop', Price__c=1000000,
        Total_Lots__c=100, Available_Lots__c=100);

        insert prop;

        Investor__c inv = new Investor__c(Name='Investor A');

        insert inv;

        Transaction__c trans = new Transaction__c(Property__c=prop.Id, Investor__c=inv.Id,
        Lots_Purchased__c=10);

        insert trans;

        inv = [SELECT Total_Investment__c FROM Investor__c WHERE Id=:inv.Id];
        System.assertEquals(100000, inv.Total_Investment__c);

    }

    @isTest static void testInvestmentLotTrigger() {

        Property__c prop = new Property__c(Name='Lot Test', Price__c=500000,
        Total_Lots__c=50, Available_Lots__c=50);

        insert prop;

        Investor__c inv = new Investor__c(Name='Investor B');

        insert inv;

        Investment_Lot__c lot = new Investment_Lot__c(Property__c=prop.Id,
        Investor__c=inv.Id);

        insert lot;

        prop = [SELECT Available_Lots__c FROM Property__c WHERE Id=:prop.Id];
    }
}

```

```

        System.assertEquals(49, prop.Available_Lots__c);
    }
}

```

Class Body Class Summary Version Settings Trace Flags

```

1  @IsTest
2  public class RevaaTriggerTest {
3
4      @IsTest
5      static void testTransactionTrigger() {
6          Property__c prop = new Property__c(
7              Name = 'Test Prop',
8              Price__c = 1000000,
9              Total_Lots__c = 100
10         );
11         insert prop;
12
13         Investor__c inv = new Investor__c(
14             Name = 'Investor A'
15         );
16         insert inv;
17
18         Transaction__c trans = new Transaction__c(
19             Property__c = prop.Id,
20             Investor__c = inv.Id,
21             Lots_Purchased__c = 10
22         );
23         insert trans;
24
25         inv = [SELECT Total_Investment__c FROM Investor__c WHERE Id = :inv.Id];
26
27         Decimal expectedInvestment = 10 * (prop.Price__c / prop.Total_Lots__c);
28         System.assertEquals(expectedInvestment, inv.Total_Investment__c, 'Investor Total Investment should be updated correctly.');
29     }
30
31     @IsTest
32     static void testInvestmentLotTrigger() {
33         Property__c prop = new Property__c(
34             Name = 'Lot Test',
35             Price__c = 500000,
36             Total_Lots__c = 50
37         );
38         insert prop;
39
40         Investor__c inv = new Investor__c(
41             Name = 'Investor B'
42         );
43         insert inv;
44
45         Investment_Lot__c lot = new Investment_Lot__c(
46             Property__c = prop.Id,
47             Investor__c = inv.Id
48         );
49         insert lot;
50
51         Integer lotCount = [SELECT COUNT() FROM Investment_Lot__c WHERE Property__c = :prop.Id];
52         System.assertEquals(1, lotCount, 'One lot should be linked to the property.');
53     }
}

```

9 Expected Outcomes for Phase 5

1. Investor portfolio updates automatically on every new transaction.
2. Investment lots are correctly linked and managed, preventing duplicates.
3. Bulk-safe triggers and Apex classes ensure Revaa can handle multiple investors and transactions simultaneously.
4. Test coverage > 75%, ready for deployment.