Problem statement: “CPQ & Order Visibility Hub for Industrial Manufacturing”

Phase5: Apex Programming (Developer)

- Classes & Objects

Purpose: Custom logic that can’t be handled easily in Flows.

Apex classes act as logic containers for complex rules that can’t be done with Flows alone.

- Handles validation when Sales Reps add Quote Lines.
- Ensures that requested quantity doesn’t exceed available stock in Inventory__c.

Why: Keeps logic centralized & reusable, instead of scattered in multiple Flows.

nventoryHelper.apxc | Log executeAnonymous @24/9/2025, 5:13:42 pm

Code Coverage: None | API Version: 64

```
public with sharing class InventoryHelper {

    public static void reduceInventory(Map<Id, Integer> qtyByInventoryId) {
        if (qtyByInventoryId == null || qtyByInventoryId.isEmpty()) return;


        List<Inventory__c> invList = [
            SELECT Id, Stock_Level__c
            FROM Inventory__c
            WHERE Id IN :qtyByInventoryId.keySet()
            FOR UPDATE
        ];

        for (Inventory__c inv : invList) {
            Integer toReduce = qtyByInventoryId.get(inv.Id);
            Decimal current = (inv.Stock_Level__c == null) ? 0 : inv.Stock_Level__c;
            Decimal updated = current - (toReduce == null ? 0 : toReduce);

            if (updated < 0) updated = 0;
            inv.Stock_Level__c = updated;
        }

        if (!invList.isEmpty()) {
            update invList;
        }
    }
```

Anonyamous code window:

Enter Apex Code

```
Inventory__c inv = new Inventory__c(
    Stock_Level__c = 10,
    Reorder_Point__c = 2
);
insert inv;
Map<Id, Integer> qtyMap = new Map<Id, Integer>{
    inv.Id => 4
};

InventoryHelper.reduceInventory(qtyMap);

Inventory__c updated = [SELECT Id, Stock_Level__c FROM Inventory__c WHERE Id = :inv.Id];

// --- Output result for screenshot ---
System.debug('Original Stock Level: 10');
System.debug('Quantity Reduced: 4');
System.debug('Updated Stock Level: ' + updated.Stock_Level__c);
```

Open Log | Execute | Execute Highlighted

Output:

InventoryHelper.apxc | Log executeAnonymous @24/9/2025, 5:13:42 pm

Execution Log

Timestamp	Event	Details
17:13:42:090	USER_DEBUG	[20]|DEBUG|Original Stock Level: 10
17:13:42:090	USER_DEBUG	[21]|DEBUG|Quantity Reduced: 4
17:13:42:090	USER_DEBUG	[22]|DEBUG|Updated Stock Level: 6.00

This Frame | Executable | Debug Only | Filter | Click here to filter the log

Logs | Tests | Checkpoints | Query Editor | View State | Progress | Problems

User	Application	Operation	Time	Status	Read	Size
sales manager	Unknown	/services/data/v64.0/tooling/executeA...	24/9/2025, 5:13:42 pm	Success		9.35 KB
sales manager	Unknown	ApexTestHandler	24/9/2025, 5:13:22 pm	Success	Unread	15.51 KB
sales manager	Unknown	ApexTestHandler	24/9/2025, 5:13:19 pm	Success	Unread	14.59 KB

File | Edit | Debug | Test | Workspace | Help

QuoteLineTriggerHandler.apxc | Log executeAnonymous @24/9/2025, 5:44:30 pm

Code Coverage: None | API Version: 64 | Go To

```
public with sharing class QuoteLineTriggerHandler {

    public static void checkInventory(List<SBQQ__QuoteLine__c> quoteLines) {
        // Collect Inventory IDs from the Quote Lines
        Set<Id> inventoryIds = new Set<Id>();
        for (SBQQ__QuoteLine__c ql : quoteLines) {
            if (ql.Inventory_Lookup__c != null) {
                inventoryIds.add(ql.Inventory_Lookup__c);
            }
        }

        // Fetch Inventory records
        Map<Id, Inventory__c> inventoryMap = new Map<Id, Inventory__c>(
            [SELECT Id, Name, Stock_Level__c
             FROM Inventory__c
             WHERE Id IN :inventoryIds]
        );

        // Compare stock with requested quantity
        for (SBQQ__QuoteLine__c ql : quoteLines) {
            if (ql.Inventory_Lookup__c != null && inventoryMap.containsKey(ql.Inventory_Lookup__c)) {
                Inventory__c inv = inventoryMap.get(ql.Inventory_Lookup__c);

                if (ql.SBQQ__Quantity__c > inv.Stock_Level__c) {
                    ql.addError(
                        'Requested quantity (' + ql.SBQQ__Quantity__c +
```

Anonyamous code window:

Enter Apex Code

```apex
try {
    Inventory__c inv = new Inventory__c(
        Stock_Level__c = 50,
        Reorder_Point__c = 5
    );
    insert inv;
    System.debug('Inventory created Id=' + inv.Id + ' Stock=' + inv.Stock_Level__c);

    SBQQ__Quote__c quote = new SBQQ__Quote__c(
        SBQQ__Status__c = 'Draft'
    );
    insert quote;
    System.debug('Quote created Id=' + quote.Id + ' Status=' + quote.SBQQ__Status__c);

    Map<String, Schema.SObjectField> qlFields = Schema.SObjectType.SBQQ__QuoteLine__c.fields.getMap();
    String invFieldName = null;
    if (qlFields.containsKey('Inventory_Lookup__c')) {
        invFieldName = 'Inventory_Lookup__c';
    } else {
        for (String fn : qlFields.keySet()) {
            if (fn.toLowerCase().contains('inventory')) {
                invFieldName = fn;
                break;
            }
        }
    }
```

Output:

File ▾ Edit ▾ Debug ▾ Test ▾ Workspace ▾ Help ▾ < >

QuoteLineTriggerHandler.apxc | Log executeAnonymous @24/9/2025, 7:05:52 pm

Execution Log

Timestamp	Event	Details
19:05:52:002	USER_DEBUG	[7]|DEBUG|QuoteLineTriggerHandler
19:05:52:047	USER_DEBUG	[16]|DEBUG|Inventory created Id=a00WU00001KIJY3YAP Stock=50
19:05:52:250	USER_DEBUG	[24]|DEBUG|Quote created Id=a12WU000007kcO4YAI Status=Draft
19:05:52:257	USER_DEBUG	[45]|DEBUG|Using QuoteLine inventory field: Inventory_Lookup__c
19:05:52:845	USER_DEBUG	[61]|DEBUG|QuoteLine inserted successfully. Id = a0yWU000000AFQnYAO
19:05:52:956	USER_DEBUG	[77]|DEBUG|Quote updated to Accepted. Quote Id=a12WU000007kcO4YAI
19:05:52:963	USER_DEBUG	[85]|DEBUG|Id=a00WU00001KIJY3YAP Stock=40.00 ReorderPoint=5.00
19:05:52:963	USER_DEBUG	[87]|DEBUG|Test END (SUCCESS)

◆ Apex Triggers (before/after insert/update/delete)

Apex Trigger – Order → Inventory

Use case: Prevent overselling, keep stock levels correct.

use triggers to automatically enforce business rules at the database level.

- Example: QuoteLineTrigger.trigger
  - Runs before insert & before update on SBQQ__QuoteLine__c.
  - Calls the handler to check if stock is sufficient.
- Why: Prevents invalid data (overselling products) from ever being saved in Salesforce.

File ▾ Edit ▾ Debug ▾ Test ▾ Workspace ▾ Help ▾ < >

ShipmentTrigger.apxt | QuoteLineTrigger.apxt | Log executeAnonymous @24/9/2025, 5:51:24 pm

Code Coverage: None ▾ API Version: 64

```
1 trigger QuoteLineTrigger on SBQQ__QuoteLine__c (before insert, before update) {
2     if (Trigger.isBefore) {
3         QuoteLineTriggerHandler.checkInventory(Trigger.new);
4     }
5 }
```

Anonyamous code window:

Enter Apex Code

```
1  SBQQ__Quote__c quote = new SBQQ__Quote__c(
2      SBQQ__Status__c = 'Draft'
3  );
4
5
6  insert quote;
7  SBQQ__QuoteLine__c ql1 = new SBQQ__QuoteLine__c(
8      SBQQ__Quote__c = quote.Id,
9      SBQQ__Quantity__c = 5
10
11 );
12 insert ql1;
13
14 // Update Quote to trigger 'Accepted' status
15 quote.SBQQ__Status__c = 'Accepted';
16 update quote;
17 SBQQ__Quote__c updatedQuote = [SELECT Id, SBQQ__Status__c FROM SBQQ__Quote__c WHERE Id = :quote.Id];
18 System.debug('Updated Quote Status: ' + updatedQuote.SBQQ__Status__c);
19
```

Open Log | Execute | Execute Highlighted

Output:

File ▾ Edit ▾ Debug ▾ Test ▾ Workspace ▾ Help ▾ < >

QuoteLineTriggerHandler.apxc | Log executeAnonymous @24/9/2025, 5:44:30 pm

Execution Log

Timestamp	Event	Details
17:44:30:823	USER_DEBUG	[21]|DEBUG|Updated Quote Status: Accepted

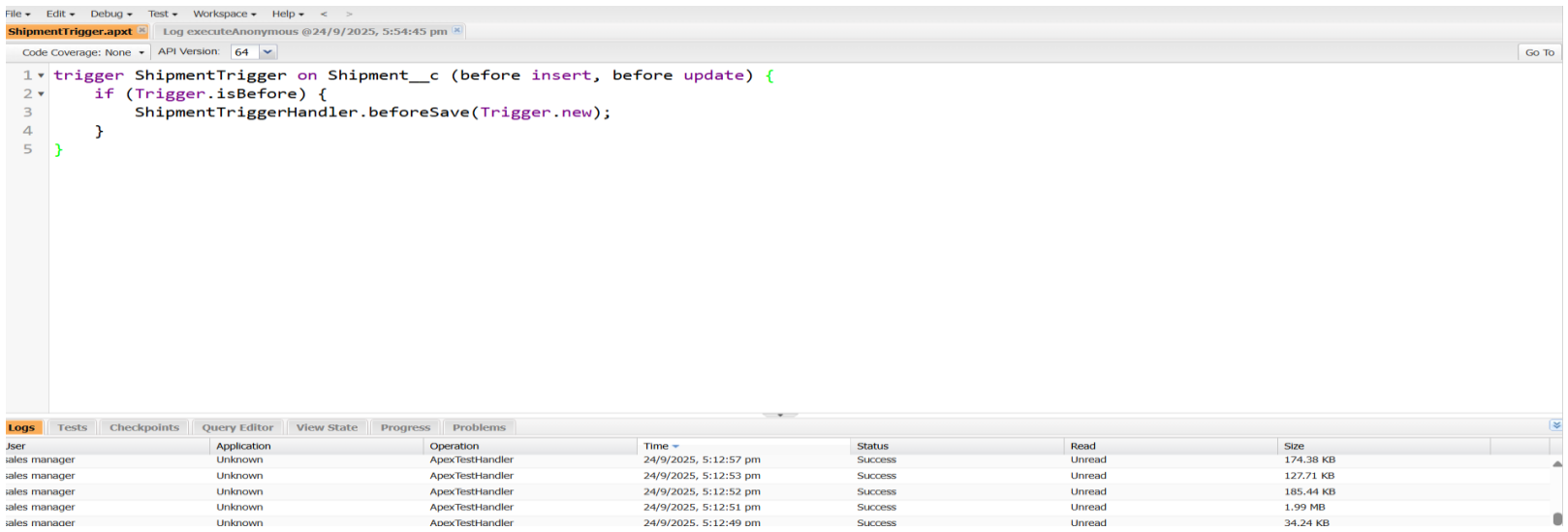
◆ Trigger Design Pattern

To avoid messy triggers, i separate logic into handler classes.

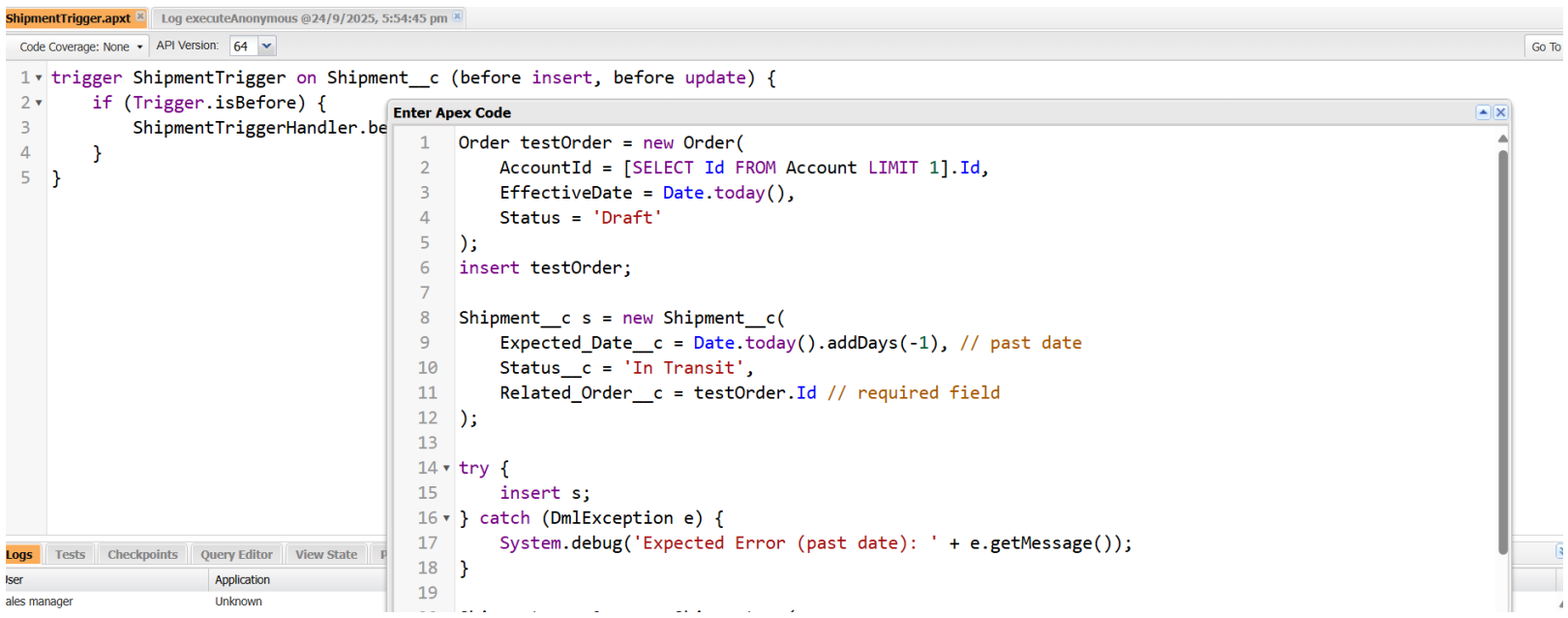
- Trigger only contains:
- if (Trigger.isBefore) {

- QuoteLineTriggerHandler.checkInventory(Trigger.new);}
- Business logic is in QuoteLineTriggerHandler.
- Why: Makes our system easier to maintain tomorrow, if we need new logic (like auto-reserve stock),

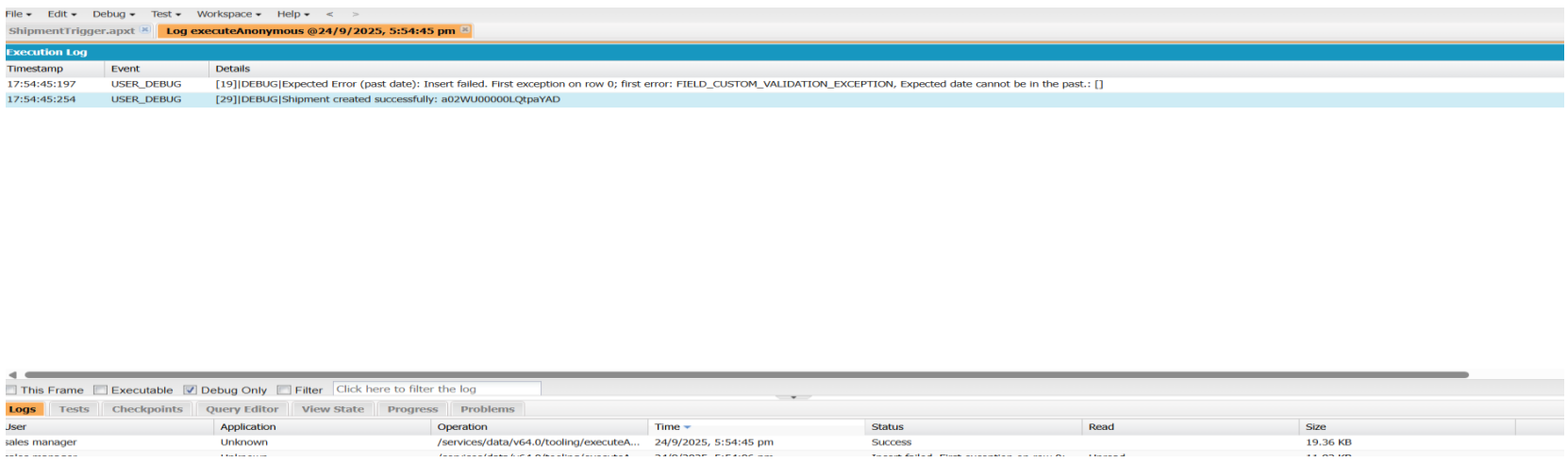
  we just add a new method in the handler.



Anonyamous code window:



Output



◆ SOQL & SOSL

The project needs to query data in real-time.

- Example: When validating Quote Lines, we use SOQL to pull stock:
- [SELECT Id, Name, Stock_Level__c FROM Inventory__c WHERE Id IN :inventoryIds]
- Why: Allows us to compare stock vs. requested quantity.
- SOSL isn't used yet, but could be applied if Sales Reps search across Products + Inventory simultaneously.

◆ Collections: List, Set, Map

Sales Reps add multiple Quote Lines or when syncing large sets of Inventory records from ERP. Apex Collections

help us bulkify our code and prevent hitting Salesforce governor limits.

- List
  - Ordered group of records.
  - Example: List<SBQQ__QuoteLine__c> quoteLines → holds all Quote Lines being validated in a trigger event.
  - Use Case: We iterate through this list to check each Quote Line against available inventory.
- Set
  - Unordered group of unique values.
  - Example: Set<Id> inventoryIds → stores all Inventory IDs linked to Quote Lines.
  - Use Case: Prevents duplicate queries when multiple Quote Lines reference the same Inventory.
- Map
  - Key-Value pairs.

- Example: Map<Id, Inventory__c> inventoryMap → links Inventory ID → Inventory record.
- Use Case: Allows fast lookup of Inventory stock level during Quote Line validation.

Why Collections?
Without them, Salesforce would run one SOQL query per Quote Line. With collections, we process all Quote Lines

in bulk using a single query making the solution scalable, governor-limit safe, and enterprise-ready.

◆ Control Statements

Control statements (if-else, loops, switch) allow us to embed business rules directly into Apex logic. In the project,

they enforce rules that Flow/Validation Rules cannot fully handle.

- If Conditions
  - Example:
  - if (ql.SBQQ__Quantity__c > inv.Stock_Level__c) {
  - ql.addError('Requested quantity exceeds stock');
  - }
  - Business Rule: Prevents overselling of inventory.
- For Loops
  - Example: Iterate through all Quote Lines being inserted/updated in bulk.
- Decision-Making
  - Could extend logic later to handle special cases, e.g.,
    - If product is “Custom Pump” → allow negative stock (since it’s built-to-order).
    - Else → enforce strict stock limits.

Why Control Statements?
They ensure that Salesforce enforces real-world manufacturing rules automatically, reducing errors and manual intervention.

Batch Apex

Large-scale async processing.

Use case: Recalculate stock nightly.

```apex
global class RecalculateStockBatch implements Database.Batchable<SObject> {
    global Database.QueryLocator start(Database.BatchableContext bc) {
        // Select only fields you need
        return Database.getQueryLocator('SELECT Id, Stock_Level__c, Reorder_Point__c FROM Inventory__c');
    }
    global void execute(Database.BatchableContext bc, List<SObject> scope) {
        List<Inventory__c> invs = (List<Inventory__c>) scope;
        for (Inventory__c inv : invs) {
            Decimal stock = inv.Stock_Level__c == null ? 0 : inv.Stock_Level__c;
            Decimal reorder = inv.Reorder_Point__c == null ? 0 : inv.Reorder_Point__c;
            if (stock < reorder) {
                inv.Stock_Level__c = reorder; // simple rule: top up to reorder point
            }
        }
        if (!invs.isEmpty()) update invs;
    }
    global void finish(Database.BatchableContext bc) {
        // Optionally send notification or log stats
    }
}
```

Anonyamous code window:

```apex
// --- Create some test Inventory records ---
List<Inventory__c> invList = new List<Inventory__c>{
    new Inventory__c(Stock_Level__c = 1, Reorder_Point__c = 5),
    new Inventory__c(Stock_Level__c = 10, Reorder_Point__c = 7),
    new Inventory__c(Stock_Level__c = 0, Reorder_Point__c = 3)
};
insert invList;


RecalculateStockBatch batch = new RecalculateStockBatch();
Database.executeBatch(batch, 200);


List<Inventory__c> updatedInv = [SELECT Id, Stock_Level__c, Reorder_Point__c FROM Inventory__c];

for (Inventory__c inv : updatedInv) {
    System.debug('Inventory Id: ' + inv.Id
                 + ' | Stock Level: ' + inv.Stock_Level__c
                 + ' | Reorder Point: ' + inv.Reorder_Point__c);
}
```

Output

File Edit Debug Test Workspace Help

RecalculateStockBatch.apxc | Log executeAnonymous @24/9/2025, 6:06:48 pm

Execution Log

Timestamp	Event	Details
18:06:48:110	USER_DEBUG	[17]|DEBUG|Inventory Id: a00WU00001JM1QwYAL | Stock Level: 100.00 | Reorder Point: 20.00
18:06:48:110	USER_DEBUG	[17]|DEBUG|Inventory Id: a00WU00001KJwKQYA1 | Stock Level: 6.00 | Reorder Point: 2.00
18:06:48:110	USER_DEBUG	[17]|DEBUG|Inventory Id: a00WU00001KLOw9YAH | Stock Level: 1.00 | Reorder Point: 5.00
18:06:48:110	USER_DEBUG	[17]|DEBUG|Inventory Id: a00WU00001KLOwAYAX | Stock Level: 10.00 | Reorder Point: 7.00
18:06:48:111	USER_DEBUG	[17]|DEBUG|Inventory Id: a00WU00001KLOwBYAX | Stock Level: 0.00 | Reorder Point: 3.00

This Frame | Executable | Debug Only | Filter | Click here to filter the log

Logs | Tests | Checkpoints | Query Editor | View State | Progress | Problems

User	Application	Operation	Time	Status	Read	Size
sales manager	Unknown	Batch Apex	24/9/2025, 6:06:50 pm	Success	Unread	3.09 KB
sales manager	Unknown	Batch Apex	24/9/2025, 6:06:49 pm	Success	Unread	3.55 KB
sales manager	Unknown	SerialBatchApexRangeChunkHandler	24/9/2025, 6:06:49 pm	Success	Unread	2.41 KB
sales manager	Unknown	SerialBatchApexRangeChunkHandler	24/9/2025, 6:06:49 pm	Success	Unread	8.66 KB
sales manager	Unknown	/services/data/v64.0/tooling/executeA...	24/9/2025, 6:06:48 pm	Success		11.75 KB
sales manager	Unknown	ApexTestHandler	24/9/2025, 6:05:10 pm	Success	Unread	9.87 MB

- **Queueable Apex**

  Lighter async jobs.

  Use case: Send async notifications for delayed shipments.

File Edit Debug Test Workspace Help

ShipmentNotifier.apxc | Log executeAnonymous @24/9/2025, 6:13:14 pm

Code Coverage: None | API Version: 64 | Go To

```
public class ShipmentNotifier implements Queueable {
    public void execute(QueueableContext qc) {
        List<Shipment__c> delayed = [
            SELECT Id, Name, Expected_Date__c
            FROM Shipment__c
            WHERE Expected_Date__c < :Date.today() AND Status__c != 'Shipped'
        ];
        // For demo, just log; in real org call notification logic or platform event
        for (Shipment__c s : delayed) {
            System.debug('ShipmentNotifier: delayed -> ' + s.Id + ' / ' + s.Name);
        }
    }
}
```

Logs | Tests | Checkpoints | Query Editor | View State | Progress | Problems

User	Application	Operation	Time	Status	Read	Size
sales manager	Unknown	ApexTestHandler	24/9/2025, 6:14:33 pm	Success	Unread	2.07 MB
sales manager	Unknown	QueueableHandler	24/9/2025, 6:13:14 pm	Success		3.22 KB
sales manager	Unknown	/services/data/v64.0/tooling/executeA...	24/9/2025, 6:13:13 pm	Success	Unread	16.78 KB

- **Scheduled Apex**

  Use case: Schedule the stock recalculation.

```
global class NightlyStockJob implements Schedulable {
    global void execute(SchedulableContext sc) {
        // run batch with size 200
        Database.executeBatch(new RecalculateStockBatch(), 200);
    }
}
```

Anonyamous code window:

```
NightlyStockJob job = new NightlyStockJob();
job.execute(null);

System.debug('NightlyStockJob executed manually');


String sch = System.now().addMinutes(1).format('s m H d M ? yyyy');
System.schedule('NightlyStockJob_Test', sch, new NightlyStockJob());
System.debug('--- NightlyStockJob scheduled to run ---');


List<Inventory__c> invList = [SELECT Id, Stock_Level__c, Reorder_Point__c FROM Inventory__c];
System.debug('--- Inventory levels after NightlyStockJob ---');
for(Inventory__c inv : invList){
    System.debug('Inventory Id: ' + inv.Id + ', Stock: ' + inv.Stock_Level__c + ', Reorder Point: ' + i
}
```

Output

Execution Log

Timestamp	Event	Details
18:21:42:094	USER_DEBUG	[5]|DEBUG|--- NightlyStockJob executed manually ---
18:21:42:155	USER_DEBUG	[10]|DEBUG|--- NightlyStockJob scheduled to run ---
18:21:42:164	USER_DEBUG	[15]|DEBUG|--- Inventory levels after NightlyStockJob ---
18:21:42:164	USER_DEBUG	[17]|DEBUG|Inventory Id: a00WU00001JM1QwYAL, Stock: 100.00, Reorder Point: 20.00
18:21:42:164	USER_DEBUG	[17]|DEBUG|Inventory Id: a00WU00001KJwKQYA1, Stock: 6.00, Reorder Point: 2.00
18:21:42:165	USER_DEBUG	[17]|DEBUG|Inventory Id: a00WU00001KLOw9YAH, Stock: 5.00, Reorder Point: 5.00
18:21:42:165	USER_DEBUG	[17]|DEBUG|Inventory Id: a00WU00001KLOwAYAX, Stock: 10.00, Reorder Point: 7.00
18:21:42:165	USER_DEBUG	[17]|DEBUG|Inventory Id: a00WU00001KLOwBYAX, Stock: 3.00, Reorder Point: 3.00

- ## Future Methods

In my Quote-to-Cash Hub:

- When a Quote is Approved, the system must send Order details to an external ERP system (e.g., SAP or Oracle).
- This external call may take a few seconds or even fail sometimes.
- If we ran this synchronously, the Sales Rep would be stuck waiting while saving/approving the Quote.
- With a Future Method, the Quote saves instantly, and ERP sync happens in the background.

◆ Exception Handling

In a manufacturing context, data mismatches are common (e.g., ERP didn't sync stock, user selected invalid product).

Instead of crashing, our Apex logic must gracefully handle errors.

- Try-Catch Blocks
  - Wrap risky code (API calls, DML operations) in try-catch.
  - Example: If ERP API fails, log the error and notify admin instead of blocking all users.
- User-Friendly Errors
  - Example: Instead of showing a generic error → ql.addError('Stock check failed: Inventory not found for this product.')
  - This guides Sales Reps to fix the issue.
- System Logging
  - Exceptions can be logged into a custom object Integration_Error__c.
  - Helps Admins track recurring issues with ERP sync or stock mismatches.

◆ Test Classes

To make project deployment-ready, I create Apex test classes.

- Example: QuoteLineTriggerHandlerTest.cls
  - Creates fake Inventory + Quote + Quote Line.
  - Tries inserting with too high quantity → expects failure.

- Tries with correct quantity → expects success.

- Why: Ensures our automation works as intended and proves 75%+ test coverage (required for deployment).

File Edit Debug Test Workspace Help < >

TestInventoryHelper.apxc

Code Coverage: None API Version: 64 Run Test Go To

```
2  private class TestInventoryHelper {
3      @isTest static void testReduceInventory() {
4          Inventory__c inv = new Inventory__c(
5              Stock_Level__c = 10,
6              Reorder_Point__c = 2
7              // Add any other required writable fields here
8          );
9          insert inv;
10
11         Map<Id, Integer> qtyMap = new Map<Id, Integer>{ inv.Id => 4 };
12
13         Test.startTest();
14             InventoryHelper.reduceInventory(qtyMap);
15         Test.stopTest();
16
17         Inventory__c updated = [SELECT Stock_Level__c FROM Inventory__c WHERE Id = :inv.Id];
18         System.assertEquals(6, Integer.valueOf(updated.Stock_Level__c));
19     }
20 }
```

Logs | Tests | Checkpoints | Query Editor | View State | Progress | Problems

Status	Test Run	Enqueued Time	Duration	Failures	Total
	707WU00000dH7tZ	Wed Sep 24 2025 17:03:58 GM...		12	53
	707WU00000dHww5	Wed Sep 24 2025 18:39:56 GM...		0	18

Overall Code Coverage

Class	Percent	Lines
Overall	**52%**	
InventoryHelper	100%	12/12
NightlyStockJob	0%	0/2
QuoteLineTrigger	100%	2/2
QuoteLineTriggerHandler	56%	9/16
QuoteTrigger	100%	2/2

File Edit Debug Test Workspace Help < >

TestShipmentTrigger.apxc

Code Coverage: None API Version: 64 Run Test Go T

```
1  @isTest
2  private class TestShipmentTrigger {
3      @isTest static void testShipmentDateValidation() {
4          Shipment__c s = new Shipment__c(
5              Expected_Date__c = Date.today().addDays(-1),
6              Status__c = 'In Transit'
7              // Add any other required writable fields here
8          );
9
10         Test.startTest();
11         try {
12             insert s;
13             System.assert(false, 'Expected insert to fail due to validation');
14         } catch (DmlException e) {
15             // expected
16         }
17         Test.stopTest();
18     }
19 }
```

Logs | Tests | Checkpoints | Query Editor | View State | Progress | Problems

Status	Test Run	Enqueued Time	Duration	Failures	Total
	707WU00000dH7tZ	Wed Sep 24 2025 17:03:58 GM...		12	53
	707WU00000dHww5	Wed Sep 24 2025 18:39:56 GM...		0	1
	707WU00000dI4Mt	Wed Sep 24 2025 18:41:01 GM...		5	17

Overall Code Coverage

Class	Percent	Lines
InventoryHelper	100%	12/12
NightlyStockJob	0%	0/2
QuoteLineTrigger	100%	2/2
QuoteLineTriggerHandler	56%	9/16
QuoteTrigger	100%	2/2
QuoteTriggerHandler	40%	8/20

◆ Asynchronous Processing

Not all processes should run instantly. In manufacturing, some tasks are heavy (ERP sync, shipment updates) or scheduled

(daily stock checks). Asynchronous Apex lets us run these in the background without slowing down the user experience.

- Future Methods / Queueable Apex
  - Use Case: After a Quote is Approved → send Order details to ERP (SAP/Oracle).
  - Runs in background → user doesn't wait for API calls.
- Batch Apex

    - Use Case: Nightly job to update Inventory__c with ERP stock levels.
    - Processes thousands of records in chunks → safe for large data volumes.
- Scheduled Apex
    - Use Case: Daily at 6 AM → check which shipments are delayed and trigger alerts.
    - This supplements our Flows with code-based scheduling for complex scenarios.

Why Asynchronous?

- Keeps the Salesforce UI fast for Sales Reps and Agents.
- Ensures large-scale ERP sync and shipment monitoring don't hit governor limits.
- Provides enterprise-level reliability for background jobs.