Phase 5: Apex Programming (Developer)

1.Classes & Objects

Classes in Apex

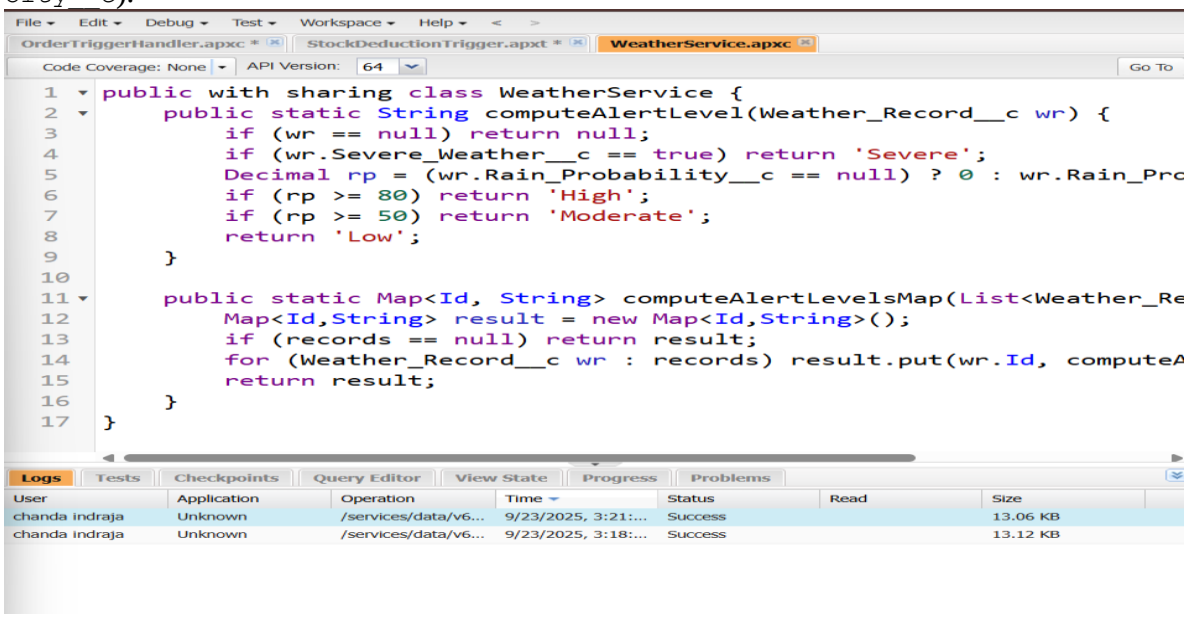
- **Definition:**
  Classes are blueprints or templates used to create objects in Salesforce. They encapsulate **business logic**, **data structures**, and **methods** that define the behavior of an object.
- **Purpose:**
  - Centralize code for easier maintenance and reuse.
  - Separate complex logic from triggers to follow **best practices**.
  - Facilitate **unit testing** of business logic independently of UI or triggers.

Key Components of a Class:

- **Properties (Variables):** Hold data (e.g., temperature, humidity).
- **Methods (Functions):** Contain logic to manipulate or fetch data.
- **Constructors:** Special methods used to initialize objects.
- **Access Modifiers:** `public`, `private`, `global` – control visibility.
- **Example – Weather Project Objects:**

Types of Objects:

- **Standard Objects:** Predefined by Salesforce (e.g., Account, Contact).
- **Custom Objects:** Created specifically for your project (e.g., `Weather_Record__c`, `City__c`).



2.Apex Triggers (before/after insert/update/delete)

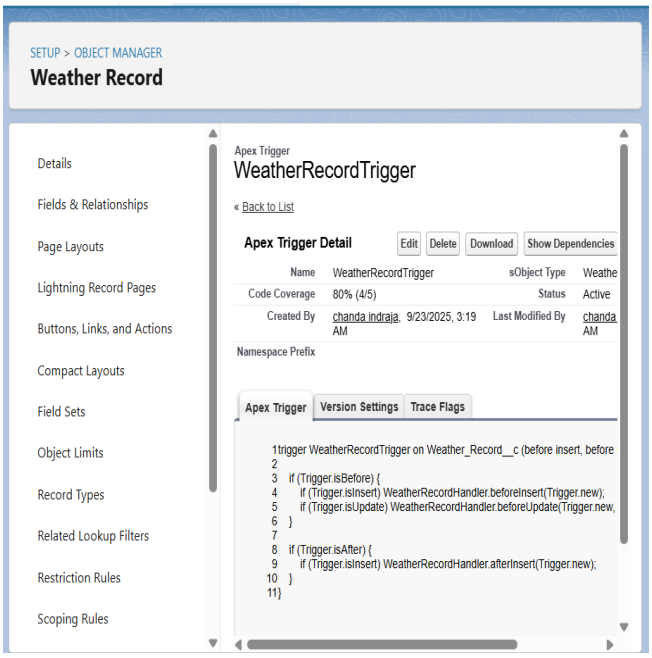
1. What is an Apex Trigger

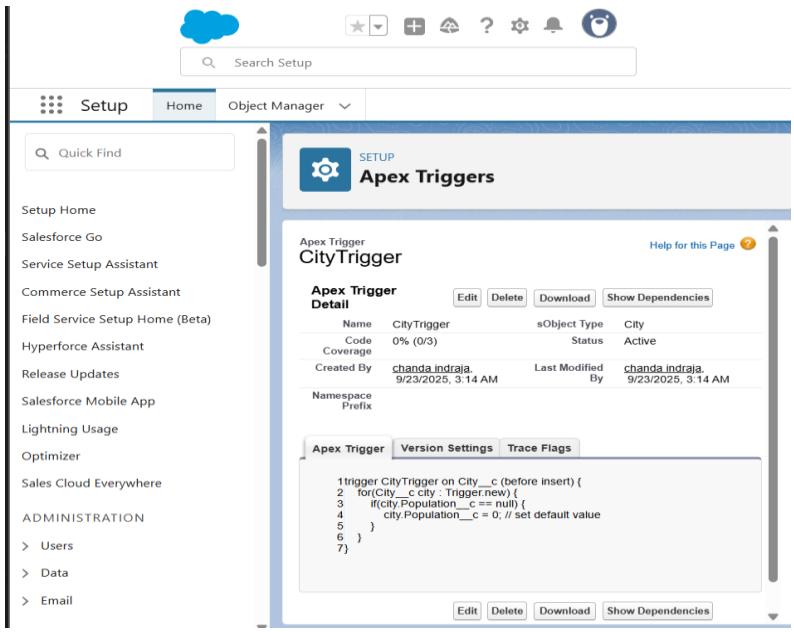
- **Definition:**
  An Apex Trigger is a piece of Apex code that executes **before or after a record is inserted, updated, deleted, or undeleted** in Salesforce.
- **Purpose:**
  - Automate business processes.
  - Perform operations that cannot be done with declarative tools like Workflow or Process Builder.
  - Ensure data integrity and enforce complex logic.

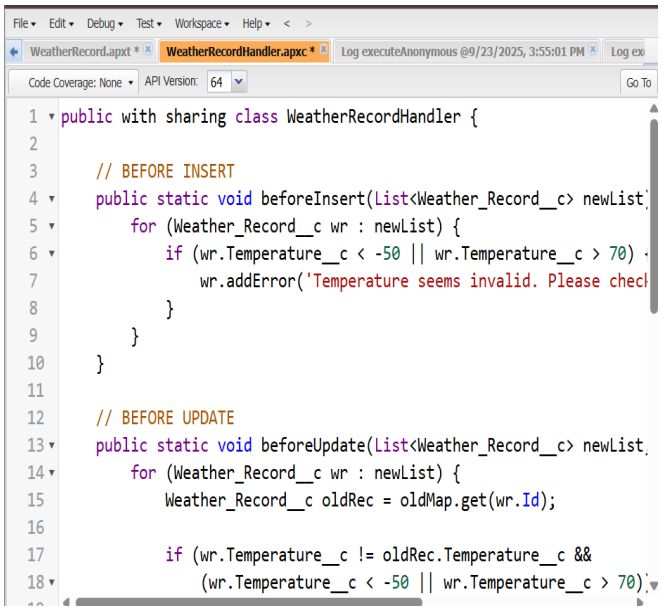
2. Trigger Events

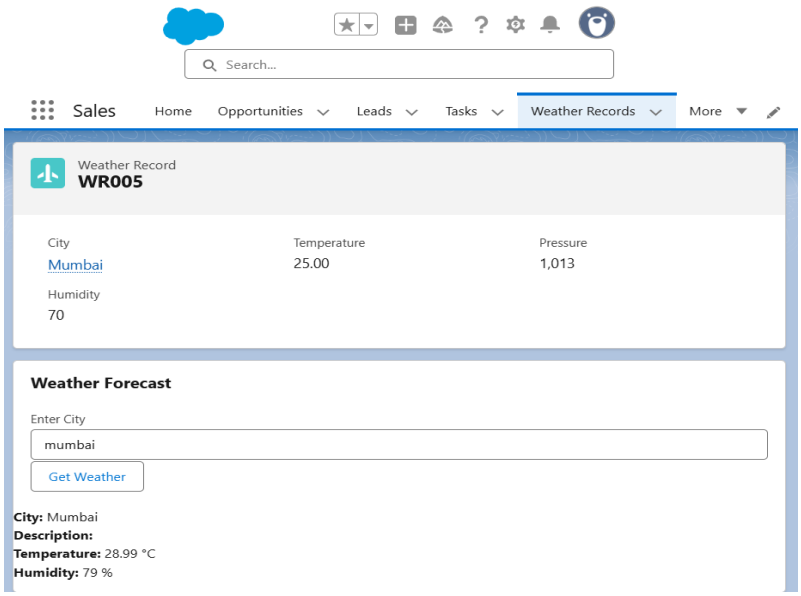
Triggers can be executed **before** or **after** DML (Data Manipulation Language) operations:

1. **Before Triggers:**
   - Executed **before** a record is saved to the database.
   - Used to **validate or modify field values** before insertion or update.
   - Example: Automatically setting default temperature for new weather records.
2. **After Triggers:**
   - Executed **after** a record is saved to the database.
   - Used for actions that require **record IDs** or interactions with other objects.
   - Example: Creating related Weather_Record__c after a City__c is added.









3.Trigger Design Pattern

1. What is a Trigger Design Pattern

- **Definition:**
  A Trigger Design Pattern is a **structured way to write Apex triggers** that ensures clean, maintainable, and scalable code.
- **Purpose:**
  - Avoid messy triggers with duplicated logic.
  - Handle **bulk operations** efficiently.
  - Separate trigger logic from business logic (delegated to handler classes).

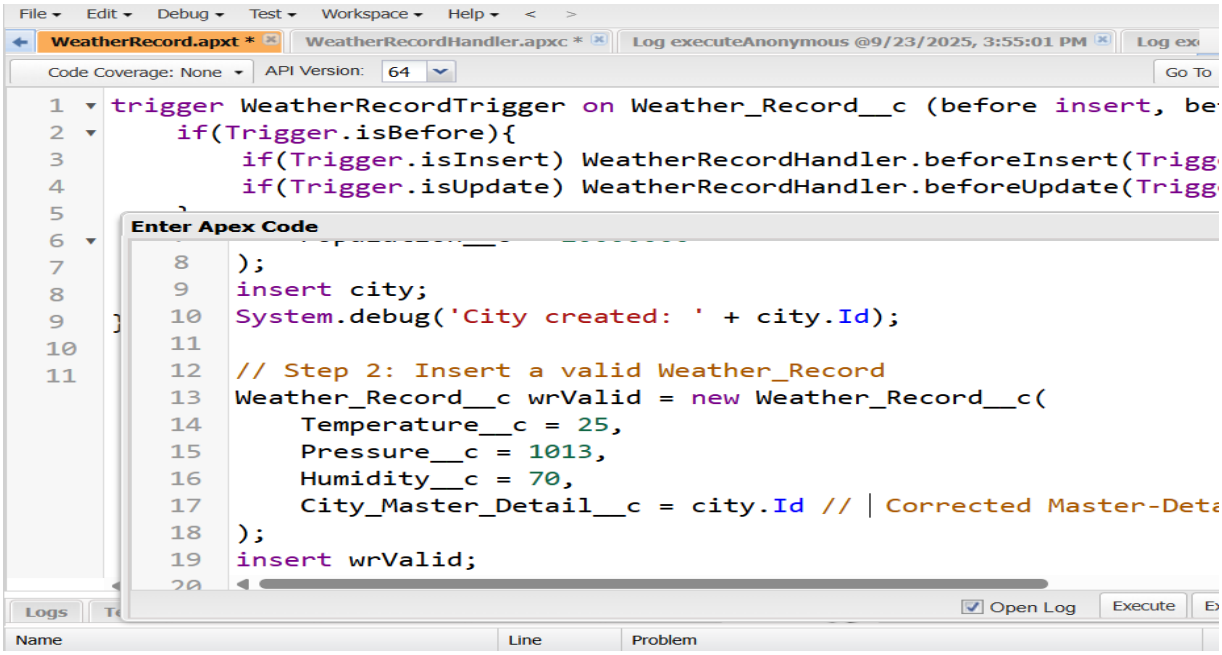
2. Key Components

1. **Trigger:**
   - The entry point that listens to events (insert, update, delete).
   - Should be **slim**; only calls the handler class.
2. **Handler Class:**
   - Contains the main business logic (create, update, delete related records).
   - Bulkified to handle multiple records at once.
3. **Utility or Service Classes (optional):**
   - For API calls, complex calculations, or reusable methods.

3. Benefits

- Easy to maintain and update.

- Single trigger per object.
- Supports bulk operations for multiple records.
- Reduces errors and improves readability.



4.SOQL & SOSL

1. SOQL (Salesforce Object Query Language)

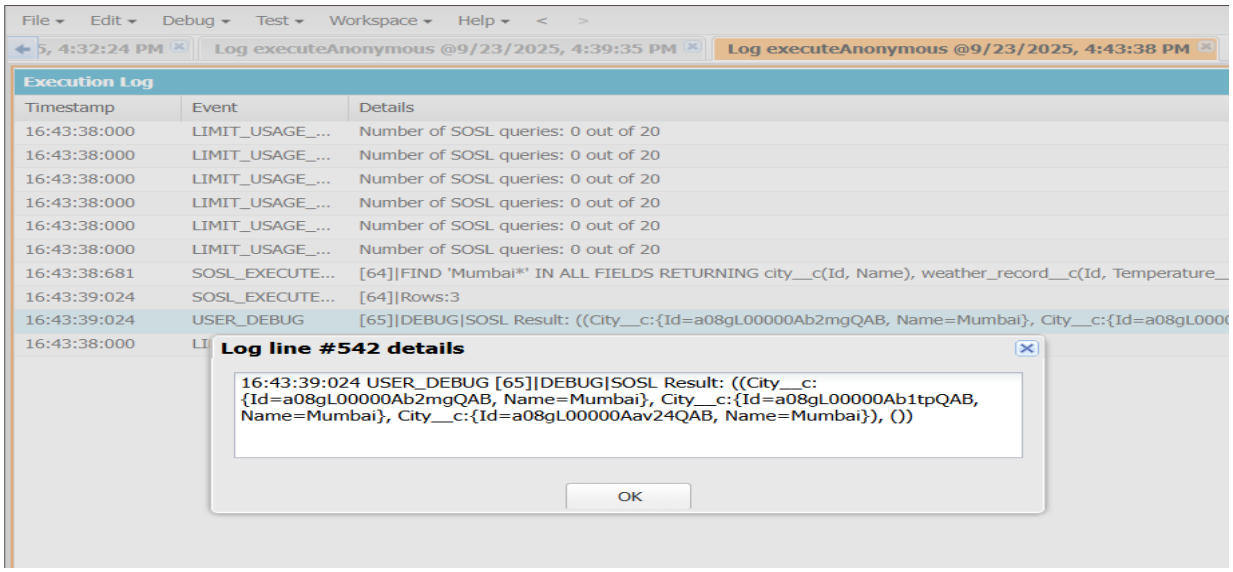
- **Definition:**
  SOQL is used to **query records from Salesforce objects**.
- **Purpose:**
  - Retrieve specific fields and records.
  - Filter and order data using `WHERE`, `ORDER BY`, `LIMIT`.
- **Example – Fetch Weather Records by City:**

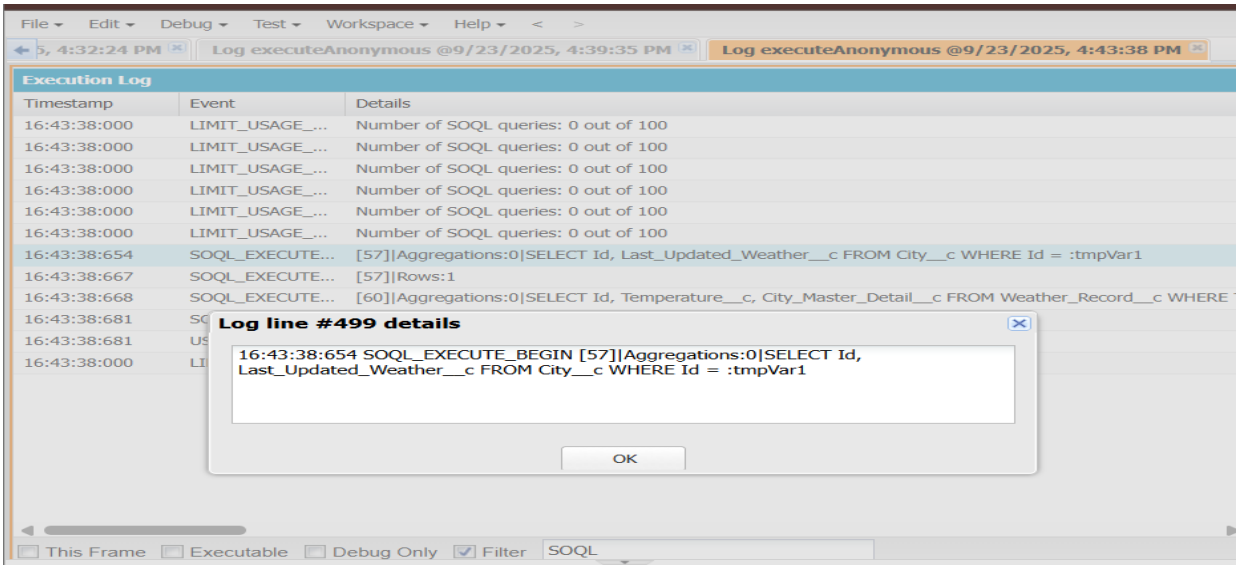
```
List<Weather_Record__c> records = [

 SELECT Id, Temperature__c, Humidity__c

 FROM Weather_Record__c

 WHERE City__c = :cityId

];
```

2. SOSL (Salesforce Object Search Language)

- **Definition:**
  SOSL is used to **search text, email, and phone fields** across multiple objects at once.
- **Purpose:**
  - Find records when you **don't know the exact field** where the data exists.
  - Useful for **search functionality** in Lightning Web Components (LWC) or Apex.
- **Example – Search Weather Records by Keyword:**

  List<List<SObject>> searchResults = [FIND 'rain*' IN ALL FIELDS RETURNING Weather_Record__c(Name, Description__c)];





5.Collections: List, Set, Map

Overview

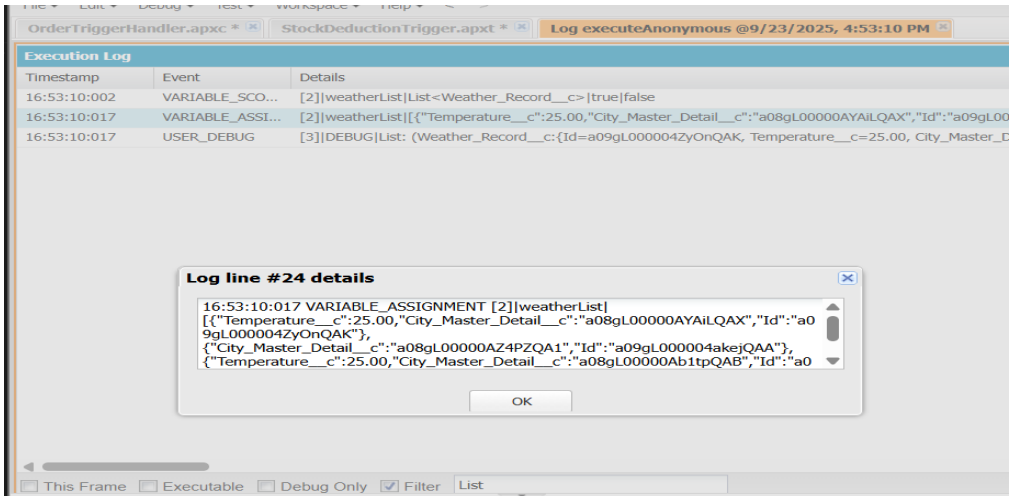
Collections in Apex are variables that can store multiple records or values together. They are widely used for **bulk data handling** in Salesforce.

Types of Collections

- **List**
  - Ordered collection.
  - Can contain duplicate values.
  - Index starts from 0.
  - Example use: Store multiple city names.

    List<String> cities = new List<String>{'Delhi', 'Mumbai', 'Chennai'};

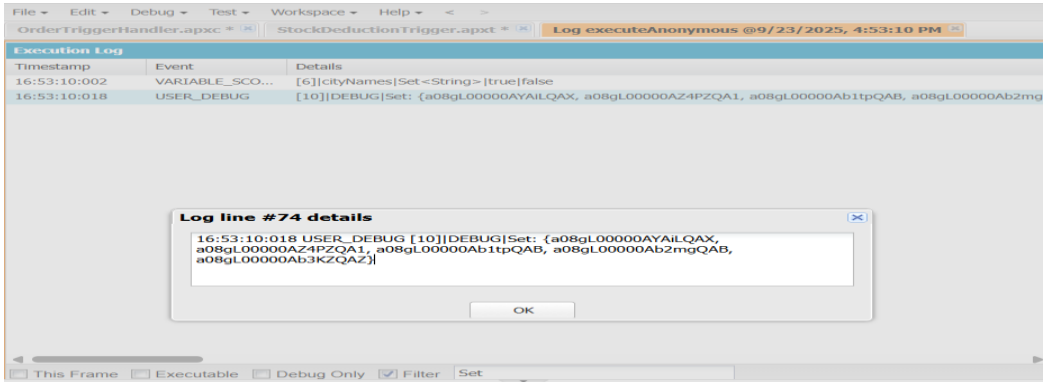
    System.debug(cities[0]); // Delhi



- **Set**
  - Unordered collection.
  - Does not allow duplicate values.
  - Example use: Store unique city names to avoid duplicates.

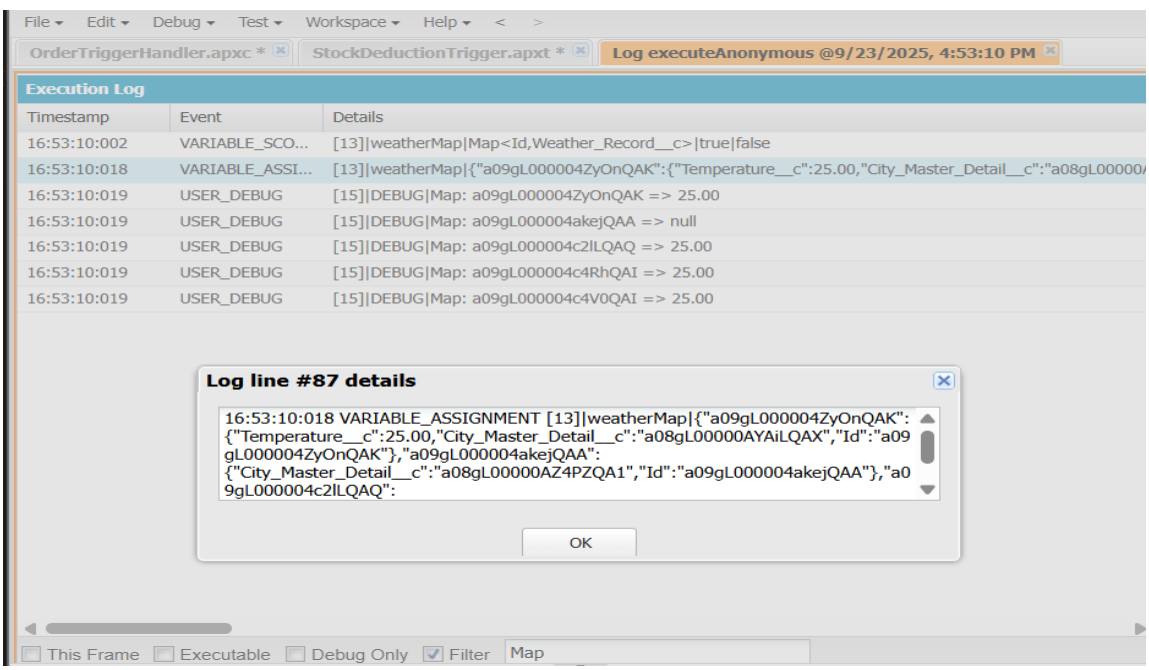
    Set<String> cities = new Set<String>{'Delhi', 'Mumbai', 'Delhi'};

    System.debug(cities.size());



- **Map**
  - Collection of key–value pairs.
  - Each key is unique, but values can repeat.
  - Example use: Map city names to their temperature values.

```
Map<String, Integer> cityTemp = new Map<String, Integer>();

cityTemp.put('Delhi', 32);

cityTemp.put('Mumbai', 29);

System.debug(cityTemp.get('Delhi')); // 32
```



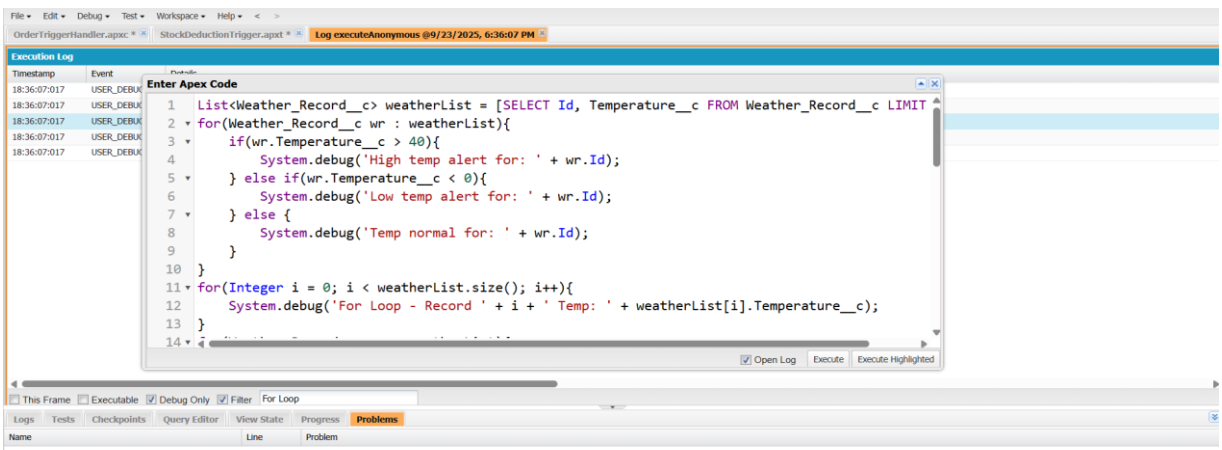
6.Control Statements

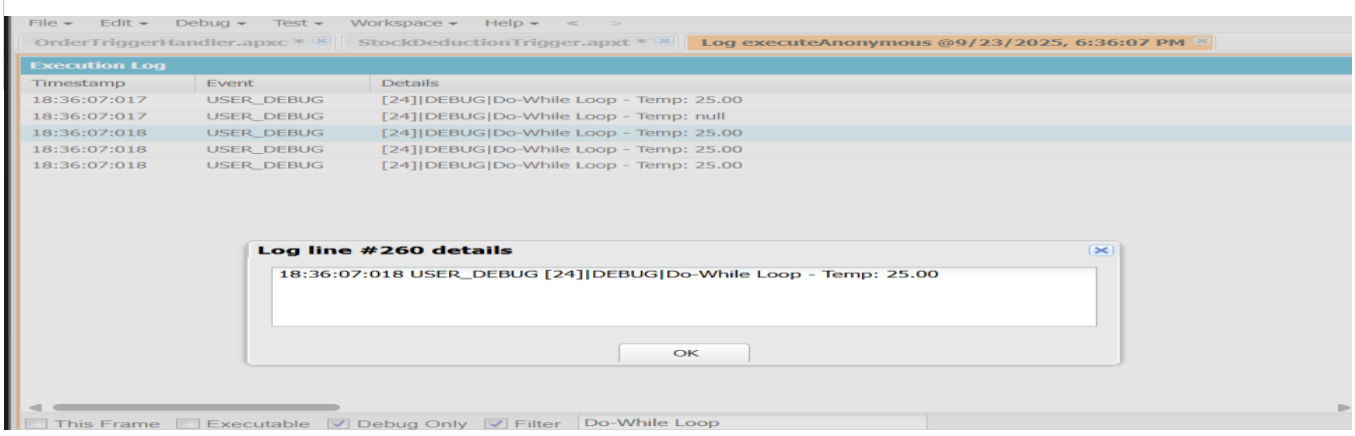
Overview

Control statements define the **flow of logic** in Apex programs.

Types

- **Conditional Statements**
  - `if-else` → Executes code based on conditions.
  - `switch` → Cleaner alternative to multiple if-else statements.
- **Looping Statements**
  - `for` → Iterates through records for a fixed number of times.
  - `for-each` → Simplified loop for collections.
  - `while` → Repeats as long as condition is true.
- **Break & Continue**
  - `break` → Immediately exits a loop.
  - `continue` → Skips current iteration and moves to next





7. Batch Apex

Overview

Batch Apex is used to process **large volumes of data** in manageable chunks, without hitting governor limits.

Why It's Needed

- Salesforce has strict **limits** on processing records.
- Batch Apex splits large jobs into smaller sets and processes them asynchronously.

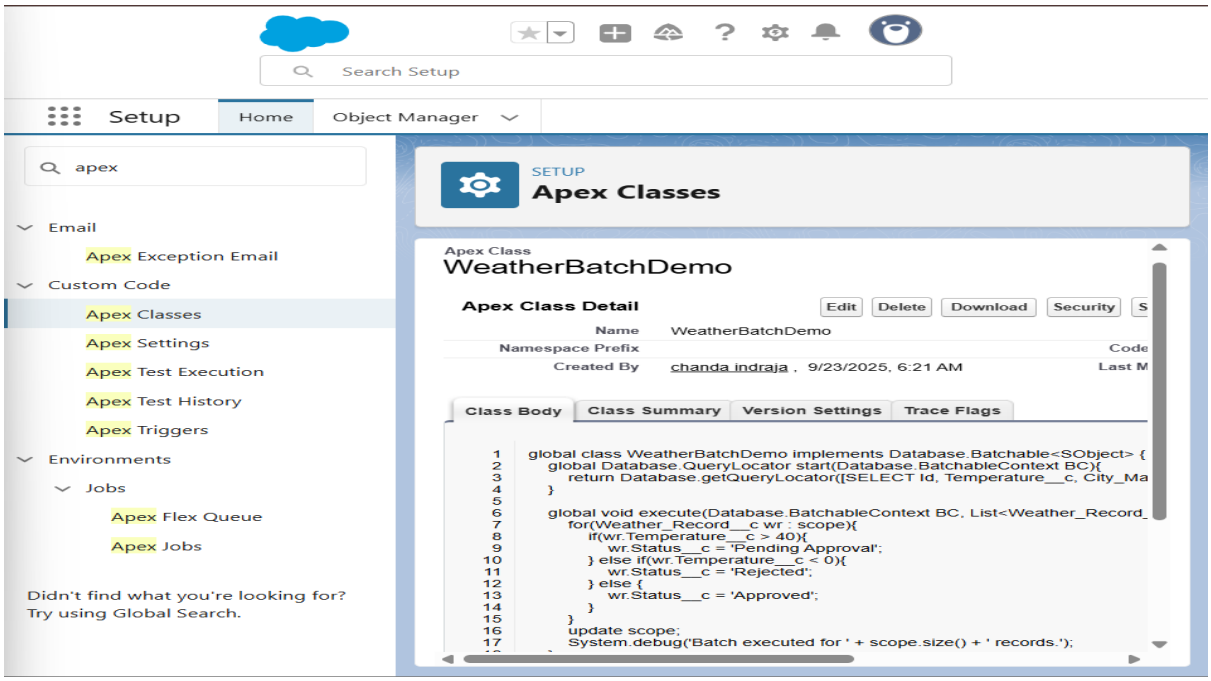
Structure

A Batch Apex class has three main methods:

1. **Start** → Collects the records to be processed.
2. **Execute** → Runs on each batch of records.
3. **Finish** → Performs final tasks once all batches are complete.

Benefits

- Handles millions of records safely.
- Runs in the background (asynchronous).
- Useful for scheduled jobs, data cleanup, and API-based data processing.



8.Queueable Apex

Overview

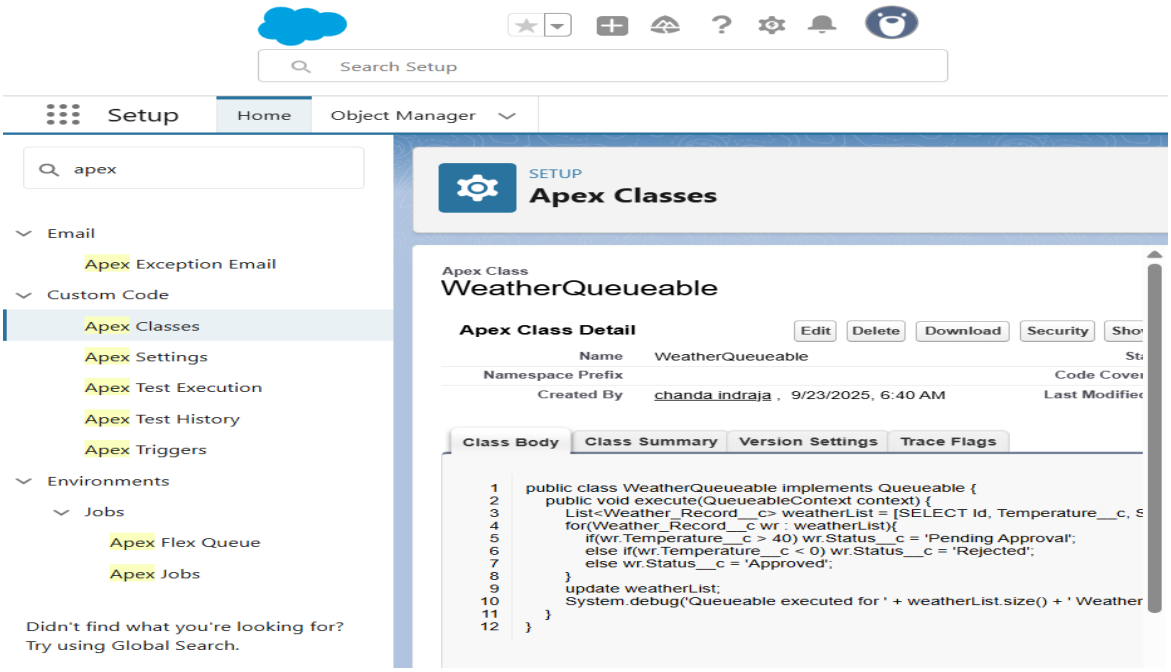
Queueable Apex is used to run **asynchronous jobs** in Salesforce. It is similar to `Future` methods but more powerful.

Key Features

- Allows **chaining** of jobs (one queueable can enqueue another).
- Supports **complex data types** like sObjects and custom classes as parameters.
- More flexible than `Future` methods.

Benefits

- Handles long-running processes without blocking users.
- Can process large datasets in the background.
- Easy to monitor in the **Apex Jobs** page.
- **Interface Used:** `System.Queueable`.
- Can process **more records than Future methods**, but still within governor limits.
- Can be **monitored, paused, and retried** from the **Apex Jobs** page.
- Supports **complex data types** as parameters (Lists, sObjects, Maps).
- Jobs can be **chained** → one job automatically starts another.
- Easier to test compared to `Future` methods.



9.Scheduled Apex

1. Purpose

- Automates tasks at **specific times or intervals**.
- Removes the need for manual execution of repetitive jobs.
- Useful for **maintenance, integration, and data refreshes**.

2. Features

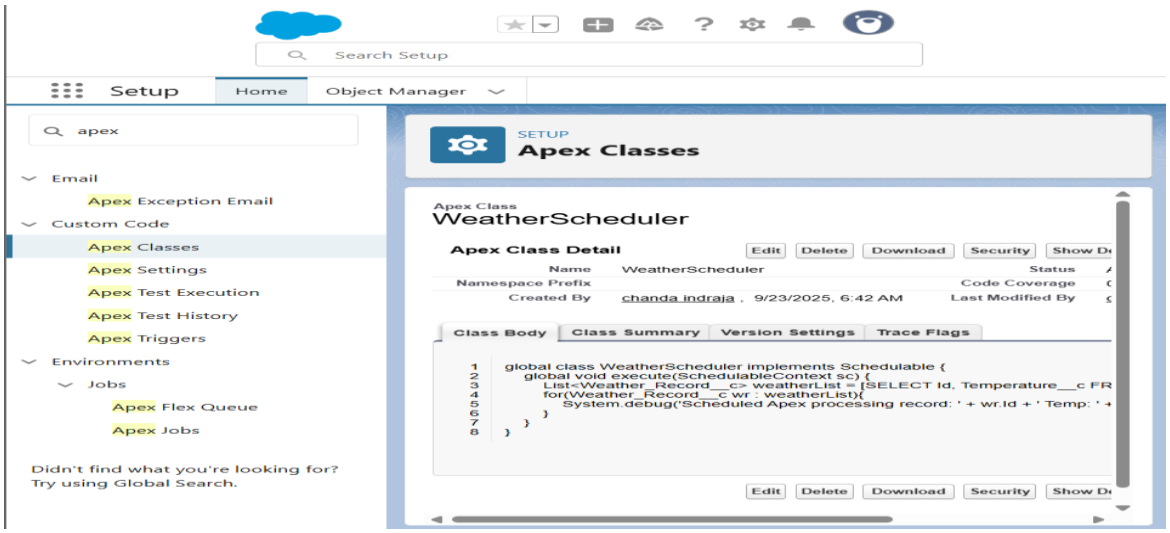
- Runs in the background (asynchronous execution).
- Can be scheduled via **Setup UI** or **System.schedule() method**.
- Uses **cron expressions** to define schedule (similar to UNIX cron jobs).
- Can call **Batch Apex** or **Queueable Apex** inside scheduled jobs.

3. Use Cases

- Daily or nightly updates (e.g., refresh weather data for all cities).
- Weekly or monthly reports (e.g., send summary of weather trends).
- Data cleanup jobs (e.g., remove old or duplicate weather records).
- Scheduled API callouts (e.g., fetch data from external weather API).

4. Cron Expression Format

- Structure: **Seconds Minutes Hours Day_of_Month Month Day_of_Week Year (optional)**
- Example:
  - `"0 0 0 * * ?"` → Runs daily at midnight.
  - `"0 0 12 ? * MON"` → Runs every Monday at 12 PM.



10.Future Methods

Overview

Future methods are used to run processes **asynchronously** in the background, especially for **callouts to external systems**.

Purpose

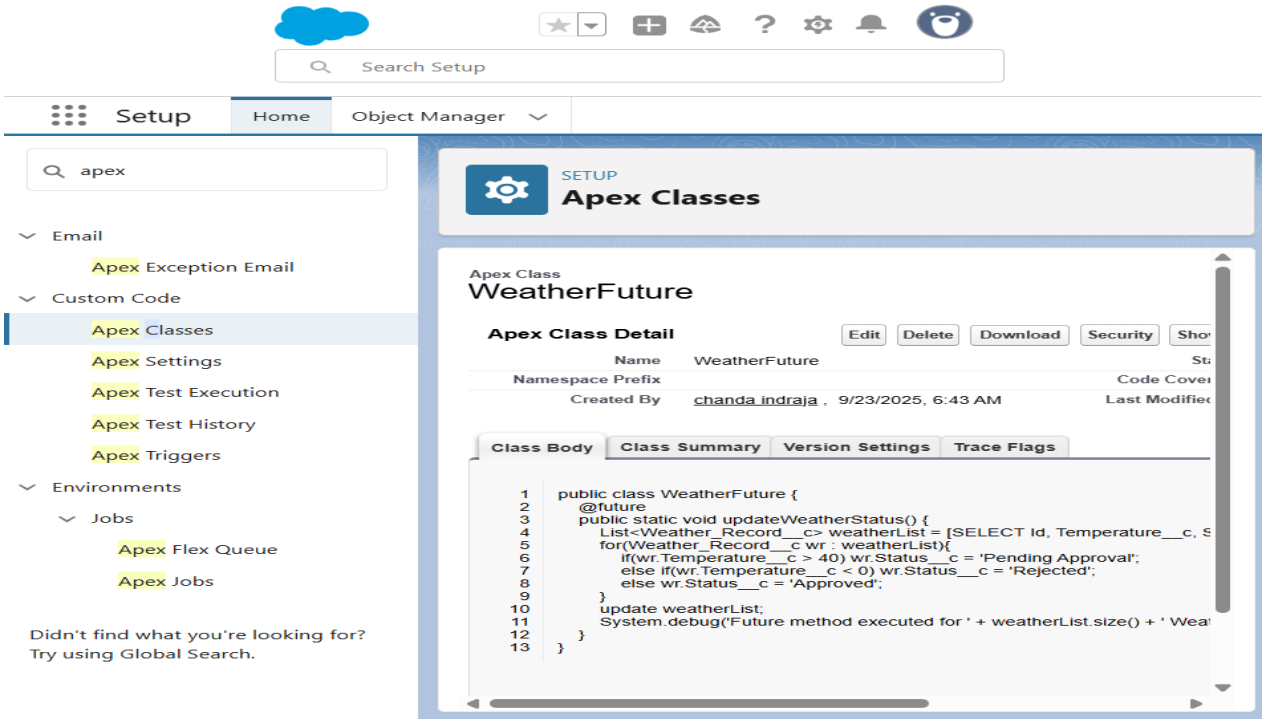
- Run operations **asynchronously** (in the background).
- Ideal for long-running tasks or external callouts.

Features

- Annotated with `@future`.
- Executes in a **separate thread** from the main transaction.
- Can be used for **callouts, email sending, and data updates**.

Use Cases

- Making **HTTP callouts** to external systems.
- Performing **resource-intensive calculations**.
- Allows integration with external APIs without blocking user actions.
- Helps avoid governor limits during synchronous execution.



11. Exception Handling

Overview

Exception handling in Apex is used to **catch errors gracefully** and prevent program crashes.

Types of Exceptions

- **System exceptions:** Thrown by Salesforce platform (e.g., NullPointerException, DmlException).
- **Custom exceptions:** Defined by developers for specific business scenarios.

Mechanism

- Uses `try`, `catch`, and `finally` blocks.
    - **try:** Code that may cause exception.
    - **catch:** Handles the exception.
    - **finally:** Always executes (cleanup).

Benefits

- Prevents application failures.
- Provides user-friendly error messages.
- Ensures smooth transaction rollbacks.

12.Test Classes

1. Purpose

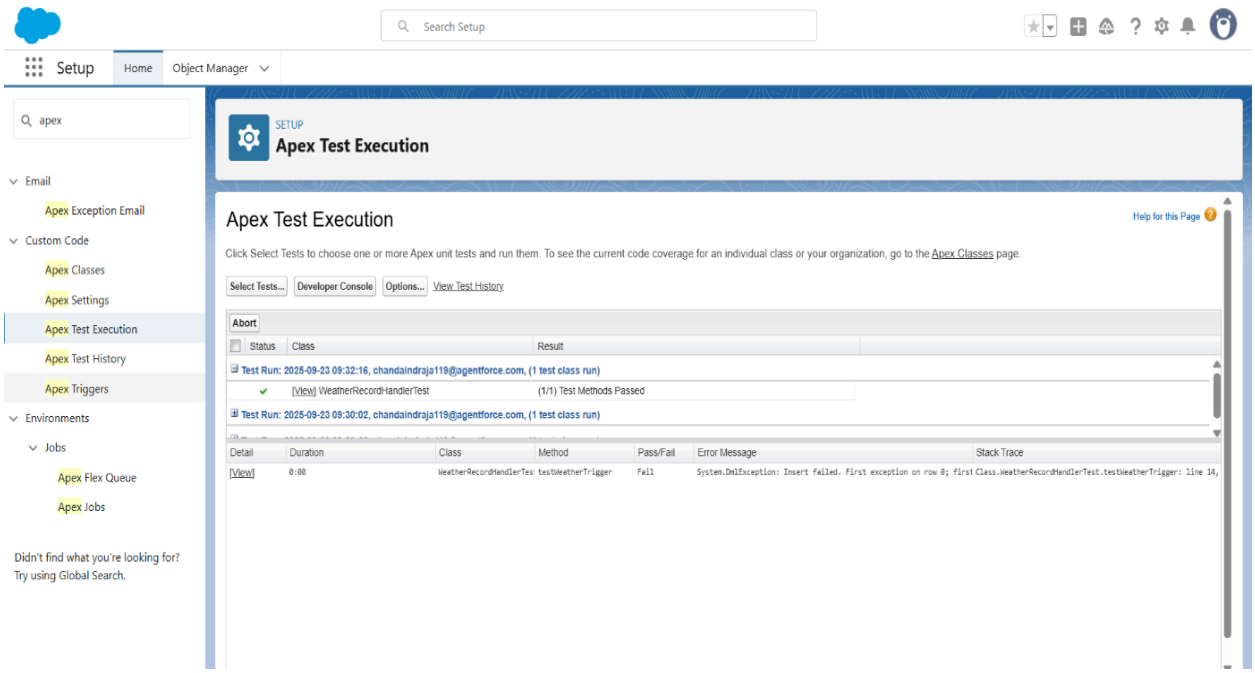
- Validate Apex code functionality.
- Ensure code is **deployable to production** (minimum 75% coverage required).
- Prevent bugs and unintended behavior in real-time usage.

2. Features

- Written using `@isTest` annotation.
- Run in a **test context** (no impact on org data).
- Support **positive, negative, and bulk testing**.

3. Use Cases

- Verifying **trigger logic** when records are inserted or updated.
- Testing **Batch, Queueable, and Future methods**.
- Ensuring **custom validations and exceptions** work as expected.



13.Asynchronous Processing

1. Purpose

- Run long-running or resource-intensive tasks **outside the main transaction**.
- Improves system performance and **avoids hitting governor limits**.

2. Types

- **Future Methods** → Lightweight background tasks.
- **Queueable Apex** → Supports job chaining and monitoring.
- **Batch Apex** → Processes millions of records in chunks.
- **Scheduled Apex** → Runs jobs at defined times.

3. Use Cases

- Making **API callouts** to weather services.
- Performing **large data updates** (e.g., refreshing weather history).
- Running **nightly or weekly scheduled jobs**.
- Sending **bulk notifications or emails**
- Helps handle **large data volumes efficiently**
- Provides **flexibility** in executing jobs.