Trigger Framework

This framework provides a **structured approach** for managing triggers in Salesforce, ensuring **modularity, reusability, and maintainability**. It is designed to work with **any Salesforce object**, making it easy for developers to follow best practices.

Key Components

Trigger (CustomObjectTrigger) → Entry Point

Controls execution based on global settings and invokes the Trigger Handler.

```apex
trigger CustomObjectTrigger on CustomObject__c (before insert, after insert, before update, after update) {
    if (!OrgWideSwitch__c.getInstance().CustomObjectTriggersDeactivated__c &&
        !TriggerFlags.stopCustomObjectTriggerExec) {

        // Call the handler
        CustomObjectTriggerHandler.customRecordTypeHandler();
    }
}
```

✓ Prevents execution if triggers are disabled
✓ Calls the Trigger Handler to process records

Trigger Handler (CustomObjectTriggerHandler) → Categorizes Records

Sorts records based on Record Type and routes them to the correct handler.

```apex
public class CustomObjectTriggerHandler {
    public static List<CustomObject__c> typeAList = new List<CustomObject__c>();
    public static List<CustomObject__c> typeBList = new List<CustomObject__c>();

    public static void customRecordTypeHandler() {
        List<CustomObject__c> recordListNew = (List<CustomObject__c>) Trigger.new;

        for (CustomObject__c record : recordListNew) {
            if (record.RecordTypeId == 'TypeA_RecordTypeId') {
                typeAList.add(record);
            } else if (record.RecordTypeId == 'TypeB_RecordTypeId') {
                typeBList.add(record);
            }
        }

        if (!typeAList.isEmpty()) {
            TriggerHandlerDispatcher.run(new TypeA_CustomObjectHandler(), Trigger.operationType, typeAList);
        }

        if (!typeBList.isEmpty()) {
            TriggerHandlerDispatcher.run(new TypeB_CustomObjectHandler(), Trigger.operationType, typeBList);
        }
    }
}
```

✓ **Categorizes records based on RecordTypeId**
✓ **Calls the Dispatcher to execute the correct handler**

Dispatcher (TriggerHandlerDispatcher) → Routes Execution

Determines the Trigger Event and calls the correct handler method.

```
public class TriggerHandlerDispatcher {
    public static void run(TriggerHandlerInterface handler, System.TriggerOperation triggerEvent, List<sObject> listNewOrOld) {
        switch on triggerEvent {
            when BEFORE_INSERT {
                handler.handleBeforeInsert(listNewOrOld);
            } when AFTER_INSERT {
                handler.handleAfterInsert(listNewOrOld);
            } when BEFORE_UPDATE {
                handler.handleBeforeUpdate(listNewOrOld, Trigger.oldMap);
            } when AFTER_UPDATE {
                handler.handleAfterUpdate(listNewOrOld, Trigger.oldMap);
            }
        }
    }
}
```

✓ **Routes execution based on trigger context**
✓ **Ensures only relevant logic runs**

Interface (TriggerHandlerInterface) → Enforces Standard Methods

Defines a contract for all handlers, ensuring consistency.

```
public interface TriggerHandlerInterface {
    void handleBeforeInsert(List<sObject> listNew);
    void handleAfterInsert(List<sObject> listNew);
    void handleBeforeUpdate(List<sObject> listNew, Map<Id, sObject> mapOld);
    void handleAfterUpdate(List<sObject> listNew, Map<Id, sObject> mapOld);
}
```

✓ **Ensures all handlers implement standard methods**
✓ **Improves maintainability and readability**

Handlers (TypeA_CustomObjectHandler) → Executes Logic

Processes records for a specific Record Type and calls utility methods.

```apex
public class TypeA_CustomObjectHandler implements TriggerHandlerInterface {
    public void handleBeforeInsert(List<sObject> listNew) {
        List<CustomObject__c> recordList = (List<CustomObject__c>) listNew;
        CustomObjectUtility.assignRecordOwner(recordList);
    }

    public void handleAfterInsert(List<sObject> listNew) {
        List<CustomObject__c> recordList = (List<CustomObject__c>) listNew;
        CustomObjectUtility.updateRecordStatus(recordList);
    }
}
```

✓ Implements TriggerHandlerInterface
✓ Executes business logic for a specific record type

Utility Classes (CustomObjectUtility) → Business Logic

Handles reusable business logic for Custom Objects.

```apex
public class CustomObjectUtility {
    public static void assignRecordOwner(List<CustomObject__c> recordList) {
        for (CustomObject__c record : recordList) {
            record.OwnerId = '005XXXXXXXXXXX'; // Example Owner Assignment
        }
        update recordList;
    }
}
```

✓ Encapsulates business logic separately from trigger execution
✓ Enhances code reusability and maintainability

How the Framework Works (Execution Flow)

Trigger (CustomObjectTrigger) fires when a record is inserted, updated, etc.
Trigger Handler (CustomObjectTriggerHandler) categorizes records based on Record Type.
Dispatcher (TriggerHandlerDispatcher) routes execution to the correct **Handler**.
Handlers (TypeA_CustomObjectHandler, etc.) process records and call **utility methods**.
Utility Classes (CustomObjectUtility) execute the actual business logic.

Benefits of This Framework

✓ **Scalable & Modular** – Easily add new logic without modifying triggers.
✓ **Performance Optimized** – Only relevant logic executes for each trigger event.
✓ **Reusable Utility Classes** – Avoids redundant code by centralizing business logic.
✓ **Maintains Separation of Concerns** – Each component has a distinct role, making debugging easier.