- **VehicleOrderTriggerHandler**

```
public class VehicleOrderTriggerHandler {

    public static void handleTrigger(List<Vehicle_Order__c> newOrders, Map<Id, Vehicle_Order__c>
oldOrders, Boolean isBefore, Boolean isAfter, Boolean isInsert, Boolean isUpdate) {

        if (isBefore) {

            if (isInsert || isUpdate) {

                preventOrderIfOutOfStock(newOrders);

            }

        }




        if (isAfter) {

            if (isInsert || isUpdate) {

                updateStockOnOrderPlacement(newOrders);

            }

        }
```

```
    }

    // Method to prevent orders when the vehicle is out of stock

    private static void preventOrderIfOutOfStock(List<Vehicle_Order__c> orders) {

        Set<Id> vehicleIds = new Set<Id>();

        for (Vehicle_Order__c order : orders) {

            if (order.Vehicle__c != null) {

                vehicleIds.add(order.Vehicle__c);

            }

        }

        if (!vehicleIds.isEmpty()) {

            Map<Id, Vehicle__c> vehicleStockMap = new Map<Id, Vehicle__c>();

            for (Vehicle__c vehicle : [SELECT Id, Stock_Quantity__c FROM Vehicle__c WHERE Id IN
:vehicleIds]) {

                vehicleStockMap.put(vehicle.Id, vehicle);

            }
```

```
        for (Vehicle_Order__c order : orders) {
            if (vehicleStockMap.containsKey(order.Vehicle__c)) {
                Vehicle__c vehicle = vehicleStockMap.get(order.Vehicle__c);
                if (vehicle.Stock_Quantity__c <= 0) {
                    order.addError('This vehicle is out of stock. Order cannot be placed.');
                }
            }
        }
    }
}
// Method to update vehicle stock when an order is placed
private static void updateStockOnOrderPlacement(List<Vehicle_Order__c> orders) {
    Set<Id> vehicleIds = new Set<Id>();
    for (Vehicle_Order__c order : orders) {
        if (order.Vehicle__c != null && order.Status__c == 'Confirmed') {
```

```
            vehicleIds.add(order.Vehicle__c);

        }

    }

    if (!vehicleIds.isEmpty()) {

        Map<Id, Vehicle__c> vehicleStockMap = new Map<Id, Vehicle__c>();

        for (Vehicle__c vehicle : [SELECT Id, Stock_Quantity__c FROM Vehicle__c WHERE Id IN :vehicleIds]) {

            vehicleStockMap.put(vehicle.Id, vehicle);

        }

        List<Vehicle__c> vehiclesToUpdate = new List<Vehicle__c>();

        for (Vehicle_Order__c order : orders) {

            if (vehicleStockMap.containsKey(order.Vehicle__c)) {

                Vehicle__c vehicle = vehicleStockMap.get(order.Vehicle__c);

                if (vehicle.Stock_Quantity__c > 0) {

                    vehicle.Stock_Quantity__c -= 1;
```

```
                vehiclesToUpdate.add(vehicle);

            }

        }

    }

    if (!vehiclesToUpdate.isEmpty()) {

        update vehiclesToUpdate;

    }

  }

}
```

- VehicleOrderTrigger

```
trigger VehicleOrderTrigger on Vehicle_Order__c (before insert,before update, after insert, after update) {

  VehicleOrderTriggerHandler.handleTrigger(Trigger.new, Trigger.oldMap, Trigger.isBefore,Trigger.isAfter, Trigger.isInsert, Trigger.isUpdate);


}
```

- VehicleOrderBatch

```
global class VehicleOrderBatch implements Database.Batchable<sObject> {

  global Database.QueryLocator start(Database.BatchableContext bc) {
    return Database.getQueryLocator([
      SELECT Id, Status__c, Vehicle__c
      FROM Vehicle_Order__c
      WHERE Status__c = 'Pending'
    ]);
  }

  global void execute(Database.BatchableContext bc, List<Vehicle_Order__c> orderList) {

    Set<Id> vehicleIds = new Set<Id>();
    for (Vehicle_Order__c order : orderList) {
      if (order.Vehicle__c != null) {
        vehicleIds.add(order.Vehicle__c);
      }
    }

    if (!vehicleIds.isEmpty()) {
      Map<Id, Vehicle__c> vehicleStockMap = new Map<Id, Vehicle__c>();
      for (Vehicle__c vehicle : [
        SELECT Id, Stock_Quantity__c
        FROM Vehicle__c
        WHERE Id IN :vehicleIds
      ]) {
        vehicleStockMap.put(vehicle.Id, vehicle);
      }
```

```
        List<Vehicle_Order__c> ordersToUpdate = new List<Vehicle_Order__c>();
        List<Vehicle__c> vehiclesToUpdate = new List<Vehicle__c>();

        for (Vehicle_Order__c order : orderList) {
            if (vehicleStockMap.containsKey(order.Vehicle__c)) {
                Vehicle__c vehicle = vehicleStockMap.get(order.Vehicle__c);

                if (vehicle.Stock_Quantity__c > 0) {
                    order.Status__c = 'Confirmed';
                    vehicle.Stock_Quantity__c -= 1;

                    ordersToUpdate.add(order);
                    vehiclesToUpdate.add(vehicle);
                }
            }
        }

        if (!ordersToUpdate.isEmpty()) {
            update ordersToUpdate;
        }

        if (!vehiclesToUpdate.isEmpty()) {
            update vehiclesToUpdate;
        }
    }
}

global void finish(Database.BatchableContext bc) {
```

```
        System.debug('Vehicle order batch job completed.');
    }
}
```

- VehicleOrderBatchScheduler

```
global class VehicleOrderBatchScheduler implements Schedulable {

    global void execute(SchedulableContext sc) {

        VehicleOrderBatch batchJob = new VehicleOrderBatch();

        Database.executeBatch(batchJob, 50); // 50 is the batch size

    }

}
```

- Batch apex

```
String cronExp = '0 0 12 * * ?'; // Runs daily at 12:00 PM

System.schedule('Daily Vehicle Order Processing', cronExp, new VehicleOrderBatchScheduler());
```