Phase5:ApexProgramming(Developer)

InPhase5,Ifocusedon**Apexprogrammingconcepts**in Salesforcetoaddbackendbusinesslogic,automation, andasynchronousprocessingtotheJobPortalproject. BelowarethedetailsoftheconceptsIimplementedalong with scenarios.

1.ClassessObjectsE

xplanation:
InApex,classesaretemplatesthatdefineobjects,their attributes,andmethods.Objectsareinstancesof classes.Theyhelpinorganizingcode,applyingreusability, andimplementing business logic.

Scenario:
IcreatedanApexclass`Job Application Handler`to manageoperationsrelatedtojobapplications,suchas validatingapplicantdetailsandassigninginterviewers.For example,whenanewapplicantrecordiscreated,the classmethodsareusedtocheckeligibilitybeforesaving.

2.Apex Triggers(Before/After Insert/Update)**Explanation:**

Triggersareusedtoperformactionsautomaticallybefore orafterDML(DataManipulationLanguage)operationslike insert,update,ordelete.

Scenario:

- **BeforeInsert:**Preventedduplicatejobapplications forthesamepositionbythesamecandidate.
- **AfterInsert:**SentanautomaticnotificationtoHR afterajobapplicationwassubmitted.

- **AppicationPreventduplicateHandlerthis**apex triggerhelpsinpreventingduplicaterecordsof applicantforthesamecontactandjobopening.

publicclassApplication_Trigger_Handler{

publicstaticvoidpreventDuplicateApplications(List newApps){

```
   //CollectallContactandJobIdsfrom the
incoming records
    Set<Id>contactIds=newSet<Id>(); Set<Id>
    jobIds = new Set<Id>();

    for(Applicationcapp:newApps){ if
        (app.Contactc != null) {
            contactIds.add(app.Contactc);
        }
        if (app.Jobc != null) {
            jobIds.add(app.Jobc);
        }
    }
    //QueryexistingApplicationswith
thosewithContactandJob combinations
    List<Applicationc>existingApps=[
        SELECT Id, Contactc, Jobc FROM
        Applicationc
        WHEREContactcIN:contactIds AND
        Jobc IN :jobIds
    ];

    //Buildasetofexistingkeys (ContactId +
JobId)
    Set<String>existingKeys=new
Set<String>();
    for(Applicationcapp:existingApps){
        existingKeys.add(app.Contactc+ '-'
+app.Jobc);
    }

    //Comparewithnewrecords→block
duplicates
    for (Applicationc app : newApps) {
        Stringkey=app.Contactc+'-' +
app.Jobc;
        if (existingKeys.contains(key)) {
            app.addError('Thiscandidatehas
alreadyappliedforthisjob posting.');
        }
    }
```

```
}

}
```

ApplicationPreventDuplicateTrigger

```
triggerApplicaiton_TriggeronApplicationc(before insert)

 {if(Trigger.isBeforeCC Trigger.isInsert)
{Application_Trigger_Handler.preventDuplicateApplicatio
ns(Trigger.new);}

 }
```

3.TriggerDesignPatternE

xplanation:
 TheTriggerDesignPatternensuresthattriggersareclean, scalable,andmaintainable.Businesslogicisseparated intohandlerclassesinsteadofwritingdirectlyinsidethe trigger.

Scenario:

FortheApplicationcobject,insteadofwritingall logicinsidethetrigger,Icreated ApplicationTriggerHandlerclass which handled validations,notifications,andupdates.Thetriggersimply calledthehandlermethods,makingitreusableand cleaner.

- **<u>CreateApplicationfromcontactcreatedandExistingJobOpening</u>-this**createsapplication automaticallywhenacontactassociatedwithajob openingisbeingcreated.

publicclassApplication_Trigger_Handler_1{

```
//MethodtocreateApplicationsfrom
Contacts who applied

public static void
createApplicationsFromContacts(List<Contact>ne
wContacts) {

    List<Applicationc>appsToCreate=new
List<Applicationc>();

    //:Loopthrough Contacts
    for(Contactc:newContacts){
        //OnlycreateApplicationif
Job_Postingc is filled
        if(c.Job_Openingc!=null){

            //:Preventduplicate
ApplicationforsameContact+Job
            List<Applicationc>existingApps
=[
                SELECTIdFROMApplicationc
                WHERE Contactc = :c.Id
                AND    Jobc
=:c.Job_Openingc
            ];
            if(existingApps.isEmpty()){
                Applicationc app = new
Applicationc();
                app.Contactc=c.Id; app.Jobc
                =
c.Job_Openingc;
                app.Applicant_Statusc=
'Applied';
                appsToCreate.add(app);
            }
        }
    }

    //Insert Applications
    if(!appsToCreate.isEmpty()){ insert
        appsToCreate;
    }
}

}
```

- **<u>ApplicationStatusHandler-</u>>**whenevertheapplicationstatusisupdatedtoshortlistedthenataskiscreatedandisassignedtotherecruiterwhowillbetakingtheinterviewasanotificationabouttheinterview.

publicclassApplication_Status_Trigger_Hander{

```
//MethodtocreateTaskwhenApplication status
changes
public static void
createTaskOnStatusChange(List<Applicationc>
newApps, Map<Id, Applicationc> oldMap) {

    List<Task>tasksToCreate=new List<Task>();

    for(Applicationcapp:newApps){

        //Compareoldvsnewstatusto detect
change
        ApplicationcoldApp=
oldMap.get(app.Id);

        if (oldApp.Applicant_Statusc!=
app.Applicant_Statusc &&
app.Applicant_Statusc=='shortlisted'&&
app.Assigned_Userc != null) {

            Task t = new Task();
            t.Subject='Followupon
shortlistedApplication';
            t.WhatId=app.Id;//Relatedto
Application
            t.OwnerId=app.Assigned_Userc;
//Assigntorecruiter(replacewithyour field API
name)
            t.Status = 'Not Started';
            t.Priority = 'High';
            t.Description='Theapplication
hasbeenapproved.Followupwiththe candidate.';
            tasksToCreate.add(t);
        }
    }

    if(!tasksToCreate.isEmpty()){ insert
        tasksToCreate;
    }
}
```

}

ApplicationstatusTrigger

triggerApplication_status_triggeronApplicationc(after update){

```
//Callhandlermethod,passTrigger.newand
Trigger.oldMap
```

Application_Status_Trigger_Hander.createTaskOnStatusChange(Trigger.new,Trigger.oldMap);}

Contacttrigger

trigger Contact_Trigger_1 on Contact (after insert, after update){

```
List<Contact>contactsWithJob=new
List<Contact>();

//Step1:Loopthroughinserted/updated contacts
for(Contactc:Trigger.new){
    if (c.Job_Openingc != null) { //
replacewithyouractualfieldAPIname
        contactsWithJob.add(c);
    }
}

//Step2:Callhandlertocreate Applications
if(!contactsWithJob.isEmpty()){

Application_Trigger_Handler_1.createApplicati
onsFromContacts(contactsWithJob);
}

}
```

4. SOQLsSOSL

Explanation:

- **SOQL(SalesforceObjectQueryLanguage):**Usedto fetchrecordsfromSalesforceobjectsbasedon conditions.
- **SOSL(SalesforceObjectSearchLanguage):**Used toperformtext-basedsearchesacrossmultiple objects.

Scenario:

- SOQLwasusedtofetchallapplicationsforagiven candidate(`SELECT Id, Status FROM Applicationc WHERE Candidatec =:candidateId`).
- SOSLwasusedtosearchapplicantdetails(like email/phone)acrossobjectswhenHRwantedto quicklyfindacandidate.

5.Collections:List,Set,MapE

xplanation:
 Collectionsaredatastructuresusedtostoremultiple records.

- **List:**Orderedcollectionallowingduplicates.
- **Set:**Unorderedcollectionwithoutduplicates.
- **Map:**Key-valuepairsforquicklookups.

Scenario:

- **List:**Usedtostoreallinterviewrecordsfora particularapplication.
- **Set:**Usedtostoreuniquecandidateemailsto preventduplicates.
- **Map:**Usedtomap`ApplicationId`→`Interview Date`forquickaccessinbulkprocessing.

6. ControlStatements

Explanation:
Controlstatementslike`if-else`,`for`,`while`,and `switch`areusedtoapplydecision-makingandlooping logic.

Scenario:
Whenassigninganinterviewer,Iusedcontrolstatements:

- Iftheapplicationstatusis"InterviewScheduled", thenassignaninterviewer.
- Elseifthestatusis"Rejected",marktheapplication as closed.

12. TestClassesE

xplanation:
TestclassesarewrittentoverifythatApexcodeworks correctlyandtomeetSalesforce’srequirementof75% codecoveragefordeployment.

Scenario:
Foreachtriggerandclass,Iwrotetestclassessuchas `TestApplicationHandler`whichtested:

- Creatingavalidapplication

- Preventingduplicateapplications
- Schedulinginterviews
  Thisensuredthatalllogicworkedasexpectedbefore deployment.

```
@IsTest
public class ATS_TestClass {

    // Utility method to create a Contact
    private static Contact createContact() {
        Contact c = new Contact(
            LastName = 'Test Candidate',
            Email = 'testcandidate@example.com'
        );
        insert c;
        return c;
    }

    // Utility method to create a Job Posting
    private static Job_Opening__c
createJobPosting() {
        Job_Opening__c job = new Job_Opening__c(
            Name = 'Software Engineer'
        );
        insert job;
        return job;
    }

    // Utility method to create an Application
    private static Application__c
createApplication(Id contactId, Id jobId, String
statusVal) {
        Application__c app = new Application__c(
            Contact__c = contactId,
            Job__c = jobId,
           Applicant_Status__c = statusVal
        );
        insert app;
        return app;
    }

    // Utility method to create Applicant Info +
Interview
    private static Interview__c createInterview(Id
appInfoId, Datetime slotTime) {
        Interview__c interview = new Interview__c(
            Applicant_Information__c = appInfoId,
            Interview_Date_Time__c = slotTime
        );
        insert interview;
        return interview;
    }

    // ---------------------------
    // TEST CASES
    // ---------------------------

    @IsTest
    static void testDuplicateApplicationPrevention()
{
        Contact c = createContact();
        Job_Opening__c job = createJobPosting();

        // Insert first application
        Application__c app1 =
createApplication(c.Id, job.Id, 'Applied');

        // Try inserting duplicate application
        Application__c app2 = new Application__c(
            Contact__c = c.Id,
            Job__c = job.Id,
           Applicant_Status__c= 'Applied'
        );

        Test.startTest();
        try {
            insert app2;
            System.assert(false, 'Duplicate should not
be inserted');
        } catch (DmlException e) {

System.assert(e.getMessage().contains('duplicate'),
'Should block duplicate application');
        }
        Test.stopTest();
    }

    @IsTest
    static void
testTaskCreationOnApprovedApplication() {
        Contact c = createContact();
        Job_Opening__c job = createJobPosting();

        Application__c app = createApplication(c.Id,
job.Id, 'Applied');

        Test.startTest();
        app.Applicant_Status__c = 'Approved';
        update app;
        Test.stopTest();

        // Check Task created
        List<Task> tasks = [SELECT Id, Subject,
WhatId FROM Task WHERE WhatId = :app.Id];
        System.assertEquals(1, tasks.size(), 'Task
should be created when Application is Approved');
        System.assertEquals('Application Approved
Notification', tasks[0].Subject, 'Task subject should
match');
    }

    @IsTest
    static void testInterviewValidation_NoOverlap()
{
        Contact c = createContact();
        Job_Opening__c job = createJobPosting();
        Application__c app = createApplication(c.Id,
job.Id, 'Applied');

        // First interview slot
        Interview__c int1 = createInterview(app.Id,
Datetime.now().addDays(1));

        // Overlapping interview slot
        Interview__c int2 = new Interview__c(
            Applicant_Information__c = app.Id,
            Interview_Date_Time__c =
int1.Interview_Date_Time__c // same time
```

```
        );

        Test.startTest();
        try {
            insert int2;
            System.assert(false, 'Should not allow
overlapping interviews');
        } catch (DmlException e) {

System.assert(e.getMessage().contains('overlap'),
'Should block overlapping interview creation');
        }
        Test.stopTest();
    }
}
```

13. AsynchronousProcessingE

xplanation:
 Asynchronousprocessing(BatchApex,Queueable, Scheduled,Futuremethods)allowsoperationstorunin thebackgroundwithoutblockingthemainexecution.

Scenario:

- BatchApex:Closinginactiveapplications.
- QueueableApex:Sendingnotificationsfornewjob postings.

- ScheduledApex:Interviewreminders.
- FutureMethod:Backgroundverificationwithexternal systems.

Thisensuredbetterperformanceandscalabilityofthe system.