Phase 5: Apex Programming (Developer)

1. Apex Classes & Objects — Service classes you should build

Keep business logic out of triggers. Use classes for reusability and unit testing.

Classes to create for project:

- ApplicationService — create application, duplicate check, business validations.
- InterviewService — schedule interview, send notifications (calls Flow or Queueable).
- TriggerHandlerBase — base class for standard trigger handler pattern.

Steps:

1. Setup → Developer Console → File → New → Apex Class.
2. Name it (e.g., ApplicationService) → write the code → Save.

ApplicationService.apxc

```
public with sharing class ApplicationService {
    public static void preventDuplicateApplications(List<Application__c> newApps) {
        if (newApps == null || newApps.isEmpty()) return;
        // collect candidate and job Ids
        Set<Id> candidateIds = new Set<Id>();
        Set<Id> jobIds = new Set<Id>();
        for (Application__c a : newApps) {
            if (a.Candidate__c != null) candidateIds.add(a.Candidate__c);
            if (a.Job__c != null) jobIds.add(a.Job__c);
        }
        if (candidateIds.isEmpty() || jobIds.isEmpty()) return;
        // query existing applications with those candidates and jobs
        List<Application__c> existing = [
            SELECT Id, Candidate__c, Job__c
            FROM Application__c
            WHERE Candidate__c IN :candidateIds
            AND Job__c IN :jobIds
        ];
        // build set of existing keys
        Set<String> existingKeys = new Set<String>();
        for (Application__c e : existing) {
            existingKeys.add(String.valueOf(e.Candidate__c) + '-' + String.valueOf(e.Job__c));
        }
        // mark errors on duplicates in the incoming set
        for (Application__c a : newApps) {
            if (a.Candidate__c == null || a.Job__c == null) continue;
            String key = String.valueOf(a.Candidate__c) + '-' + String.valueOf(a.Job__c);
            if (existingKeys.contains(key)) {
                a.addError('This candidate has already applied for this job.');
            }
        }
    }
    // small helper used by tests to ensure ApplicationService is callable
    public static void someUtilityMethod() {
        System.debug('ApplicationService.someUtilityMethod executed');
    }
}
```

InterviewService.apxc

```
public with sharing class InterviewService {
    public static void validateInterviewDates(List<Interview__c> newRecords) {
        if (newRecords == null) return;
        for (Interview__c i : newRecords) {
            // Use Interview_Date__c as earlier logs indicated that's the field API name
            if (i.Interview_Date__c != null && i.Interview_Date__c < Date.today()) {
                i.addError('Interview date cannot be in the past.');
            }
        }
    }

    // small helper used by tests to ensure InterviewService is callable
    public static void someMethod() {
        System.debug('InterviewService.someMethod executed');
    }
}
```

TriggerHandlerBase.apxc

```
public abstract class TriggerHandlerBase {
    /**
     * Use this static flag from tests to skip trigger/handler logic
     * when creating test data in @TestSetup. Tests should set it
     * to true during setup and set to false when they want trigger
     * behavior to run.
     */
    public static Boolean SKIP_TRIGGERS = false;
}
```

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

Respond to DML events (create/update/delete) on records.

Create one trigger per object — e.g., ApplicationTrigger, InterviewTrigger.

Steps (Developer Console):

1. Setup → Developer Console → File → New → Apex Trigger.
2. Select Object → Name, e.g., ApplicationTrigger → Save.
3. In trigger body call handler methods.

ApplicationTrigger.apxt

Code Coverage: None | API Version: 64

```
trigger ApplicationTrigger on Application__c (
    before insert,
    before update,
    after insert,
    after update
) {
    if (Trigger.isBefore) {
        if (Trigger.isInsert) {
            ApplicationTriggerHandler.beforeInsert(Trigger.new);
        }
        if (Trigger.isUpdate) {
            ApplicationTriggerHandler.beforeUpdate(Trigger.new, Trigger.oldMap);
        }
    }
    if (Trigger.isAfter) {
        if (Trigger.isInsert) {
            ApplicationTriggerHandler.afterInsert(Trigger.new);
        }
        if (Trigger.isUpdate) {
            ApplicationTriggerHandler.afterUpdate(Trigger.new, Trigger.oldMap);
        }
    }
}
```

InterviewTrigger.apxt

Code Coverage: None | API Version: 64

```
trigger InterviewTrigger on Interview__c (
    before insert,
    before update,
    after insert
) {
    if (Trigger.isBefore) {
        if (Trigger.isInsert) {
            InterviewTriggerHandler.beforeInsert(Trigger.new);
        }
        if (Trigger.isUpdate) {
            InterviewTriggerHandler.beforeUpdate(Trigger.new, Trigger.oldMap);
        }
    }

    if (Trigger.isAfter) {
        if (Trigger.isInsert) {
            InterviewTriggerHandler.afterInsert(Trigger.new);
        }
    }
}
```

3. SOQL & SOSL — queries & searches

SOQL (Salesforce Object Query Language) — use to fetch records.
SOSL (Salesforce Object Search Language) — use to search text across fields (like resume text).

SOQL — get applications for candidate Ids:

```
if (candidateIds.isEmpty() || jobIds.isEmpty()) return;
// query existing applications with those candidates and jobs
List<Application__c> existing = [
    SELECT Id, Candidate__c, Job__c
    FROM Application__c
    WHERE Candidate__c IN :candidateIds
    AND Job__c IN :jobIds
];
```

4. Collections: List, Set, Map

Efficient processing and to avoid duplicates.
Examples:

Set<Id> changedRecIds when candidate or job changed, re-check duplicates.
List<Application__c> appsToUpdate to batch update records.
Map<Id, Job__c> jobById to map parent records for quick lookup.

```
// If candidate or job changed, re-check duplicates
Set<Id> changedRecIds = new Set<Id>();
for (Application__c a : newList) {
    Application__c oldA = oldMap.get(a.Id);
    if (oldA == null) continue;
    if (a.Candidate__c != oldA.Candidate__c || a.Job__c != oldA.Job__c) changedRecIds.add(a.Id);
}
if (!changedRecIds.isEmpty()) {
    List<Application__c> changed = new List<Application__c>();
    for (Application__c a : newList) if (changedRecIds.contains(a.Id)) changed.add(a);
    ApplicationService.preventDuplicateApplications(changed);
}
isExecuting = false;
```

5. Batch Apex:

Batch Apex — processing large data sets

- Bulk update Applications (e.g., mark old applications as Expired)
- Mass email or data cleanup (when >50,000 records)

Steps to create Batch Apex:

1. Create an Apex class implementing Database.Batchable<sObject>.
2. Implement start, execute, finish.
3. Optionally implement Database.Stateful to preserve state across execute batches.

ExpireOldApplicationsBatch.apxc

Code Coverage: None | API Version: 64

```apex
global class ExpireOldApplicationsBatch implements Database.Batchable<sObject>, Database.AllowsCallouts {
    global Database.QueryLocator start(Database.BatchableContext bc) {
        String q = 'SELECT Id, Status__c, Application_Date__c FROM Application__c WHERE Status__c IN (\'Submitted\', \'Under Review\')';
        return Database.getQueryLocator(q);
    }

    global void execute(Database.BatchableContext bc, List<sObject> scope) {
        List<Application__c> updates = new List<Application__c>();
        Date today = Date.today();
        for (sObject s : scope) {
            Application__c a = (Application__c)s;
            if (a.Application_Date__c != null && a.Application_Date__c.addDays(60) < today) {
                Application__c up = new Application__c(Id = a.Id, Status__c = 'Expired');
                updates.add(up);
            }
        }
        if (!updates.isEmpty()) {
            update updates;
        }
    }

    global void finish(Database.BatchableContext bc) {
        // Optionally enqueue follow-up queueable or send email summary
    }
}
```

6. Queueable Apex

Process resume attachments (heavy CPU or callout)
Chain jobs (e.g., after batch finish enqueue further processing)
Replace some use-cases of future methods with more control

```apex
@IsTest
static void testInterviewQueueable() {
    // Create Candidate + Job + Application first
    Candidate__c cand = new Candidate__c(Name='Cand Q', Email__c='q@app.com');
    insert cand;

    Job__c job = new Job__c(Name='Dev Intern');
    insert job;

    Application__c app = new Application__c(
        Candidate__c=cand.Id,
        Job__c=job.Id,
        Status__c='Submitted',
        Application_Date__c = Date.today()
    );
    insert app;
```

7. Schedule Apex

- Run Batch Apex daily/weekly to expire applications, refresh counters.
- Update dashboard data at night.

ScheduledExpireOldApps.apxc

Code Coverage: None | API Version: 64

```apex
global class ScheduledExpireOldApps implements Schedulable {
    global void execute(SchedulableContext sc) {
        Database.executeBatch(new ExpireOldApplicationsBatch(), 200);
    }
}
```

```apex
@IsTest
static void testScheduledExpireOldApps() {
    Test.startTest();
    String jobId = System.schedule(
        'TestSched',
        '0 0 12 * * ?',
        new ScheduledExpireOldApps()
    );
    Test.stopTest();
    System.assertNotEquals(null, jobId, 'Scheduled job should run');
}
```

8. @future Methods (Callouts) — for external integrations

- Call external resume parsing API after interview creation.
- Send data to third-party ATS.

```apex
public with sharing class ExternalIntegration {
    // Synchronous callout method – tests will mock this using HttpCalloutMock
    public static void callExternalAPI() {
        Http http = new Http();
        HttpRequest req = new HttpRequest();
        req.setEndpoint('https://example.com/api'); // dummy endpoint
        req.setMethod('GET');
        req.setTimeout(60000);
        try {
            HttpResponse res = http.send(req);
            System.debug('ExternalIntegration response: ' + res.getBody());
        } catch (Exception ex) {
            System.debug('Callout failed: ' + ex.getMessage());
        }
    }

    @future(callout=true)
    public static void sendResumeForParsing(String resumeUrl, Id candidateId) {
        if (String.isBlank(resumeUrl) || candidateId == null) return;
        try {
            Http http = new Http();
            HttpRequest req = new HttpRequest();
            req.setEndpoint(resumeUrl); // Replace with actual parsing endpoint
            req.setMethod('GET');
            HttpResponse res = http.send(req);
            // handle response (store parsed data) – omitted for brevity
        } catch (Exception ex) {
            System.debug('Callout failed: ' + ex.getMessage());
        }
    }
}
```

```apex
@IsTest
public class ExternalIntegrationTest {

    @IsTest
    static void testExternalCallout() {
        // Mock the HTTP callout
        Test.setMock(HttpCalloutMock.class, new MockHttpResponseGenerator());

        Test.startTest();
        ExternalIntegration.callExternalAPI();
        Test.stopTest();

        System.assert(true, 'External API callout executed with mock');
    }

    // Simple mock response
    private class MockHttpResponseGenerator implements HttpCalloutMock {
        public HTTPResponse respond(HTTPRequest req) {
            HttpResponse res = new HttpResponse();
            res.setStatusCode(200);
            res.setBody('{"success":true}');
            return res;
        }
    }
}
```

9. Test Classes

- Use `@isTest` classes.
- Do not use `SeeAllData=true` (create all test data in test).
- Use `Test.startTest()` and `Test.stopTest()` to run async jobs, batch, queueable, future.
- Assert expected behavior using `System.assert`.
- Cover positive and negative cases (duplicates, invalid dates, bulk insert).

```apex
@IsTest
public class BatchAndSchedTest {

    @TestSetup
    static void setupData() {
        // Candidate
        Candidate__c cand = new Candidate__c(
            Name='Batch Candidate',
            Email__c='batch@app.com'
        );
        insert cand;

        // Job
        Job__c job = new Job__c(Name='QA Intern');
        insert job;

        // Application with old Application_Date__c (more than 60 days old)
        Application__c oldApp = new Application__c(
            Candidate__c = cand.Id,
            Job__c       = job.Id,
            Status__c    = 'Submitted',
            Application_Date__c = Date.today().addDays(-70)
        );
        insert oldApp;
    }

    @IsTest
    static void testExpireOldApplicationsBatch() {
        Test.startTest();
        Database.executeBatch(new ExpireOldApplicationsBatch(), 100);
        Test.stopTest();

        // Verify the old application was expired
        Application__c checkApp = [
            SELECT Id, Status__c
            FROM Application__c
            WHERE Status__c = 'Expired'
            LIMIT 1
        ];
        System.assertEquals('Expired', checkApp.Status__c);
    }
```

```apex
@IsTest
public class InterviewTest {
    @TestSetup
    static void setupData() {
        Candidate__c cand = new Candidate__c(Name='Interview Cand', Email__c='int@app.com');
        insert cand;

        Job__c job = new Job__c(Name='QA Intern');
        insert job;

        Application__c app = new Application__c(
            Candidate__c = cand.Id,
            Job__c = job.Id,
            Status__c = 'Submitted',
            Application_Date__c = Date.today()
        );
        insert app;

        // Insert a valid interview with required fields
        Interview__c interview = new Interview__c(
            Application__c = app.Id,
            Interview_Date__c = Date.today().addDays(2)
        );
        insert interview;
    }

    @IsTest
    static void testInterviewInsert() {
        Application__c app = [SELECT Id FROM Application__c LIMIT 1];

        Interview__c newInt = new Interview__c(
            Application__c = app.Id,
            Interview_Date__c = Date.today().addDays(3)
        );

        Test.startTest();
        insert newInt;
        Test.stopTest();

        System.assertNotEquals(null, newInt.Id);
    }
```

10.Asynchronous Processing

Batch Apex — use when you need to process many Application records (>50k) or heavy updates. Example: monthly cleanup, mass status changes.
Queueable — for medium-sized chaining tasks and complex processing per record (resume processing, 3rd party calls).
Scheduled Apex — to kick off Batch or Queueable at scheduled times.
Future — for simple fire-and-forget callouts (legacy; Queueable preferred now).