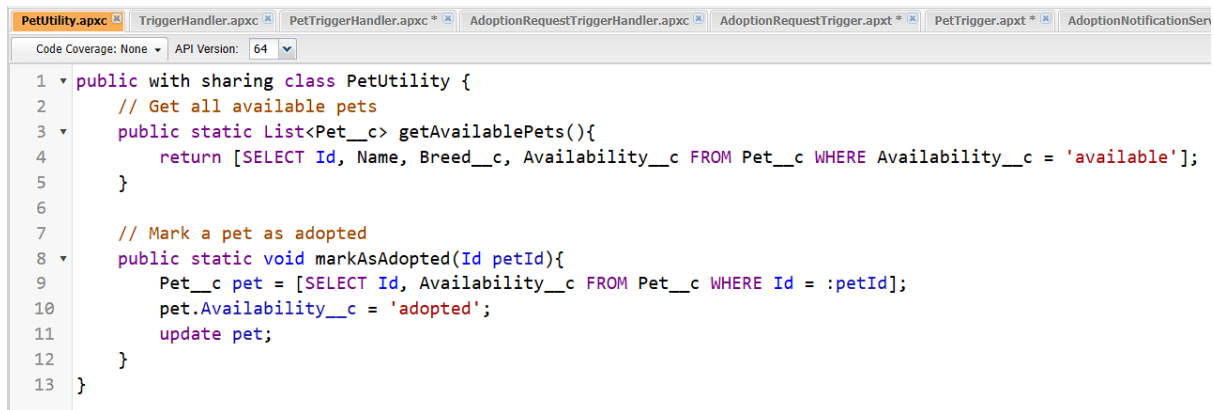


Phase 5: Apex Programming (Developer)

1.Apex Classes & Objects

1. PetUtility.cls: To check number of available pets to adopt



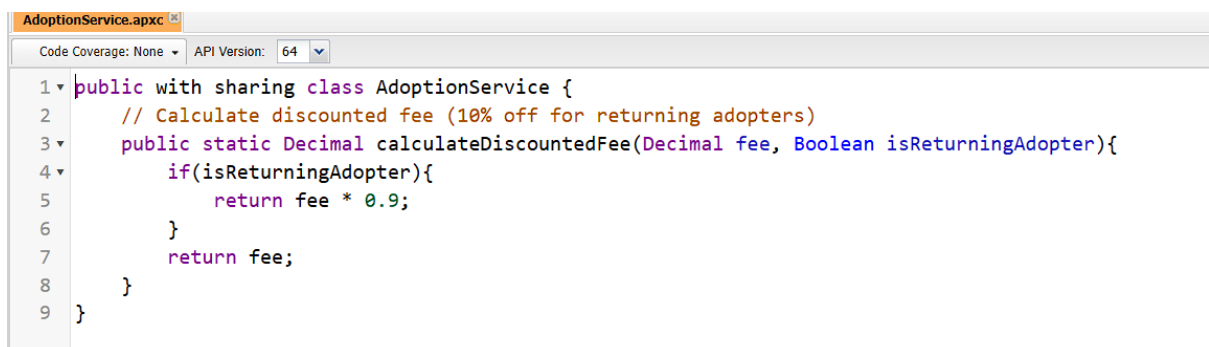
```
1 public with sharing class PetUtility {
2     // Get all available pets
3     public static List<Pet__c> getAvailablePets(){
4         return [SELECT Id, Name, Breed__c, Availability__c FROM Pet__c WHERE Availability__c = 'available'];
5     }
6
7     // Mark a pet as adopted
8     public static void markAsAdopted(Id petId){
9         Pet__c pet = [SELECT Id, Availability__c FROM Pet__c WHERE Id = :petId];
10        pet.Availability__c = 'adopted';
11        update pet;
12    }
13 }
```

2. AdopterService.cls : To Count how many pets adopter has adopted



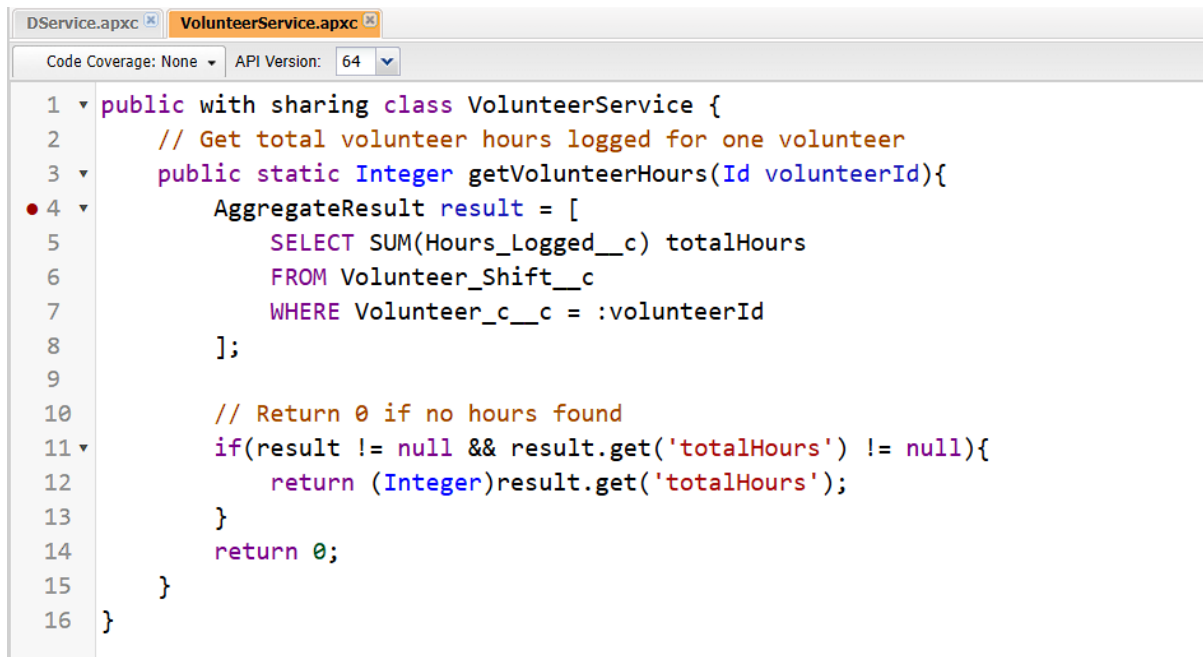
```
1 public with sharing class AdopterService {
2     // Count how many pets adopter has adopted
3     public static Integer getTotalPetsAdopted(Id adopterId){
4         return [SELECT COUNT() FROM Adoption_Request__c WHERE Adopter__c = :adopterId AND Status__c = 'Approved'];
5     }
6 }
```

3. AdoptionService.cls : This class created to get discount for potential adopters



```
1 public with sharing class AdoptionService {
2     // Calculate discounted fee (10% off for returning adopters)
3     public static Decimal calculateDiscountedFee(Decimal fee, Boolean isReturningAdopter){
4         if(isReturningAdopter){
5             return fee * 0.9;
6         }
7         return fee;
8     }
9 }
```

4. VolunteerService.cls : To check Number of hours staff worked

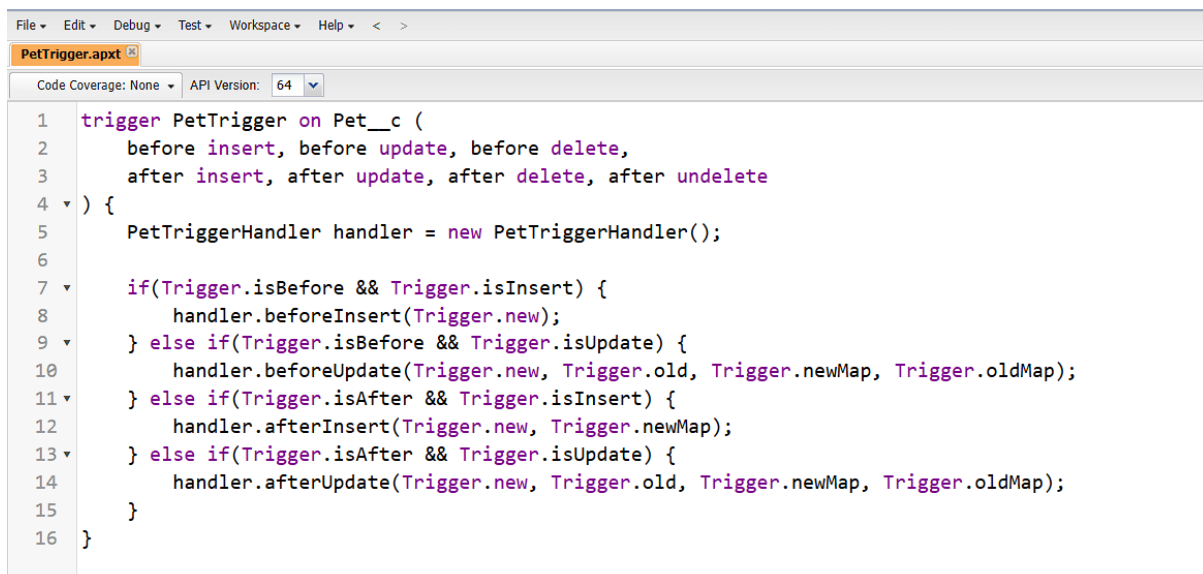


```
1 public with sharing class VolunteerService {
2     // Get total volunteer hours logged for one volunteer
3     public static Integer getVolunteerHours(Id volunteerId){
4         AggregateResult result = [
5             SELECT SUM(Hours_Logged__c) totalHours
6             FROM Volunteer_Shift__c
7             WHERE Volunteer_c__c = :volunteerId
8         ];
9
10        // Return 0 if no hours found
11        if(result != null && result.get('totalHours') != null){
12            return (Integer)result.get('totalHours');
13        }
14        return 0;
15    }
16 }
```

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

1. PetTrigger.trigger

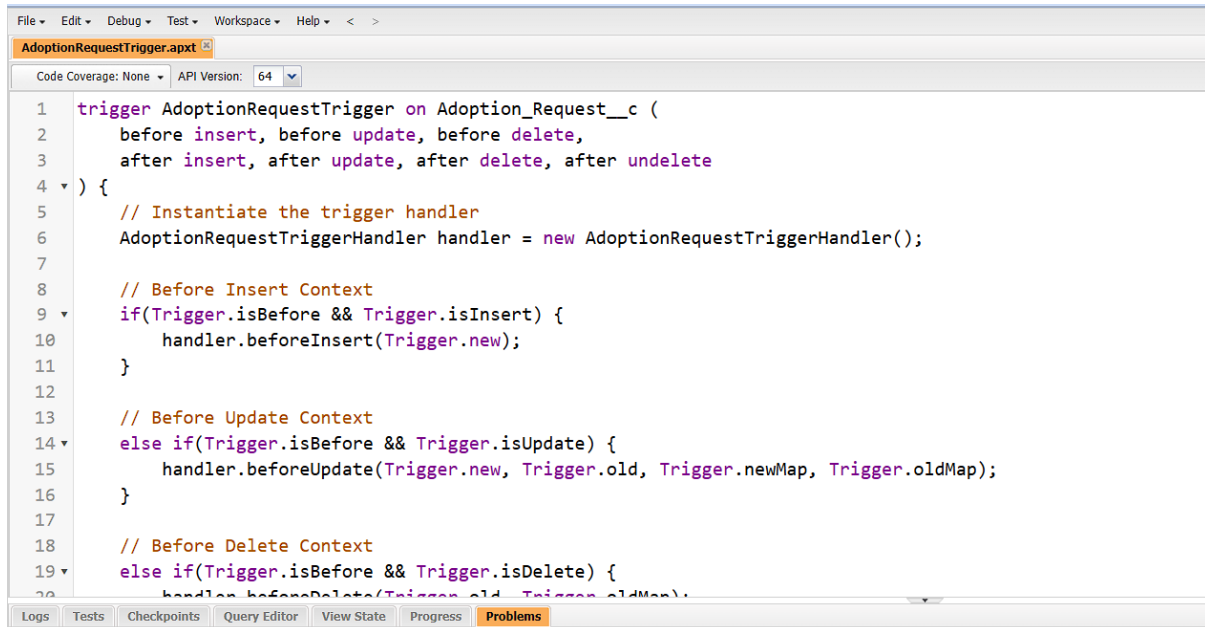
Purpose: To handle automation and business logic for Pet records



```
1 trigger PetTrigger on Pet__c (
2     before insert, before update, before delete,
3     after insert, after update, after delete, after undelete
4 ) {
5     PetTriggerHandler handler = new PetTriggerHandler();
6
7     if(trigger.isBefore && trigger.isInsert) {
8         handler.beforeInsert(trigger.new);
9     } else if(trigger.isBefore && trigger.isUpdate) {
10        handler.beforeUpdate(trigger.new, trigger.old, trigger.newMap, trigger.oldMap);
11    } else if(trigger.isAfter && trigger.isInsert) {
12        handler.afterInsert(trigger.new, trigger.newMap);
13    } else if(trigger.isAfter && trigger.isUpdate) {
14        handler.afterUpdate(trigger.new, trigger.old, trigger.newMap, trigger.oldMap);
15    }
16 }
```

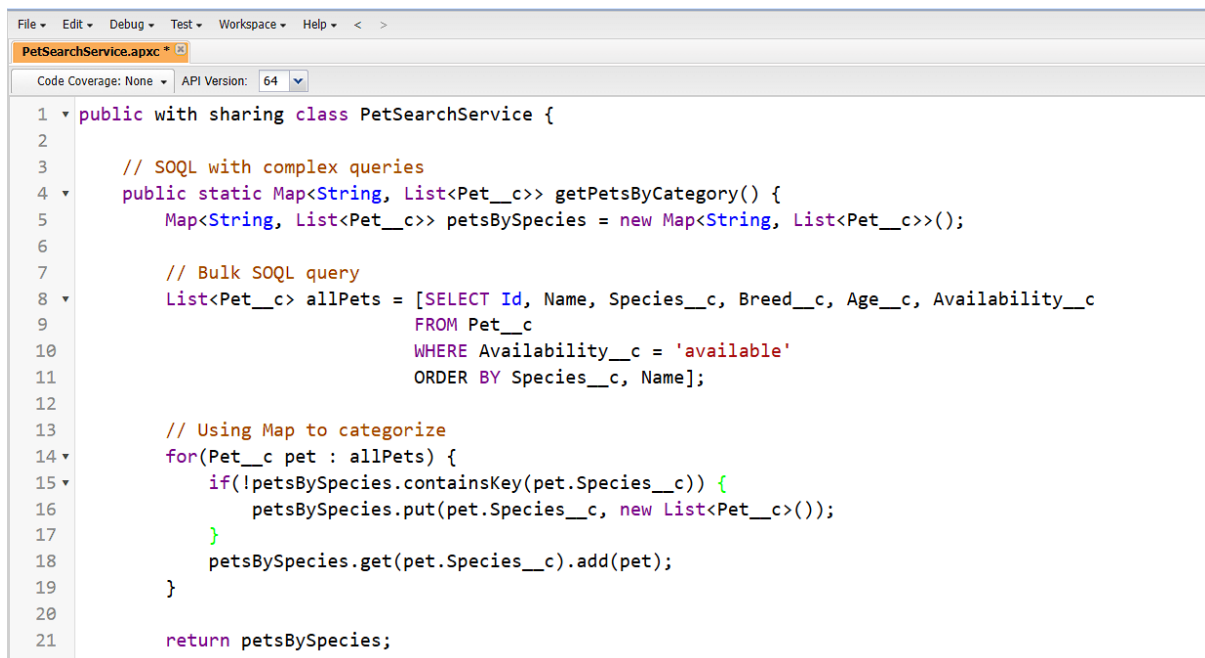
2. AdoptionRequestTrigger.trigger

Purpose: To manage the complete adoption request lifecycle



```
1 trigger AdoptionRequestTrigger on Adoption_Request__c (
2     before insert, before update, before delete,
3     after insert, after update, after delete, after undelete
4 ) {
5     // Instantiate the trigger handler
6     AdoptionRequestTriggerHandler handler = new AdoptionRequestTriggerHandler();
7
8     // Before Insert Context
9     if(Trigger.isBefore && Trigger.isInsert) {
10         handler.beforeInsert(Trigger.new);
11     }
12
13     // Before Update Context
14     else if(Trigger.isBefore && Trigger.isUpdate) {
15         handler.beforeUpdate(Trigger.new, Trigger.old, Trigger.newMap, Trigger.oldMap);
16     }
17
18     // Before Delete Context
19     else if(Trigger.isBefore && Trigger.isDelete) {
20         handler.beforeDelete(Trigger.old, Trigger.oldMap);
21     }
22 }
```

SOQL & SOSL - Implement Search Functionality

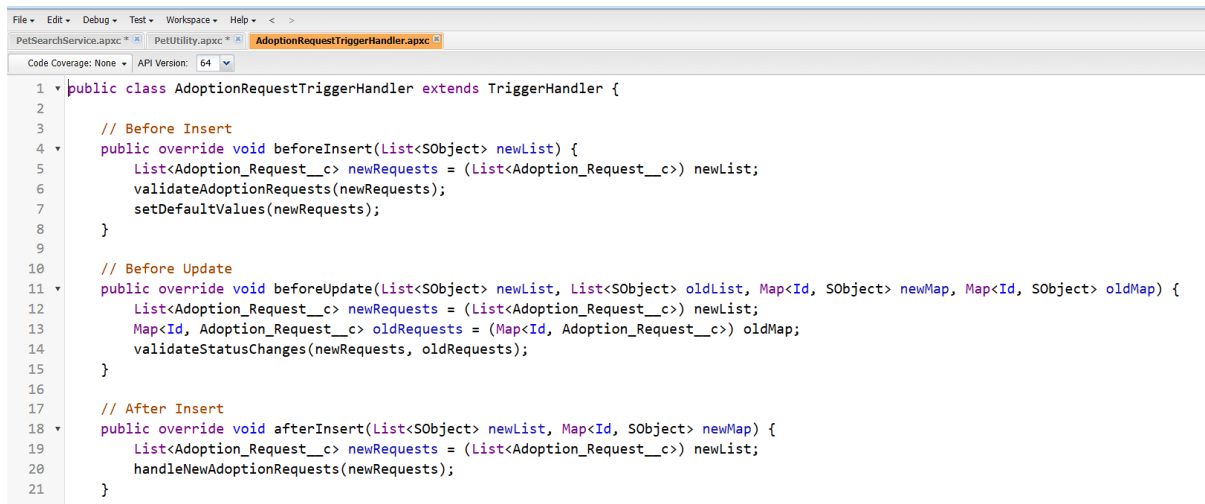


```
1 public with sharing class PetSearchService {
2
3     // SOQL with complex queries
4     public static Map<String, List<Pet__c>> getPetsByCategory() {
5         Map<String, List<Pet__c>> petsBySpecies = new Map<String, List<Pet__c>>();
6
7         // Bulk SOQL query
8         List<Pet__c> allPets = [SELECT Id, Name, Species__c, Breed__c, Age__c, Availability__c
9                                FROM Pet__c
10                                WHERE Availability__c = 'available'
11                                ORDER BY Species__c, Name];
12
13         // Using Map to categorize
14         for(Pet__c pet : allPets) {
15             if(!petsBySpecies.containsKey(pet.Species__c)) {
16                 petsBySpecies.put(pet.Species__c, new List<Pet__c>());
17             }
18             petsBySpecies.get(pet.Species__c).add(pet);
19         }
20
21         return petsBySpecies;
22     }
23 }
```

What it does:

- **SOQL:** Query Salesforce database to get pet records
- **WHERE clause:** Filter only available pets
- **IN clause:** Get multiple pets at once (bulk query)

Collections (List, Set, Map)

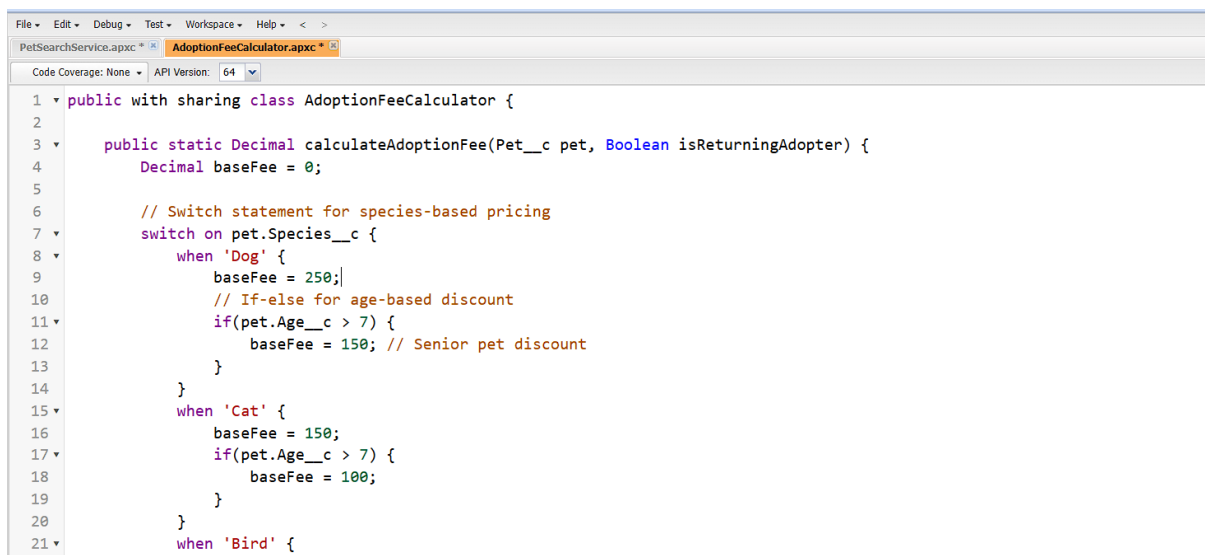


```

1 public class AdoptionRequestTriggerHandler extends TriggerHandler {
2
3     // Before Insert
4     public override void beforeInsert(List<SObject> newList) {
5         List<Adoption_Request__c> newRequests = (List<Adoption_Request__c>) newList;
6         validateAdoptionRequests(newRequests);
7         setDefaultValues(newRequests);
8     }
9
10    // Before Update
11    public override void beforeUpdate(List<SObject> newList, List<SObject> oldList, Map<Id, SObject> newMap, Map<Id, SObject> oldMap) {
12        List<Adoption_Request__c> newRequests = (List<Adoption_Request__c>) newList;
13        Map<Id, Adoption_Request__c> oldRequests = (Map<Id, Adoption_Request__c>) oldMap;
14        validateStatusChanges(newRequests, oldRequests);
15    }
16
17    // After Insert
18    public override void afterInsert(List<SObject> newList, Map<Id, SObject> newMap) {
19        List<Adoption_Request__c> newRequests = (List<Adoption_Request__c>) newList;
20        handleNewAdoptionRequests(newRequests);
21    }
22
23    --
  
```

- **List:** Maintains order of adoption requests
- **Set:** Ensures no duplicate pet IDs
- **Map:** Quick lookup of pets by ID (instead of looping through lists)

Control Statements



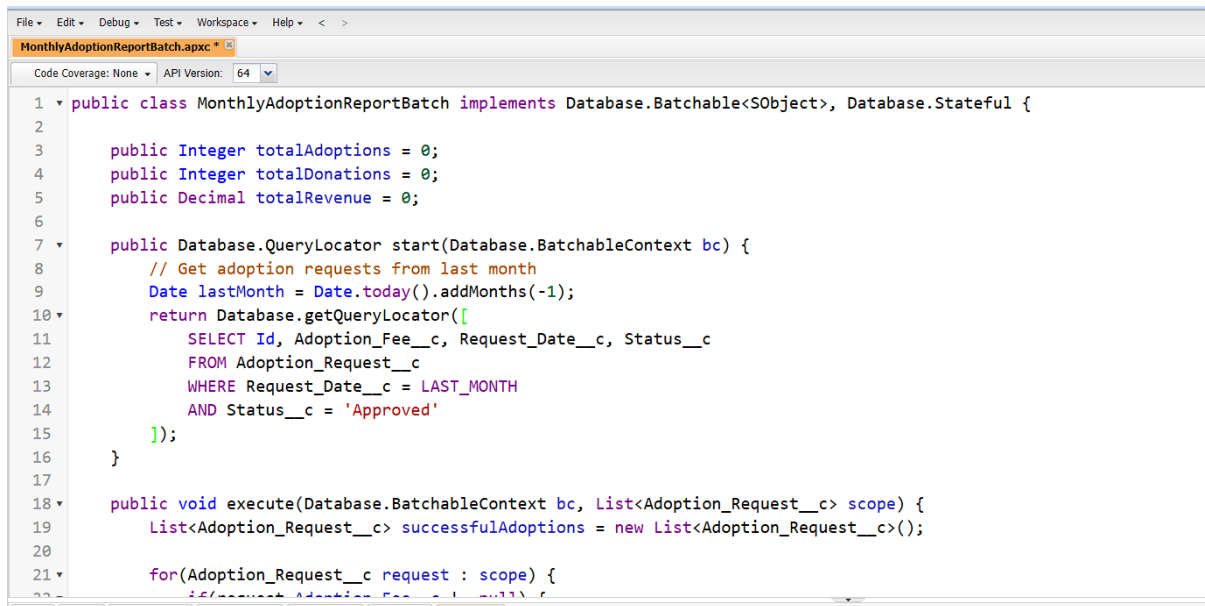
```

1 public with sharing class AdoptionFeeCalculator {
2
3     public static Decimal calculateAdoptionFee(Pet__c pet, Boolean isReturningAdopter) {
4         Decimal baseFee = 0;
5
6         // Switch statement for species-based pricing
7         switch on pet.Species__c {
8             when 'Dog' {
9                 baseFee = 250;
10                // If-else for age-based discount
11                if(pet.Age__c > 7) {
12                    baseFee = 150; // Senior pet discount
13                }
14            }
15            when 'Cat' {
16                baseFee = 150;
17                if(pet.Age__c > 7) {
18                    baseFee = 100;
19                }
20            }
21            when 'Bird' {
22                --
  
```

What it does:

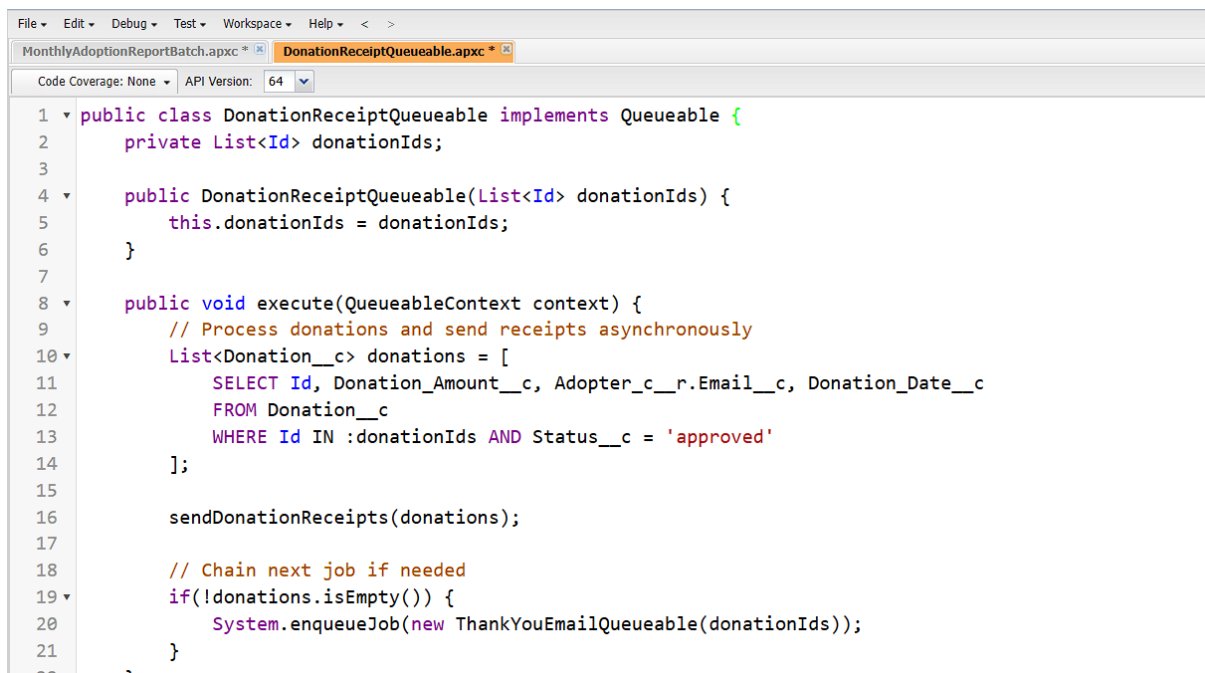
- **Conditional logic:** Validate business rules
- **Looping:** Process multiple records efficiently

Batch Apex (Data Maintenance)



```
1 public class MonthlyAdoptionReportBatch implements Database.Batchable<SObject>, Database.Stateful {
2
3     public Integer totalAdoptions = 0;
4     public Integer totalDonations = 0;
5     public Decimal totalRevenue = 0;
6
7     public Database.QueryLocator start(Database.BatchableContext bc) {
8         // Get adoption requests from last month
9         Date lastMonth = Date.today().addMonths(-1);
10        return Database.getQueryLocator([
11            SELECT Id, Adoption_Fee__c, Request_Date__c, Status__c
12            FROM Adoption_Request__c
13            WHERE Request_Date__c = LAST_MONTH
14            AND Status__c = 'Approved'
15        ]);
16    }
17
18    public void execute(Database.BatchableContext bc, List<Adoption_Request__c> scope) {
19        List<Adoption_Request__c> successfulAdoptions = new List<Adoption_Request__c>();
20
21        for(Adoption_Request__c request : scope) {
22            if(request.Adoption_Fee__c > 0) {
23                // ...
24            }
25        }
26    }
27 }
```

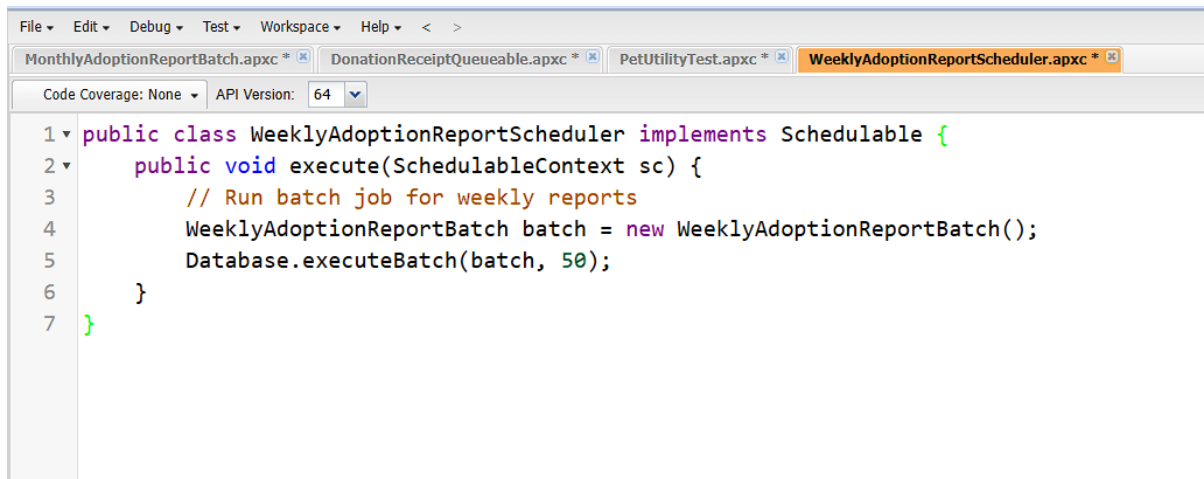
Queueable Apex



```
1 public class DonationReceiptQueueable implements Queueable {
2     private List<Id> donationIds;
3
4     public DonationReceiptQueueable(List<Id> donationIds) {
5         this.donationIds = donationIds;
6     }
7
8     public void execute(QueueableContext context) {
9         // Process donations and send receipts asynchronously
10        List<Donation__c> donations = [
11            SELECT Id, Donation_Amount__c, Adopter__c.r.Email__c, Donation_Date__c
12            FROM Donation__c
13            WHERE Id IN :donationIds AND Status__c = 'approved'
14        ];
15
16        sendDonationReceipts(donations);
17
18        // Chain next job if needed
19        if(!donations.isEmpty()) {
20            System.enqueueJob(new ThankYouEmailQueueable(donationIds));
21        }
22    }
23 }
```

- **Donation receipt chains:** Send receipt → Thank you email → Follow-up
- **Adoption process chains:** Approval → Notification → Welcome kit
- **Volunteer shift reminders**

Scheduled Apex



```

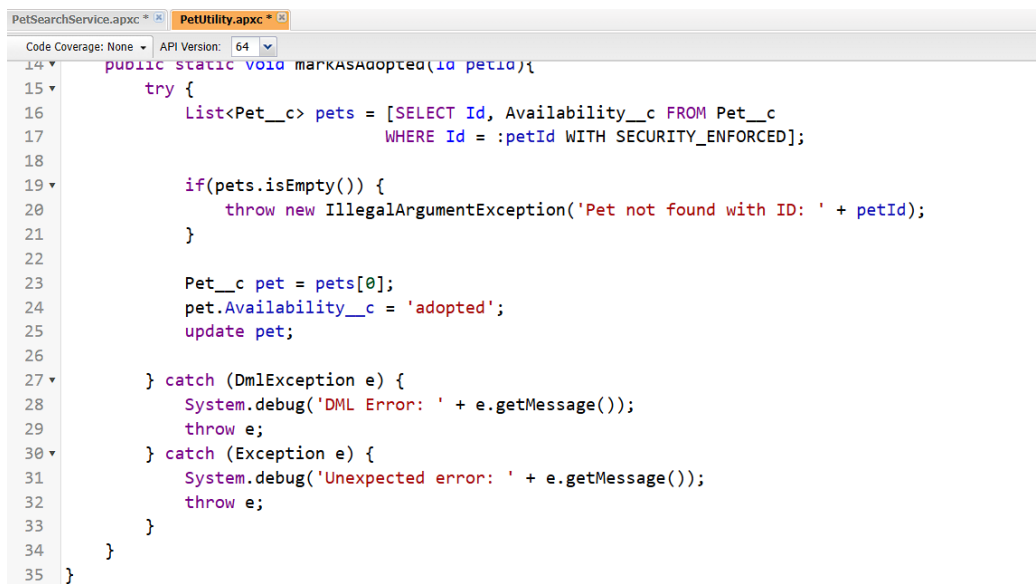
1 public class WeeklyAdoptionReportScheduler implements Schedulable {
2     public void execute(SchedulableContext sc) {
3         // Run batch job for weekly reports
4         WeeklyAdoptionReportBatch batch = new WeeklyAdoptionReportBatch();
5         Database.executeBatch(batch, 50);
6     }
7 }

```

Scheduled Apex Needed For:

- **Daily shelter cleanup tasks**
- **Weekly adoption statistics**
- **Monthly volunteer hour calculations**

Exception Handling



```

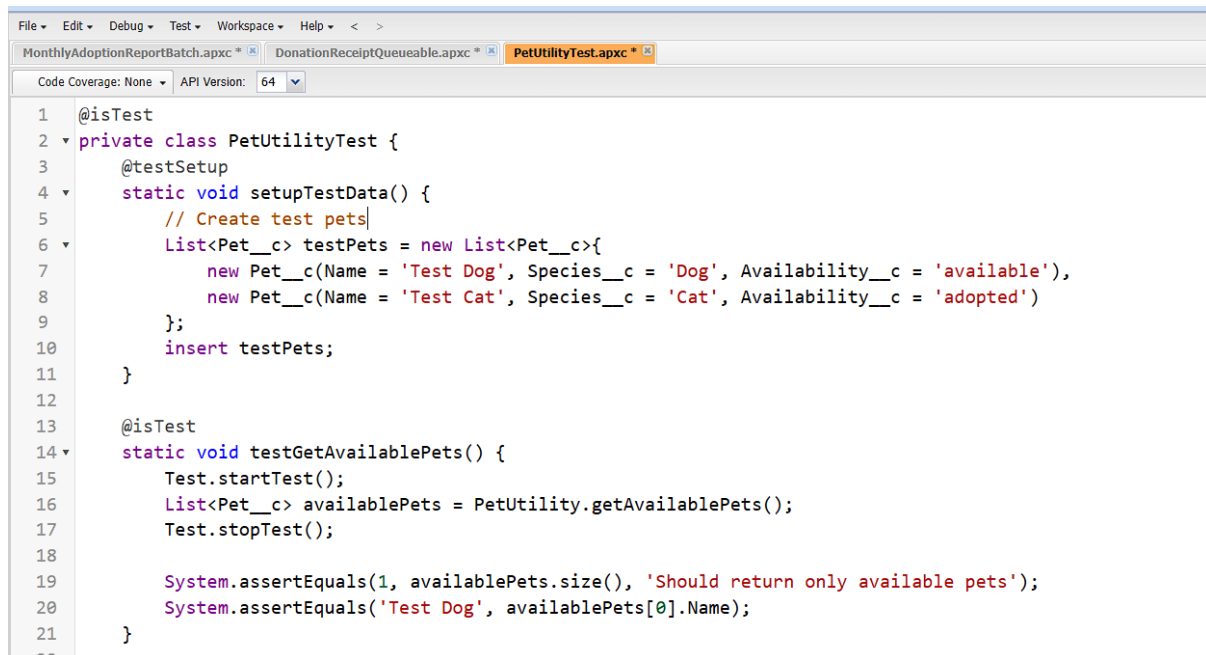
14 public static void markAsAdopted(id petId){
15     try {
16         List<Pet__c> pets = [SELECT Id, Availability__c FROM Pet__c
17                             WHERE Id = :petId WITH SECURITY_ENFORCED];
18
19         if(pets.isEmpty()) {
20             throw new IllegalArgumentException('Pet not found with ID: ' + petId);
21         }
22
23         Pet__c pet = pets[0];
24         pet.Availability__c = 'adopted';
25         update pet;
26
27     } catch (DmlException e) {
28         System.debug('DML Error: ' + e.getMessage());
29         throw e;
30     } catch (Exception e) {
31         System.debug('Unexpected error: ' + e.getMessage());
32         throw e;
33     }
34 }
35 }

```

Exception Handling Needed For:

- **Prevent system crashes** when data is missing
- **Graceful error messages** for users
- **Error logging** for debugging

Test Classes



```
1  @isTest
2  private class PetUtilityTest {
3      @testSetup
4      static void setupTestData() {
5          // Create test pets
6          List<Pet__c> testPets = new List<Pet__c>{
7              new Pet__c(Name = 'Test Dog', Species__c = 'Dog', Availability__c = 'available'),
8              new Pet__c(Name = 'Test Cat', Species__c = 'Cat', Availability__c = 'adopted')
9          };
10         insert testPets;
11     }
12
13     @isTest
14     static void testGetAvailablePets() {
15         Test.startTest();
16         List<Pet__c> availablePets = PetUtility.getAvailablePets();
17         Test.stopTest();
18
19         System.assertEquals(1, availablePets.size(), 'Should return only available pets');
20         System.assertEquals('Test Dog', availablePets[0].Name);
21     }
22 }
```

Deployment to production

- **Code quality assurance**
- **Preventing bugs in live system**