**Student Registration Portal - Phase 5: Apex Programming (Developer)**

**Apex Classes**

**1. StudentRegistrationController.cls**

/\*\*  
 \* Controller class for Student Registration functionality  
 \* Handles student registration, course enrollment, and application processing  
 \*/  
public with sharing class StudentRegistrationController {  
  
 /\*\*  
 \* Register a new student with application  
 \* @param studentData Map containing student information  
 \* @param applicationData Map containing application information  
 \* @return String applicationId or error message  
 \*/  
 @AuraEnabled  
 public static String registerStudent(Map<String, Object> studentData,   
 Map<String, Object> applicationData) {  
 try {  
 // Create Student record  
 Student\_\_c newStudent = new Student\_\_c();  
 newStudent.First\_Name\_\_c = (String) studentData.get('firstName');  
 newStudent.Last\_Name\_\_c = (String) studentData.get('lastName');  
 newStudent.Email\_\_c = (String) studentData.get('email');  
 newStudent.Phone\_\_c = (String) studentData.get('phone');  
 newStudent.Date\_of\_Birth\_\_c = Date.valueOf((String) studentData.get('dateOfBirth'));  
 newStudent.Address\_\_c = (String) studentData.get('address');  
 newStudent.Previous\_Education\_\_c = (String) studentData.get('previousEducation');  
 newStudent.Application\_Status\_\_c = 'Draft';  
  
 insert newStudent;  
  
 // Create Application record  
 Application\_\_c newApplication = new Application\_\_c();  
 newApplication.Student\_\_c = newStudent.Id;  
 newApplication.Intended\_Program\_\_c = (String) applicationData.get('program');  
 newApplication.Intended\_Major\_\_c = (String) applicationData.get('major');  
 newApplication.Previous\_GPA\_\_c = (Decimal) applicationData.get('gpa');  
 newApplication.Personal\_Statement\_\_c = (String) applicationData.get('personalStatement');  
 newApplication.Review\_Status\_\_c = 'Submitted';  
  
 insert newApplication;  
  
 return newApplication.Application\_Number\_\_c;  
  
 } catch (Exception e) {  
 throw new AuraHandledException('Error registering student: ' + e.getMessage());  
 }  
 }  
  
 /\*\*  
 \* Get available courses for enrollment  
 \* @param program Student's program (Undergraduate/Graduate)  
 \* @param semester Target semester  
 \* @return List of available Course\_\_c records  
 \*/  
 @AuraEnabled(cacheable=true)  
 public static List<Course\_\_c> getAvailableCourses(String program, String semester) {  
 try {  
 String courseCodePrefix = program == 'Graduate' ? 'GRAD%' : '%';  
  
 return [SELECT Id, Course\_Code\_\_c, Course\_Name\_\_c, Credits\_\_c,   
 Department\_\_c, Course\_Fee\_\_c, Max\_Capacity\_\_c,   
 Current\_Enrollment\_\_c, Status\_\_c  
 FROM Course\_\_c   
 WHERE Semester\_\_c = :semester   
 AND Status\_\_c = 'Active'  
 AND Course\_Code\_\_c LIKE :courseCodePrefix  
 AND Current\_Enrollment\_\_c < Max\_Capacity\_\_c  
 ORDER BY Department\_\_c, Course\_Code\_\_c];  
  
 } catch (Exception e) {  
 throw new AuraHandledException('Error fetching courses: ' + e.getMessage());  
 }  
 }  
  
 /\*\*  
 \* Enroll student in selected courses  
 \* @param studentId Student ID  
 \* @param courseIds List of Course IDs  
 \* @return Map with success status and enrollment details  
 \*/  
 @AuraEnabled  
 public static Map<String, Object> enrollInCourses(Id studentId, List<Id> courseIds) {  
 List<Enrollment\_\_c> enrollments = new List<Enrollment\_\_c>();  
 Map<String, Object> result = new Map<String, Object>();  
  
 try {  
 // Check for existing enrollments  
 List<Enrollment\_\_c> existingEnrollments = [  
 SELECT Course\_\_c FROM Enrollment\_\_c   
 WHERE Student\_\_c = :studentId AND Course\_\_c IN :courseIds  
 ];  
  
 Set<Id> existingCourseIds = new Set<Id>();  
 for (Enrollment\_\_c enrollment : existingEnrollments) {  
 existingCourseIds.add(enrollment.Course\_\_c);  
 }  
  
 Decimal totalFee = 0;  
 List<String> enrolledCourses = new List<String>();  
  
 for (Id courseId : courseIds) {  
 if (!existingCourseIds.contains(courseId)) {  
 Enrollment\_\_c enrollment = new Enrollment\_\_c();  
 enrollment.Student\_\_c = studentId;  
 enrollment.Course\_\_c = courseId;  
 enrollment.Enrollment\_Status\_\_c = 'Registered';  
 enrollment.Payment\_Status\_\_c = 'Pending';  
  
 enrollments.add(enrollment);  
 }  
 }  
  
 if (!enrollments.isEmpty()) {  
 insert enrollments;  
  
 // Calculate total fees  
 for (Course\_\_c course : [SELECT Course\_Name\_\_c, Course\_Fee\_\_c   
 FROM Course\_\_c WHERE Id IN :courseIds]) {  
 totalFee += course.Course\_Fee\_\_c;  
 enrolledCourses.add(course.Course\_Name\_\_c);  
 }  
 }  
  
 result.put('success', true);  
 result.put('enrollmentCount', enrollments.size());  
 result.put('totalFee', totalFee);  
 result.put('enrolledCourses', enrolledCourses);  
  
 } catch (Exception e) {  
 result.put('success', false);  
 result.put('error', e.getMessage());  
 }  
  
 return result;  
 }  
}

**2. CourseEnrollmentService.cls**

/\*\*  
 \* Service class for Course Enrollment business logic  
 \* Handles enrollment calculations, capacity management, and prerequisites  
 \*/  
public with sharing class CourseEnrollmentService {  
  
 /\*\*  
 \* Check course prerequisites for student  
 \* @param studentId Student ID  
 \* @param courseId Course ID to check prerequisites for  
 \* @return Boolean indicating if prerequisites are met  
 \*/  
 public static Boolean checkPrerequisites(Id studentId, Id courseId) {  
 Course\_\_c course = [SELECT Prerequisites\_\_c FROM Course\_\_c WHERE Id = :courseId LIMIT 1];  
  
 if (String.isBlank(course.Prerequisites\_\_c)) {  
 return true; // No prerequisites required  
 }  
  
 // Parse prerequisites and check completed courses  
 List<String> prerequisiteCodes = course.Prerequisites\_\_c.split(',');  
 List<String> completedCourses = getCompletedCourses(studentId);  
  
 for (String prereq : prerequisiteCodes) {  
 if (!completedCourses.contains(prereq.trim())) {  
 return false;  
 }  
 }  
  
 return true;  
 }  
  
 /\*\*  
 \* Get completed courses for student  
 \* @param studentId Student ID  
 \* @return List of completed course codes  
 \*/  
 private static List<String> getCompletedCourses(Id studentId) {  
 List<String> completedCourses = new List<String>();  
  
 for (Enrollment\_\_c enrollment : [SELECT Course\_\_r.Course\_Code\_\_c   
 FROM Enrollment\_\_c   
 WHERE Student\_\_c = :studentId   
 AND Enrollment\_Status\_\_c = 'Completed'  
 AND Grade\_\_c IN ('A+','A','B+','B','C+','C')]) {  
 completedCourses.add(enrollment.Course\_\_r.Course\_Code\_\_c);  
 }  
  
 return completedCourses;  
 }  
  
 /\*\*  
 \* Calculate enrollment fees with discounts  
 \* @param studentId Student ID  
 \* @param courseIds List of Course IDs  
 \* @return Decimal total amount after discounts  
 \*/  
 public static Decimal calculateEnrollmentFees(Id studentId, List<Id> courseIds) {  
 Decimal totalFees = 0;  
 Decimal discount = 0;  
  
 // Get course fees  
 for (Course\_\_c course : [SELECT Course\_Fee\_\_c FROM Course\_\_c WHERE Id IN :courseIds]) {  
 totalFees += course.Course\_Fee\_\_c;  
 }  
  
 // Apply discounts based on enrollment count  
 Integer enrollmentCount = courseIds.size();  
 if (enrollmentCount >= 5) {  
 discount = totalFees \* 0.10; // 10% discount for 5+ courses  
 } else if (enrollmentCount >= 3) {  
 discount = totalFees \* 0.05; // 5% discount for 3+ courses  
 }  
  
 return totalFees - discount;  
 }  
  
 /\*\*  
 \* Update course capacity after enrollment  
 \* @param courseIds List of Course IDs to update  
 \*/  
 public static void updateCourseCapacity(List<Id> courseIds) {  
 List<Course\_\_c> coursesToUpdate = new List<Course\_\_c>();  
  
 // Aggregate enrollment counts  
 Map<Id, Integer> courseEnrollmentCounts = new Map<Id, Integer>();  
  
 for (AggregateResult ar : [SELECT Course\_\_c, COUNT(Id) enrollmentCount  
 FROM Enrollment\_\_c   
 WHERE Course\_\_c IN :courseIds   
 AND Enrollment\_Status\_\_c = 'Registered'  
 GROUP BY Course\_\_c]) {  
 courseEnrollmentCounts.put((Id)ar.get('Course\_\_c'), (Integer)ar.get('enrollmentCount'));  
 }  
  
 // Update course records  
 for (Course\_\_c course : [SELECT Id, Current\_Enrollment\_\_c, Max\_Capacity\_\_c, Status\_\_c   
 FROM Course\_\_c WHERE Id IN :courseIds]) {  
 Integer currentCount = courseEnrollmentCounts.get(course.Id);  
 if (currentCount != null) {  
 course.Current\_Enrollment\_\_c = currentCount;  
 course.Status\_\_c = (currentCount >= course.Max\_Capacity\_\_c) ? 'Full' : 'Active';  
 coursesToUpdate.add(course);  
 }  
 }  
  
 if (!coursesToUpdate.isEmpty()) {  
 update coursesToUpdate;  
 }  
 }  
}

**3. ApplicationReviewHelper.cls**

/\*\*  
 \* Helper class for Application Review processes  
 \* Contains utility methods for application processing and scoring  
 \*/  
public with sharing class ApplicationReviewHelper {  
  
 /\*\*  
 \* Calculate application score based on criteria  
 \* @param applicationId Application ID  
 \* @return Decimal application score (0-100)  
 \*/  
 public static Decimal calculateApplicationScore(Id applicationId) {  
 Application\_\_c app = [SELECT Previous\_GPA\_\_c, Test\_Scores\_\_c,   
 Recommendation\_Letters\_\_c, Personal\_Statement\_\_c  
 FROM Application\_\_c WHERE Id = :applicationId];  
  
 Decimal score = 0;  
  
 // GPA Score (40% weight)  
 if (app.Previous\_GPA\_\_c != null) {  
 score += (app.Previous\_GPA\_\_c / 4.0) \* 40;  
 }  
  
 // Test Scores (30% weight) - assuming standardized format  
 if (!String.isBlank(app.Test\_Scores\_\_c)) {  
 // Parse test score (simplified logic)  
 score += parseTestScore(app.Test\_Scores\_\_c) \* 30;  
 }  
  
 // Recommendation Letters (15% weight)  
 if (app.Recommendation\_Letters\_\_c != null) {  
 score += Math.min(app.Recommendation\_Letters\_\_c \* 5, 15);  
 }  
  
 // Personal Statement (15% weight)  
 if (!String.isBlank(app.Personal\_Statement\_\_c)) {  
 score += Math.min(app.Personal\_Statement\_\_c.length() / 100, 15);  
 }  
  
 return Math.round(score);  
 }  
  
 /\*\*  
 \* Parse test score from string format  
 \* @param testScore Test score string  
 \* @return Decimal normalized score (0-1)  
 \*/  
 private static Decimal parseTestScore(String testScore) {  
 // Simplified parsing - assumes format like "SAT: 1400" or "GRE: 320"  
 if (testScore.contains('SAT')) {  
 Decimal satScore = extractNumericValue(testScore);  
 return satScore != null ? Math.min(satScore / 1600, 1) : 0;  
 } else if (testScore.contains('GRE')) {  
 Decimal greScore = extractNumericValue(testScore);  
 return greScore != null ? Math.min(greScore / 340, 1) : 0;  
 }  
 return 0;  
 }  
  
 /\*\*  
 \* Extract numeric value from string  
 \* @param input String containing numeric value  
 \* @return Decimal numeric value  
 \*/  
 private static Decimal extractNumericValue(String input) {  
 Pattern numPattern = Pattern.compile('\d+');  
 Matcher matcher = numPattern.matcher(input);  
 if (matcher.find()) {  
 return Decimal.valueOf(matcher.group());  
 }  
 return null;  
 }  
  
 /\*\*  
 \* Auto-assign applications to reviewers based on workload  
 \* @param applicationIds List of Application IDs to assign  
 \*/  
 public static void assignApplicationReviewers(List<Id> applicationIds) {  
 List<Application\_\_c> applications = [SELECT Id, Intended\_Major\_\_c, Reviewer\_\_c   
 FROM Application\_\_c   
 WHERE Id IN :applicationIds   
 AND Reviewer\_\_c = null];  
  
 // Get available reviewers by department  
 Map<String, List<User>> reviewersByDepartment = getReviewersByDepartment();  
  
 // Get current workload for reviewers  
 Map<Id, Integer> reviewerWorkload = getReviewerWorkload();  
  
 List<Application\_\_c> applicationsToUpdate = new List<Application\_\_c>();  
  
 for (Application\_\_c app : applications) {  
 List<User> departmentReviewers = reviewersByDepartment.get(app.Intended\_Major\_\_c);  
  
 if (departmentReviewers != null && !departmentReviewers.isEmpty()) {  
 // Find reviewer with lowest workload  
 User selectedReviewer = null;  
 Integer minWorkload = Integer.MAX\_VALUE;  
  
 for (User reviewer : departmentReviewers) {  
 Integer currentWorkload = reviewerWorkload.get(reviewer.Id);  
 if (currentWorkload == null) currentWorkload = 0;  
  
 if (currentWorkload < minWorkload) {  
 minWorkload = currentWorkload;  
 selectedReviewer = reviewer;  
 }  
 }  
  
 if (selectedReviewer != null) {  
 app.Reviewer\_\_c = selectedReviewer.Id;  
 applicationsToUpdate.add(app);  
  
 // Update workload count  
 reviewerWorkload.put(selectedReviewer.Id, minWorkload + 1);  
 }  
 }  
 }  
  
 if (!applicationsToUpdate.isEmpty()) {  
 update applicationsToUpdate;  
 }  
 }  
  
 /\*\*  
 \* Get reviewers organized by department  
 \* @return Map of department to list of reviewers  
 \*/  
 private static Map<String, List<User>> getReviewersByDepartment() {  
 Map<String, List<User>> reviewersByDept = new Map<String, List<User>>();  
  
 for (User reviewer : [SELECT Id, Name, Department FROM User   
 WHERE Profile.Name = 'Application Reviewer'   
 AND IsActive = true]) {  
 if (!reviewersByDept.containsKey(reviewer.Department)) {  
 reviewersByDept.put(reviewer.Department, new List<User>());  
 }  
 reviewersByDept.get(reviewer.Department).add(reviewer);  
 }  
  
 return reviewersByDept;  
 }  
  
 /\*\*  
 \* Get current workload for reviewers  
 \* @return Map of reviewer ID to current application count  
 \*/  
 private static Map<Id, Integer> getReviewerWorkload() {  
 Map<Id, Integer> workloadMap = new Map<Id, Integer>();  
  
 for (AggregateResult ar : [SELECT Reviewer\_\_c, COUNT(Id) appCount  
 FROM Application\_\_c   
 WHERE Review\_Status\_\_c = 'Under Review'  
 AND Reviewer\_\_c != null  
 GROUP BY Reviewer\_\_c]) {  
 workloadMap.put((Id)ar.get('Reviewer\_\_c'), (Integer)ar.get('appCount'));  
 }  
  
 return workloadMap;  
 }  
}

**Apex Triggers**

**1. StudentTrigger.trigger**

/\*\*  
 \* Trigger for Student\_\_c object  
 \* Handles student record lifecycle events  
 \*/  
trigger StudentTrigger on Student\_\_c (before insert, before update, after insert, after update) {  
  
 if (Trigger.isBefore) {  
 StudentTriggerHandler.handleBeforeEvents(Trigger.new, Trigger.oldMap);  
 }  
  
 if (Trigger.isAfter) {  
 StudentTriggerHandler.handleAfterEvents(Trigger.new, Trigger.oldMap, Trigger.isInsert, Trigger.isUpdate);  
 }  
}

**2. StudentTriggerHandler.cls**

/\*\*  
 \* Handler class for Student trigger  
 \* Implements trigger design pattern for better maintainability  
 \*/  
public with sharing class StudentTriggerHandler {  
  
 /\*\*  
 \* Handle before insert and update events  
 \* @param newRecords List of new Student records  
 \* @param oldMap Map of old Student records  
 \*/  
 public static void handleBeforeEvents(List<Student\_\_c> newRecords, Map<Id, Student\_\_c> oldMap) {  
 validateStudentData(newRecords);  
 generateStudentIds(newRecords, oldMap);  
 }  
  
 /\*\*  
 \* Handle after insert and update events  
 \* @param newRecords List of new Student records  
 \* @param oldMap Map of old Student records  
 \* @param isInsert Boolean indicating if this is insert operation  
 \* @param isUpdate Boolean indicating if this is update operation  
 \*/  
 public static void handleAfterEvents(List<Student\_\_c> newRecords, Map<Id, Student\_\_c> oldMap,   
 Boolean isInsert, Boolean isUpdate) {  
 if (isInsert) {  
 createStudentPortalAccounts(newRecords);  
 }  
  
 if (isUpdate) {  
 handleStatusChanges(newRecords, oldMap);  
 }  
 }  
  
 /\*\*  
 \* Validate student data before saving  
 \* @param students List of Student records  
 \*/  
 private static void validateStudentData(List<Student\_\_c> students) {  
 for (Student\_\_c student : students) {  
 // Additional validation beyond validation rules  
 if (!String.isBlank(student.Email\_\_c)) {  
 if (!student.Email\_\_c.endsWith('@university.edu') &&   
 !student.Email\_\_c.endsWith('@student.university.edu')) {  
 student.addError('Email must be a valid university domain');  
 }  
 }  
  
 // Validate age calculation  
 if (student.Date\_of\_Birth\_\_c != null) {  
 Integer age = Date.today().year() - student.Date\_of\_Birth\_\_c.year();  
 if (age < 16 || age > 100) {  
 student.Date\_of\_Birth\_\_c.addError('Invalid date of birth');  
 }  
 }  
 }  
 }  
  
 /\*\*  
 \* Generate Student IDs for new records  
 \* @param students List of Student records  
 \* @param oldMap Map of old records for update context  
 \*/  
 private static void generateStudentIds(List<Student\_\_c> students, Map<Id, Student\_\_c> oldMap) {  
 for (Student\_\_c student : students) {  
 if (String.isBlank(student.Student\_ID\_\_c) &&   
 (oldMap == null || String.isBlank(oldMap.get(student.Id)?.Student\_ID\_\_c))) {  
 // Student ID will be auto-generated by custom number field  
 // Additional logic can be added here if needed  
 }  
 }  
 }  
  
 /\*\*  
 \* Create portal user accounts for approved students  
 \* @param students List of new Student records  
 \*/  
 private static void createStudentPortalAccounts(List<Student\_\_c> students) {  
 List<User> usersToCreate = new List<User>();  
  
 for (Student\_\_c student : students) {  
 if (student.Application\_Status\_\_c == 'Approved') {  
 User portalUser = new User();  
 portalUser.FirstName = student.First\_Name\_\_c;  
 portalUser.LastName = student.Last\_Name\_\_c;  
 portalUser.Email = student.Email\_\_c;  
 portalUser.Username = student.Email\_\_c + '.student';  
 portalUser.Alias = (student.First\_Name\_\_c.substring(0,1) +   
 student.Last\_Name\_\_c.substring(0,1)).toLowerCase();  
 portalUser.TimeZoneSidKey = 'America/New\_York';  
 portalUser.LocaleSidKey = 'en\_US';  
 portalUser.EmailEncodingKey = 'UTF-8';  
 portalUser.LanguageLocaleKey = 'en\_US';  
 portalUser.ProfileId = getStudentProfileId();  
  
 usersToCreate.add(portalUser);  
 }  
 }  
  
 if (!usersToCreate.isEmpty()) {  
 try {  
 insert usersToCreate;  
 } catch (Exception e) {  
 System.debug('Error creating portal users: ' + e.getMessage());  
 }  
 }  
 }  
  
 /\*\*  
 \* Handle application status changes  
 \* @param students List of Student records  
 \* @param oldMap Map of old Student records  
 \*/  
 private static void handleStatusChanges(List<Student\_\_c> students, Map<Id, Student\_\_c> oldMap) {  
 List<String> statusChangeNotifications = new List<String>();  
  
 for (Student\_\_c student : students) {  
 Student\_\_c oldStudent = oldMap.get(student.Id);  
  
 if (student.Application\_Status\_\_c != oldStudent.Application\_Status\_\_c) {  
 // Log status change  
 statusChangeNotifications.add('Student ' + student.Student\_ID\_\_c +   
 ' status changed from ' + oldStudent.Application\_Status\_\_c +   
 ' to ' + student.Application\_Status\_\_c);  
 }  
 }  
  
 // Send notifications if needed  
 if (!statusChangeNotifications.isEmpty()) {  
 sendStatusChangeNotifications(statusChangeNotifications);  
 }  
 }  
  
 /\*\*  
 \* Get Student Profile ID  
 \* @return Id of Student Portal profile  
 \*/  
 private static Id getStudentProfileId() {  
 Profile studentProfile = [SELECT Id FROM Profile WHERE Name = 'Student Portal User' LIMIT 1];  
 return studentProfile.Id;  
 }  
  
 /\*\*  
 \* Send status change notifications  
 \* @param notifications List of notification messages  
 \*/  
 private static void sendStatusChangeNotifications(List<String> notifications) {  
 // Implementation for sending notifications  
 // Could integrate with custom notification system or email  
 System.debug('Status change notifications: ' + notifications);  
 }  
}

**3. EnrollmentTrigger.trigger & Handler**

/\*\*  
 \* Trigger for Enrollment\_\_c object  
 \*/  
trigger EnrollmentTrigger on Enrollment\_\_c (after insert, after update, after delete) {  
  
 if (Trigger.isAfter) {  
 if (Trigger.isInsert || Trigger.isUpdate) {  
 EnrollmentTriggerHandler.updateCourseEnrollmentCounts(Trigger.new);  
 }  
  
 if (Trigger.isDelete) {  
 EnrollmentTriggerHandler.updateCourseEnrollmentCounts(Trigger.old);  
 }  
 }  
}  
  
/\*\*  
 \* Handler for Enrollment trigger  
 \*/  
public with sharing class EnrollmentTriggerHandler {  
  
 /\*\*  
 \* Update course enrollment counts when enrollments change  
 \* @param enrollments List of Enrollment records  
 \*/  
 public static void updateCourseEnrollmentCounts(List<Enrollment\_\_c> enrollments) {  
 Set<Id> courseIds = new Set<Id>();  
  
 for (Enrollment\_\_c enrollment : enrollments) {  
 courseIds.add(enrollment.Course\_\_c);  
 }  
  
 // Use service class method  
 CourseEnrollmentService.updateCourseCapacity(new List<Id>(courseIds));  
 }  
}

**SOQL & SOSL Queries**

**Complex SOQL Examples**

// 1. Get student enrollment summary with course details  
public static List<StudentEnrollmentSummary> getStudentEnrollmentSummary(Id studentId) {  
 return [SELECT Id, Student\_\_r.Name, Student\_\_r.Student\_ID\_\_c,  
 Course\_\_r.Course\_Code\_\_c, Course\_\_r.Course\_Name\_\_c,  
 Course\_\_r.Credits\_\_c, Enrollment\_Status\_\_c, Grade\_\_c,  
 Payment\_Status\_\_c, Payment\_Amount\_\_c  
 FROM Enrollment\_\_c   
 WHERE Student\_\_c = :studentId  
 ORDER BY Course\_\_r.Course\_Code\_\_c];  
}  
  
// 2. Department-wise course enrollment statistics  
public static List<AggregateResult> getDepartmentEnrollmentStats() {  
 return [SELECT Course\_\_r.Department\_\_c dept,  
 COUNT(Id) totalEnrollments,  
 AVG(Course\_\_r.Credits\_\_c) avgCredits,  
 SUM(Payment\_Amount\_\_c) totalRevenue  
 FROM Enrollment\_\_c   
 WHERE Enrollment\_Status\_\_c = 'Registered'  
 GROUP BY Course\_\_r.Department\_\_c  
 ORDER BY COUNT(Id) DESC];  
}  
  
// 3. Find students with pending applications  
public static List<Student\_\_c> getStudentsWithPendingApplications() {  
 return [SELECT Id, Name, Email\_\_c, Application\_Status\_\_c,  
 (SELECT Application\_Number\_\_c, Review\_Status\_\_c,   
 Intended\_Program\_\_c, Intended\_Major\_\_c  
 FROM Applications\_\_r   
 WHERE Review\_Status\_\_c IN ('Submitted', 'Under Review')  
 ORDER BY Application\_Date\_\_c DESC)  
 FROM Student\_\_c   
 WHERE Application\_Status\_\_c IN ('Submitted', 'Under Review')];  
}

**SOSL Examples**

// Search across multiple objects for student information  
public static List<List<SObject>> searchStudentInfo(String searchTerm) {  
 return [FIND :searchTerm IN ALL FIELDS   
 RETURNING Student\_\_c(Id, Name, Email\_\_c, Student\_ID\_\_c),  
 Application\_\_c(Application\_Number\_\_c, Intended\_Major\_\_c),  
 Enrollment\_\_c(Id, Course\_\_r.Course\_Name\_\_c)];  
}

**Collections and Data Structures**

**Map Usage Examples**

// Using Maps for efficient data processing  
public static Map<Id, Decimal> calculateStudentTotalFees(List<Id> studentIds) {  
 Map<Id, Decimal> studentFeeMap = new Map<Id, Decimal>();  
  
 for (AggregateResult ar : [SELECT Student\_\_c, SUM(Payment\_Amount\_\_c) totalFee  
 FROM Enrollment\_\_c   
 WHERE Student\_\_c IN :studentIds  
 GROUP BY Student\_\_c]) {  
 studentFeeMap.put((Id)ar.get('Student\_\_c'), (Decimal)ar.get('totalFee'));  
 }  
  
 return studentFeeMap;  
}  
  
// Using Set for deduplication  
public static Set<String> getUniqueDepartments(List<Course\_\_c> courses) {  
 Set<String> departments = new Set<String>();  
  
 for (Course\_\_c course : courses) {  
 if (!String.isBlank(course.Department\_\_c)) {  
 departments.add(course.Department\_\_c);  
 }  
 }  
  
 return departments;  
}

**Exception Handling**

**Custom Exception Classes**

// Custom exception for enrollment-related errors  
public class EnrollmentException extends Exception {  
 public String enrollmentId;  
 public String courseCode;  
  
 public EnrollmentException(String message, String enrollmentId, String courseCode) {  
 this(message);  
 this.enrollmentId = enrollmentId;  
 this.courseCode = courseCode;  
 }  
}  
  
// Usage in service methods  
public static void processEnrollment(Id enrollmentId) {  
 try {  
 // Enrollment processing logic  
 Enrollment\_\_c enrollment = [SELECT Id, Course\_\_r.Course\_Code\_\_c   
 FROM Enrollment\_\_c WHERE Id = :enrollmentId];  
  
 if (enrollment == null) {  
 throw new EnrollmentException('Enrollment not found', enrollmentId, null);  
 }  
  
 // Additional processing...  
  
 } catch (EnrollmentException e) {  
 System.debug('Enrollment Error: ' + e.getMessage() +   
 ' for Enrollment: ' + e.enrollmentId);  
 // Handle specific enrollment error  
  
 } catch (Exception e) {  
 System.debug('General Error in processEnrollment: ' + e.getMessage());  
 throw e; // Re-throw if cannot handle  
 }  
}