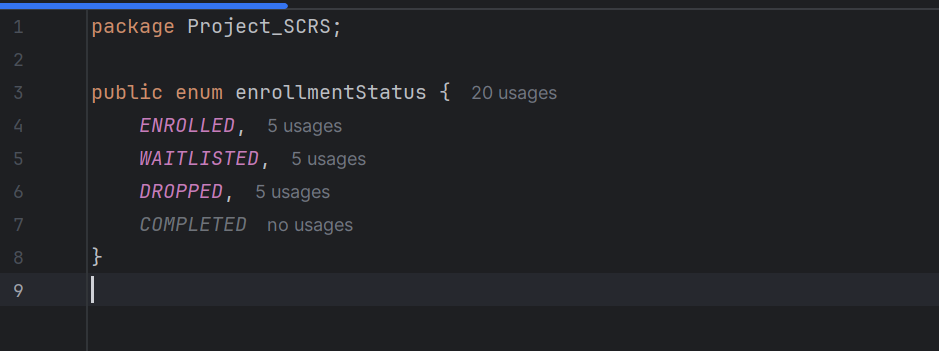
**Day 3 – Enrollment API’s**

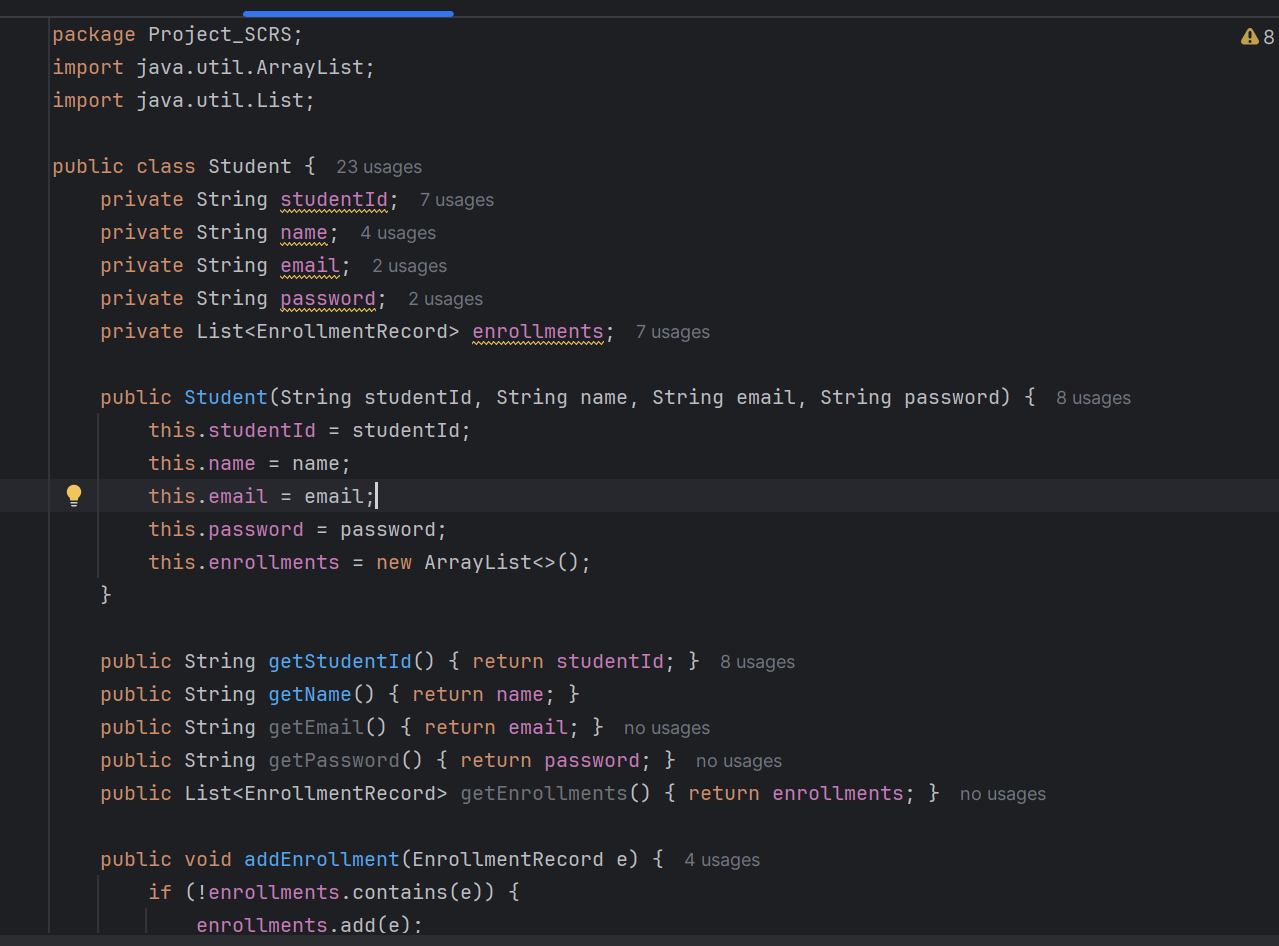
**EnrollmentStatus.java**

public enum enrollmentStatus {  
 *ENROLLED*,  
 *WAITLISTED*,  
 *DROPPED*,  
 *COMPLETED*}

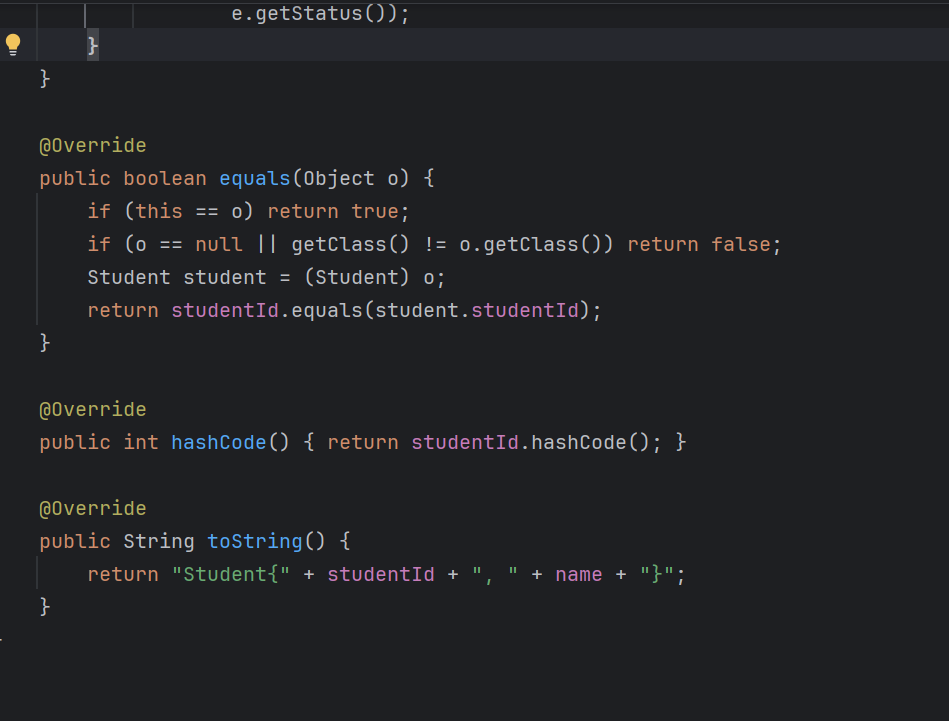


**Student.java**

import java.util.ArrayList;  
import java.util.List;  
  
public class Student {  
 private String studentId;  
 private String name;  
 private String email;  
 private String password;  
 private List<EnrollmentRecord> enrollments;  
  
 public Student(String studentId, String name, String email, String password) {  
 this.studentId = studentId;  
 this.name = name;  
 this.email = email;  
 this.password = password;  
 this.enrollments = new ArrayList<>();  
 }  
  
 public String getStudentId() { return studentId; }  
 public String getName() { return name; }  
 public String getEmail() { return email; }  
 public String getPassword() { return password; }  
 public List<EnrollmentRecord> getEnrollments() { return enrollments; }  
  
 public void addEnrollment(EnrollmentRecord e) {  
 if (!enrollments.contains(e)) {  
 enrollments.add(e);  
 }  
 }  
  
 public void removeEnrollment(EnrollmentRecord e) {  
 enrollments.remove(e);  
 }  
  
 public void viewDashboard() {  
 System.*out*.println("\nDashboard for student: " + studentId + " - " + name);  
 if (enrollments.isEmpty()) {  
 System.*out*.println(" No enrollments yet.");  
 return;  
 }  
 for (EnrollmentRecord e : enrollments) {  
 System.*out*.printf(" Course: %s (%s) - Status: %s%n",  
 e.getCourse().getCourseName(),  
 e.getCourse().getCourseId(),  
 e.getStatus());  
 }  
 }  
  
 @Override  
 public boolean equals(Object o) {  
 if (this == o) return true;  
 if (o == null || getClass() != o.getClass()) return false;  
 Student student = (Student) o;  
 return studentId.equals(student.studentId);  
 }  
  
 @Override  
 public int hashCode() { return studentId.hashCode(); }  
  
 @Override  
 public String toString() {  
 return "Student{" + studentId + ", " + name + "}";  
 }  
}

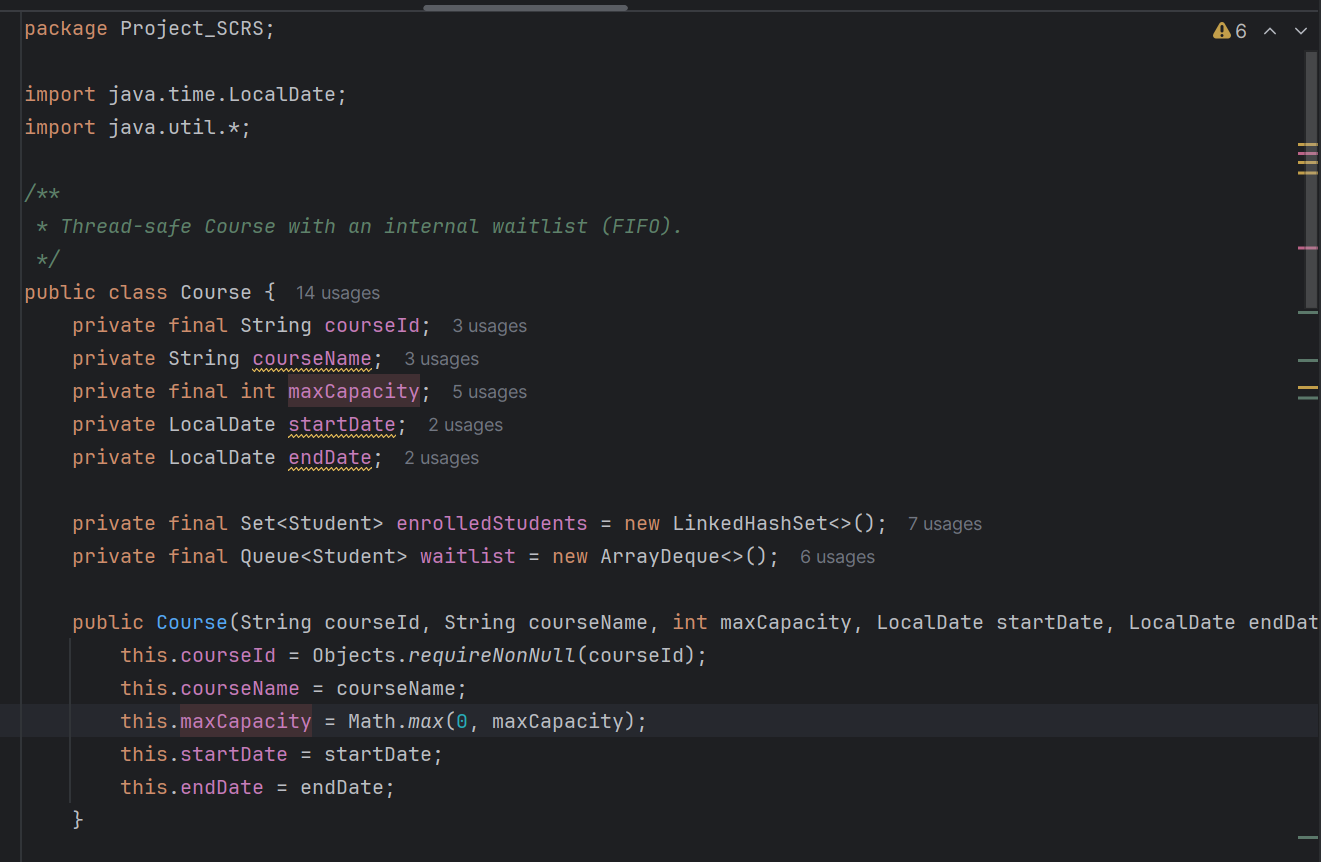


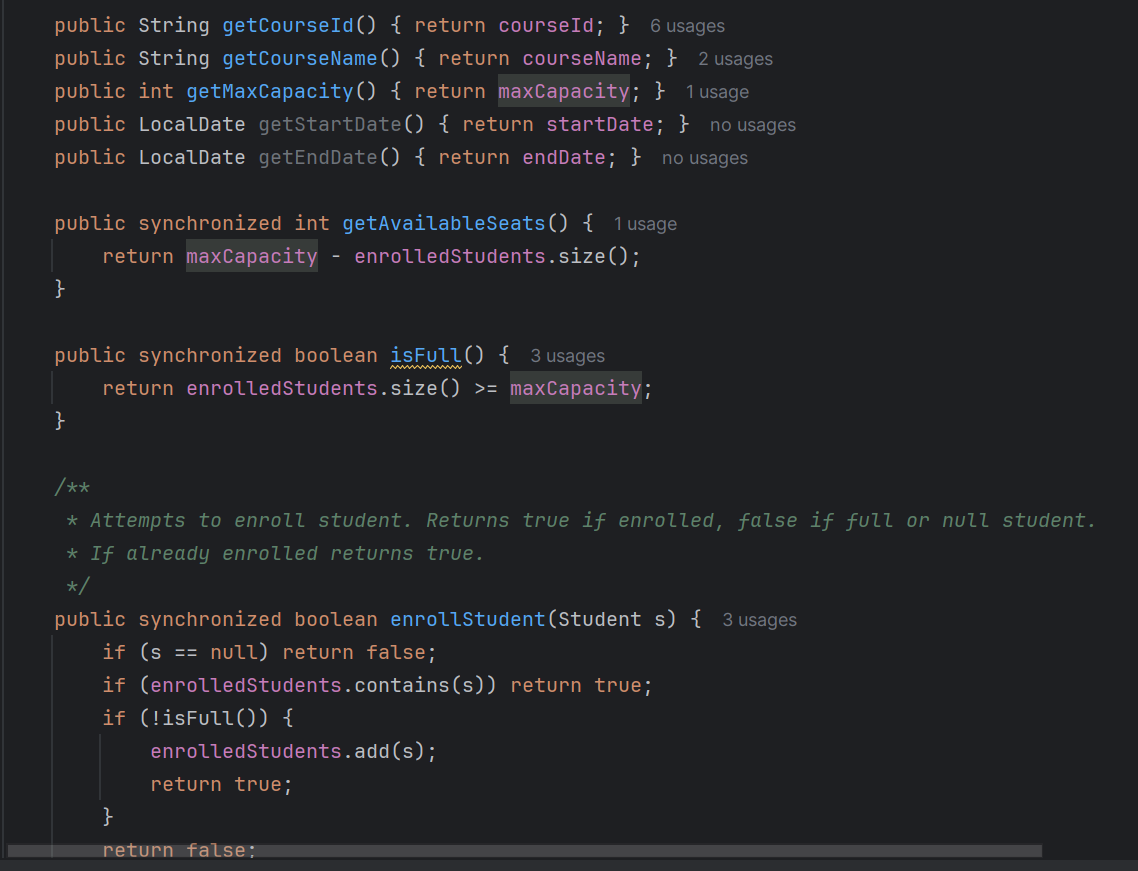


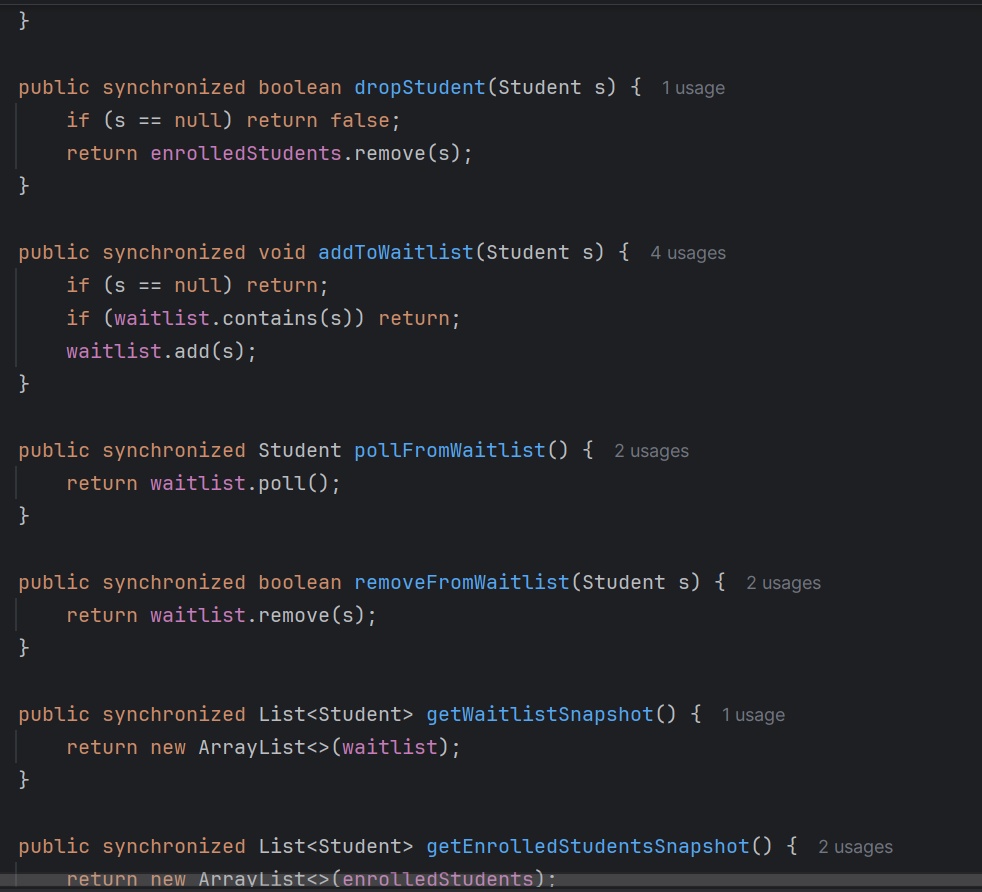


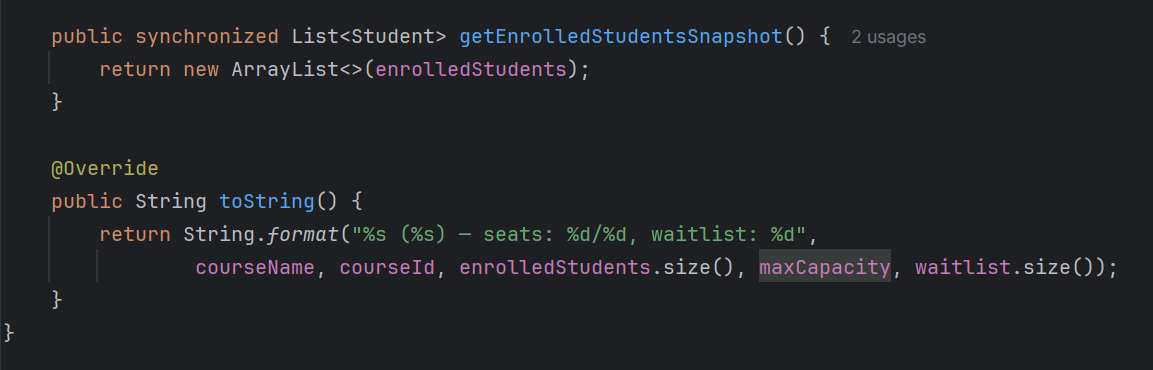
**Course.java**

import java.time.LocalDate;  
import java.util.\*;  
  
*/\*\*  
 \* Thread-safe Course class.  
 \* Combines simple Day3 fields with advanced Day4 features like enrolled students and waitlist.  
 \*/*public class Course {  
 private final String courseId;  
 private String name;  
 private final int capacity;  
 private int enrolledCount; // keeps Day3 compatibility  
 private LocalDate startDate;  
 private LocalDate endDate;  
  
 private final Set<Student> enrolledStudents = new LinkedHashSet<>();  
 private final Queue<Student> waitlist = new ArrayDeque<>();  
  
 public Course(String courseId, String name, int capacity, LocalDate startDate, LocalDate endDate) {  
 this.courseId = Objects.*requireNonNull*(courseId);  
 this.name = name;  
 this.capacity = Math.*max*(0, capacity);  
 this.enrolledCount = 0;  
 this.startDate = startDate;  
 this.endDate = endDate;  
 }  
  
 // --- Day3 getters ---  
 public String getCourseId() { return courseId; }  
 public String getCourseName() { return name; }  
 public int getMaxCapacity() { return capacity; }  
 public int getEnrolledCount() { return enrolledCount; }  
  
 public LocalDate getStartDate() {  
 return startDate;  
 }  
  
 public LocalDate getEndDate() {  
 return endDate;  
 }  
  
 public synchronized void incrementEnrolledCount() { enrolledCount++; }  
 public void decrementEnrolledCount() { enrolledCount--; }  
  
  
 // --- Advanced Day4 methods ---  
  
 public synchronized int getAvailableSeats() {  
 return capacity - enrolledStudents.size();  
 }  
  
 public synchronized boolean isFull() {  
 return enrolledStudents.size() >= capacity;  
 }  
  
 public synchronized boolean enrollStudent(Student s) {  
 if (s == null) return false;  
 if (enrolledStudents.contains(s)) return true;  
 if (!isFull()) {  
 enrolledStudents.add(s);  
 enrolledCount++; // keep Day3 counter updated  
 return true;  
 }  
 // if full, automatically add to waitlist  
 addToWaitlist(s);  
 return false;  
 }  
  
 public synchronized boolean dropStudent(Student s) {  
 if (s == null) return false;  
 boolean removed = enrolledStudents.remove(s);  
 if (removed) {  
 enrolledCount--; // update Day3 counter  
 // Promote next student from waitlist  
 if (!waitlist.isEmpty()) {  
 Student next = waitlist.poll();  
 enrollStudent(next);  
 }  
 }  
 return removed;  
 }  
  
 public synchronized void addToWaitlist(Student s) {  
 if (s == null) return;  
 if (waitlist.contains(s)) return;  
 waitlist.add(s);  
 }  
  
 public synchronized Student pollFromWaitlist() {  
 return waitlist.poll();  
 }  
  
 public synchronized boolean removeFromWaitlist(Student s) {  
 return waitlist.remove(s);  
 }  
  
 public synchronized List<Student> getWaitlistSnapshot() {  
 return Collections.*unmodifiableList*(new ArrayList<>(waitlist));  
 }  
  
 public synchronized List<Student> getEnrolledStudentsSnapshot() {  
 return Collections.*unmodifiableList*(new ArrayList<>(enrolledStudents));  
 }  
  
 @Override  
 public String toString() {  
 return String.*format*("%s (ID:%s) — seats: %d/%d, waitlist: %d",  
 name, courseId, enrolledStudents.size(), capacity, waitlist.size());  
 }  
}



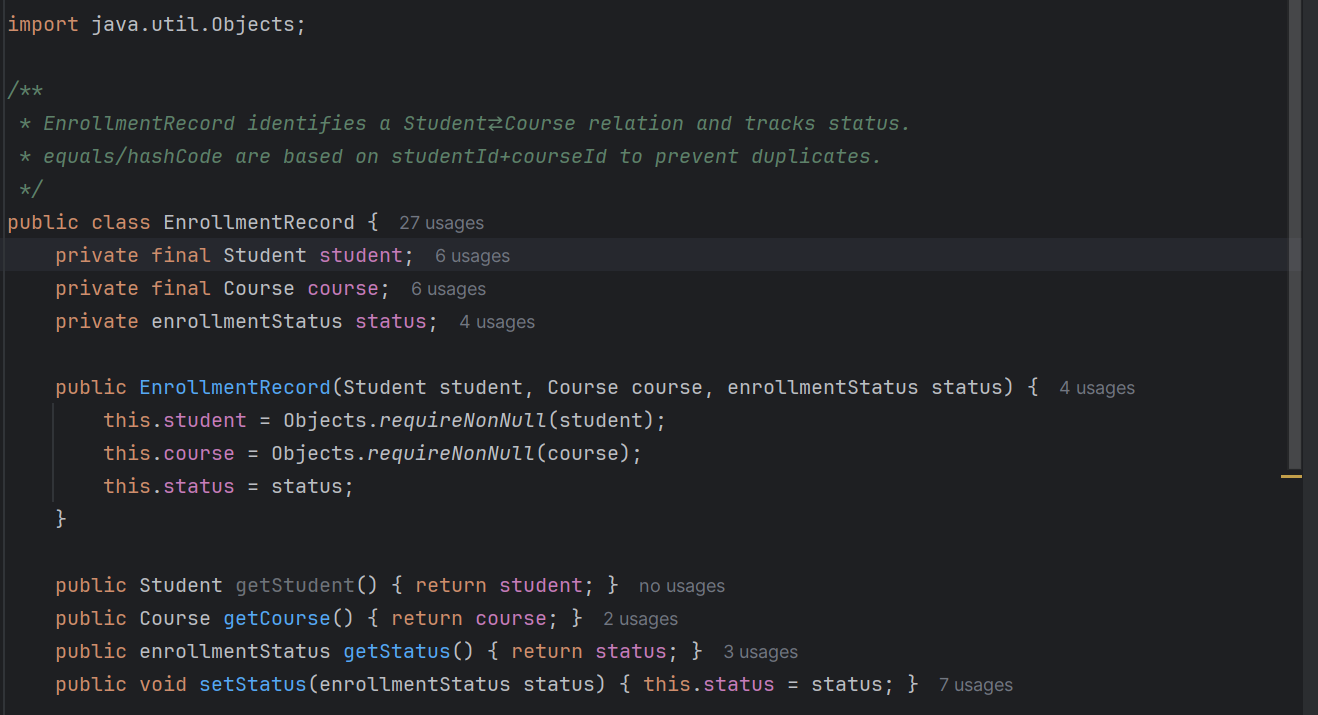


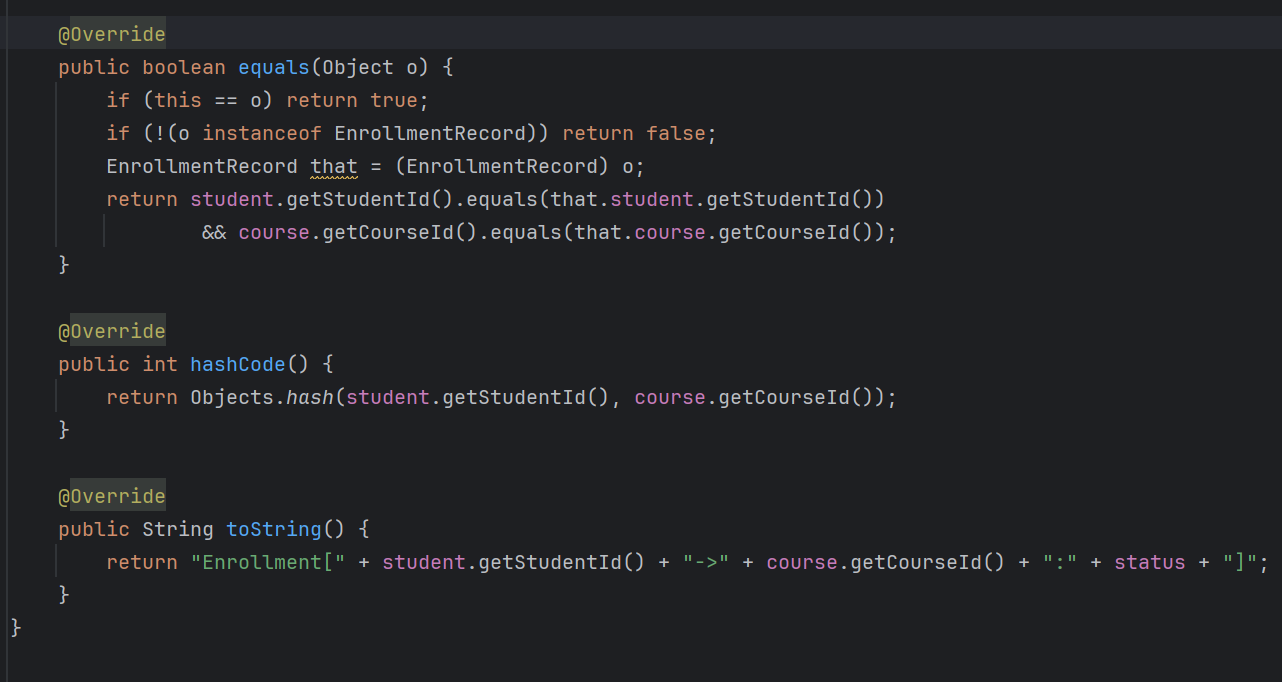




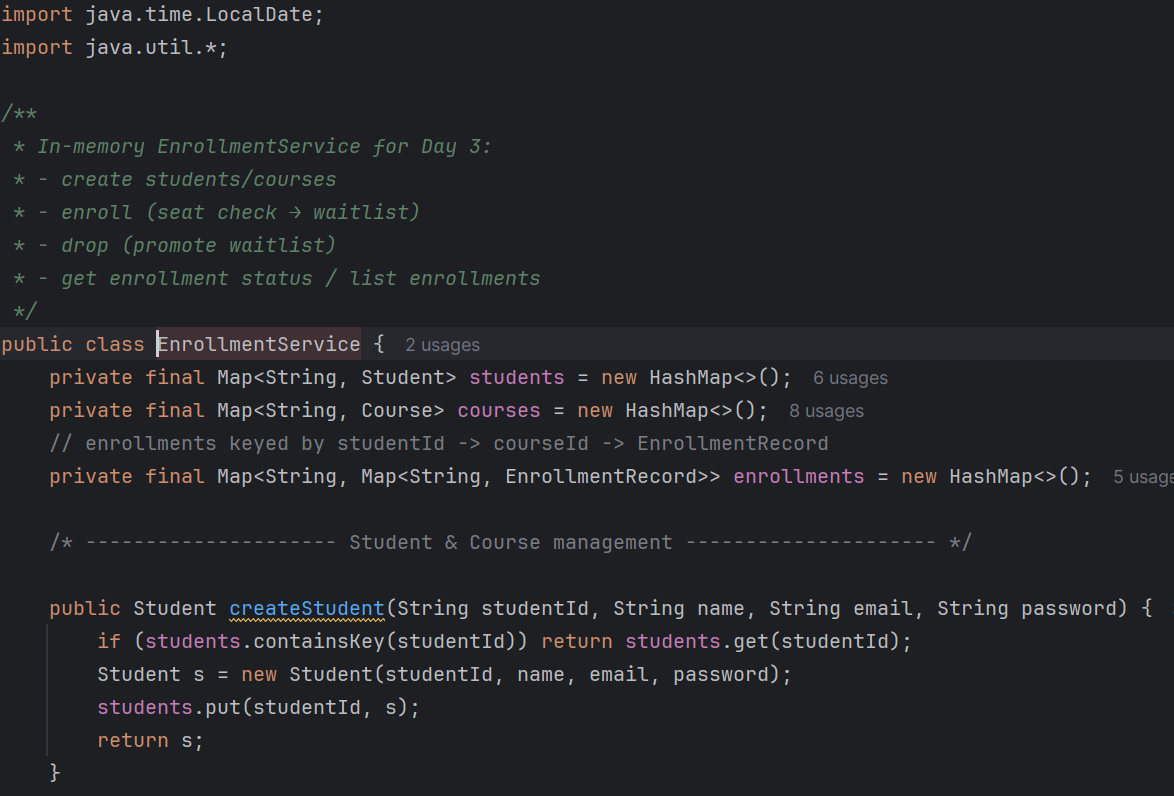
**EnrollmentRecord.java**

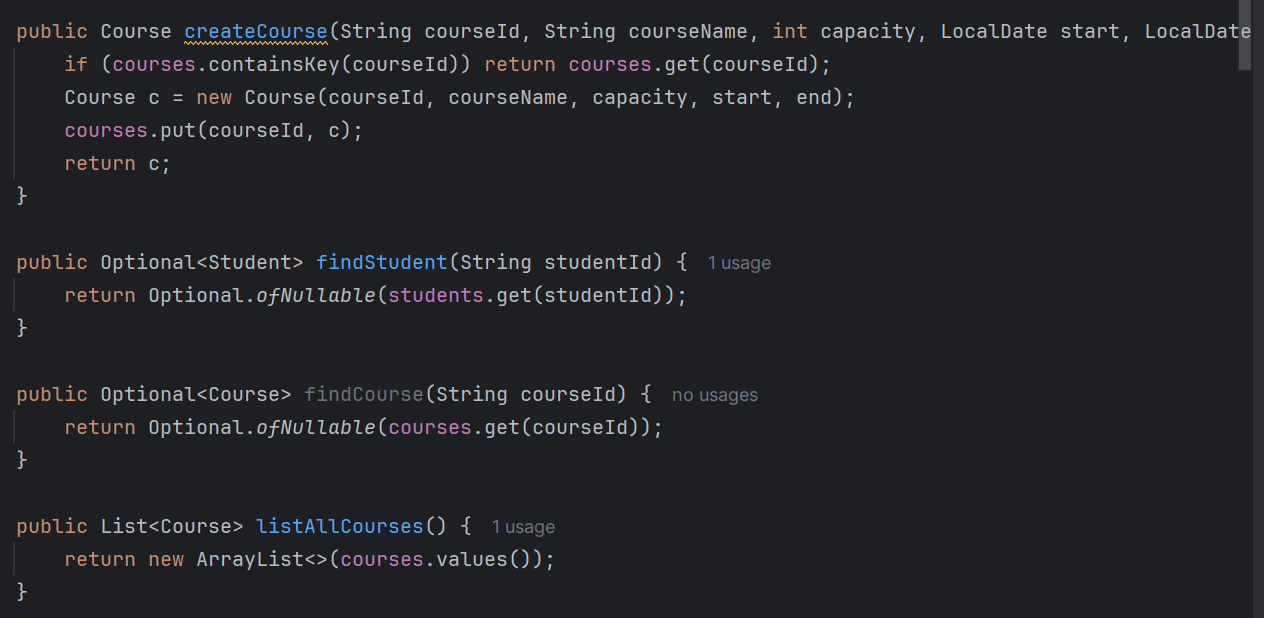
import java.util.Objects;  
  
*/\*\*  
 \* EnrollmentRecord identifies a Student⇄Course relation and tracks status.  
 \* equals/hashCode are based on studentId+courseId to prevent duplicates.  
 \*/*public class EnrollmentRecord {  
 private final Student student;  
 private final Course course;  
 private enrollmentStatus status;  
  
 public EnrollmentRecord(Student student, Course course, enrollmentStatus status) {  
 this.student = Objects.*requireNonNull*(student);  
 this.course = Objects.*requireNonNull*(course);  
 this.status = status;  
 }  
  
 public Student getStudent() { return student; }  
 public Course getCourse() { return course; }  
 public enrollmentStatus getStatus() { return status; }  
 public void setStatus(enrollmentStatus status) { this.status = status; }  
  
 @Override  
 public boolean equals(Object o) {  
 if (this == o) return true;  
 if (!(o instanceof EnrollmentRecord)) return false;  
 EnrollmentRecord that = (EnrollmentRecord) o;  
 return student.getStudentId().equals(that.student.getStudentId())  
 && course.getCourseId().equals(that.course.getCourseId());  
 }  
  
 @Override  
 public int hashCode() {  
 return Objects.*hash*(student.getStudentId(), course.getCourseId());  
 }  
  
 @Override  
 public String toString() {  
 return "Enrollment[" + student.getStudentId() + "->" + course.getCourseId() + ":" + status + "]";  
 }  
}

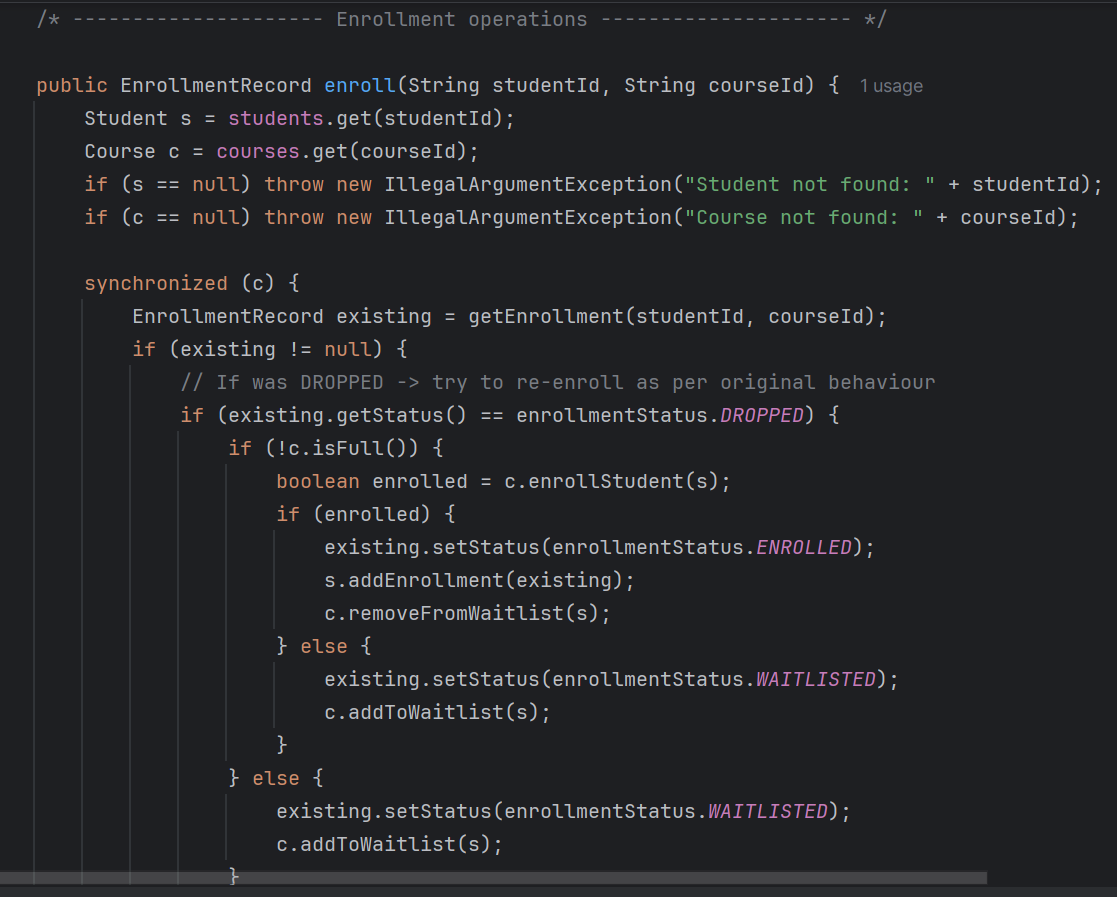


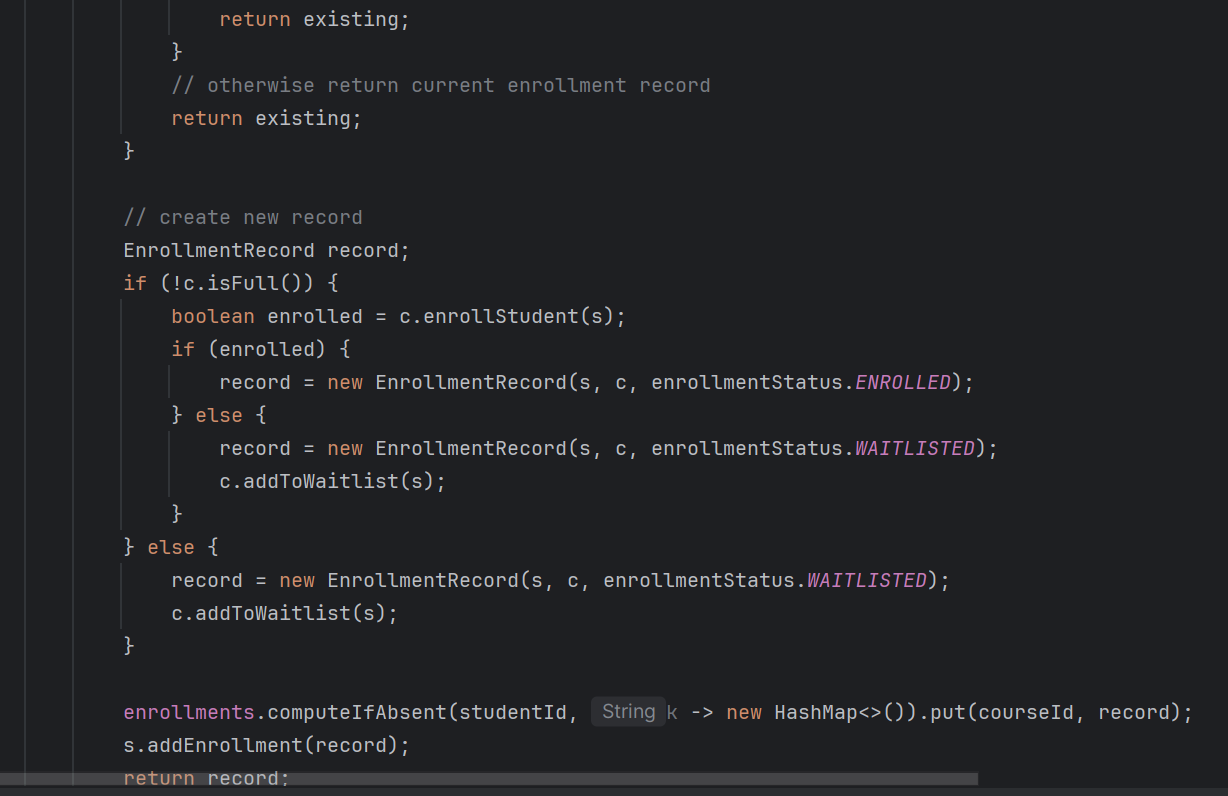


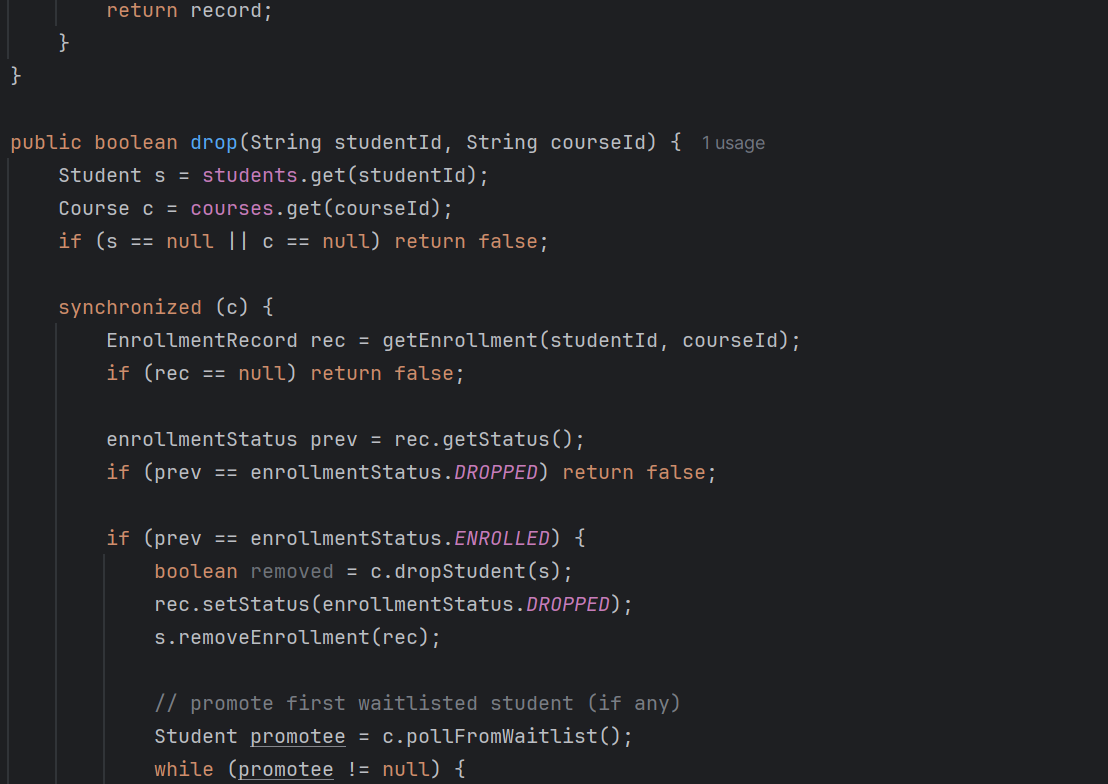
**EnrollmentService.java**

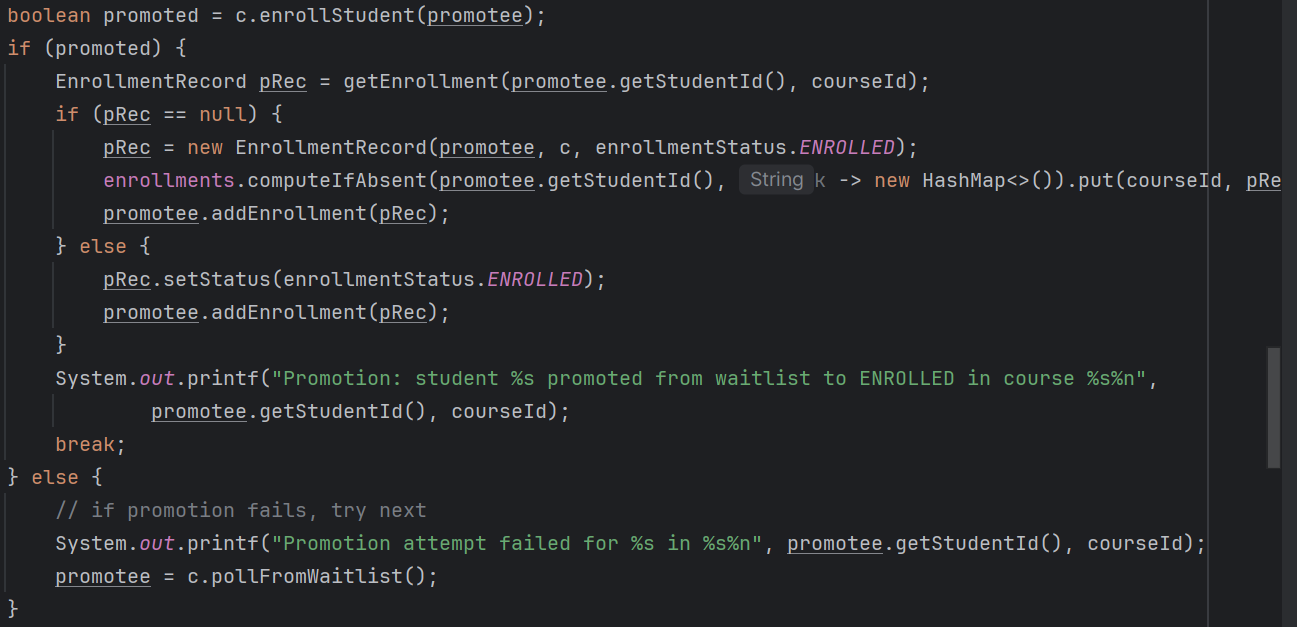


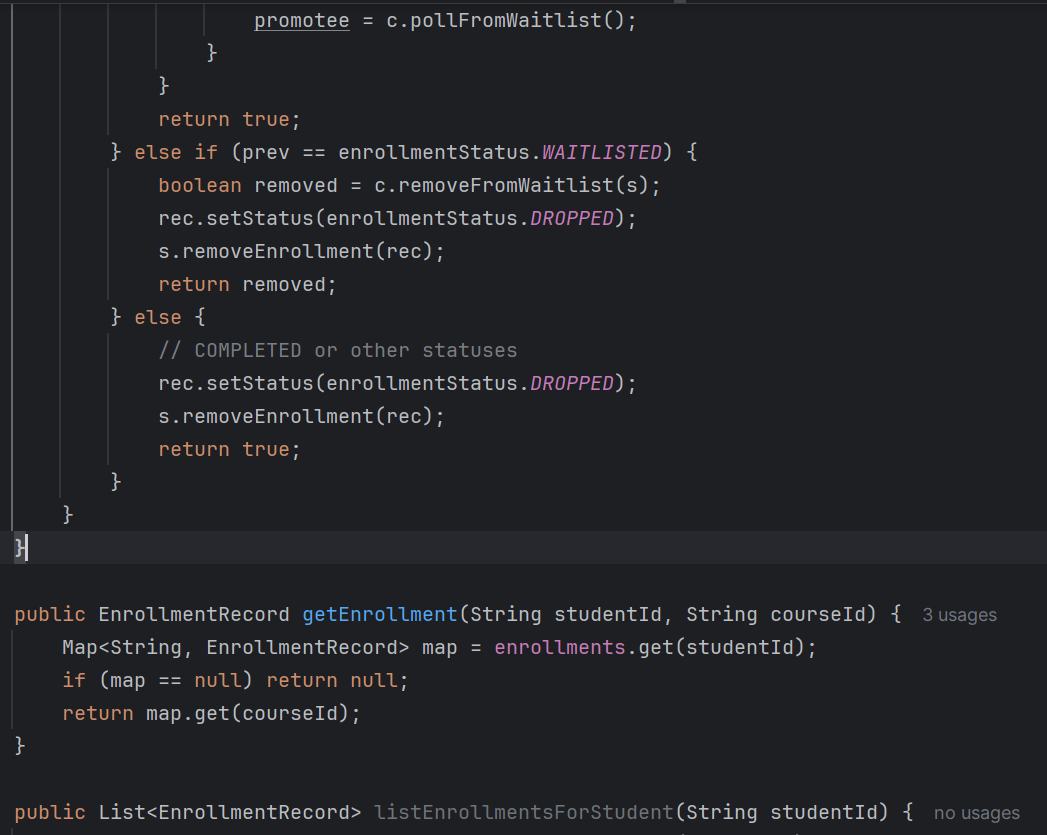




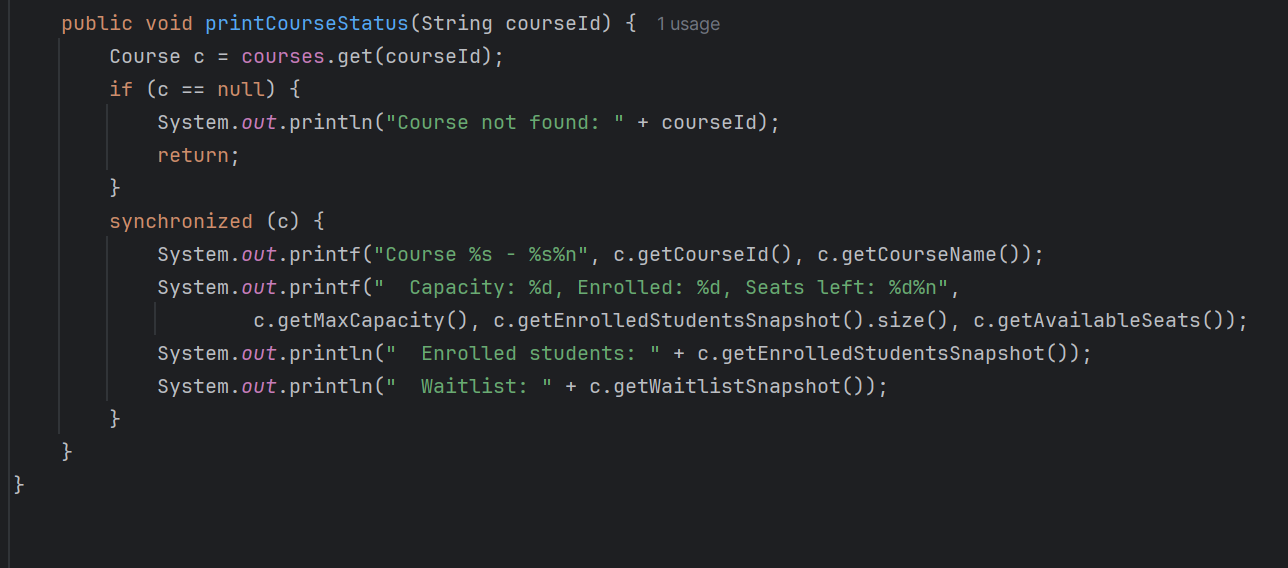






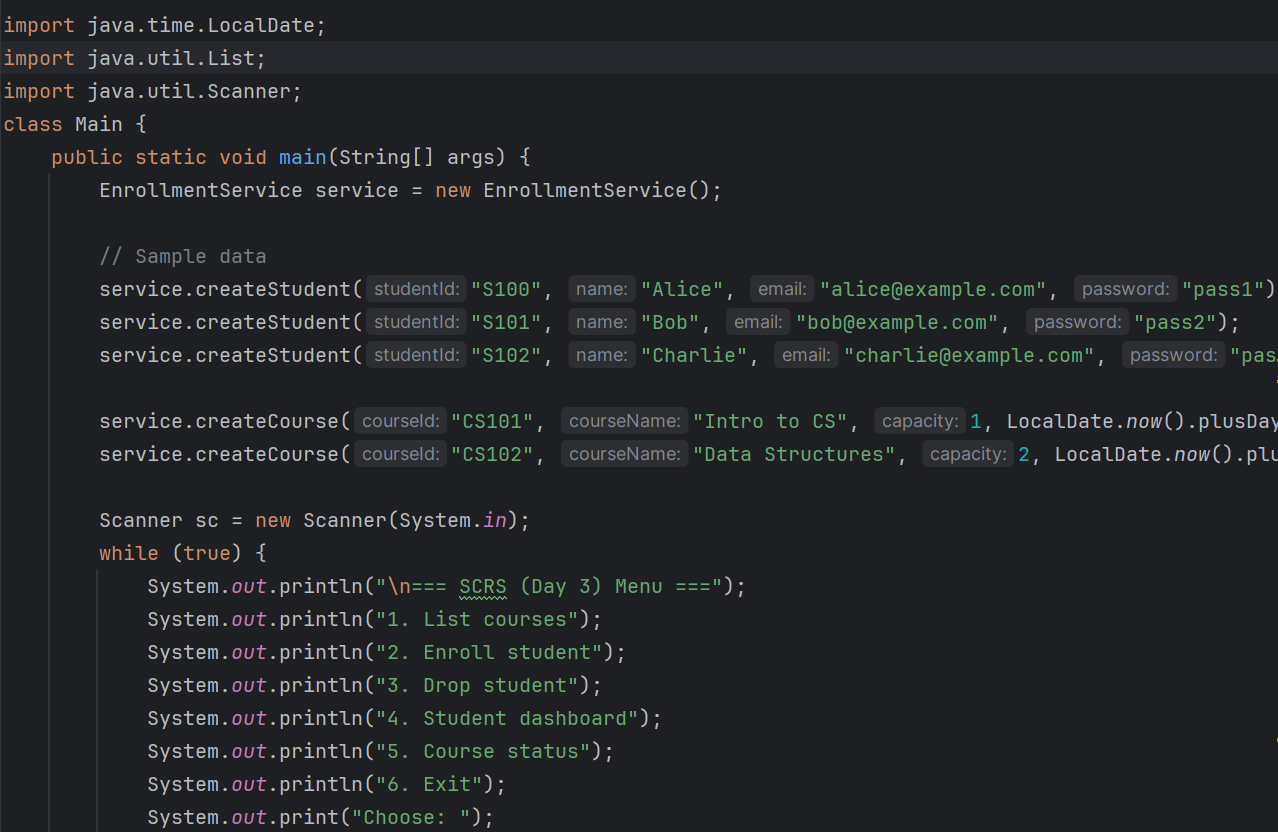




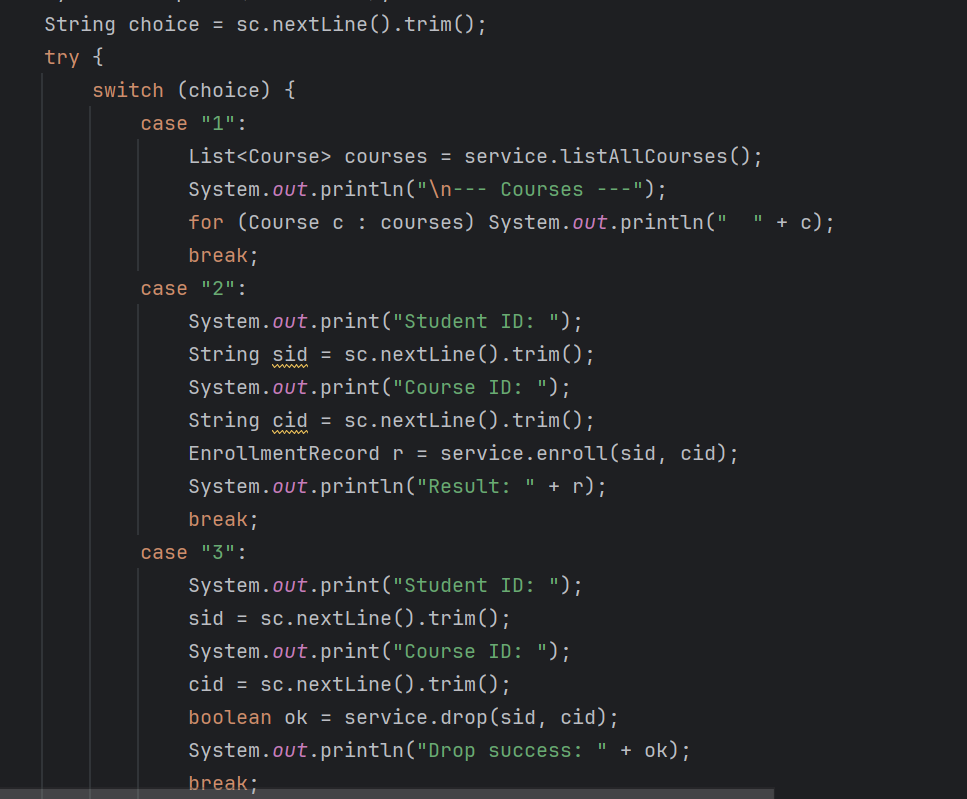


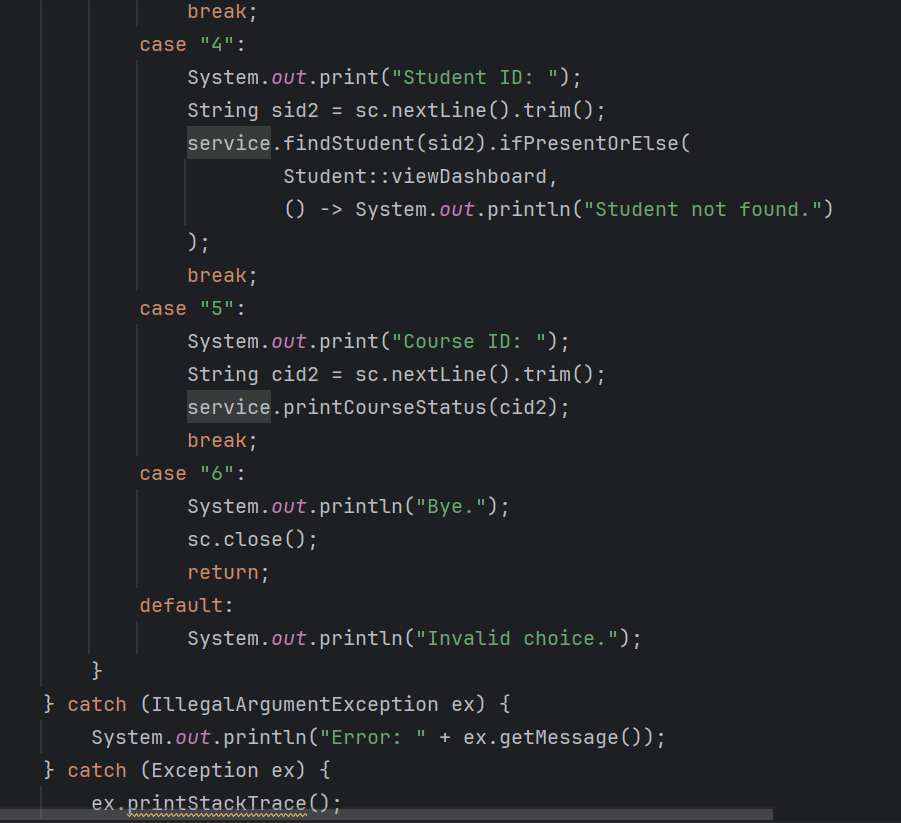
**Main\_3.java**

import java.time.LocalDate;  
import java.util.Scanner;  
  
public class Main\_3 {  
 public static void main(String[] args) {  
  
 // Initialize Enrollment Service  
 EnrollmentService enrollmentService = new EnrollmentService();  
  
 // Create students  
 enrollmentService.createStudent("S1", "Alice", "alice@example.com", "pwd123");  
 enrollmentService.createStudent("S2", "Bob", "bob@example.com", "pwd456");  
 enrollmentService.createStudent("S3", "Charlie", "charlie@example.com", "pwd789");  
  
 // Create courses  
 enrollmentService.createCourse("C1", "Data Structures", 2,  
 LocalDate.*now*(), LocalDate.*now*().plusDays(30));  
 enrollmentService.createCourse("C2", "Algorithms", 1,  
 LocalDate.*now*(), LocalDate.*now*().plusDays(30));  
  
 Scanner sc = new Scanner(System.*in*);  
 while (true) {  
 System.*out*.println("\n--- COURSE ENROLLMENT SYSTEM (DAY 3) ---");  
 System.*out*.println("1. Enroll Student");  
 System.*out*.println("2. Drop Student");  
 System.*out*.println("3. View Student Enrollments");  
 System.*out*.println("4. View Course Status");  
 System.*out*.println("5. Exit");  
 System.*out*.print("Enter your choice: ");  
  
 int choice = sc.nextInt();  
 sc.nextLine(); // consume newline  
  
 switch (choice) {  
 case 1 -> {  
 System.*out*.print("Enter Student ID: ");  
 String sid = sc.nextLine();  
 System.*out*.print("Enter Course ID: ");  
 String cid = sc.nextLine();  
 try {  
 EnrollmentRecord record = enrollmentService.enroll(sid, cid);  
 System.*out*.println("Enrollment Successful: " + record.getStatus());  
 } catch (Exception e) {  
 System.*out*.println("Error: " + e.getMessage());  
 }  
 }  
  
 case 2 -> {  
 System.*out*.print("Enter Student ID: ");  
 String sid = sc.nextLine();  
 System.*out*.print("Enter Course ID: ");  
 String cid = sc.nextLine();  
 boolean dropped = enrollmentService.drop(sid, cid);  
 if (dropped) System.*out*.println("Student dropped successfully.");  
 else System.*out*.println("Drop failed.");  
 }  
  
 case 3 -> {  
 System.*out*.print("Enter Student ID: ");  
 String sid = sc.nextLine();  
 System.*out*.println("Enrollments:");  
 for (EnrollmentRecord rec : enrollmentService.listEnrollmentsForStudent(sid)) {  
 System.*out*.printf("Course: %s, Status: %s%n",  
 rec.getCourse().getCourseName(), rec.getStatus());  
 }  
 }  
  
 case 4 -> {  
 System.*out*.print("Enter Course ID: ");  
 String cid = sc.nextLine();  
 enrollmentService.printCourseStatus(cid);  
 }  
  
 case 5 -> {  
 System.*out*.println("Exiting...");  
 return;  
 }  
  
 default -> System.*out*.println("Invalid choice!");  
 }  
 }  
 }  
}









**Day 3 – Basic Student & Course Setup + Enrollment Record**

1. **Student class**
   * Stores student information: studentId, name, email, password.
   * Provides getters and basic constructors.
2. **Course class**
   * Stores course information: courseId, courseName, capacity.
   * Tracks enrolled students with a simple counter (enrolledCount) and list of students.
   * Methods to increment/decrement enrollment count.
3. **EnrollmentRecord class**
   * Links a Student with a Course.
   * Maintains enrollmentStatus (ENROLLED, DROPPED, etc.).
4. **EnrollmentDAO / Service**
   * Provides methods to save and retrieve enrollments.
   * Handles enrolling a student in a course (without waitlist yet).
   * Handles dropping a student from a course (simple decrement).
5. **Main3 (Day 3 Main)**
   * Creates students and courses.
   * Enrolls students up to the course capacity.
   * Drops a student (simple drop, no waitlist yet).

**Key Concept Learned:**

* Object relationships between Student, Course, and EnrollmentRecord.
* Simple enrollment and drop mechanics without queues or waitlists.