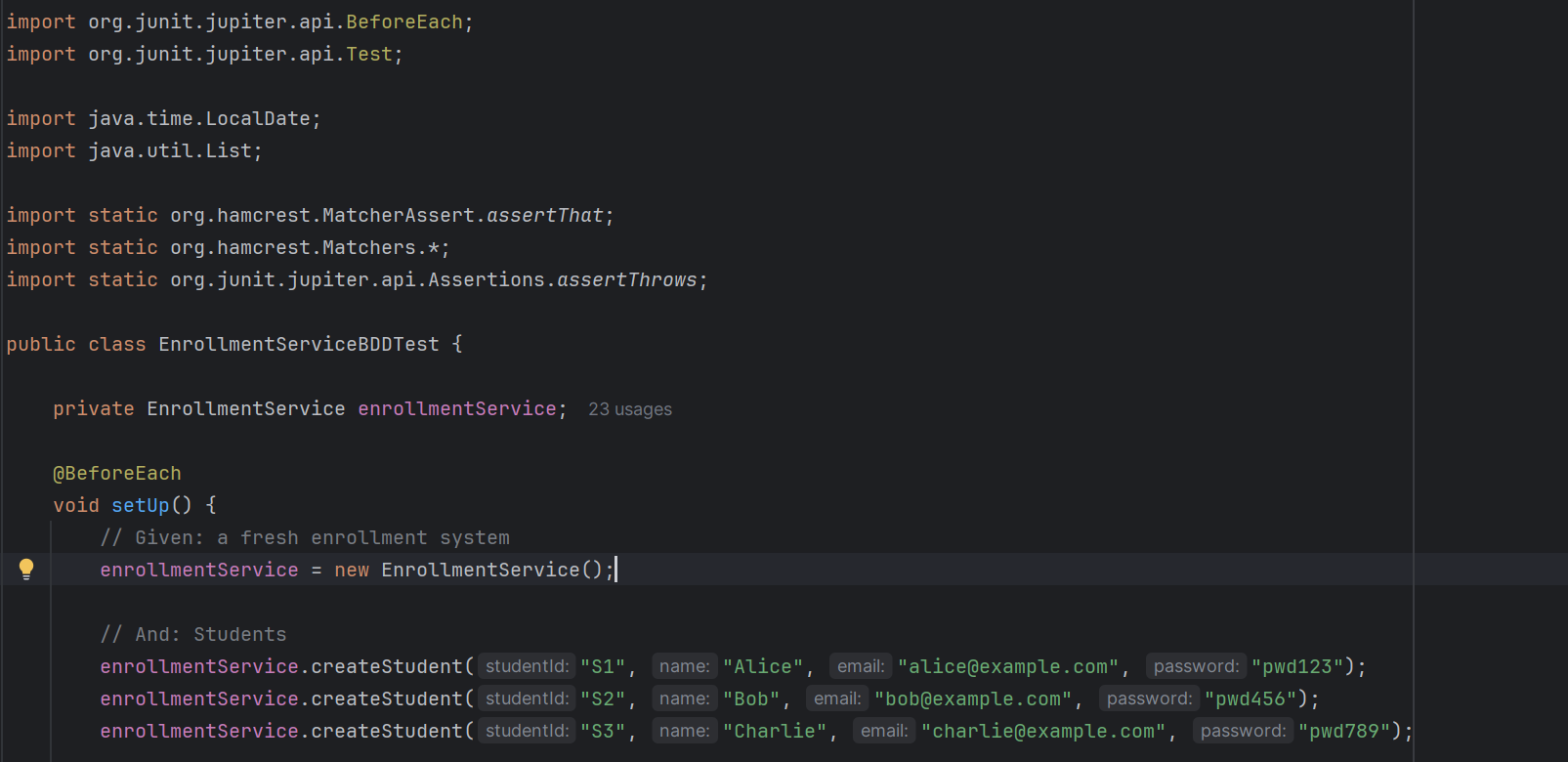
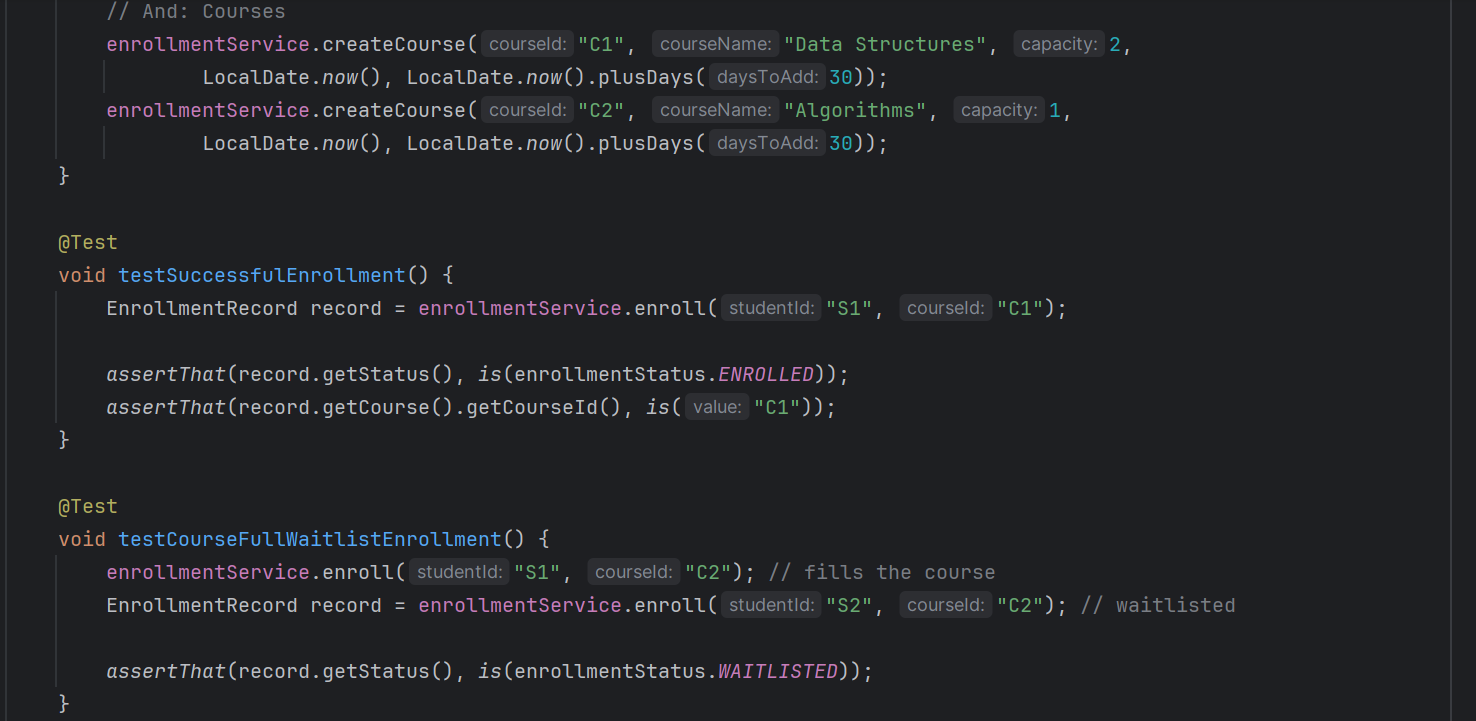
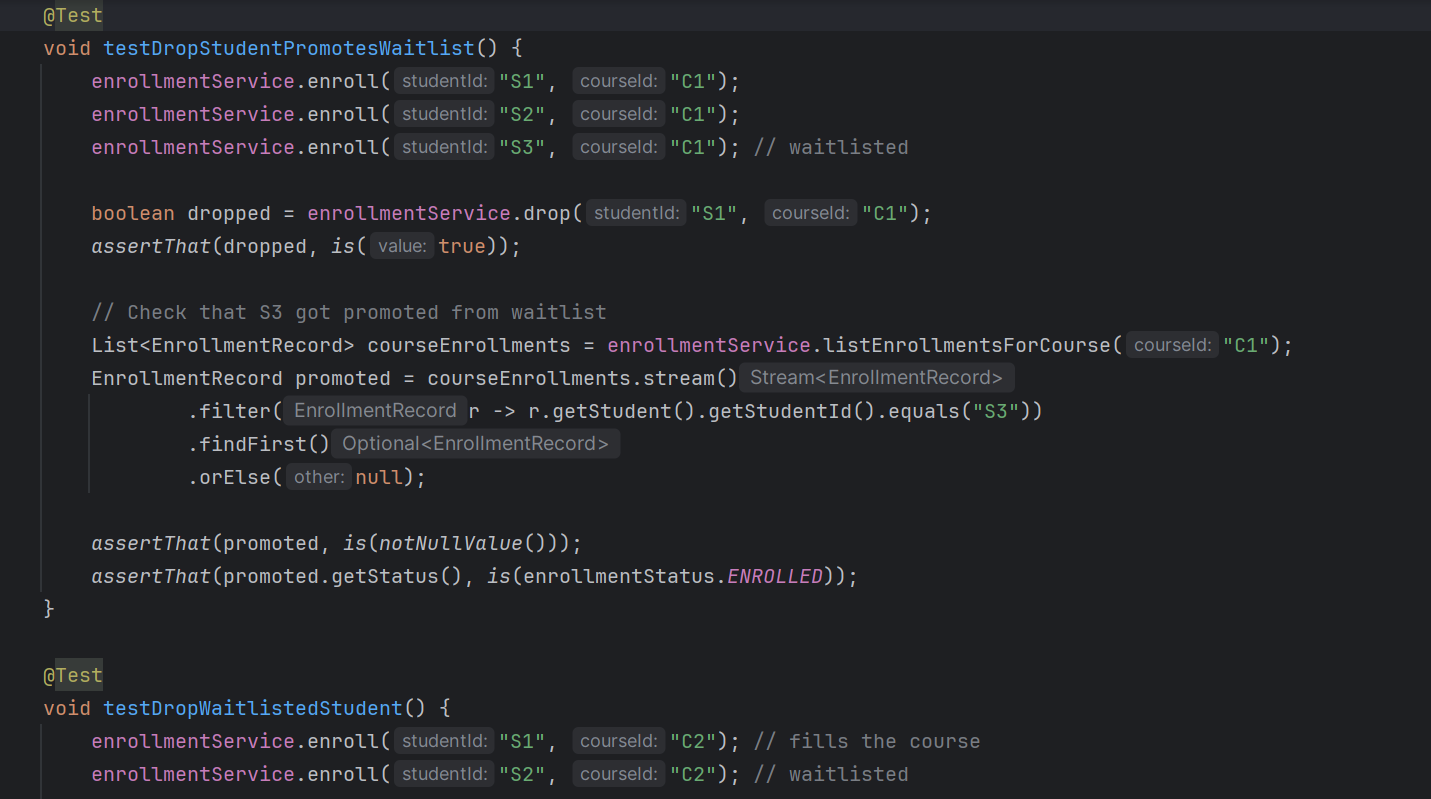
**Day 8 – BDD – Junit for enrollment scenario.**

**EnrollmentServiceBDDTest.java**

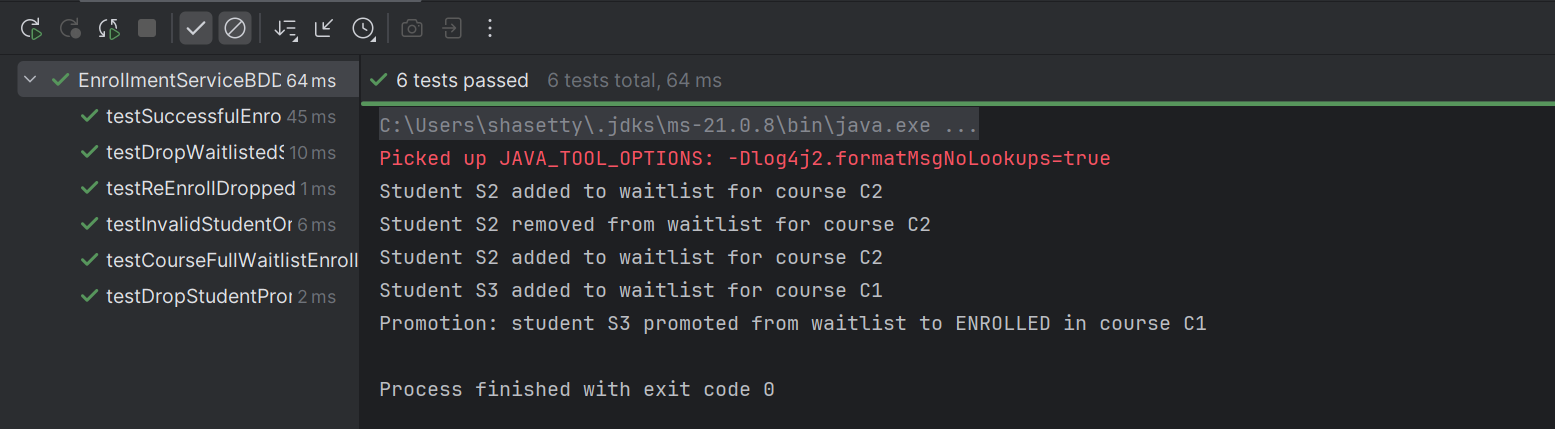
import org.junit.jupiter.api.BeforeEach;  
import org.junit.jupiter.api.Test;  
  
import java.time.LocalDate;  
import java.util.List;  
  
import static org.hamcrest.MatcherAssert.*assertThat*;  
import static org.hamcrest.Matchers.\*;  
import static org.junit.jupiter.api.Assertions.*assertThrows*;  
  
public class EnrollmentServiceBDDTest {  
  
 private EnrollmentService enrollmentService;  
  
 @BeforeEach  
 void setUp() {  
 // Given: a fresh enrollment system  
 enrollmentService = new EnrollmentService();  
  
 // And: 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");  
  
 // And: Courses  
 enrollmentService.createCourse("C1", "Data Structures", 2,  
 LocalDate.*now*(), LocalDate.*now*().plusDays(30));  
 enrollmentService.createCourse("C2", "Algorithms", 1,  
 LocalDate.*now*(), LocalDate.*now*().plusDays(30));  
 }  
  
 @Test  
 void testSuccessfulEnrollment() {  
 EnrollmentRecord record = enrollmentService.enroll("S1", "C1");  
  
 *assertThat*(record.getStatus(), *is*(enrollmentStatus.*ENROLLED*));  
 *assertThat*(record.getCourse().getCourseId(), *is*("C1"));  
 }  
  
 @Test  
 void testCourseFullWaitlistEnrollment() {  
 enrollmentService.enroll("S1", "C2"); // fills the course  
 EnrollmentRecord record = enrollmentService.enroll("S2", "C2"); // waitlisted  
  
 *assertThat*(record.getStatus(), *is*(enrollmentStatus.*WAITLISTED*));  
 }  
  
 @Test  
 void testDropStudentPromotesWaitlist() {  
 enrollmentService.enroll("S1", "C1");  
 enrollmentService.enroll("S2", "C1");  
 enrollmentService.enroll("S3", "C1"); // waitlisted  
  
 boolean dropped = enrollmentService.drop("S1", "C1");  
 *assertThat*(dropped, *is*(true));  
  
 // Check that S3 got promoted from waitlist  
 List<EnrollmentRecord> courseEnrollments = enrollmentService.listEnrollmentsForCourse("C1");  
 EnrollmentRecord promoted = courseEnrollments.stream()  
 .filter(r -> r.getStudent().getStudentId().equals("S3"))  
 .findFirst()  
 .orElse(null);  
  
 *assertThat*(promoted, *is*(*notNullValue*()));  
 *assertThat*(promoted.getStatus(), *is*(enrollmentStatus.*ENROLLED*));  
 }  
  
 @Test  
 void testDropWaitlistedStudent() {  
 enrollmentService.enroll("S1", "C2"); // fills the course  
 enrollmentService.enroll("S2", "C2"); // waitlisted  
  
 boolean dropped = enrollmentService.drop("S2", "C2");  
 *assertThat*(dropped, *is*(true));  
  
 EnrollmentRecord rec = enrollmentService.getEnrollment("S2", "C2");  
 *assertThat*(rec.getStatus(), *is*(enrollmentStatus.*DROPPED*));  
 }  
  
 @Test  
 void testInvalidStudentOrCourse() {  
 IllegalArgumentException ex1 = *assertThrows*(IllegalArgumentException.class, () ->  
 enrollmentService.enroll("INVALID", "C1"));  
 *assertThat*(ex1.getMessage(), *containsString*("Student not found"));  
  
 IllegalArgumentException ex2 = *assertThrows*(IllegalArgumentException.class, () ->  
 enrollmentService.enroll("S1", "INVALID"));  
 *assertThat*(ex2.getMessage(), *containsString*("Course not found"));  
 }  
  
 @Test  
 void testReEnrollDroppedStudent() {  
 enrollmentService.enroll("S1", "C1");  
 enrollmentService.drop("S1", "C1");  
  
 EnrollmentRecord rec = enrollmentService.enroll("S1", "C1");  
 *assertThat*(rec.getStatus(), *is*(enrollmentStatus.*ENROLLED*));  
 }  
}











**Test Cases**

1. **testSuccessfulEnrollment()**
   * Enrolls a student in a course with available seats.
   * Asserts the status is ENROLLED.
   * Asserts course ID matches.
2. **testCourseFullWaitlistEnrollment()**
   * Fills a course.
   * Attempts to enroll another student.
   * Asserts that the student is WAITLISTED.
3. **testDropStudent()**
   * Drops an enrolled student.
   * Asserts that status is DROPPED.
   * Ensures student’s enrollment record reflects the drop.
4. **(Optional in your setup)**
   * Could test invalid student/course IDs.
   * Could test promotion from waitlist after drop.

**Key Features**

* **Automated testing** of enrollment and waitlist logic.
* Validates **business rules**:
  + Capacity check
  + Waitlist behavior
  + Drop & promotion
* Uses **Hamcrest matchers** for readable assertions (is(), hasSize(), etc.).