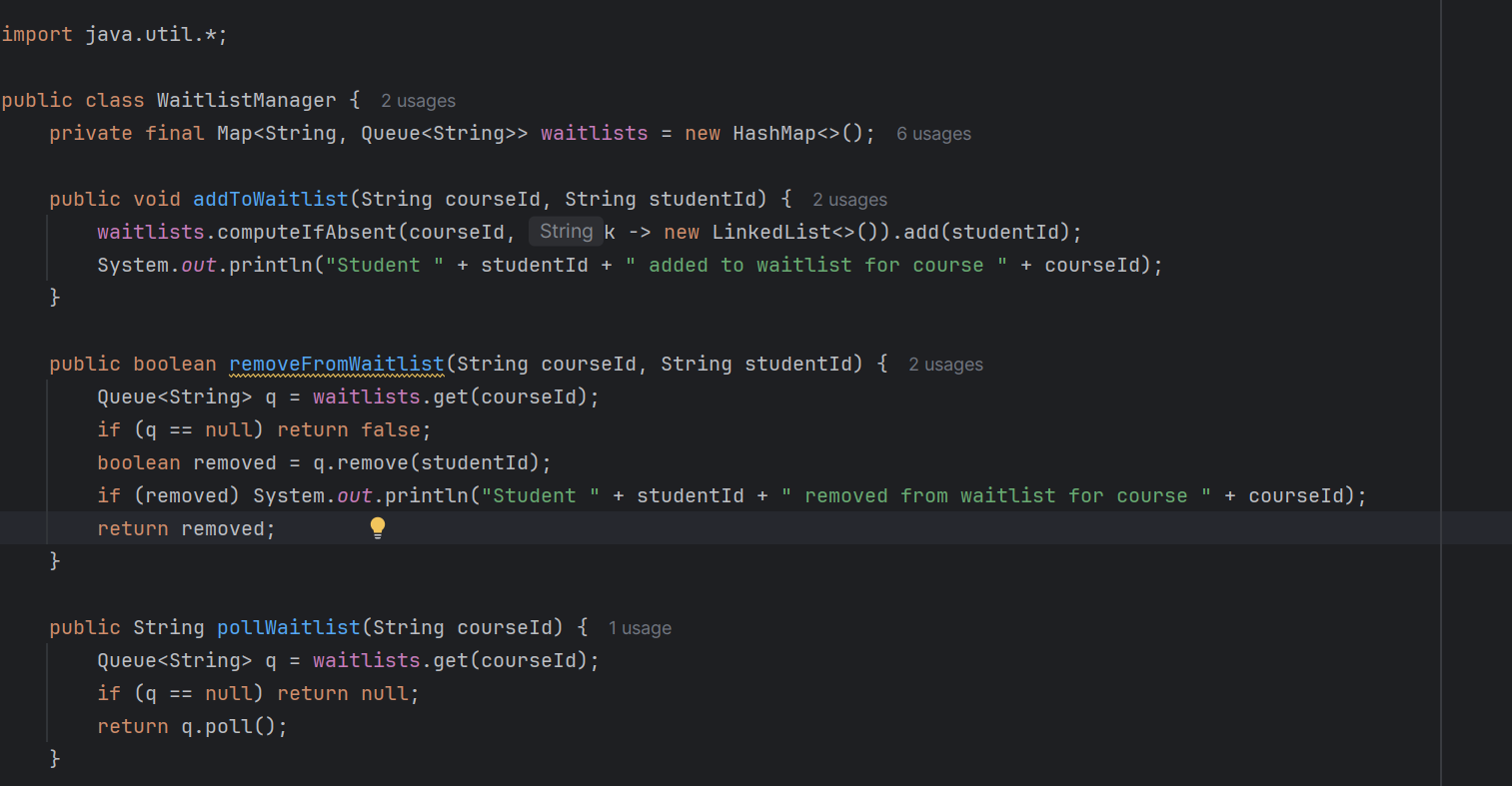
**Day 6 - Queue- Manage waitlist with DS.**

**WaitlistManager.java**

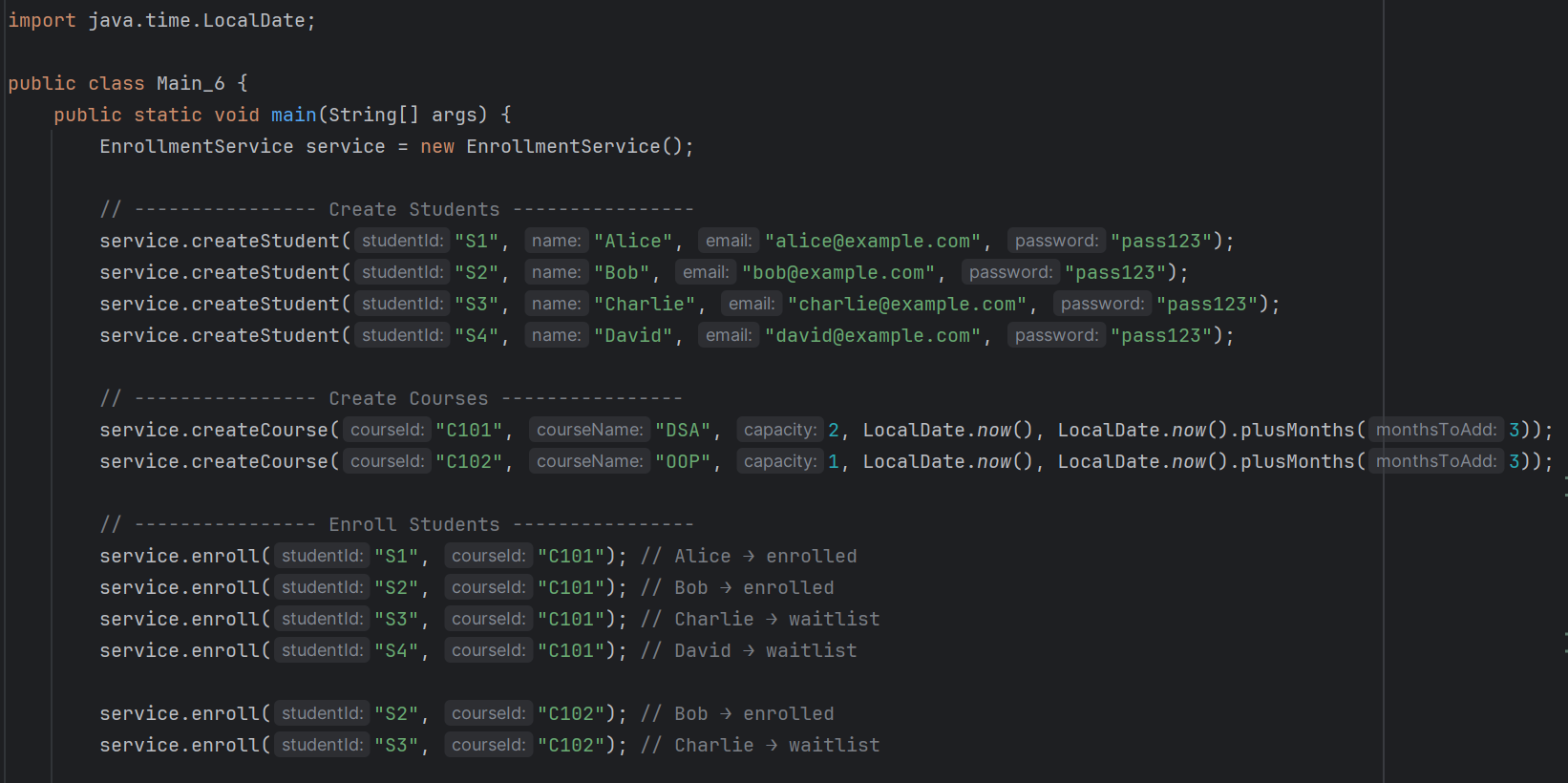
import java.util.\*;  
  
public class WaitlistManager {  
 private final Map<String, Queue<String>> waitlists = new HashMap<>();  
  
 public void addToWaitlist(String courseId, String studentId) {  
 waitlists.computeIfAbsent(courseId, k -> new LinkedList<>()).add(studentId);  
 System.*out*.println("Student " + studentId + " added to waitlist for course " + courseId);  
 }  
  
 public boolean removeFromWaitlist(String courseId, String studentId) {  
 Queue<String> q = waitlists.get(courseId);  
 if (q == null) return false;  
 boolean removed = q.remove(studentId);  
 if (removed) System.*out*.println("Student " + studentId + " removed from waitlist for course " + courseId);  
 return removed;  
 }  
  
 public String pollWaitlist(String courseId) {  
 Queue<String> q = waitlists.get(courseId);  
 if (q == null) return null;  
 return q.poll();  
 }  
  
 public List<String> getWaitlist(String courseId) {  
 Queue<String> q = waitlists.get(courseId);  
 if (q == null) return Collections.*emptyList*();  
 return new ArrayList<>(q);  
 }  
  
 public void printWaitlist() {  
 System.*out*.println("\n--- Waitlists ---");  
 for (String courseId : waitlists.keySet()) {  
 System.*out*.println("Course " + courseId + ": " + waitlists.get(courseId));  
 }  
 }  
}

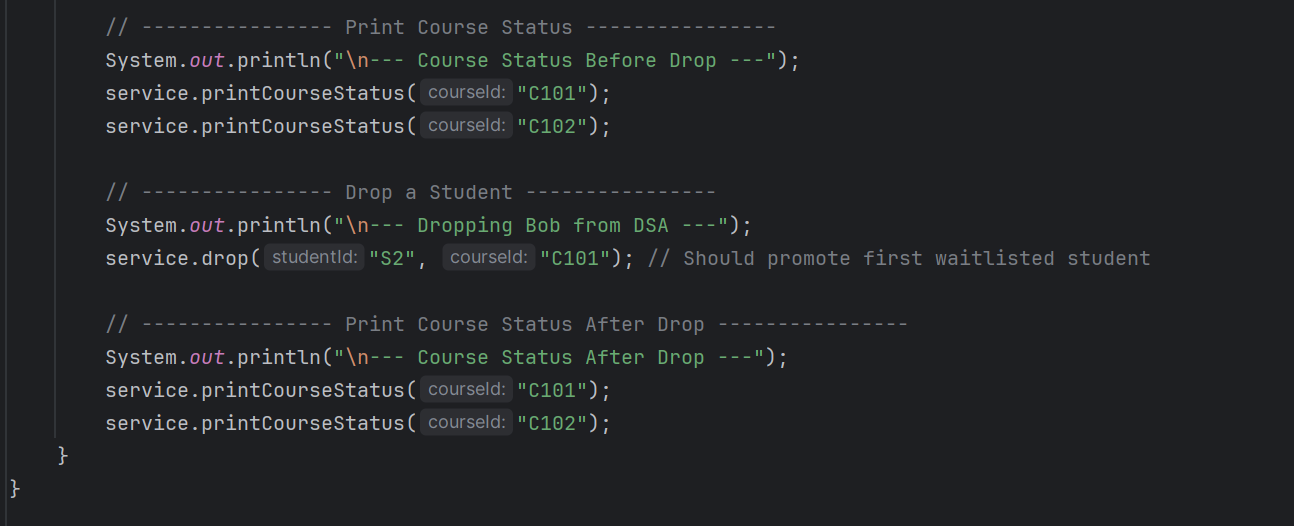




**Main\_6.java**

import java.time.LocalDate;  
  
public class Main\_6 {  
 public static void main(String[] args) {  
 EnrollmentService service = new EnrollmentService();  
  
 // ---------------- Create Students ----------------  
 service.createStudent("S1", "Alice", "alice@example.com", "pass123");  
 service.createStudent("S2", "Bob", "bob@example.com", "pass123");  
 service.createStudent("S3", "Charlie", "charlie@example.com", "pass123");  
 service.createStudent("S4", "David", "david@example.com", "pass123");  
  
 // ---------------- Create Courses ----------------  
 service.createCourse("C101", "DSA", 2, LocalDate.*now*(), LocalDate.*now*().plusMonths(3));  
 service.createCourse("C102", "OOP", 1, LocalDate.*now*(), LocalDate.*now*().plusMonths(3));  
  
 // ---------------- Enroll Students ----------------  
 service.enroll("S1", "C101"); // Alice → enrolled  
 service.enroll("S2", "C101"); // Bob → enrolled  
 service.enroll("S3", "C101"); // Charlie → waitlist  
 service.enroll("S4", "C101"); // David → waitlist  
  
 service.enroll("S2", "C102"); // Bob → enrolled  
 service.enroll("S3", "C102"); // Charlie → waitlist  
  
 // ---------------- Print Course Status ----------------  
 System.*out*.println("\n--- Course Status Before Drop ---");  
 service.printCourseStatus("C101");  
 service.printCourseStatus("C102");  
  
 // ---------------- Drop a Student ----------------  
 System.*out*.println("\n--- Dropping Bob from DSA ---");  
 service.drop("S2", "C101"); // Should promote first waitlisted student  
  
 // ---------------- Print Course Status After Drop ----------------  
 System.*out*.println("\n--- Course Status After Drop ---");  
 service.printCourseStatus("C101");  
 service.printCourseStatus("C102");  
 }  
}





**WaitlistManager**

Uses a **Map of courseId → Queue of studentIds** to track waitlists

Key methods:

addToWaitlist(courseId, studentId) → add a student to the waitlistremoveFromWaitlist(courseId, studentId) → remove a student

pollWaitlist(courseId) → get and remove first student from waitlist (for promotion)

getWaitlist(courseId) → get current waitlist as list

printWaitlist() → prints all waitlists

**Main\_6 Workflow**

1. **Create Students**
   * Alice, Bob, Charlie, David
2. **Create Courses**
   * C101 DSA (capacity 2)
   * C102 OOP (capacity 1)
3. **Enroll Students**
   * Enroll up to course capacity → ENROLLED
   * Extra students → added to waitlist (FIFO)
4. **Print Course Status**
   * Shows enrolled students and current waitlist
5. **Drop a Student**
   * Bob is dropped from C101
   * First waitlisted student (Charlie) is **promoted automatically**
6. **Print Course Status Again**
   * Updates enrollment and waitlist after promotion

**Key Features**

* **Queue-based waitlist** ensures **first-come, first-served promotion**
* Supports **enrolled & waitlisted states**
* Automatically updates enrollment when a seat becomes available