



CC-211L Object Oriented Programming Lab

Department of Computer Science

University of the Punjab

Fall 2026

Lab 2: Classes, Encapsulation, and Valid Objects

After completing this lab, you will be able to:

- Design a class from a UML specification and keep its data private.
- Use default, overloaded, and delegating constructors with member-initializer lists.
- Enforce valid state during construction and through public member functions.
- Explain the relationship between a class's private state and its public interface.
- Test boundary cases and debug a class whose interface permits invalid state.

Lab duration: 1 hour 45 minutes

Part A: Design and implementation (50 marks)

Part B: Required test cases (35 marks)

Part C: Interface review and debugging (15 marks)

Grading Scheme

Question:	1	2	3	4	5	6	Total
Points:	10	25	15	35	10	5	100
Score:							

Important instructions

- Use C++17 and compile with `-std=c++17 -Wall -Wextra -pedantic`.
- Do not use inheritance, operator overloading, templates, STL containers, exceptions, or file I/O.
- Keep all data members **private**. Client code must use the public interface.
- Do not use global variables, inheritance, dynamic allocation, pointers, or references in this lab.
- Submit one PDF containing written answers, UML, and evidence, plus `course.cpp`.
- Name the PDF `roll_number_oop_lab2.pdf`.

1 Part A: The Course Class

You will implement a small class that represents one course registration. The class must never expose an invalid object to its client.

1. [10 marks] A1. Design before code

Draw a UML class diagram for `Course`. Your diagram must include the following private data members and public operations. Use `-` for private and `+` for public members.

Member	Required meaning
<code>code: string</code>	non-empty course code, such as CC211
<code>title: string</code>	non-empty course title
<code>creditHours: int</code>	an integer from 1 through 3
<code>capacity: int</code>	an integer greater than 0
<code>enrolled: int</code>	an integer from 0 through <code>capacity</code>

Include these operations in the diagram: `Course()`, `Course(code, title, creditHours, capacity)`, `Course(code, title)`, `enroll()`, `drop()`, `isFull()`, `getCode()`, `getTitle()`, `getCreditHours()`, `getCapacity()`, and `getEnrolled()`. Mark return types and parameter types. State in one sentence why `enrolled` must not be public.

2. [25 marks] A2. Implement a valid Course

Create `course.cpp`. Implement the class from A1 and a `main` function that demonstrates its interface.

Construction rules:

- The default constructor creates `GEN000`, `Unspecified Course`, 3 credit hours, capacity 30, and 0 enrolled.
- The four-argument constructor receives `code`, `title`, `credit hours`, and `capacity`; it starts with 0 enrolled.
- The two-argument constructor receives `code` and `title` and delegates to the four-argument constructor using 3 credit hours and capacity 30.
- Use member-initializer lists. Do not assign all members in the constructor body.
- If the supplied `code` or `title` is empty, use `UNKNOWN` and `Untitled Course` respectively.
- If credit hours are outside 1–3, use 3. If capacity is not positive, use 30.

Behavior rules:

- `enroll()` increases `enrolled` by one and returns `true` only when a seat was available.
- `drop()` decreases `enrolled` by one and returns `true` only when at least one student was enrolled.
- `isFull()` returns whether `enrolled == capacity`.
- Getter functions return state without modifying the object.
- Use `const` on getter member functions.

Your `main` must create and test:

- a) one default-constructed course;
- b) one course using the two-argument constructor;
- c) one course using the four-argument constructor with invalid values, demonstrating the fallback rules;
- d) a small-capacity course that successfully enrolls students until full, rejects one extra enrollment, drops one student, and enrolls again.

Print enough state to make every result visible. Do not access a data member directly from `main`.

3. [15 marks] **A3. Interface review**

Answer the following questions using your implementation:

- a) Which constructor is called by `Course c{"CC211", "OOP"}`? Why?
- b) Why is delegation preferable to duplicating the default values in a second constructor?
- c) Give one example of an invalid state that the class prevents and identify the method that prevents it.
- d) Why can a `const Course` object call a getter but not `enroll()`?

2 Part B: Required Test Cases

Your program must print one clearly labelled line for each test below. Use the following format so that the result can be checked quickly:

```
TEST 1: PASS
TEST 2: PASS
...
TEST 10: PASS
```

Each test must check the stated condition with an `if` statement. Do not print `PASS` unconditionally. A failed condition must print `FAIL` and the observed values. Print **All tests passed** only when all ten tests pass.

4. [35 marks] **B1. Implement the fixed test table**

Add the following ten tests to `main`. The numbers and expected values are fixed; do not replace them with interactive input.

- i) Default construction gives `GEN000`, `Unspecified Course`, 3, 30, and 0.
- ii) Two-argument construction of `"CC211"`, `"OOP"` gives 3 credit hours, capacity 30, and 0 enrolled.
- iii) Four-argument construction of `"CC213"`, `"Data Structures"`, 2, 40 preserves all valid input.
- iv) Empty code and title use `UNKNOWN` and `Untitled Course`.
- v) Credit hours 0 use 3, and capacity -5 uses 30.
- vi) A course with capacity 2 accepts the first two calls to `enroll()`.
- vii) The third call to `enroll()` is rejected and enrolled remains 2.
- viii) A call to `drop()` decreases enrolled from 2 to 1.
- ix) Two further calls to `drop()` leave enrolled at 0 and reject the last call.
- x) A full course makes `isFull()` return `true`; after a successful drop it returns `false`.

Use one fixed course with capacity 2 for tests vi–x. Print the expected and observed values in a `FAIL` line.

3 Part C: Debugging and Evidence

5. [10 marks] **C1. Find the design defects**

The following class is intentionally broken:

```
class Course {
public:
    Course(std::string c, int seats)
        : code{c}, capacity{seats}, enrolled{0} {}

    void enroll() { ++enrolled; }
    int getEnrolled() { return enrolled; }

    int capacity;
    int enrolled;

private:
    std::string code;
};
```

Identify at least four defects or design weaknesses. At least one must concern encapsulation, one must concern valid state, and one must concern the public interface. For each, state a concrete correction. You do not need to implement this broken class.

6. [5 marks] **C2. Testing and code quality**

Submit `course.cpp`, the UML diagram, the compile command, and successful output. The output must show ten numbered test results and **All tests passed**. Code is graded for readable names, private state, small focused member functions, correct use of constructors and initializer lists, and warnings-free compilation.

Submission checklist

Submit one PDF and `course.cpp` only. Do not submit executable files. The PDF must contain your full name, roll number, UML diagram, written answers, compile commands, and output evidence. Name it exactly `roll_number_oop_lab2.pdf`.

Suggested timing

A1: UML and design decisions	15 minutes
A2–A3: implementation and interface review	35 minutes
B1: fixed test table	25 minutes
C1–C2: debugging, tests, and packaging	20 minutes
Total	105 minutes