

Session 8

Implementing class diagrams in code

Sessions 8 to 10 turn diagrams into code and tables. The mapping is mechanical once you know the rules: a class becomes a class, an attribute a field, an operation a method, and each association end a reference or a collection.

Objectives

✖ Do not copy. Read for understanding and the viva

- Complete questions 19 to 20 of the manual: implementing class diagrams in code
- Prepare the deliverable before the lab and finish it during the session
- Be ready to explain every step in the viva

Questions Covered

✖ Do not copy. Read for understanding and the viva

Question	Requirement	Status
Q19	Implement the following Class Diagram in C++/Java	Complete
Q20	Implement the Class Diagram of figure 1.18, in C++ or Java	Complete

Preparation

✖ Do not copy. Read for understanding and the viva

- The manual asks for a problem description of 300 to 500 words and a list of assumptions before every diagram. Write both first; they fix the scope the evaluator marks you against.
- The manual says C++ or Java. Every program here is given in C++, Rust, Python and TypeScript, and any of the four is acceptable in the lab; pick one and keep it for all three sessions.
- Map multiplicity: `1` becomes a single reference, `*` becomes a `List` (Java) or `std::vector` (C++).
- Write a small `main` that creates a few objects and exercises every operation so the program actually runs.

Question 19

Problem Statement

■ Write in lab record

Implement the following Class Diagram in C++/Java.

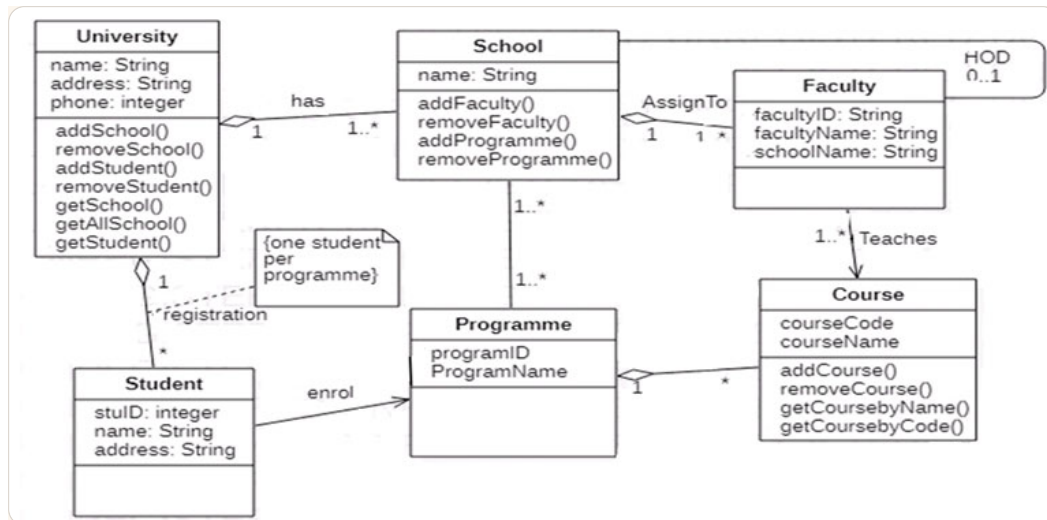


Figure 1.15: Class Diagram for Student Registration

Solution

Write in lab record

Assumptions

The system keeps the academic structure of one university. A university has a name, an address and a phone number and is made up of one or more schools. Each school is named and is responsible for a set of faculty members and a set of programmes. A faculty member has an ID, a name and the name of the school that employs them; at most one other faculty member is their head of department. Faculty members teach courses. A programme has an ID and a name, and a course has a code and a name. Students register with the university; each student has a roll number, a name and an address and enrolls in a programme. The program must let an administrator add and remove schools and students at university level, add and remove faculty and programmes at school level, keep a searchable catalogue of courses, and look up a school by name, a student by roll number and a course by name or code. Every relationship in the figure must be navigable from the code in the direction the figure shows.

- The reference listing is C++17, compiled with `clang++ -std=c++17 -Wall -Wextra`; the same classes, links and `main` are given in Rust, Python and TypeScript, and all four print the same output.
- Objects are created in `main` and linked with pointers; no object owns another, so there is no `new` or `delete`.
- A `1` or `0..1` end is a pointer; a `*` or `1..*` end is a `std::vector` of pointers.
- The four operations shown inside `Course` (`addCourse`, `removeCourse`, `getCoursebyName`, `getCoursebyCode`) manage the list of all courses, so they are `static` and work on one shared catalogue.
- The aggregation between `Course` and `Programme` is implemented exactly as drawn: the diamond is on `Course` with multiplicity `1`, and `*` is on `Programme`, so `Course` holds the vector of programmes.
- The `Teaches` arrow points from `Faculty` to `Course`, so only `Faculty` holds the link. The `enrol` arrow points from `Student` to `Programme`, so only `Student` holds the link.
- The constraint `one student per programme` on the registration link is read as: one registration binds a student to exactly one programme. `Student::enrol` refuses a second programme.
- `getAllSchool` returns the vector of schools; `getSchool` and `getStudent` return `nullptr` when nothing matches.

Diagram elements

Class	Attributes	Operations	Association ends
University	name: String, address: String, phone: integer	addSchool, removeSchool, addStudent, removeStudent, getSchool, getAllSchool, getStudent	has 1 to 1..* School (aggregation); registration 1 to * Student (aggregation, constraint one student per programme)
School	name: String	addFaculty, removeFaculty, addProgramme, removeProgramme	AssignTo 1 to 1..* Faculty (aggregation); 1..* to 1..* Programme
Faculty	facultyID: String, facultyName: String, schoolName: String	none	HOD 0..1 to Faculty (reflexive); Teaches 1..* Faculty to Course (arrow to Course)
Programme	programID, ProgramName	none	* Programme to 1 Course (diamond on Course)
Course	courseCode, courseName	addCourse, removeCourse, getCoursebyName, getCoursebyCode	see Programme and Faculty
Student	stuID: integer, name: String, address: String	none	enrol Student to Programme (arrow to Programme)

Steps

1. Create the folder `session-8` and save the listing below as `student_registration.cpp`.
2. Compile: `clang++ -std=c++17 -Wall -Wextra -o student_registration student_registration.cpp`. The build must print nothing.
3. Run `./student_registration` and paste the output into the record.
4. For another language, save the matching tab and run it: `rustc -O --edition 2021 student_registration.rs && ./student_registration`, `python3 student_registration.py`, or `node student_registration.ts` (Node 22.18 or later strips the types natively, no compiler needed).

Program

- Lab record: write one language only. Pick yours once and every page opens on it; the other tabs are the same solution for comparison.

C++

student_registration.cpp

```

1 // student_registration.cpp -- MCSL-222 Session 8, Q19
2 // Figure 1.15 (Student Registration) implemented in C++17.
3 // Build: clang++ -std=c++17 -Wall -Wextra -o student_registration student_registration.cpp
4 //
5 // Mapping used throughout:
6 //   class           → class
7 //   attribute       → public data member (same name as the figure)
8 //   operation       → member function (same name as the figure)
9 //   association end  → pointer for 1 / 0..1, std::vector of pointers for * / 1..*
10 // Objects are created in main and never deleted here; the pointers are links only.
11
12 #include <algorithm>
13 #include <iostream>
14 #include <string>
15 #include <vector>
16
17 class School;
18 class Faculty;
19 class Programme;
20 class Course;
21 class Student;
22
23 // Removes one link from an association vector. Every remove* operation uses it.
24 template <class T>
25 static void unlink(std::vector<T*>& v, T* p) {
26     v.erase(std::remove(v.begin(), v.end(), p), v.end());
27 }
28
29 // ----- Course
30 class Course {
31 public:
32     std::string courseCode;
33     std::string courseName;
34     std::vector<Programme*> programmes; // Course (1) <-- (*) Programme, as drawn
35
36     Course(std::string code, std::string name)
37         : courseCode(std::move(code)), courseName(std::move(name)) {}
38
39     // The four operations in the figure manage the list of courses, so they
40     // work on one shared catalogue rather than on a single Course object.
41     inline static std::vector<Course*> catalogue;
42
43     static void addCourse(Course& c) { catalogue.push_back(&c); }
44     static void removeCourse(Course& c) { unlink(catalogue, &c); }
45     static Course* getCoursebyName(const std::string& name) {
46         for (Course* c : catalogue)
47             if (c->courseName == name) return c;
48         return nullptr;
49     }
50     static Course* getCoursebyCode(const std::string& code) {
51         for (Course* c : catalogue)
52             if (c->courseCode == code) return c;
53         return nullptr;
54     }
55 };
56
57 // ----- Programme
58 class Programme {
59 public:
60     std::string programID;
61     std::string ProgramName;
62     std::vector<School*> schools; // School (1..*) --- (1..*) Programme
63
64     Programme(std::string id, std::string name)

```

```

65         : programID(std::move(id)), ProgramName(std::move(name)) {}
66     };
67
68     // ----- Faculty
69     class Faculty {
70     public:
71         std::string facultyID;
72         std::string facultyName;
73         std::string schoolName;
74         Faculty* hod = nullptr; // HOD 0..1 (reflexive association)
75         std::vector<Course*> courses; // Teaches: Faculty (1..*) → Course
76
77         Faculty(std::string id, std::string name)
78             : facultyID(std::move(id)), facultyName(std::move(name)) {}
79
80         void teaches(Course& c) { courses.push_back(&c); }
81     };
82
83     // ----- School
84     class School {
85     public:
86         std::string name;
87         std::vector<Faculty*> faculties; // AssignTo: School (1) ⇔ (1..*) Faculty
88         std::vector<Programme*> programmes; // School (1..*) --- (1..*) Programme
89
90         explicit School(std::string n) : name(std::move(n)) {}
91
92         void addFaculty(Faculty& f) {
93             faculties.push_back(&f);
94             f.schoolName = name; // the figure keeps the school name inside Faculty
95         }
96         void removeFaculty(Faculty& f) {
97             unlink(faculties, &f);
98             f.schoolName.clear();
99         }
100
101         void addProgramme(Programme& p) { // both ends are 1..*, so update both
102
103             programmes.push_back(&p);
104
105             p.schools.push_back(this);
106         }
107
108         void removeProgramme(Programme& p) {
109
110             unlink(programmes, &p);
111
112             unlink(p.schools, this);
113         }
114     };
115
116     // ----- Student
117     class Student {
118     public:
119
120         int stuID;

```

```

11
4   std::string name;
11
5   std::string address;
11
6   Programme* programme = nullptr; // enrol: Student → Programme
11
7
11
8   Student(int id, std::string n, std::string addr)
11
9       : stuID(id), name(std::move(n)), address(std::move(addr)) {}
12
0
12
1   // Constraint {one student per programme}: a registration binds a student
12
2   // to exactly one programme, so a second enrol() is refused.
12
3   bool enrol(Programme& p) {
12
4       if (programme != nullptr) {
12
5           std::cout << "   refused: " << name << " is already enrolled in "
12
6               << programme->ProgramName << " (one student per programme)\n";
12
7           return false;
12
8       }
12
9       programme = &p;
13
0       return true;
13
1   }
13
2 };
13
3
13
4 // ----- University
13
5 class University {
13
6 public:
13
7     std::string name;
13
8     std::string address;
13
9     int phone;
14
0     std::vector<School*> schools; // has: University (1) <-- (1..*) School
14
1     std::vector<Student*> students; // registration: University (1) <-- (*) Student
14
2
14
3     University(std::string n, std::string addr, int ph)
14
4         : name(std::move(n)), address(std::move(addr)), phone(ph) {}
14
5

```

```

14
6    void addSchool(School& s) { schools.push_back(&s); }
14
7    void removeSchool(School& s) { unlink(schools, &s); }
14
8    void addStudent(Student& s) { students.push_back(&s); }
14
9    void removeStudent(Student& s) { unlink(students, &s); }
15
0    School* getSchool(const std::string& n) const {
15
1        for (School* s : schools)
15
2            if (s->name == n) return s;
15
3        return nullptr;
15
4    }
15
5    const std::vector<School*>& getAllSchool() const { return schools; }
15
6    Student* getStudent(int id) const {
15
7        for (Student* s : students)
15
8            if (s->stuID == id) return s;
15
9        return nullptr;
16
0    }
16
1 };
16
2
16
3 // ----- main
16
4 static void printUniversity(const University& u) {
16
5     std::cout << u.name << ", " << u.address << ", phone " << u.phone << "\n";
16
6     for (const School* s : u.getAllSchool()) {
16
7         std::cout << " School: " << s->name << "\n";
16
8         for (const Faculty* f : s->faculties) {
16
9             std::cout << " Faculty " << f->facultyID << " " << f->facultyName
17
0                 << " (school " << f->schoolName << ")"
17
1                 << (f->hod ? ", HOD " + f->hod->facultyName : "") << " teaches";
17
2             for (const Course* c : f->courses) std::cout << " " << c->courseCode;
17
3             std::cout << "\n";
17
4         }
17
5         for (const Programme* p : s->programmes)
17
6             std::cout << " Programme " << p->programID << " " << p->ProgramName << "\n";
17
7     }

```

```
17
18     for (const Student* st : u.students)
17
19         std::cout << "  Student " << st->stuID << " " << st->name << ", " << st->address
18
19             << " → " << (st->programme ? st->programme->ProgramName : "not enrolled")
18
19             << "\n";
18
19 }
18
19
18
19 int main() {
18
19     University ignou("IGNOU", "Maidan Garhi, New Delhi", 29572514);
18
19
19
18     // Two schools
18
19     School socis("SOCIS");
18
19     School soms("SOMS");
19
19     ignou.addSchool(socis);
19
19     ignou.addSchool(soms);
19
19
19
19     // Faculties, with a HOD link inside SOCIS
19
19     Faculty f1("F01", "Dr. Sharma");
19
19     Faculty f2("F02", "Dr. Verma");
19
19     Faculty f3("F03", "Dr. Iyer");
19
19     socis.addFaculty(f1);
19
19     socis.addFaculty(f2);
19
19     soms.addFaculty(f3);
20
19     f2.hod = &f1; // HOD 0..1
20
19
19
19     // Programmes
20
19     Programme mca("MCA", "Master of Computer Applications");
20
19     Programme mba("MBA", "Master of Business Administration");
20
19     socis.addProgramme(mca);
20
19     soms.addProgramme(mba);
20
19
19
19     // Courses and the catalogue operations
20
19     Course c1("MCS-217", "Software Engineering");
```

```

21
0   Course c2("MCSL-222", "OOAD and Web Technologies Lab");
21
1   Course c3("MMPC-001", "Management Concepts");
21
2   Course::addCourse(c1);
21
3   Course::addCourse(c2);
21
4   Course::addCourse(c3);
21
5   c1.programmes.push_back(&mca); // Course (1) <-- (*) Programme
21
6   c2.programmes.push_back(&mca);
21
7   c3.programmes.push_back(&mca);
21
8   f1.teaches(c1);
21
9   f2.teaches(c2);
22
0   f3.teaches(c3);
22
1
22
2   // Students and registration
22
3   Student s1(2401, "Asha", "Jaipur");
22
4   Student s2(2402, "Ravi", "Pune");
22
5   ignou.addStudent(s1);
22
6   ignou.addStudent(s2);
22
7   s1.enrol(mca);
22
8   s2.enrol(mba);
22
9
23
0   std::cout << "--- after setup ---\n";
23
1   printUniversity(ignou);
23
2
23
3   std::cout << "--- constraint check ---\n";
23
4   s1.enrol(mba); // refused: already in MCA
23
5
23
6   std::cout << "--- lookups ---\n";
23
7   std::cout << "getSchool(\"SOMS\") → " << ignou.getSchool("SOMS")→name << "\n";
23
8   std::cout << "getSchool(\"SOL\") → "
23
9       << (ignou.getSchool("SOL") ? "found" : "nullptr") << "\n";
24
0   std::cout << "getStudent(2402) → " << ignou.getStudent(2402)→name << "\n";
24
1   std::cout << "getCoursebyCode(\"MCSL-222\") → "

```

```
24
2      << Course::getCoursebyCode("MCSL-222")→courseName << "\n";
24
3      std::cout << "getCoursebyName(\"Management Concepts\") → "
24
4      << Course::getCoursebyName("Management Concepts")→courseCode << "\n";
24
5      std::cout << "catalogue size: " << Course::catalogue.size() << "\n";
24
6
24
7      std::cout << "--- removals ---\n";
24
8      Course::removeCourse(c3);
24
9      socis.removeFaculty(f2);
25
0      soms.removeProgramme(mba);
25
1      ignou.removeStudent(s2);
25
2      ignou.removeSchool(soms);
25
3      std::cout << "catalogue size: " << Course::catalogue.size()
25
4      << ", f2.schoolName is \"" << f2.schoolName << "\""
25
5      << ", mba.schools.size() = " << mba.schools.size() << "\n";
25
6      printUniversity(ignou);
25
7      return 0;
25
8 }
```

Output

Compiled with zero warnings and run here; this is the real output. All four implementations print exactly this; the outputs were diffed and are identical byte for byte.

```
1 --- after setup ---
2 IGNOU, Maidan Garhi, New Delhi, phone 29572514
3   School: SOCIS
4     Faculty F01 Dr. Sharma (school SOCIS) teaches MCS-217
5     Faculty F02 Dr. Verma (school SOCIS), HOD Dr. Sharma teaches MCSL-222
6     Programme MCA Master of Computer Applications
7   School: SOMS
8     Faculty F03 Dr. Iyer (school SOMS) teaches MMPC-001
9     Programme MBA Master of Business Administration
10  Student 2401 Asha, Jaipur → Master of Computer Applications
11  Student 2402 Ravi, Pune → Master of Business Administration
12 --- constraint check ---
13   refused: Asha is already enrolled in Master of Computer Applications (one student per programme)
14 --- lookups ---
15   getSchool("SOMS") → SOMS
16   getSchool("SOL") → nullptr
17   getStudent(2402) → Ravi
18   getCoursebyCode("MCSL-222") → OOAD and Web Technologies Lab
19   getCoursebyName("Management Concepts") → MMPC-001
20   catalogue size: 3
21 --- removals ---
22   catalogue size: 2, f2.schoolName is "", mba.schools.size() = 0
23   IGNOU, Maidan Garhi, New Delhi, phone 29572514
24     School: SOCIS
25       Faculty F01 Dr. Sharma (school SOCIS) teaches MCS-217
26       Programme MCA Master of Computer Applications
27       Student 2401 Asha, Jaipur → Master of Computer Applications
```

Explanation

Diagram element	Where it is in the code
Six classes	Six <code>class</code> definitions in the same order as the figure's dependencies: <code>Course</code> , <code>Programme</code> , <code>Faculty</code> , <code>School</code> , <code>Student</code> , <code>University</code> . Forward declarations at the top let each class name the others before they are defined.
Attributes	Public data members with the exact names of the figure, for example <code>std::string facultyID</code> and <code>int phone</code> .
University has 1..* School	<code>std::vector<School*> schools</code> in <code>University</code> ; <code>addSchool</code> and <code>removeSchool</code> change it.
University registration * Student	<code>std::vector<Student*> students</code> ; <code>addStudent</code> , <code>removeStudent</code> , <code>getStudent</code> .
Constraint one student per programme	<code>Student::programme</code> is a single pointer and <code>enrol</code> returns <code>false</code> with a message when it is already set. The output line under <code>constraint check</code> shows the refusal.
School AssignTo 1..* Faculty	<code>std::vector<Faculty*> faculties</code> ; <code>addFaculty</code> also fills <code>Faculty::schoolName</code> , and <code>removeFaculty</code> clears it, so the attribute in the figure always agrees with the link.
School 1..* to 1..* Programme	Both ends are collections: <code>School::programmes</code> and <code>Programme::schools</code> . <code>addProgramme</code> and <code>removeProgramme</code> update both, which is why <code>mba.schools.size()</code> is 0 after removal.
Faculty HOD 0..1	<code>Faculty* hod = nullptr</code> inside <code>Faculty</code> itself: a reflexive association is a pointer to the same class.
Faculty Teaches Course	<code>std::vector<Course*> courses</code> and <code>teaches()</code> in <code>Faculty</code> only; the arrow is one-way.
Course 1 to * Programme	<code>std::vector<Programme*> programmes</code> in <code>Course</code> , filled in <code>main</code> with <code>push_back</code> .
Course operations	<code>inline static std::vector<Course*> catalogue</code> plus four <code>static</code> functions. <code>getCoursebyName</code> and <code>getCoursebyCode</code> scan the catalogue and return <code>nullptr</code> on a miss.
Remove operations	All call one helper, <code>unlink</code> , which is <code>std::remove</code> followed by <code>erase</code> .

Diagram element to code, where the four languages differ:

Element	C++	Rust	Python	TypeScript
Object and its links	Locals in <code>main</code> , links are raw pointers	Every object is an <code>Rc<RefCell<T>></code> ; a link is an <code>Rc</code> clone, read with <code>borrow()</code> and changed with <code>borrow_mut()</code>	Objects, links are plain references	Objects, links are typed references; every field, parameter and return carries a type and <code>tsc --strict</code> checks them
<code>1</code> or <code>0..1</code> end (<code>hod</code> , <code>programme</code>)	<code>Faculty* hod = nullptr</code>	<code>Option<Rc<RefCell<Faculty>>></code> , <code>None</code> when empty	<code>Optional[Faculty] = None</code>	union field <code>hod</code> : <code>Faculty</code> or <code>null</code> , starting as <code>null</code> ; a lookup hit in <code>main</code> needs <code>!</code> before <code>.name</code> because <code>getSchool</code> returns the same union
<code>*</code> or <code>1..*</code> end (<code>schools</code> , <code>courses</code>)	<code>std::vector<School*></code>	<code>Vec<Rc<RefCell<School>>></code>	<code>list[School]</code> with <code>field(default_factory=list)</code>	typed array <code>readonly schools</code> : <code>School[] = []</code> ; <code>readonly</code> because the array is never replaced, only pushed to
Back end of the two-way School to Programme link	<code>std::vector<School*></code> in <code>Programme</code>	<code>Vec<Weak<RefCell<School>>></code> ; <code>Weak</code> so the two <code>Rc</code> lists do not form a cycle, and <code>addProgramme</code> takes the school's <code>Rc</code> as <code>this</code> to make it	plain list	<code>readonly schools</code> : <code>School[]</code> in <code>Programme</code>
Course catalogue and its four operations	<code>inline static</code> <code>std::vector<Course*></code> and <code>static</code> member functions	<code>thread_local!</code> <code>static</code> <code>CATALOGUE</code> : <code>RefCell<Vec<..>></code> and associated functions called as <code>Course::addCourse(&c1)</code>	<code>catalogue</code> : <code>ClassVar[list[Course]]</code> and <code>@staticmethod</code>	<code>static readonly</code> <code>catalogue</code> : <code>Course[]</code> and <code>static</code> methods; the two getters return <code>Course</code> or <code>null</code>
Removing one link	<code>unlink</code> : <code>std::remove</code> then <code>erase</code>	<code>Vec::retain</code> with <code>Rc::ptr_eq</code>	<code>list.remove</code> , identity based because every class is <code>@dataclass(eq=False)</code>	generic <code>unlink<T>(arr: T[], p: T):</code> <code>indexOf</code> then <code>splice</code>
Attributes shared by two classes	Nothing special	Nothing special	Nothing special	<code>interface</code> <code>Addressed</code>

Element	C++	Rust	Python	TypeScript
(name , address in Student and University)				implemented by both, so one where() helper prints either
Figure names such as getCoursebyName	As drawn	As drawn, under #! [allow(non_snake_case)]	As drawn	As drawn

Question 20

Problem Statement

Write in lab record

Implement the Class Diagram of figure 1.18, in C++ or Java.

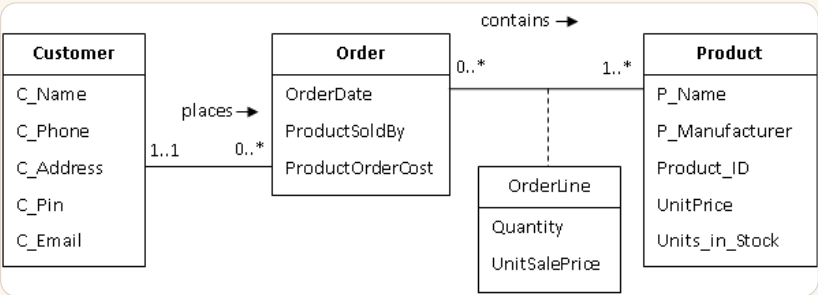


Figure 1.18: Customer Order Association Class

Solution

Write in lab record

Assumptions

A stationery shop records what its customers buy. A customer is identified by name, phone, address, PIN code and e-mail. Each customer places any number of orders, and every order belongs to exactly one customer. An order records the date, who sold it and the total cost. An order contains one or more products; a product has a name, a manufacturer, a product ID, a unit price and the units in stock. The same product can appear in many orders and at a different price each time, and the quantity bought is specific to that order and that product. Those two values, quantity and unit sale price, belong to neither the order nor the product but to the pair, so the figure attaches them to the association as the class OrderLine. The program must create products and a customer, let the customer place orders, add products to an order with a quantity and sale price, reduce stock, refuse a line that asks for more than the stock, and print each order with its lines and total.

- Types are not shown in the figure. Product_ID and Units_in_Stock are int, prices are double, all other attributes are std::string. OrderDate is a string in YYYY-MM-DD form.
- OrderLine is its own class with a pointer to its Order, a pointer to its Product, and the two attributes of the figure.
- places is two-way in the code: Customer::orders holds the 0..* end and Order::customer the 1..1 end. Customer::places sets both.
- contains is navigable towards Product, so Order holds its OrderLine objects by value in a std::vector and Product holds no back link.
- ProductOrderCost is derived: contains adds quantity times sale price to it. The figure names the attribute, so it is stored rather than recomputed.
- The 1..* at Product cannot be checked by the compiler; an order starts empty and the program relies on main adding at least one line.

Diagram elements

Class	Attributes	Association ends
Customer	C_Name, C_Phone, C_Address, C_Pin, C_Email	places: Customer 1..1 to 0..* Order (arrow to Order)
Order	OrderDate, ProductSoldBy, ProductOrderCost	contains: Order 0..* to 1..* Product (arrow to Product)
Product	P_Name, P_Manufacturer, Product_ID, UnitPrice, Units_in_Stock	none
OrderLine (association class)	Quantity, UnitSalePrice	attached to contains by a dashed line

Steps

1. Save the listing below as `customer_order.cpp` in the same `session-8` folder.
2. Compile: `clang++ -std=c++17 -Wall -Wextra -o customer_order customer_order.cpp`.
3. Run `./customer_order` and paste the output.
4. For another language, save the matching tab and run it: `rustc -0 --edition 2021 customer_order.rs && ./customer_order`, `python3 customer_order.py`, Or `node customer_order.ts`.

Program

- Lab record: write one language only. Pick yours once and every page opens on it; the other tabs are the same solution for comparison.

C++

`customer_order.cpp`

```

1 // customer_order.cpp -- MCSL-222 Session 8, Q20
2 // Figure 1.18 (Customer places Order, Order contains Product, OrderLine
3 // association class) implemented in C++17.
4 // Build: clang++ -std=c++17 -Wall -Wextra -o customer_order customer_order.cpp
5
6 #include <iomanip>
7 #include <iostream>
8 #include <string>
9 #include <vector>
10
11 class Order;
12 class Product;
13
14 // ----- Product
15 class Product {
16 public:
17     std::string P_Name;
18     std::string P_Manufacturer;
19     int Product_ID;
20     double UnitPrice;
21     int Units_in_Stock;
22
23     Product(std::string name, std::string maker, int id, double price, int stock)
24         : P_Name(std::move(name)), P_Manufacturer(std::move(maker)), Product_ID(id),
25           UnitPrice(price), Units_in_Stock(stock) {}
26 };
27
28 // ----- OrderLine
29 // The association class: one object per (Order, Product) pair, holding the
30 // attributes that belong to the link and not to either end.
31 class OrderLine {
32 public:
33     Order* order;
34     Product* product;
35     int Quantity;
36     double UnitSalePrice;
37
38     double lineTotal() const { return Quantity * UnitSalePrice; }
39 };
40
41 // ----- Order
42 class Order {
43 public:
44     std::string OrderDate;
45     std::string ProductSoldBy;
46     double ProductOrderCost = 0.0;
47     class Customer* customer = nullptr; // back link of "places" (1..1)
48     std::vector<OrderLine> lines; // contains: Order (0..*) → (1..*) Product
49
50     Order(std::string date, std::string soldBy)
51         : OrderDate(std::move(date)), ProductSoldBy(std::move(soldBy)) {}
52
53     // Adds one Product to this Order through an OrderLine.
54     bool contains(Product& p, int qty, double salePrice) {
55         if (qty ≤ 0 || qty > p.Units_in_Stock) {
56             std::cout << " refused: only " << p.Units_in_Stock << " x " << p.P_Name
57               << " in stock, asked for " << qty << "\n";
58             return false;
59         }
60         lines.push_back(OrderLine{this, &p, qty, salePrice});
61         p.Units_in_Stock -= qty;
62         ProductOrderCost += qty * salePrice;
63         return true;
64     }

```

```

65 };
66
67 // ----- Customer
68 class Customer {
69 public:
70     std::string C_Name;
71     std::string C_Phone;
72     std::string C_Address;
73     std::string C_Pin;
74     std::string C_Email;
75     std::vector<Order*> orders; // places: Customer (1..1) → (0..*) Order
76
77     Customer(std::string name, std::string phone, std::string addr, std::string pin,
78             std::string email)
79         : C_Name(std::move(name)), C_Phone(std::move(phone)), C_Address(std::move(addr)),
80           C_Pin(std::move(pin)), C_Email(std::move(email)) {}
81
82     void places(Order& o) {
83         orders.push_back(&o);
84         o.customer = this;
85     }
86 };
87
88 // ----- main
89 static void printOrder(const Order& o) {
90     std::cout << "Order dated " << o.OrderDate << " sold by " << o.ProductSoldBy
91               << " for " << (o.customer ? o.customer→C_Name : "nobody") << "\n";
92     for (const OrderLine& l : o.lines)
93         std::cout << " " << std::left << std::setw(12) << l.product→P_Name
94                   << std::right << std::setw(3) << l.Quantity << " x "
95                   << std::fixed << std::setprecision(2) << std::setw(9) << l.UnitSalePrice
96                   << " = " << std::setw(9) << l.lineTotal() << "\n";
97     std::cout << " ProductOrderCost = " << std::fixed << std::setprecision(2)
98               << o.ProductOrderCost << "\n";
99 }
100
101 0
102
103 1 int main() {
104
105     2 Product pen("Pen", "Cello", 101, 10.00, 500);
106
107     3 Product notebook("Notebook", "Classmate", 102, 45.00, 40);
108
109     4 Product stapler("Stapler", "Kangaro", 103, 120.00, 3);
110
111
112     6 Customer asha("Asha", "9876543210", "12 MG Road, Jaipur", "302001", "asha@example.com");
113
114
115     8 Order o1("2026-09-01", "Store counter");
116
117     9 Order o2("2026-09-15", "Online");
118
119     0 asha.places(o1);
120
121     1 asha.places(o2);
122
123
124     3 o1.contains(pen, 20, 9.50); // discounted sale price

```

```

11
4    o1.contains(notebook, 5, 45.00);
11
5    o2.contains(stapler, 2, 115.00);
11
6    o2.contains(stapler, 2, 115.00); // refused: only 1 left
11
7
11
8    for (const Order* o : asha.orders) printOrder(*o);
11
9
12
0    std::cout << "Stock left: pen " << pen.Units_in_Stock << ", notebook "
12
1    << notebook.Units_in_Stock << ", stapler " << stapler.Units_in_Stock << "\n";
12
2    std::cout << asha.C_Name << " has placed " << asha.orders.size() << " orders\n";
12
3    return 0;
12
4 }

```

Output

Compiled with zero warnings and run here; this is the real output. All four implementations print exactly this; the outputs were diffed and are identical byte for byte.

```

1  refused: only 1 x Stapler in stock, asked for 2
2  Order dated 2026-09-01 sold by Store counter for Asha
3  Pen          20 x      9.50 =    190.00
4  Notebook     5 x     45.00 =    225.00
5  ProductOrderCost = 415.00
6  Order dated 2026-09-15 sold by Online for Asha
7  Stapler      2 x    115.00 =    230.00
8  ProductOrderCost = 230.00
9  Stock left: pen 480, notebook 35, stapler 1
10 Asha has placed 2 orders

```

Check by hand: $20 \times 9.50 + 5 \times 45.00 = 415.00$; stock of pens $500 - 20 = 480$; the second stapler line asks for 2 when only $3 - 2 = 1$ is left, so it is refused.

Explanation

Diagram element	Where it is in the code
Product	<code>class Product</code> with the five attributes and a constructor; it has no link back to orders because the <code>contains</code> arrow points at it.
OrderLine association class	<code>class OrderLine</code> with <code>Order* order</code> , <code>Product* product</code> , <code>int Quantity</code> , <code>double UnitSalePrice</code> . One object exists per (order, product) pair. <code>lineTotal()</code> is the only extra.
Order contains 1..* Product	<code>std::vector<OrderLine> lines</code> in <code>Order</code> . <code>Order::contains(Product&, qty, price)</code> creates the line, reduces <code>Units_in_Stock</code> and adds to <code>ProductOrderCost</code> .
Customer places 0..* Order	<code>std::vector<Order*> orders</code> in <code>Customer</code> and <code>Order::customer</code> for the 1..1 end. <code>Customer::places</code> sets both, which is why <code>printOrder</code> can print the customer's name from the order.
Stock guard	The <code>if</code> at the top of <code>contains</code> refuses <code>qty</code> below 1 or above stock, which produced the first line of the output.
Printing	<code>printOrder</code> walks <code>lines</code> ; <code>std::setw</code> and <code>std::setprecision(2)</code> line up the money columns.

Diagram element to code, where the four languages differ:

Element	C++	Rust	Python	TypeScript
OrderLine's two links	<code>Order* order</code> , <code>Product* product</code>	<code>Weak<RefCell<Order>></code> back to the order, <code>Rc<RefCell<Product>></code> to the product	<code>order: Order</code> , <code>product: Product</code> fields of a <code>@dataclass(eq=False)</code>	<code>readonly order: Order</code> and <code>readonly product: Product</code> , set once in the constructor
Order contains 1..* Product	<code>std::vector<OrderLine></code> by value	<code>Vec<OrderLine></code> by value	<code>list[OrderLine]</code>	typed array <code>readonly lines: OrderLine[] = []</code>
places, two-way	<code>std::vector<Order*></code> plus <code>Customer*</code> <code>customer</code>	<code>Vec<Rc<..>></code> plus <code>Option<Weak<RefCell<Customer>>></code> ; <code>places</code> and <code>contains</code> are associated functions taking <code>this: &Rc<..></code> because the back link needs the <code>Rc</code>	list plus <code>Optional[Customer]</code>	<code>readonly orders: Order[]</code> plus the union field <code>customer: Customer</code> or <code>null</code>
Stock and cost updates through a link	write through the pointer	<code>p.borrow_mut().Units_in_Stock -= qty</code>	attribute assignment	field assignment; <code>Units_in_Stock</code> and <code>ProductOrderCost</code> are the only fields without <code>readonly</code> , which is exactly the set <code>contains</code> changes
Money columns	<code>std::fixed</code> , <code>setprecision(2)</code> , <code>setw(9)</code>	<code>{:>9.2}</code>	<code>f"{x:>9.2f}"</code>	<code>x.toFixed(2).padStart(9)</code>
Attributes the figure lists but <code>main</code> never reads (<code>C_Phone</code> , <code>P_Manufacturer</code>)	No warning	<code>#![allow(dead_code)]</code> , otherwise <code>rustc</code> warns about unread fields	No warning	No warning

Viva Questions

✗ Do not copy. Read for understanding and the viva

Q: How does a multiplicity of `*` differ from `1` in the code? **A:** A `1` or `0..1` end becomes a single pointer (`Faculty* hod`); a `*` or `1..*` end becomes a `std::vector` of pointers (`std::vector<School*> schools`).

Q: Why is an association class a separate class and not extra attributes on Order? **A:** Quantity and UnitSalePrice belong to one (order, product) pair. Putting them on Order would allow only one product per order; putting them on Product would give every order the same quantity. A separate `OrderLine` holds one pair.

Q: Where does aggregation show up in the code? **A:** Nowhere special: an aggregation is implemented the same way as a plain association, a pointer or vector in the whole. The diamond documents a whole-part meaning; the compiler does not enforce it.

Q: How is the reflexive HOD association implemented? **A:** As a pointer to the same class, `Faculty* hod`, set to `nullptr` for a faculty member with no head.

Q: Why are `addCourse` and the other Course operations static? **A:** The figure puts them inside Course, but they act on the set of all courses, not on one course. A static function on a shared catalogue matches that meaning.

Q: How is the constraint one student per programme enforced? **A:** `Student::programme` is one pointer, and `enrol` refuses when it is already set. The program prints the refusal so the evaluator can see it.

Q: Why are there no `new` or `delete` calls? **A:** All objects are locals in `main`; associations store non-owning pointers. That mirrors the diagram, where a link is a reference, not ownership.

Q: What does a one-way arrow change in the code? **A:** Only the class at the tail keeps a member for the link. Faculty holds `courses`; Course has no `faculties` vector.

Common Mistakes

✗ Do not copy. Read for understanding and the viva

- Storing a `*` end as a fixed array or a single pointer; the multiplicity says the number is open, so use `std::vector`.
- Updating only one end of a two-way association (for example pushing to `School::programmes` and forgetting `Programme::schools`), so removals leave dangling links.
- Flattening `OrderLine` into extra fields on `Order`, which silently limits an order to one product.
- Renaming the attributes and operations (for example `getCourseByCode` instead of the figure's `getCoursebyCode`); the evaluator compares against the figure.
- Submitting code that compiles with warnings. Fix every `-Wall -Wextra` message, especially unused parameters.
- Leaving `main` empty. The manual wants a program that runs, so exercise every operation and paste the output.

Session Summary

■ Write in lab record

- Question 19: `student_registration.cpp`, `.rs`, `.py` and `.ts`, six classes of figure 1.15 with every attribute, operation and association end, the enrol constraint enforced, run output attached (identical in all four languages)
- Question 20: `customer_order.cpp`, `.rs`, `.py` and `.ts`, Customer, Order, Product and the `OrderLine` association class of figure 1.18, run output attached (identical in all four languages)
- Problem description and assumptions for both figures, plus a diagram-element table for each

✎ Edit this page

< Session 7

Session 9 >