

Session 9

Implementing associations

Associations carry the interesting decisions: which side holds the reference, whether both sides do, and how an association class is represented. This session implements two small but complete examples.

Objectives

✖ Do not copy. Read for understanding and the viva

- Complete questions 21 to 22 of the manual: implementing associations
- 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
Q21	Implement the following Associations using C++/Java	Complete
Q22	Implement the following Associations using C++/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; use the language you chose in Session 8.
- Decide navigability: one-way associations need a reference on one side only; two-way ones need both plus code to keep them consistent.
- An association class (OrderLine style) becomes its own class holding references to both ends.

Question 21

Problem Statement

■ Write in lab record

Implement the following Associations using C++/Java.

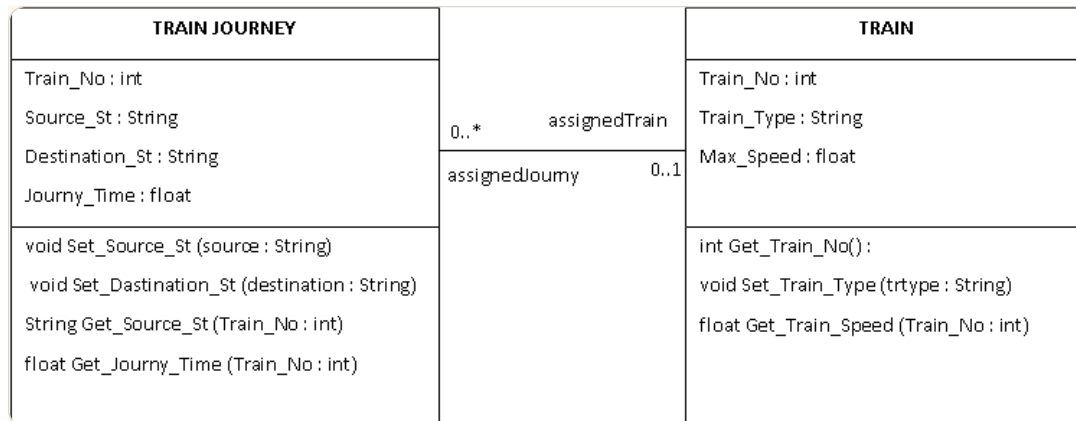


Figure 1.16: Train-Journey Association

Solution

■ Write in lab record

Assumptions

A railway keeps a list of trains and a list of journeys. A train has a number, a type such as Rajdhani or Shatabdi, and a maximum speed. A journey has a source station, a destination station and a journey time in hours, and records the number of the train that runs it. One train can be assigned to any number of journeys, including none, and a journey is assigned to at most one train at a time. Both ends of the link have role names in the figure, `assignedTrain` on the train side and `assignedJourney` on the journey side, and the line has no arrowhead, so the link must be navigable in both directions: from a journey you reach its train, and from a train you list its journeys. The program must let the operator assign a journey to a train, move a journey to another train, take a journey off its train, set and read the stations and the train type, and query a journey time or a train speed by train number. Whatever the sequence of operations, the two ends must never disagree.

- Two-way association: `TrainJourney::assignedTrain` is a pointer (`0..1`) and `Train::assignedJourney` is a `std::vector` of pointers (`0..*`).
- Both ends change only inside two free functions, `assign` and `unassign`. `main` never touches the pointers or the vector directly.
- `assign` first calls `unassign`, so a journey can never appear under two trains.
- The figure gives `Train_No` as an attribute of `TrainJourney` and also as a parameter of the getters. `assign` copies the train's number into the journey, and the getters answer only when the number passed matches; otherwise they return a marker (`not this train`) or `-1`).
- Attribute and operation names keep the figure's spelling, including `Journey_Time` and `Set_Dastination_St`.

Diagram elements

Class	Attributes	Operations
TrainJourney	Train_No: int, Source_St: String, Destination_St: String, Journey_Time: float	Set_Source_St(source: String), Set_Dastination_St(destination: String), Get_Source_St(Train_No: int): String, Get_Journey_Time(Train_No: int): float
Train	Train_No: int, Train_Type: String, Max_Speed: float	Get_Train_No(): int, Set_Train_Type(trtype: String), Get_Train_Speed(Train_No: int): float

Association: TrainJourney `0..*` (role `assignedJourney`) to Train `0..1` (role `assignedTrain`), no arrowhead, so two-way.

How the objects point at each other after the first three `assign` calls in `main` (each arrow is a pointer stored in the object at its tail):

```

1  rajdhani (Train 12951)          shatabdi (Train 12009)
2  assignedJourney: [ j1, j2 ]    assignedJourney: [ j3 ]
3      |      |                    |
4      v      v                    v
5      j1     j2                   j3
6  assignedTrain ---> rajdhani     assignedTrain ---> shatabdi
7  assignedTrain ---> rajdhani

```

Every journey in a train's vector points back at that train, and no journey is in two vectors. `assign` and `unassign` are the only code that may change this picture.

Steps

1. Save the listing below as `train_journey.cpp` in a `session-9` folder.
2. Compile: `clang++ -std=c++17 -Wall -Wextra -o train_journey train_journey.cpp`.
3. Run `./train_journey` and paste the output.
4. For another language, save the matching tab and run it: `rustc -0 --edition 2021 train_journey.rs && ./train_journey`, `python3 train_journey.py`, or `node train_journey.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++

train_journey.cpp

```

1 // train_journey.cpp -- MCSL-222 Session 9, Q21
2 // Figure 1.16 (Train Journey -- Train) implemented in C++17 as a TWO-WAY
3 // association: TrainJourney.assignedTrain (0..1) and Train.assignedJourney (0..*).
4 // Build: clang++ -std=c++17 -Wall -Wextra -o train_journey train_journey.cpp
5
6 #include <algorithm>
7 #include <iostream>
8 #include <string>
9 #include <vector>
10
11 class Train;
12
13 // ----- TrainJourney
14 class TrainJourney {
15 public:
16     int Train_No = 0;
17     std::string Source_St;
18     std::string Destination_St;
19     float Journey_Time = 0.0f;
20     Train* assignedTrain = nullptr; // role assignedTrain, multiplicity 0..1
21
22     TrainJourney(std::string src, std::string dst, float hours)
23         : Source_St(std::move(src)), Destination_St(std::move(dst)), Journey_Time(hours) {}
24
25     void Set_Source_St(const std::string& source) { Source_St = source; }
26     void Set_Destination_St(const std::string& destination) { Destination_St = destination; }
27     // The figure passes Train_No to the getters, so they answer only for
28     // the train this journey is assigned to.
29     std::string Get_Source_St(int train_no) const {
30         return train_no == Train_No ? Source_St : std::string("(not this train)");
31     }
32     float Get_Journey_Time(int train_no) const {
33         return train_no == Train_No ? Journey_Time : -1.0f;
34     }
35 };
36
37 // ----- Train
38 class Train {
39 public:
40     int Train_No;
41     std::string Train_Type;
42     float Max_Speed;
43     std::vector<TrainJourney*> assignedJourney; // role assignedJourney, multiplicity 0..*
44
45     Train(int no, std::string type, float speed)
46         : Train_No(no), Train_Type(std::move(type)), Max_Speed(speed) {}
47
48     int Get_Train_No() const { return Train_No; }
49     void Set_Train_Type(const std::string& trtype) { Train_Type = trtype; }
50     float Get_Train_Speed(int train_no) const {
51         return train_no == Train_No ? Max_Speed : -1.0f;
52     }
53 };
54
55 // ----- keeping both ends in step
56 // Both ends change in one place, so a journey can never point at a train
57 // that does not list it, and vice versa.
58 static void unassign(TTrainJourney& j) {
59     if (Train* t = j.assignedTrain) {
60         t->assignedJourney.erase(

```

```

61         std::remove(t->assignedJourney.begin(), t->assignedJourney.end(), &j),
62         t->assignedJourney.end());
63     j.assignedTrain = nullptr;
64     j.Train_No = 0;
65 }
66 }
67
68 static void assign(Train& t, TrainJourney& j) {
69     unassign(j); // a journey has at most one train (0..1)
70     j.assignedTrain = &t;
71     j.Train_No = t.Train_No;
72     t.assignedJourney.push_back(&j);
73 }
74
75 // ----- main
76 static void printTrain(const Train& t) {
77     std::cout << "Train " << t.Get_Train_No() << " (" << t.Train_Type << ", "
78         << t.Max_Speed << " km/h) runs " << t.assignedJourney.size() << " journey(s)\n";
79     for (const TrainJourney* j : t.assignedJourney)
80         std::cout << " " << j->Source_St << " → " << j->Destination_St << ", "
81             << j->Journey_Time << " h, Train_No stored in journey = " << j->Train_No
82             << "\n";
83 }
84
85 int main() {
86     Train rajdhani(12951, "Rajdhani", 130.0f);
87     Train shatabdi(12009, "Shatabdi", 150.0f);
88
89     TrainJourney j1("Mumbai", "Delhi", 15.5f);
90     TrainJourney j2("Delhi", "Mumbai", 15.75f);
91     TrainJourney j3("Mumbai", "Ahmedabad", 6.25f);
92
93     assign(rajdhani, j1);
94     assign(rajdhani, j2);
95     assign(shatabdi, j3);
96
97     std::cout << "--- after assignment ---\n";
98     printTrain(rajdhani);
99     printTrain(shatabdi);
100
101     std::cout << "--- operations from the figure ---\n";
102
103     j3.Set_Source_St("Mumbai Central");
104
105     j3.Set_Destination_St("Ahmedabad Jn");
106
107     shatabdi.Set_Train_Type("Shatabdi Express");
108
109     std::cout << "j3.Get_Source_St(12009) = " << j3.Get_Source_St(12009) << "\n";
110
111     std::cout << "j3.Get_Source_St(12951) = " << j3.Get_Source_St(12951) << "\n";
112
113     std::cout << "j3.Get_Journey_Time(12009) = " << j3.Get_Journey_Time(12009) << "\n";
114
115     std::cout << "shatabdi.Get_Train_Speed(12009) = " << shatabdi.Get_Train_Speed(12009) << "\n";
116
117     std::cout << "j1.assignedTrain->Train_Type = " << j1.assignedTrain->Train_Type << "\n";
118
119 }

```

```

11
1    std::cout << "--- move j2 to the Shatabdi (0..1 keeps only one train) ---\n";
11
2    assign(shatabdi, j2);
11
3    printTrain(rajdhani);
11
4    printTrain(shatabdi);
11
5
11
6    std::cout << "--- unassign j3 ---\n";
11
7    unassign(j3);
11
8    std::cout << "j3.assignedTrain is " << (j3.assignedTrain ? "set" : "nullptr")
11
9                << ", shatabdi lists " << shatabdi.assignedJourney.size() << " journey(s)\n";
12
10   return 0;
12
11 }

```

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 assignment ---
2  Train 12951 (Rajdhani, 130 km/h) runs 2 journey(s)
3   Mumbai → Delhi, 15.5 h, Train_No stored in journey = 12951
4   Delhi → Mumbai, 15.75 h, Train_No stored in journey = 12951
5  Train 12009 (Shatabdi, 150 km/h) runs 1 journey(s)
6   Mumbai → Ahmedabad, 6.25 h, Train_No stored in journey = 12009
7  --- operations from the figure ---
8  j3.Get_Source_St(12009) = Mumbai Central
9  j3.Get_Source_St(12951) = (not this train)
10 j3.Get_Journey_Time(12009) = 6.25
11 shatabdi.Get_Train_Speed(12009) = 150
12 j1.assignedTrain→Train_Type = Rajdhani
13 --- move j2 to the Shatabdi (0..1 keeps only one train) ---
14 Train 12951 (Rajdhani, 130 km/h) runs 1 journey(s)
15   Mumbai → Delhi, 15.5 h, Train_No stored in journey = 12951
16 Train 12009 (Shatabdi Express, 150 km/h) runs 2 journey(s)
17   Mumbai Central → Ahmedabad Jn, 6.25 h, Train_No stored in journey = 12009
18   Delhi → Mumbai, 15.75 h, Train_No stored in journey = 12009
19 --- unassign j3 ---
20 j3.assignedTrain is nullptr, shatabdi lists 1 journey(s)

```

Explanation

Diagram element	Where it is in the code
Role assignedTrain, 0..1	<code>Train* assignedTrain = nullptr</code> in <code>TrainJourney</code> . A pointer that may be null is exactly <code>0..1</code> .
Role assignedJourney, 0..*	<code>std::vector<TrainJourney*> assignedJourney</code> in <code>Train</code> .
Two-way navigability	Both members exist. <code>j1.assignedTrain→Train_Type</code> walks journey to train; <code>printTrain</code> walks train to journeys.
Consistency	<code>assign</code> sets the pointer, copies <code>Train_No</code> and pushes into the vector in one place. <code>unassign</code> erases from the vector and clears the pointer. Nothing else writes the two ends.
0..1 upper bound	<code>assign</code> starts with <code>unassign(j)</code> , so moving <code>j2</code> to the Shatabdi removes it from the Rajdhani, as the output after the move shows: Rajdhani 1 journey, Shatabdi 2.
Getters with a Train_No parameter	<code>Get_Source_St(12951)</code> on a Shatabdi journey returns <code>(not this train)</code> ; with the matching number it returns the station.
Setters	<code>Set_Source_St</code> , <code>Set_Dastination_St</code> and <code>Set_Train_Type</code> assign the string; the change shows in the second print of the Shatabdi.

Diagram element to code, where the four languages differ:

Element	C++	Rust	Python	TypeScript
assignedTrain, 0..1	Train* assignedTrain = nullptr	Option<Weak<RefCell<Train>>> ; Weak because the train already holds an Rc to the journey, and a train() helper upgrades it	Optional[Train] = None	union field assignedTrain: Train or null , starting as null ; main writes j1.assignedTrain! where it knows the link is set
assignedJourney, 0..*	std::vector<TrainJourney*>	Vec<Rc<RefCell<TrainJourney>>>	List[TrainJourney]	typed array readonly assignedJourney: TrainJourney[] = []
assign and unassign	free functions writing through pointers	free functions taking &Rc<..> and using borrow_mut() on both ends	free functions	typed free functions, assign(t: Train, j: TrainJourney): void
Train_No drawn in both boxes	plain member in each class	plain field in each struct	plain field in each class	interface TrainNumbered implemented by both classes; the isTrain helper the getters call accepts either
Removing the journey in unassign	std::remove then erase	retain with Rc::ptr_eq	List.remove (identity, eq=False)	splice(indexOf(j), 1)
Printing Max_Speed and Journey_Time as 130 and 15.5	default stream format	{ } prints 130 and 15.5	print(130.0) would give 130.0 , so a one-line helper g(x) returns f" {x:g}"	template literal prints 130

Question 22

Problem Statement

Write in lab record

Implement the following Associations using C++/Java.

Person		BankAccount
Person_ID : String	1 has *	Acc_No : String
Name : String		Acc_Balance : double
void addAccount (BankAccount a)		void Credit(double amount) void Withdraw(double amount)

Figure 1.17: Person and Bank Account

Solution

■ Write in lab record

Assumptions

A bank keeps, for each customer, the accounts that customer holds. A person has an ID and a name. A bank account has an account number and a balance, and supports a credit that increases the balance and a withdrawal that decreases it. One person can hold any number of accounts and every account belongs to exactly one person. The figure gives Person an operation `addAccount(BankAccount a)` and gives BankAccount no operation or attribute that refers to a person, so the link is navigable from person to account only. From a person you can reach and total all their accounts; from an account you cannot ask who owns it. The program must create a person and two accounts, attach the accounts, credit and withdraw amounts, refuse a withdrawal that exceeds the balance and a credit that is not positive, and print the accounts with a total.

- One-way association: `Person::accounts` is a `std::vector` of pointers (`*`). `BankAccount` has no member that refers to `Person`.
- `Credit` rejects an amount of zero or less. `Withdraw` rejects an amount of zero or less and any amount above the balance; the account never goes negative.
- The `1` at the Person end (every account has exactly one owner) cannot be checked from the account side in a one-way design. The program trusts `main` to add each account to one person only. Making the link two-way is the fix if that check is required; Q21 shows how.
- `totalBalance` is an extra helper for printing; the figure's attributes and operations are otherwise unchanged.

Diagram elements

Class	Attributes	Operations
Person	Person_ID: String, Name: String	addAccount(BankAccount a)
BankAccount	Acc_No: String, Acc_Balance: double	Credit(double amount), Withdraw(double amount)

Association: Person `1` has BankAccount `*`, navigable from Person only.

Pointers after the two `addAccount` calls in `main`:

```

1  asha (Person P001)
2  accounts: [ savings, current ]
3      |           |
4      v           v
5  savings  current      (no arrow back to asha:
6  SB-1001  CA-2001      BankAccount has no Person member)
```

Steps

1. Save the listing below as `person_account.cpp` in the `session-9` folder.
2. Compile: `clang++ -std=c++17 -Wall -Wextra -o person_account person_account.cpp`.

3. Run `./person_account` and paste the output.

4. For another language, save the matching tab and run it: `rustc -0 --edition 2021 person_account.rs && ./person_account`, `python3 person_account.py`, Or `node person_account.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++

person_account.cpp

```

1 // person_account.cpp -- MCSL-222 Session 9, Q22
2 // Figure 1.17 (Person 1 has * BankAccount) implemented in C++17 as a ONE-WAY
3 // association: Person knows its accounts, BankAccount knows nothing of Person.
4 // Build: clang++ -std=c++17 -Wall -Wextra -o person_account person_account.cpp
5
6 #include <iomanip>
7 #include <iostream>
8 #include <string>
9 #include <vector>
10
11 // ----- BankAccount
12 class BankAccount {
13 public:
14     std::string Acc_No;
15     double Acc_Balance;
16
17     BankAccount(std::string no, double opening) : Acc_No(std::move(no)), Acc_Balance(opening) {}
18
19     void Credit(double amount) {
20         if (amount ≤ 0) {
21             std::cout << " refused: credit amount must be positive\n";
22             return;
23         }
24         Acc_Balance += amount;
25     }
26     void Withdraw(double amount) {
27         if (amount ≤ 0 || amount > Acc_Balance) {
28             std::cout << " refused: cannot withdraw " << amount << " from " << Acc_No
29                 << " (balance " << Acc_Balance << ")\n";
30             return;
31         }
32         Acc_Balance -= amount;
33     }
34 };
35
36 // ----- Person
37 class Person {
38 public:
39     std::string Person_ID;
40     std::string Name;
41     std::vector<BankAccount*> accounts; // has: Person (1) → (*) BankAccount
42
43     Person(std::string id, std::string name) : Person_ID(std::move(id)), Name(std::move(name)) {}
44
45     void addAccount(BankAccount& a) { accounts.push_back(&a); }
46
47     double totalBalance() const {
48         double sum = 0;
49         for (const BankAccount* a : accounts) sum += a->Acc_Balance;
50         return sum;
51     }
52 };
53
54 // ----- main
55 static void printPerson(const Person& p) {
56     std::cout << p.Name << " (" << p.Person_ID << ") holds " << p.accounts.size()
57         << " account(s)\n";
58     for (const BankAccount* a : p.accounts)
59         std::cout << " " << a->Acc_No << " balance " << std::fixed << std::setprecision(2)
60             << a->Acc_Balance << "\n";

```

```
61     std::cout << "   total " << std::fixed << std::setprecision(2) << p.totalBalance() << "\n";
62 }
63
64 int main() {
65     Person asha("P001", "Asha");
66     BankAccount savings("SB-1001", 5000.00);
67     BankAccount current("CA-2001", 12000.00);
68
69     asha.addAccount(savings);
70     asha.addAccount(current);
71
72     std::cout << "--- after addAccount ---\n";
73     printPerson(asha);
74
75     std::cout << "--- Credit and Withdraw ---\n";
76     savings.Credit(1500.00);
77     current.Withdraw(2000.00);
78     savings.Withdraw(9000.00); // refused: more than balance
79     current.Credit(-50.00);    // refused: not positive
80     printPerson(asha);
81
82     // One-way navigation: from an account there is no way back to Asha.
83     // The line below would not compile, which is the point of the figure:
84     // std::cout << savings.owner->Name;
85     return 0;
86 }
```

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 addAccount ---
2 Asha (P001) holds 2 account(s)
3   SB-1001  balance 5000.00
4   CA-2001  balance 12000.00
5   total 17000.00
6 --- Credit and Withdraw ---
7   refused: cannot withdraw 9000.00 from SB-1001 (balance 6500.00)
8   refused: credit amount must be positive
9 Asha (P001) holds 2 account(s)
10  SB-1001  balance 6500.00
11  CA-2001  balance 10000.00
12  total 16500.00
```

Check by hand: $5000 + 1500 = 6500$ on the savings account, $12000 - 2000 = 10000$ on the current account, total 16500 .

Explanation

Diagram element	Where it is in the code
Person 1 has * BankAccount	<code>std::vector<BankAccount*> accounts</code> in <code>Person</code> ; <code>addAccount(BankAccount& a)</code> pushes a pointer.
One-way navigability	<code>BankAccount</code> has no <code>Person*</code> member. The commented line at the end of <code>main</code> , <code>savings.owner →Name</code> , would not compile, which is the point.
Credit(double amount)	Adds to <code>Acc_Balance</code> after the positive check.
Withdraw(double amount)	Subtracts after checking <code>amount</code> is positive and not above <code>Acc_Balance</code> ; the refusal for <code>9000</code> on a balance of <code>6500</code> is the first refused line of the output.
Printing	<code>printPerson</code> walks the vector from the <code>Person</code> side, which is the only direction available.

Diagram element to code, where the four languages differ:

Element	C++	Rust	Python	TypeScript
Person has * BankAccount	<code>std::vector<BankAccount*></code>	<code>Vec<Rc<RefCell<BankAccount>>></code> ; <code>Person</code> itself is a plain struct because nothing links back to it	<code>List[BankAccount]</code>	typed array <code>readonly</code> <code>accounts:</code> <code>BankAccount[] =</code> <code>[]</code>
Credit and Withdraw changing the balance	member function on the object	<code>savings.borrow_mut().Credit(1500.0)</code> ; the account is shared with <code>Person</code> , so mutation goes through the <code>RefCell</code>	method	typed method <code>Credit(amount: number): void</code> ; <code>Acc_Balance</code> is the one field without <code>readonly</code>
What the commented <code>savings.owner</code> line would do	compile error	compile error	<code>AttributeError</code> at run time	<code>tsc</code> error, <code>owner</code> is not a property of <code>BankAccount</code> ; a bare <code>node</code> run strips types without checking and throws <code>TypeError</code> at run time
<code>totalBalance</code>	loop	<code>iter().map(..).sum()</code>	<code>sum(generator)</code>	<code>reduce</code>
Money format	<code>std::fixed</code> , <code>setprecision(2)</code>	<code>{:.2}</code>	<code>f"{x:.2f}"</code>	<code>toFixed(2)</code>

Viva Questions

✗ Do not copy. Read for understanding and the viva

Q: What decides whether an association is one-way or two-way in code? **A:** Navigability. An arrowhead, or an operation on one side only, means only that class stores the link. Role names on both ends with no arrowhead, as in figure 1.16, mean both classes store it.

Q: Why does the train program change both ends inside `assign` and never in `main`? **A:** A two-way link is two members that must agree. If any code can set one without the other, they drift apart. One function that always updates both is the only way to guarantee consistency.

Q: What does `0..1` become in C++? **A:** A pointer that may be `nullptr`. `1` is a pointer that must not be null; `*` and `0..*` are a `std::vector`.

Q: What is lost with one-way navigation in figure 1.17? **A:** The `1` at `Person` cannot be enforced or even checked from the account, and you cannot find an account's owner without scanning every person.

Q: What would change to make `Person` and `BankAccount` two-way? **A:** Add `Person* owner` to `BankAccount`, set it in `addAccount`, and refuse `addAccount` when `owner` is already set.

Q: Why keep the misspelt names such as `Journey_Time`? **A:** The lab record is checked against the figure. Matching names show the mapping is exact; fixing spellings is a separate remark, not a silent change.

Q: Why does `assign` call `unassign` first? **A:** The `0..1` on the train end means a journey has at most one train. Removing the old link before adding the new one keeps that bound.

Common Mistakes

✗ Do not copy. Read for understanding and the viva

- Setting `assignedTrain` on the journey and forgetting to push into `assignedJourney`, or the reverse; the printed lists then disagree with the pointers.
- Erasing from a vector inside a range-for loop over the same vector. Use the erase-remove idiom on a copy of the pointer, as `unassign` does.
- Adding an owner pointer to `BankAccount` when the figure shows a one-way link, then claiming it matches the figure.
- Letting `Withdraw` drive the balance negative because the check compares the wrong way round.
- Comparing floats printed with different precision and thinking the values changed; set the precision once.

Session Summary

■ Write in lab record

- Question 21: `train_journey.cpp`, `.rs`, `.py` and `.ts`, two-way `TrainJourney` to `Train` association with `assign` and `unassign` keeping both ends in step, run output attached (identical in all four languages)
- Question 22: `person_account.cpp`, `.rs`, `.py` and `.ts`, one-way `Person` to `BankAccount` association with guarded `Credit` and `Withdraw`, 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

< Previous
Session 8

Next >
Session 10