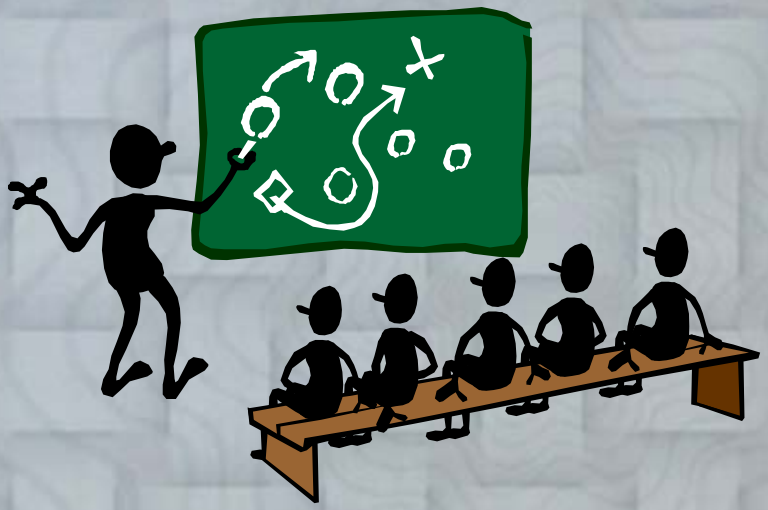
CSC 330 Object Oriented Software Design



Software Design Phase

Overview

- Overview
- Design and abstraction
- Action-oriented design
- Data flow analysis
- Transaction analysis
- Data-oriented design
- Object-oriented design
- Elevator problem: object-oriented design
- Air Gourmet Case Study: object-oriented design

Data and Actions

- Two aspects of a product
  - Actions which operate on data
  - Data on which actions operate
- The two basic ways of designing a product
  - Action-oriented design
  - Data-oriented design
- Third way
  - Hybrid methods
  - For example, object-oriented design

Design Activities

- Architectural design
- Detailed design
- Design testing

Architectural Design

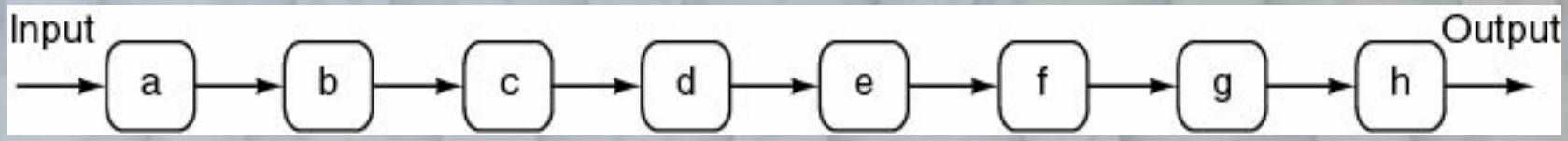
- Input: Specifications
- Output: Modular decomposition
- Abstraction

Detailed Design

- Each module is designed
  - Specific algorithms
  - Data structures

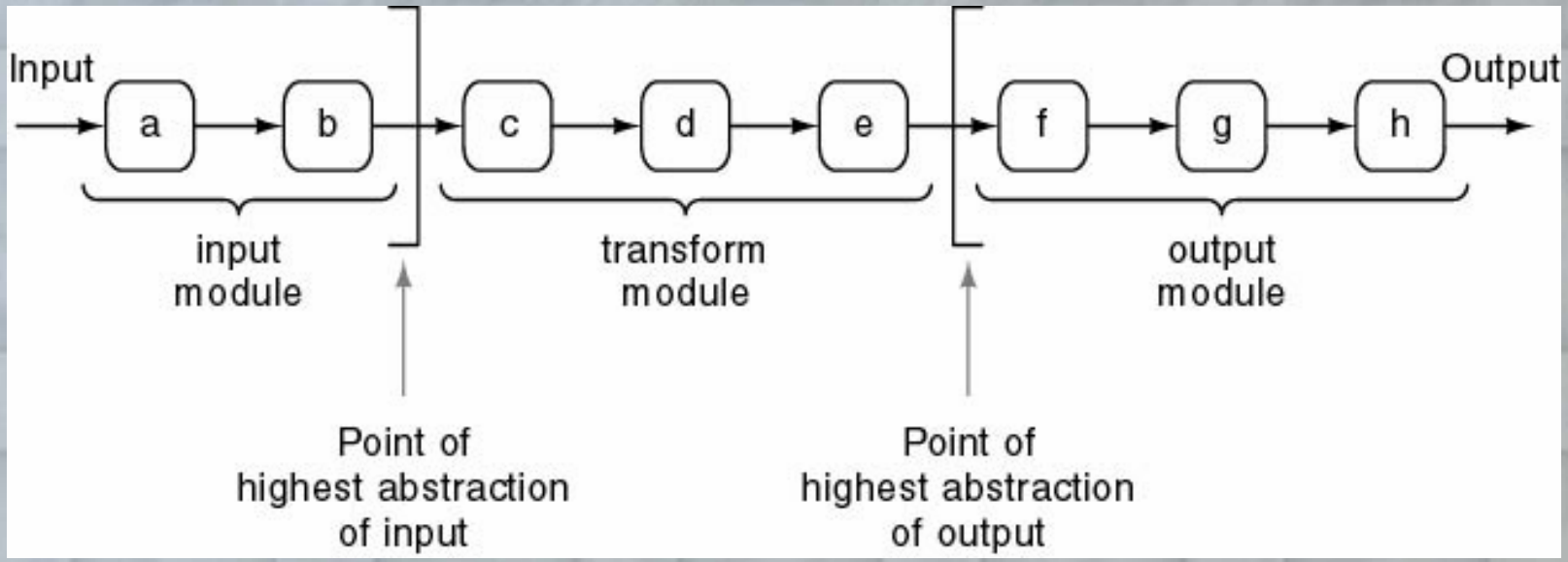
Action-Oriented Design Methods

- Data flow analysis
- When to use it
  - With most specification methods (Structured Systems Analysis here)
- Key point: Have detailed action



Data Flow Analysis

- Product transforms input into output
- Determine
  - "Point of highest abstraction of input"
  - "Point of highest abstract of output"



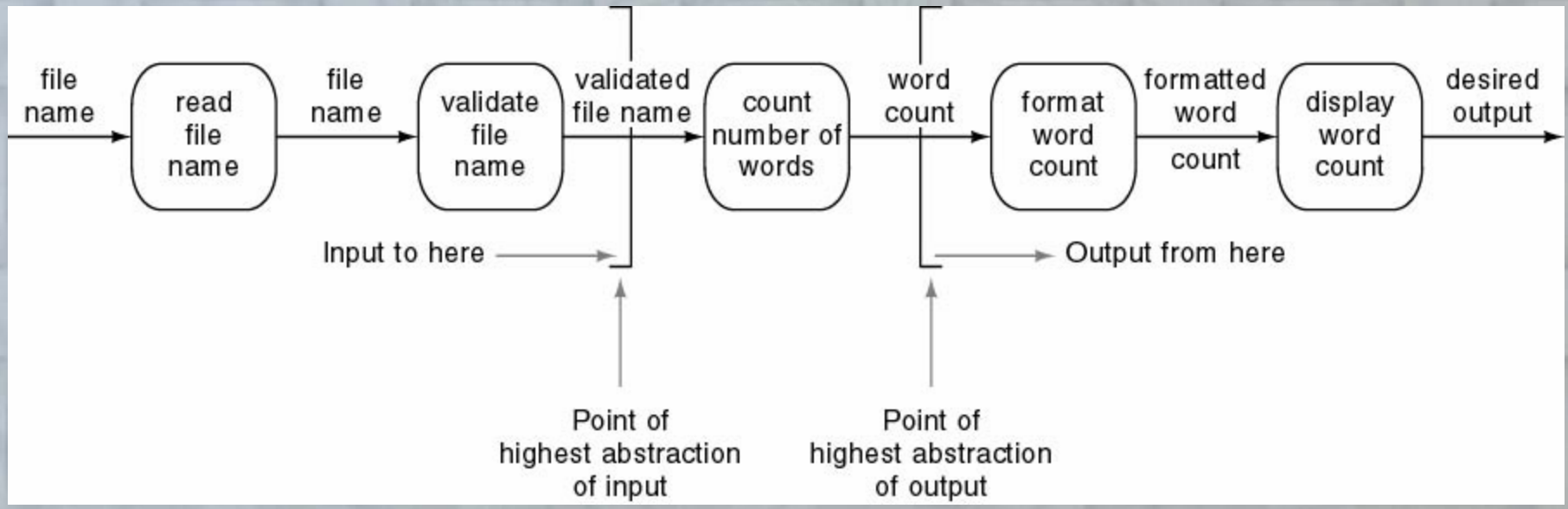
Data Flow Analysis cont'd

- Decompose into three modules
- Repeat stepwise until each module has high cohesion
  - Minor modifications may be needed to lower coupling

Data Flow Analysis cont'd

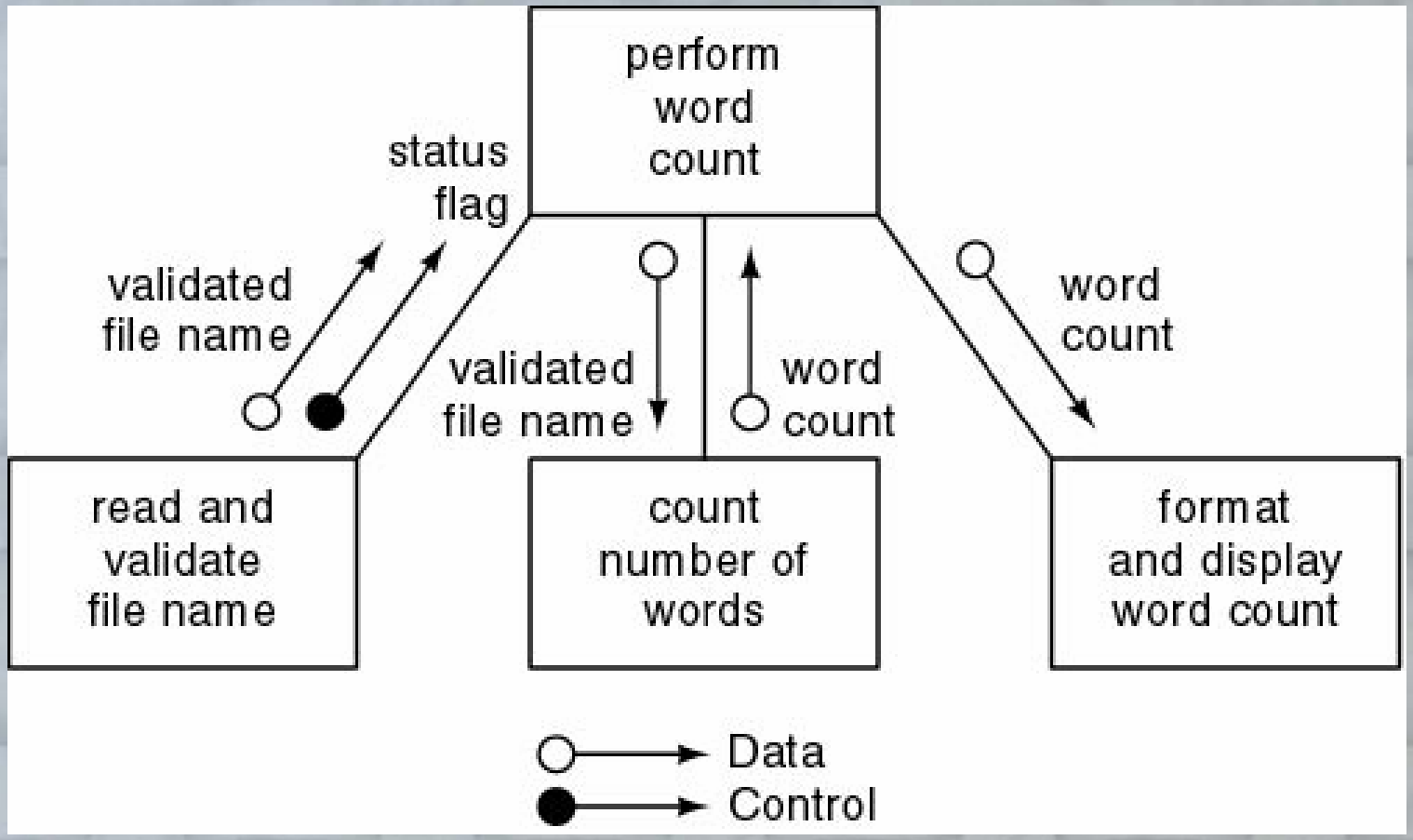
- Example

  Design a product which takes as input a file name, and returns the number of words in that file



Data Flow Analysis Example cont'd

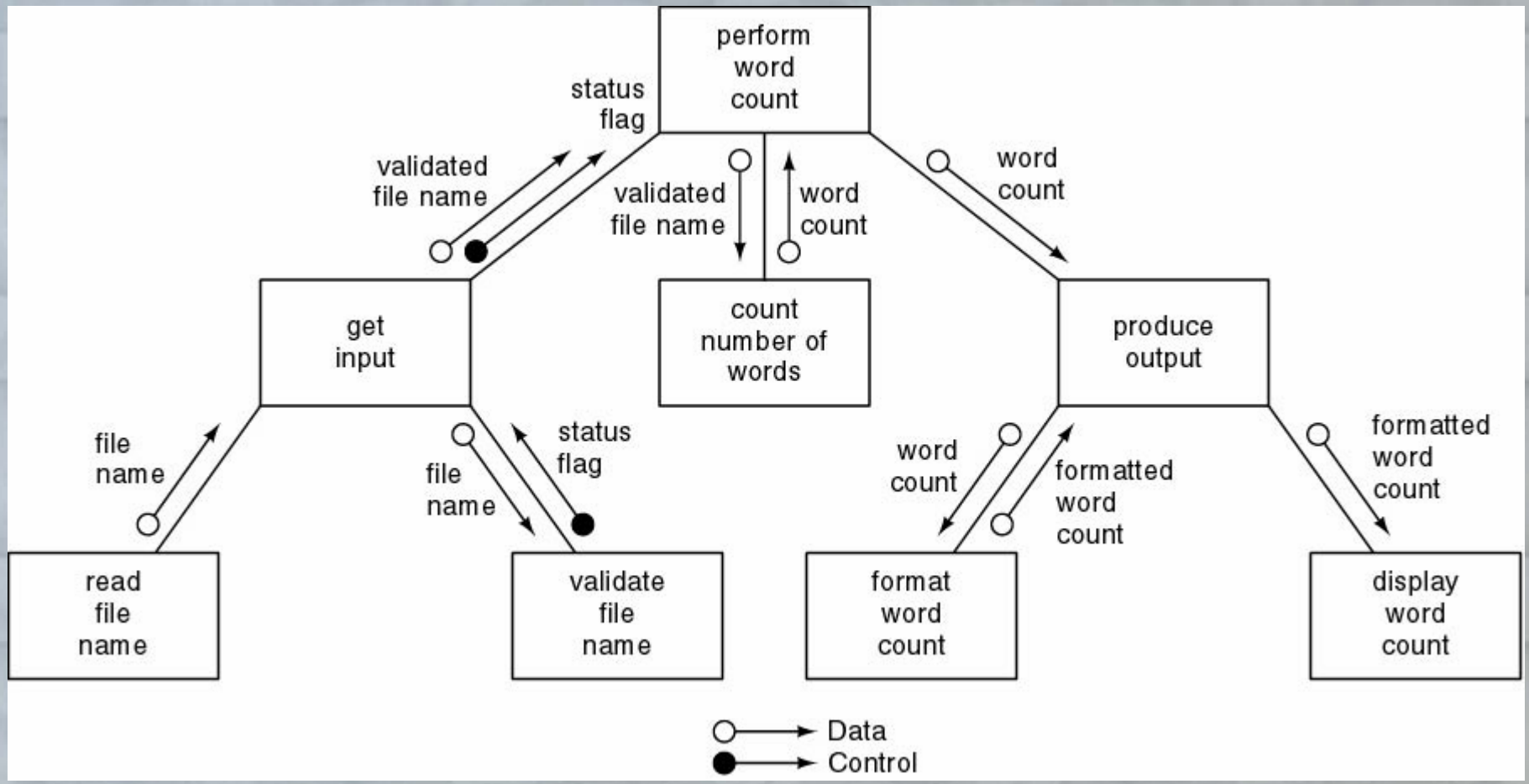
- First refinement



- Refine modules of communicational cohesion

Data Flow Analysis Example cont'd

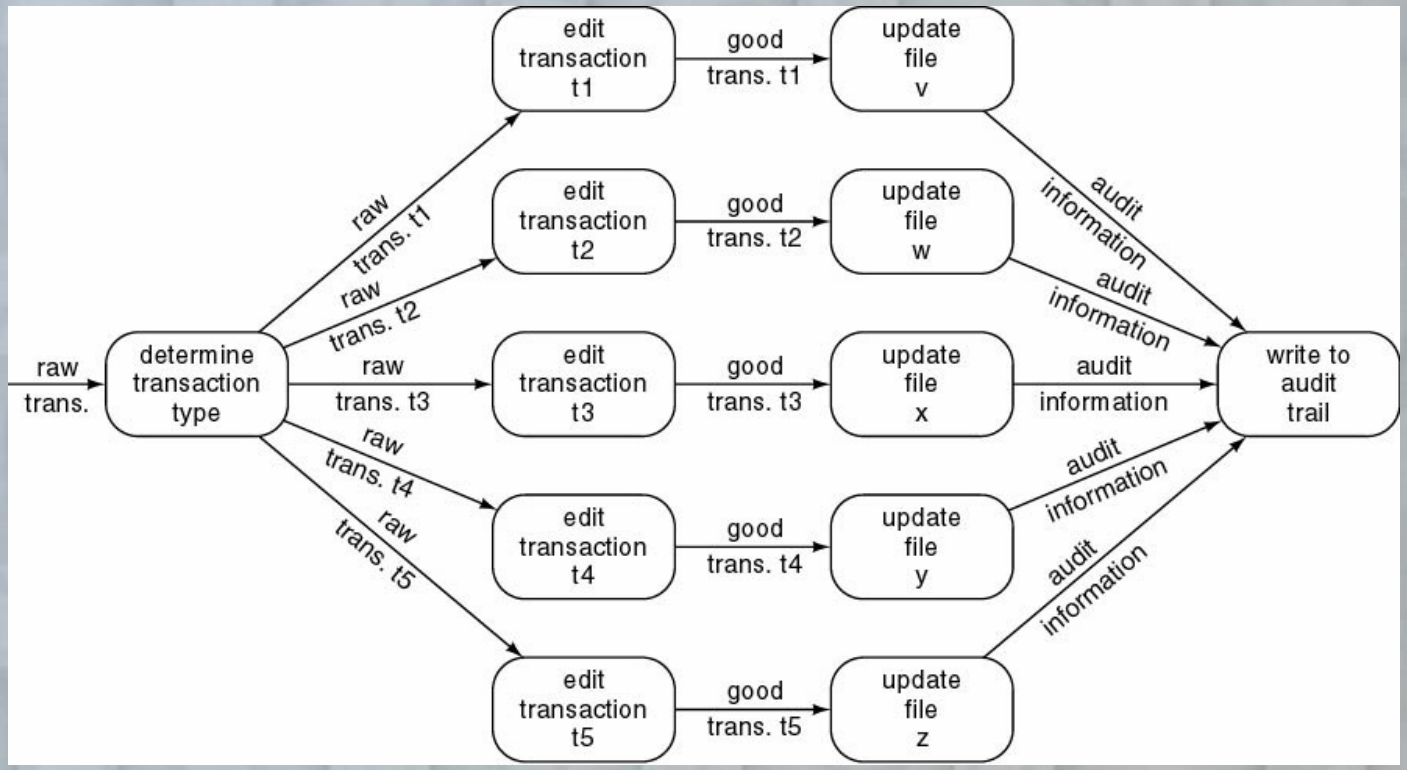
- Second refinement



- All eight modules have functional cohesion

Transaction Analysis

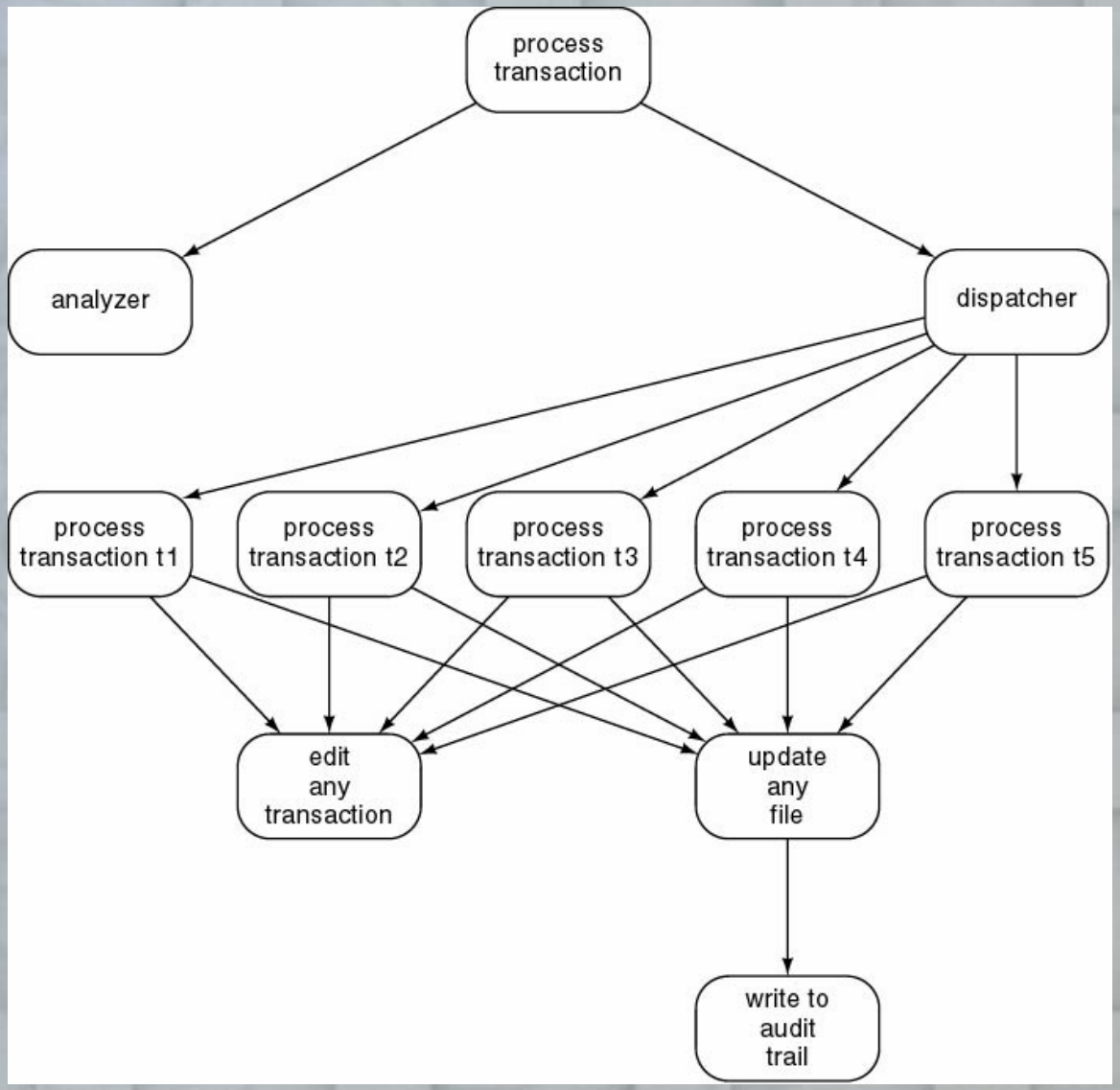
- DFA poor for transaction processing products
  - Example: ATM (Automatic Teller Machine)



- Poor design
  - Logical cohesion, control coupling

Corrected Design Using Transaction Analysis

- Software reuse



Data-Oriented Design

- Basic principle
  - The structure of a product must conform to the structure of its data
- Three very similar methods
  - Warnier
  - Orr
  - Jackson
- Data-oriented design
  - Has never been as popular as action-oriented design
  - With the rise of OOD, data-oriented design has largely fallen out of fashion

Object-Oriented Design Steps

- OOD consists of four steps:
  - 1. Construct interaction diagrams for each scenario
  - 2. Construct the detailed class diagram
  - 3. Design the product in terms of clients of objects
  - 4. Proceed to the detailed design

Elevator Problem: OOD

- Step 1. Construct interaction diagrams for each scenario
- Sequence diagrams
- Collaboration diagrams
  - Both show the same thing
  - Objects and messages passed between them
- But in a different way

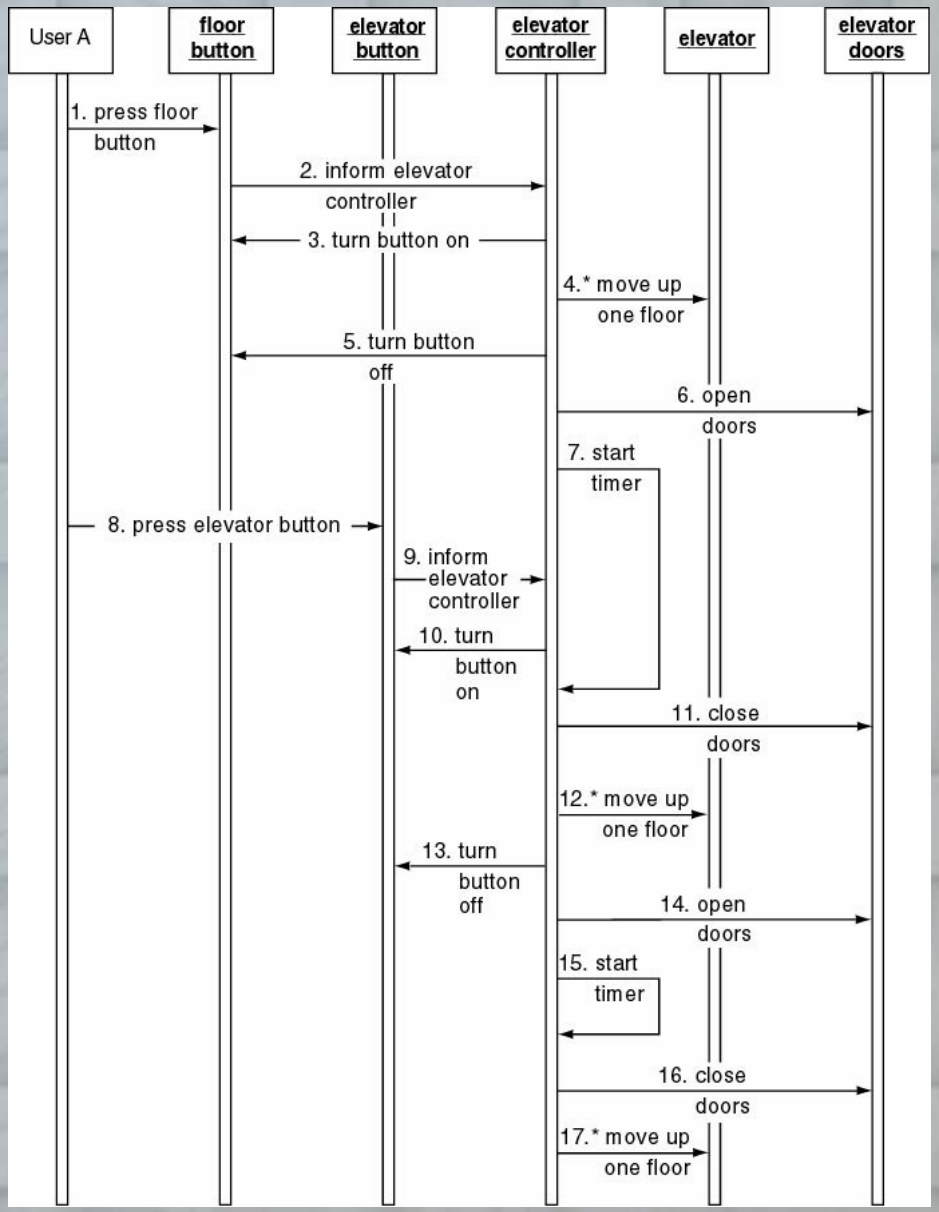
Elevator Problem: OOD cont'd

1. User A presses the Up floor button at floor 3 to request an elevator. User A wishes to go to floor 7.
2. The floor button informs the elevator controller that the floor button has been pushed.
3. The elevator controller sends a message to the Up floor button to turn itself on.
4. The elevator controller sends a series of messages to the elevator to move itself up to floor 3. The elevator contains User B, who has entered the elevator at floor 1 and pressed the elevator button for floor 9.
5. The elevator controller sends a message to the Up floor button to turn itself off.
6. The elevator controller sends a message to the elevator doors to open themselves.
7. The elevator control starts the timer.
   User A enters the elevator.
8. User A presses elevator button for floor 7.
9. The elevator button informs the elevator controller that the elevator button has been pushed.
10. The elevator controller sends a message to the elevator button for floor 7 to turn itself on.
11. The elevator controller sends a message to the elevator doors to close themselves after a timeout.
12. The elevator controller sends a series of messages to the elevator to move itself up to floor 7.
13. The elevator controller sends a message to the elevator button for floor 7 to turn itself off.
14. The elevator controller sends a message to the elevator doors to open themselves to allow User A to exit from the elevator.
15. The elevator controller starts the timer.
    User A exits from the elevator.
16. The elevator controller sends a message to the elevator doors to close themselves after a timeout.
17. The elevator controller sends a series of messages to the elevator to move itself up to floor 9 with User B.

- Normal scenario

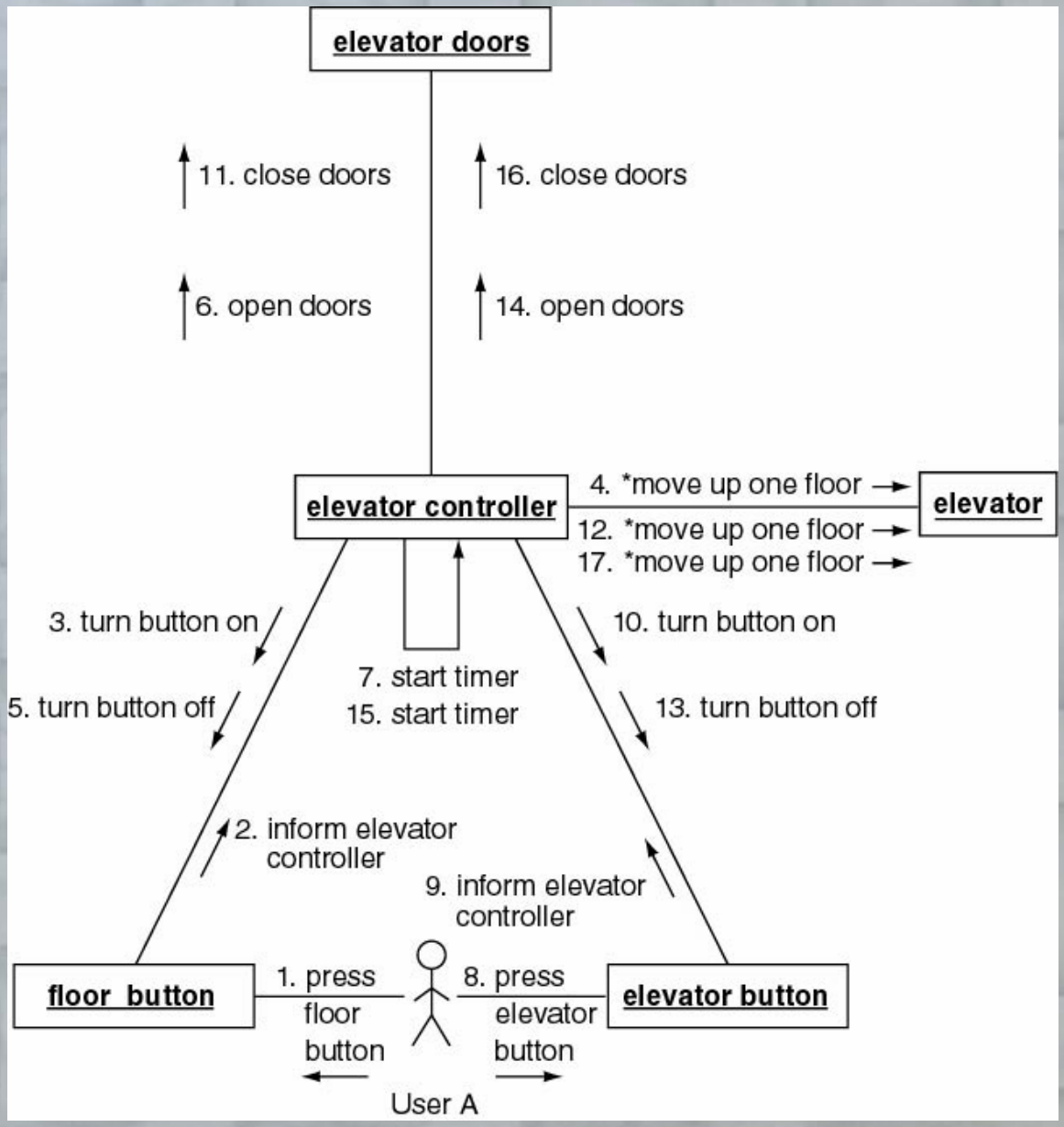
Elevator Problem: OOD cont'd

- Sequence diagram



Elevator Problem: OOD cont'd

- Collaboration diagram



Elevator Problem: OOD cont'd

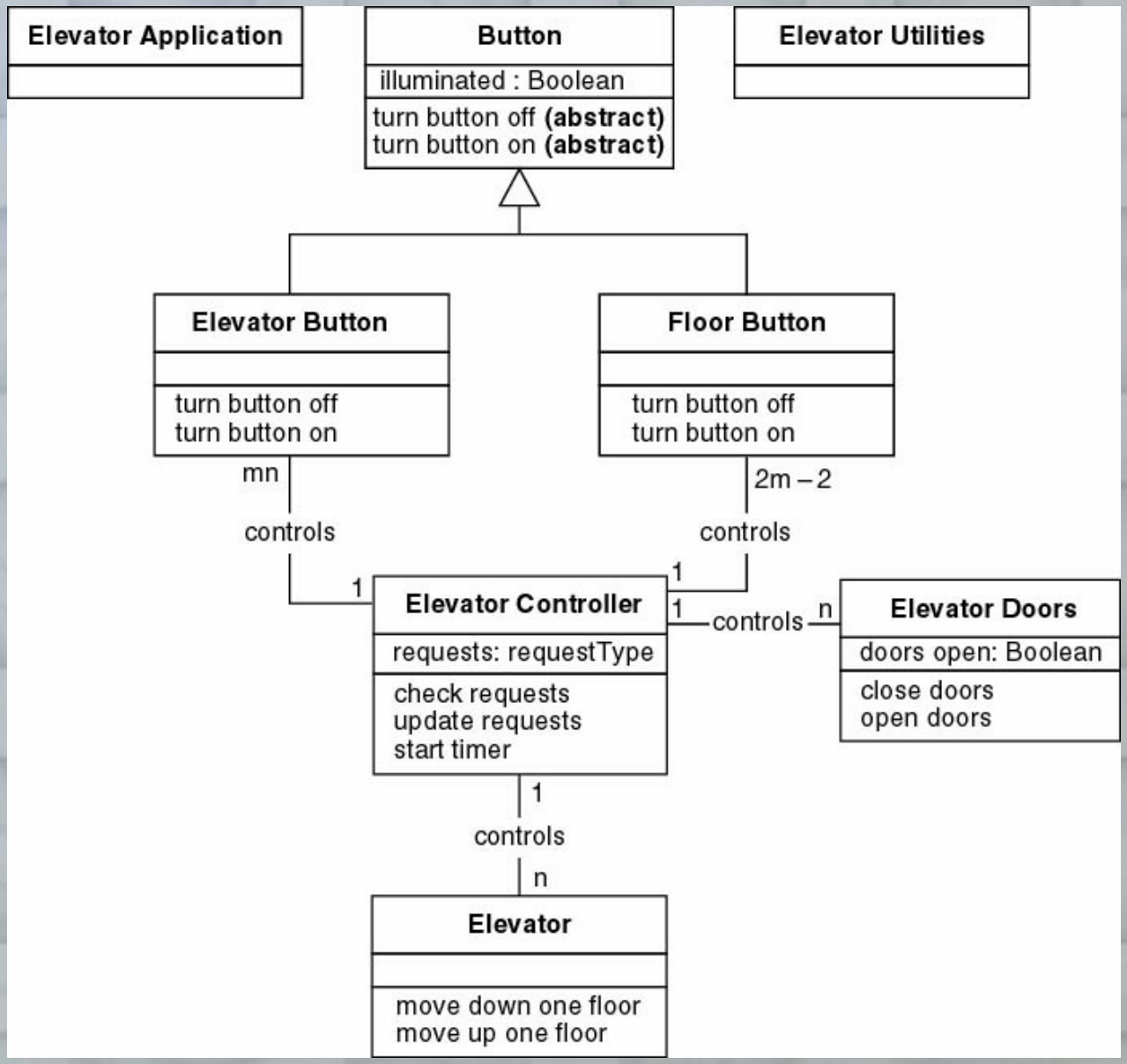
- Step 2. Construct Detailed Class Diagram
- Do we assign an action to a class or to a client of that class?
- Criteria
  - Information hiding
  - Reducing number of copies of action
  - Responsibility-driven design
- Examples

  close doors is assigned to **Elevator Doors**

  move one floor down is assigned to **Elevator**

Elevator Problem: OOD cont'd

- Detailed class diagram

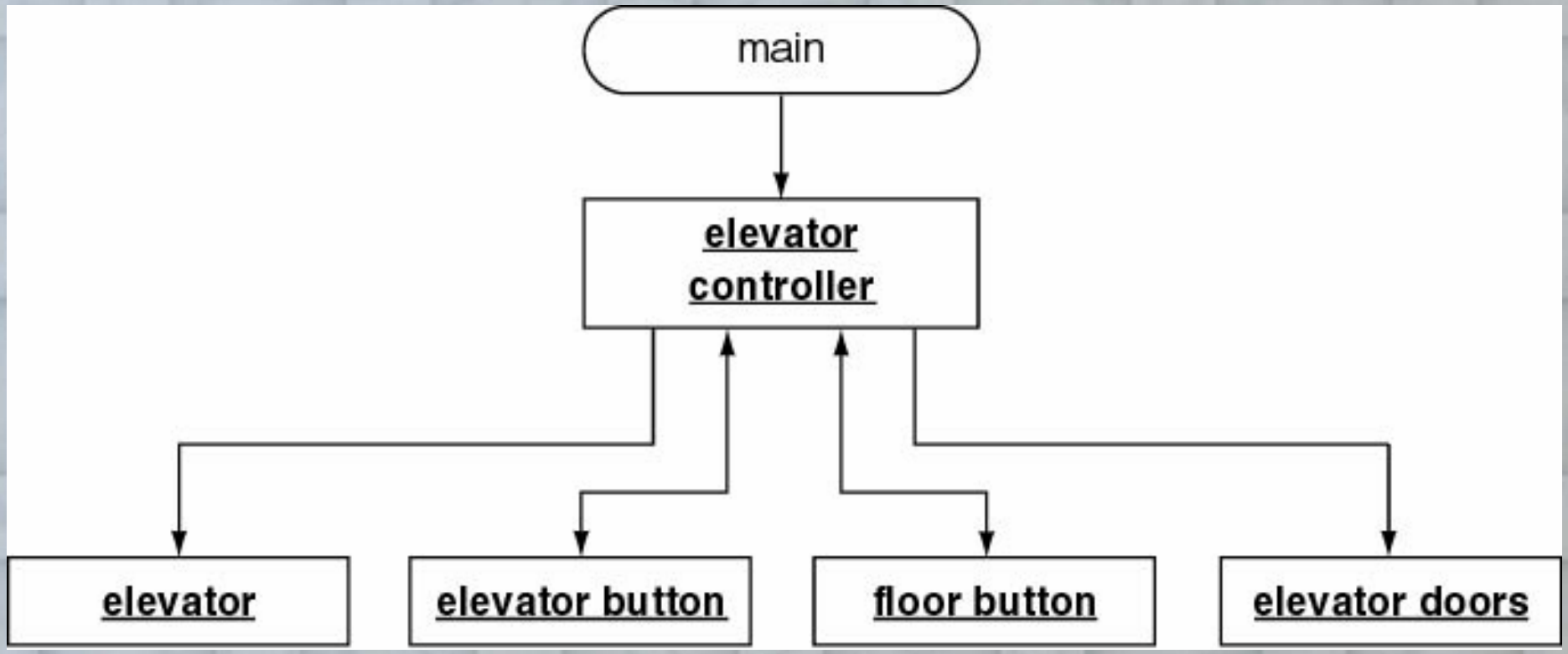


Elevator Problem: OOD cont'd

- Step 3. Design product in terms of clients of objects
- Draw an arrow from an object to a client of that object
- Objects that are not clients of any object have to be initiated, probably by the main method
  - Additional methods may be needed

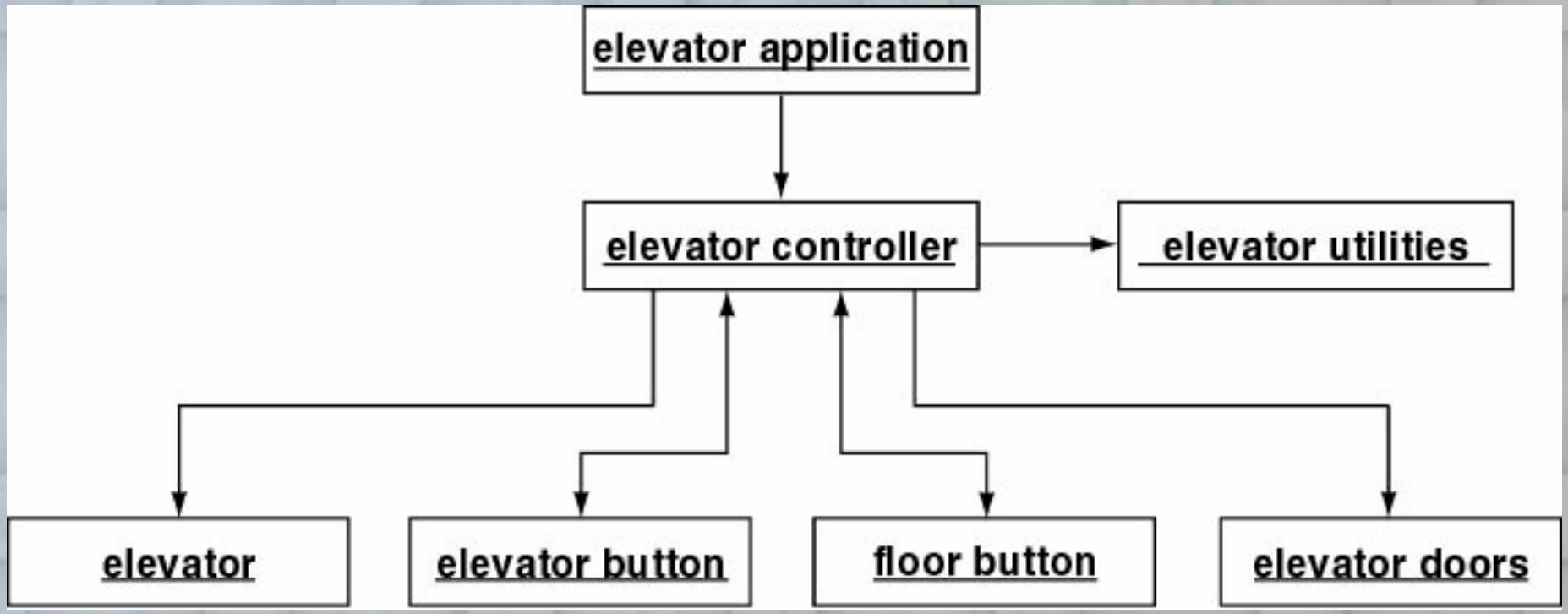
Elevator Problem: OOD cont'd

- C++ Client-object relations



Elevator Problem: OOD cont'd

- Java Client-object relations



Elevator Problem: OOD cont'd

- elevator controller needs method elevator control loop so that *main* (or elevator application) can call it

Elevator Problem: OOD cont'd

- Step 4. Perform detailed design
- Detailed design of method elevator controller loop

```
void elevator event loop (void)
{
  while (TRUE)
  {
    if (a button has been pressed)
      if (button is not on)
      {
        update requests;
        button::turn button on;
      }
    else if (elevator is moving up)
    {
      if (there is no request to stop at floor f)
        elevator::move one floor up;
      else
      {
        stop elevator by not sending a message to move;
        elevator doors::open doors;
        start timer;
        if (elevator button is on)
          elevator button::turn button off;
        update requests;
      }
    }
    else if (elevator is moving down)
      [similar to up case]
    else if (elevator is stopped and request is pending)
    {
      elevator doors::close doors;
      if (floor button is on)
        floor button::turn button off;
      determine direction of next request;
      elevator::move one floor up/down;
    }
    else if (elevator is at rest and not (request is pending))
      elevator doors::close doors;
    else
      there are no requests, elevator is stopped with elevator doors closed, so do nothing;
  }
}
```

Formal Techniques for Detailed Design

- Implementing a complete product and then proving it correct is hard
- However, use of formal techniques during detailed design can help
  - Correctness proving can be applied to module-sized pieces
  - The design should have fewer faults if it is developed in parallel with a correctness proof
  - If the same programmer does the detailed design and implementation
    - The programmer will have a positive attitude to the detailed design
    - Should lead to fewer faults

Design of Real-Time Systems

- Difficulties associated with real-time systems
  - Inputs come from real world
    - Software has no control over timing of inputs
  - Frequently implemented on distributed software
    - Communications implications
    - Timing issues
  - Problems of synchronization
    - Race conditions
    - Deadlock (deadly embrace)
- Major difficulty in design of real-time systems
  - Determining whether the timing constraints are met by the design

Real-Time Design Methods

- Usually, extensions of nonreal-time methods to real-time
- We have limited experience in use of any real-time methods
- The state-of-the-art is not where we would like it to be

Testing during the Design Phase

- Design reviews
  - Design must correctly reflect specifications
  - Design itself must be correct
  - Transaction-driven inspections

CASE Tools for the Design Phase

- UpperCASE tools
  - Built around data dictionary
  - Consistency checker
  - Screen, report generators
  - Modern tools represent OOD using UML
  - Examples: Software through Pictures, Rational Rose

Example

Air Gourmet

Air Gourmet Case Study: Object-Oriented Design

- OOD consists of four steps:
  - 1. Construct interaction diagrams for each scenario
  - 2. Construct the detailed class diagram
  - 3. Design the product in terms of clients of objects
  - 4. Proceed to the detailed design

Step 1. Interaction Diagrams

- Extended scenario for making a reservation

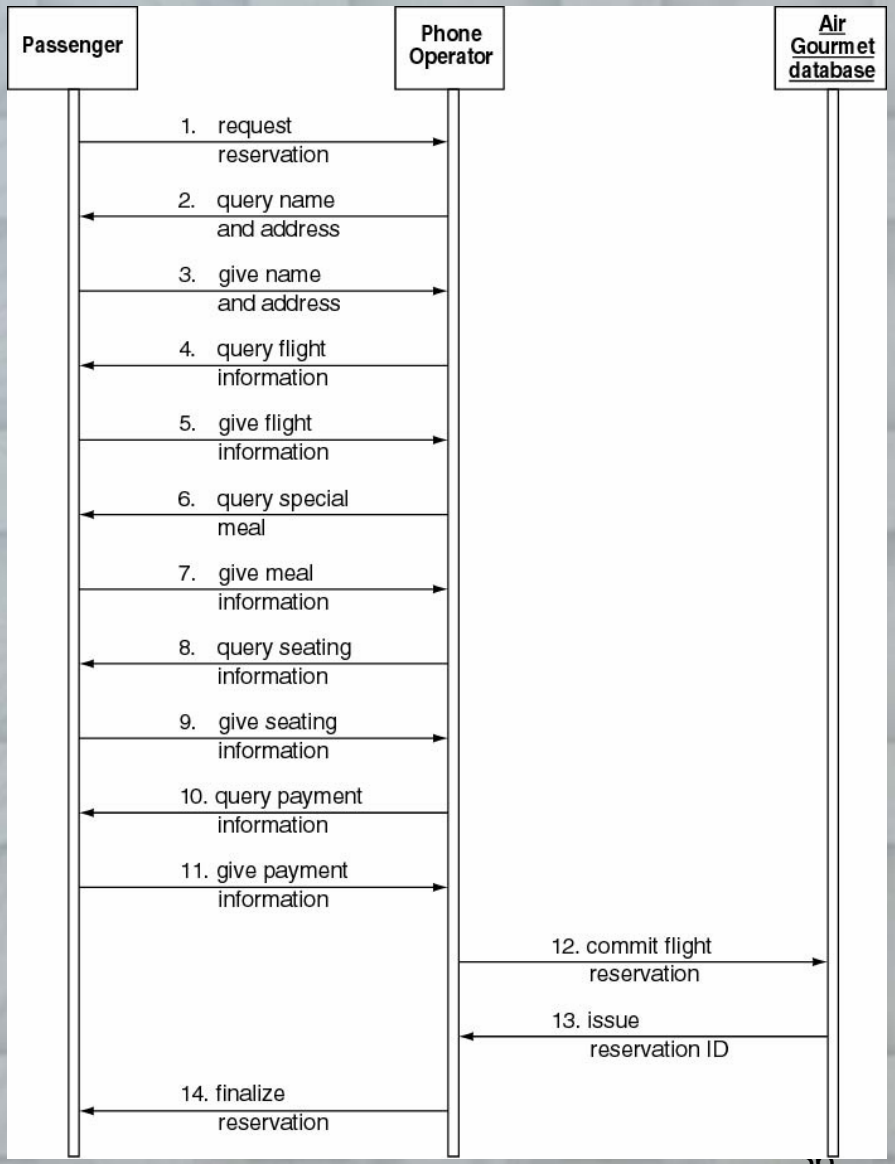
1. The Passenger contacts the Air Gourmet call center Phone Operator and expresses the desire to reserve a flight.
2. The Phone Operator requests the name and address of the Passenger.
3. The Passenger provides the requested information.
4. The Phone Operator asks the Passenger the date of the flight and the flight number.
5. The Passenger provides the requested information.
6. The Phone Operator asks the Passenger if a special meal is desired.
7. The Passenger states what kind of special meal, if any, is desired.
8. The Phone Operator asks the Passenger for his or her seating preference.
9. The Passenger provides his or her seating preference.
10. The Phone Operator requests payment information.
11. The Passenger provides a valid credit card number.
12. The Phone Operator commits the reservation to the Air Gourmet database.
13. The Air Gourmet database issues the reservation ID to the Phone Operator.
14. The Phone Operator gives the reservation ID to the Passenger and informs the Passenger that the ticket will be mailed soon.

Possible Alternatives

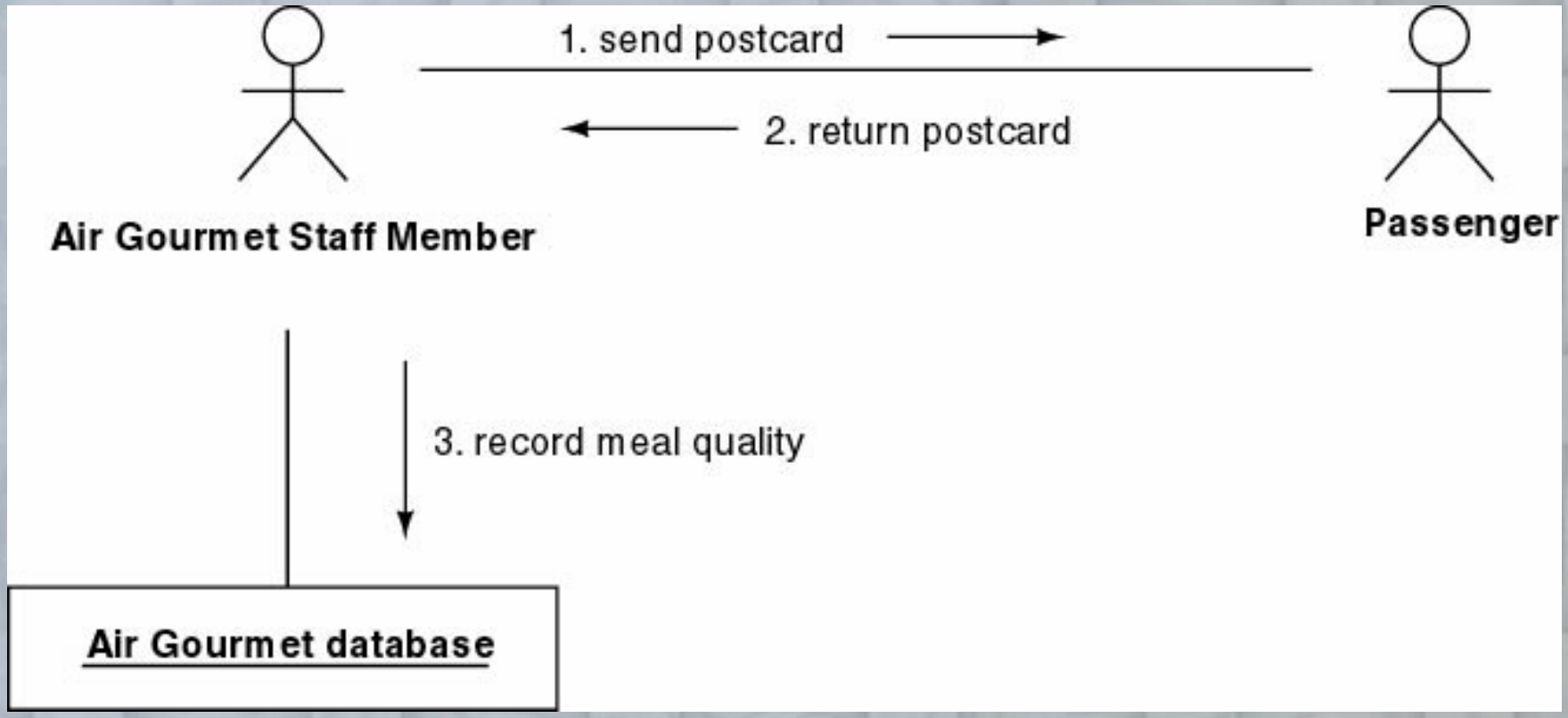
A. The flight number requested does not exist on the particular day that the Passenger wants to fly.
B. The flight that the Passenger requested already is full.
C. The Passenger has an unusual meal request that Air Gourmet cannot fulfill.
D. There are problems with the credit card number provided by the Passenger.

Step 1. Interaction Diagrams cont'd

- Sequence diagram for making a reservation

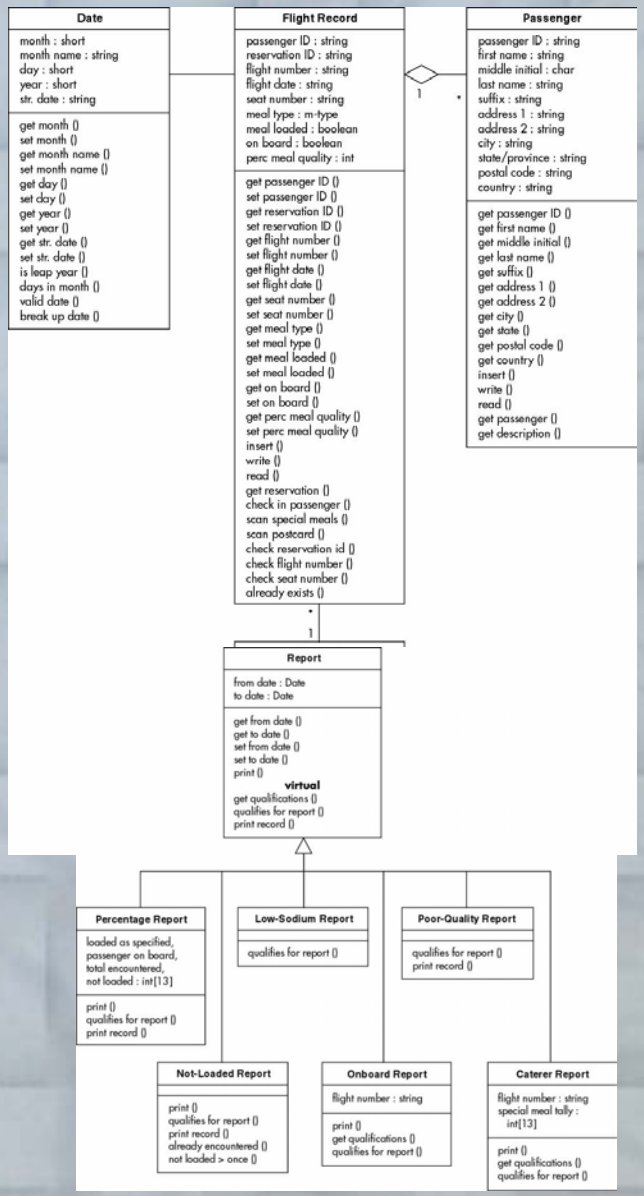


Step 1. Interaction Diagrams (contd)

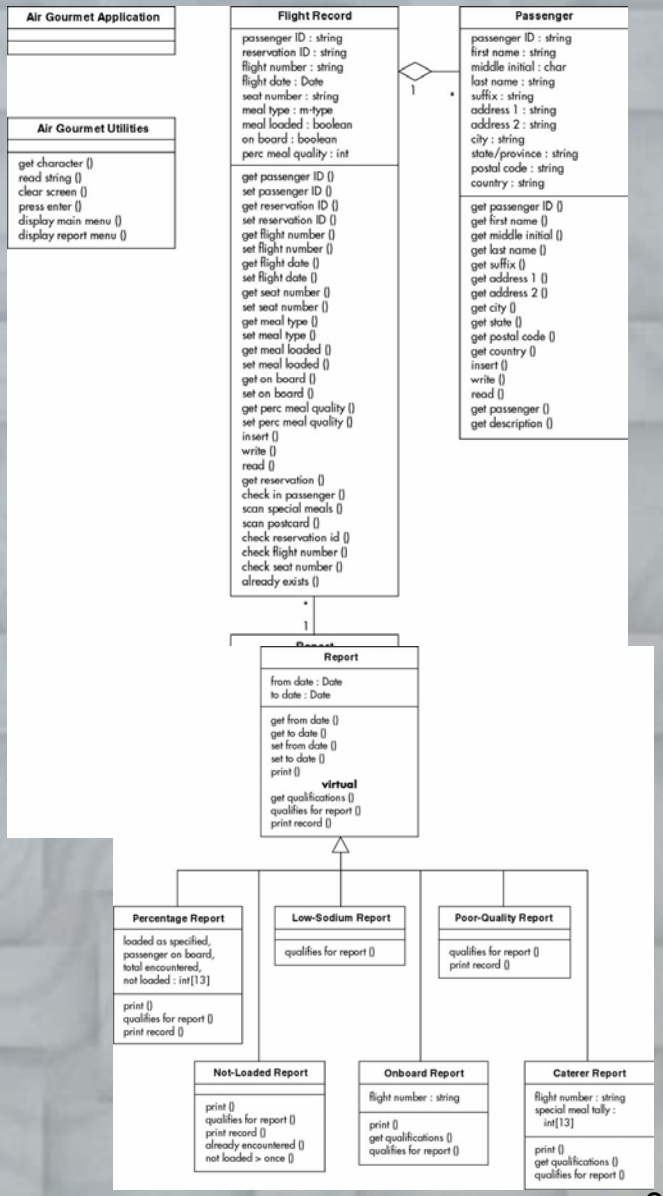


- Collaboration diagram for sending and returning a postcard

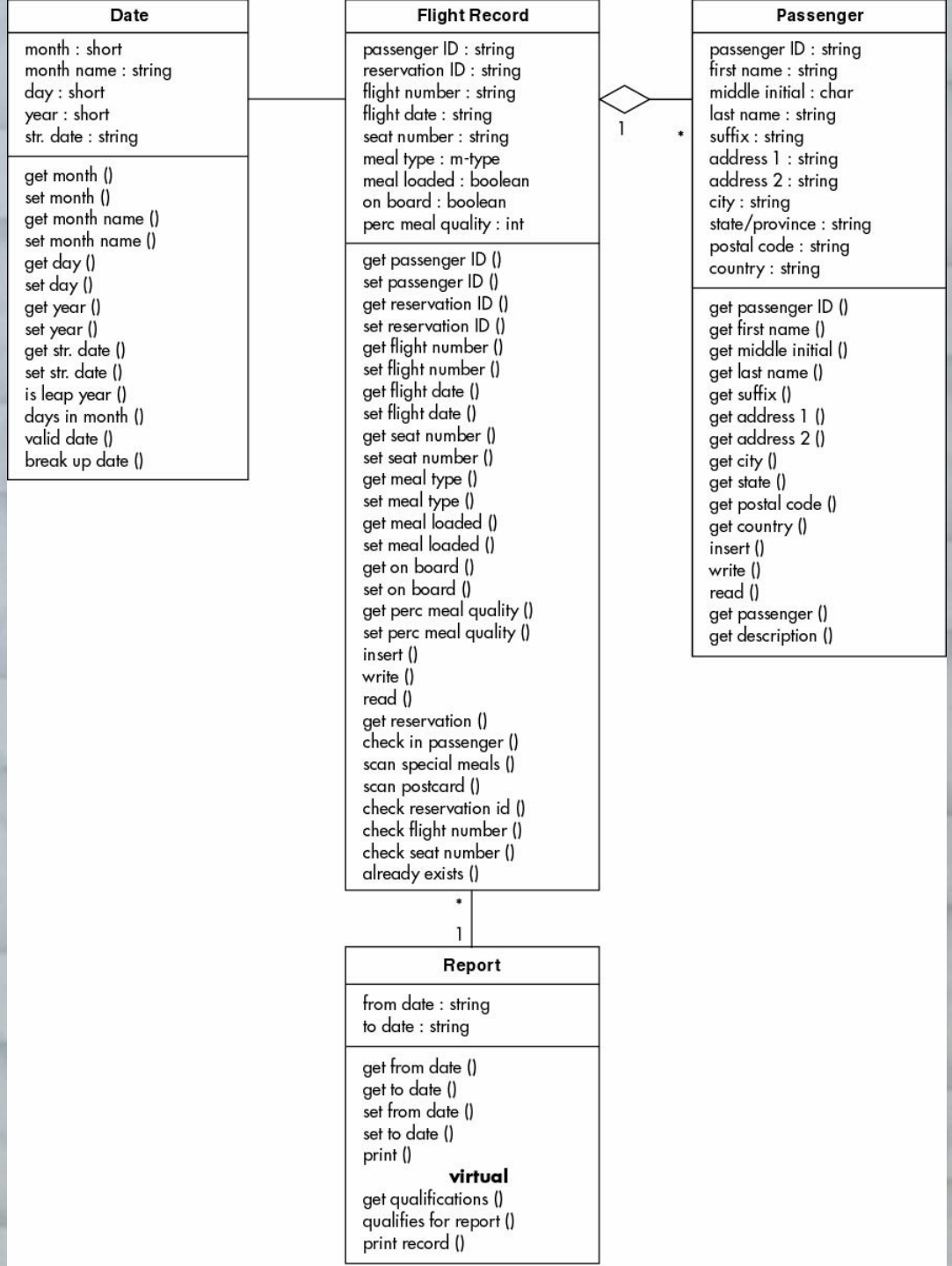
Step 2. Detailed Class Diagram

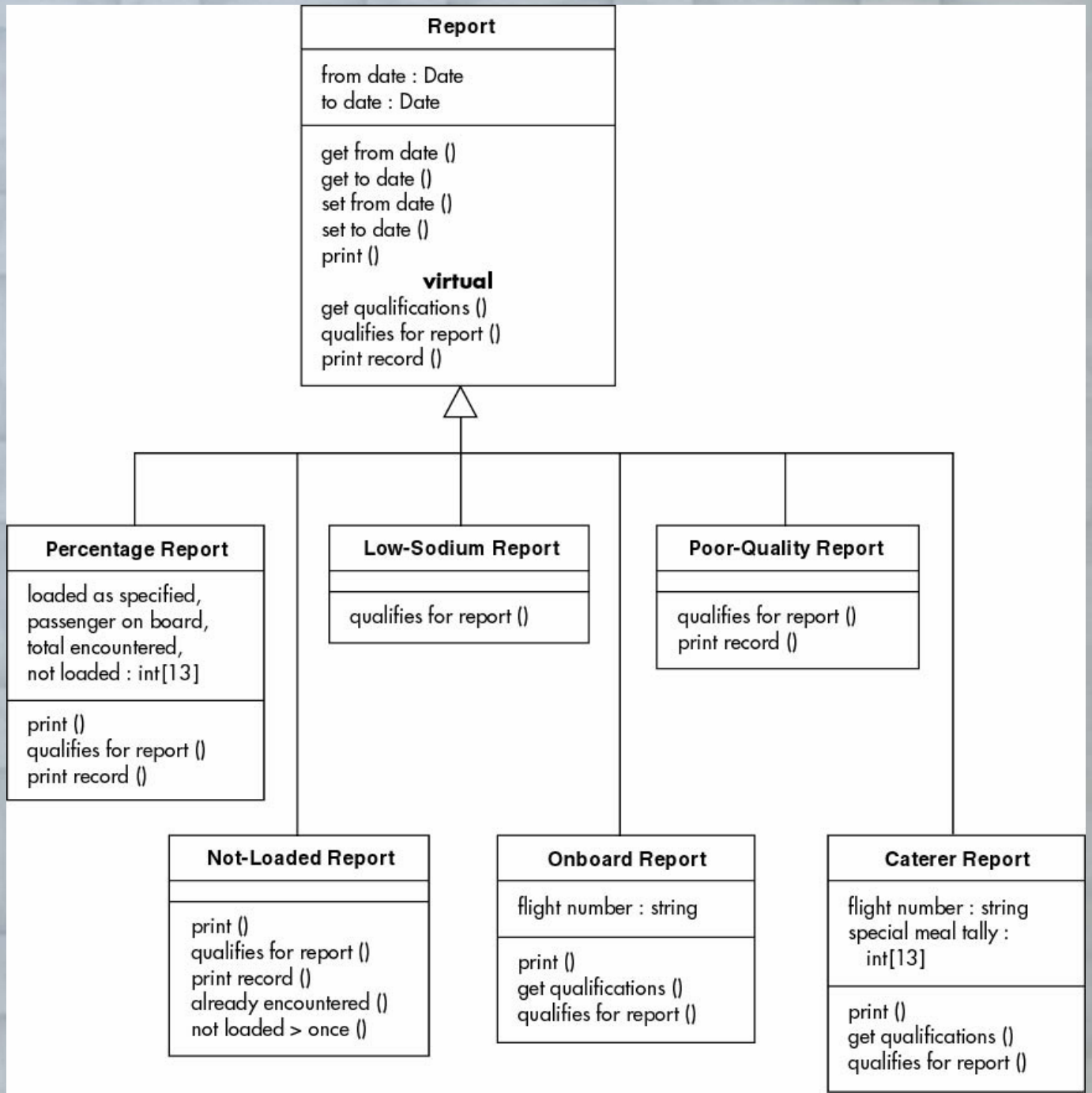


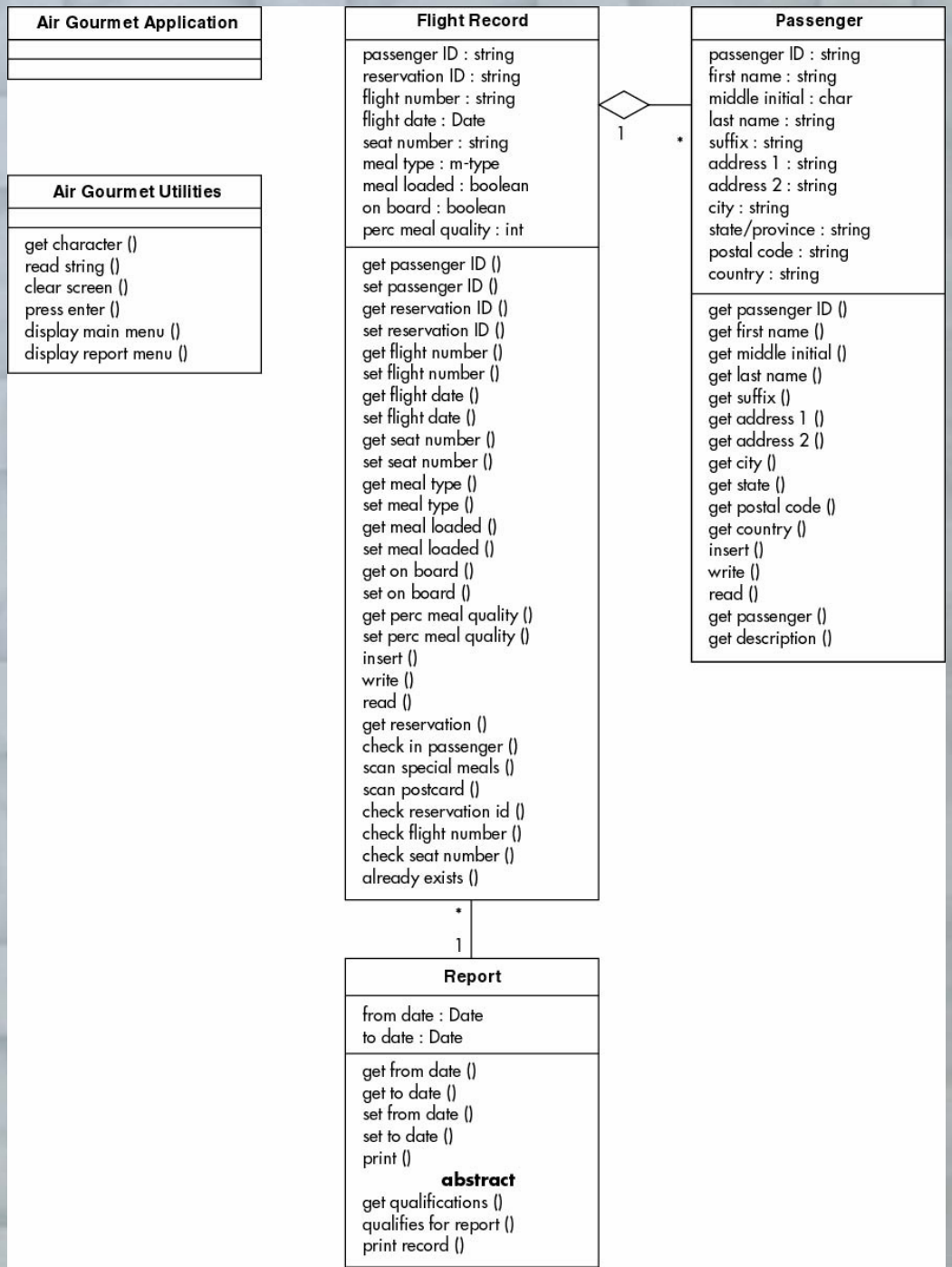
C++ version

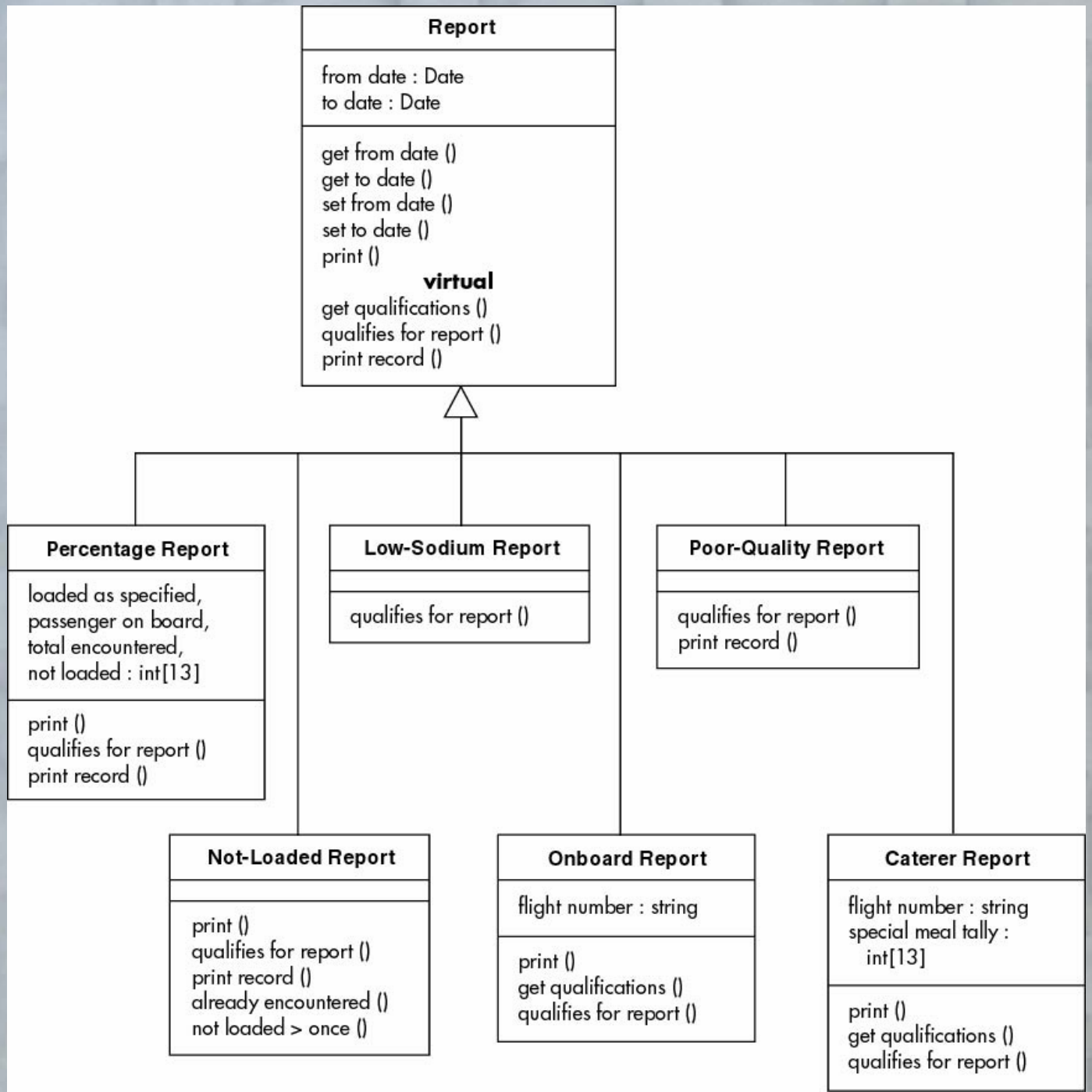


Java version

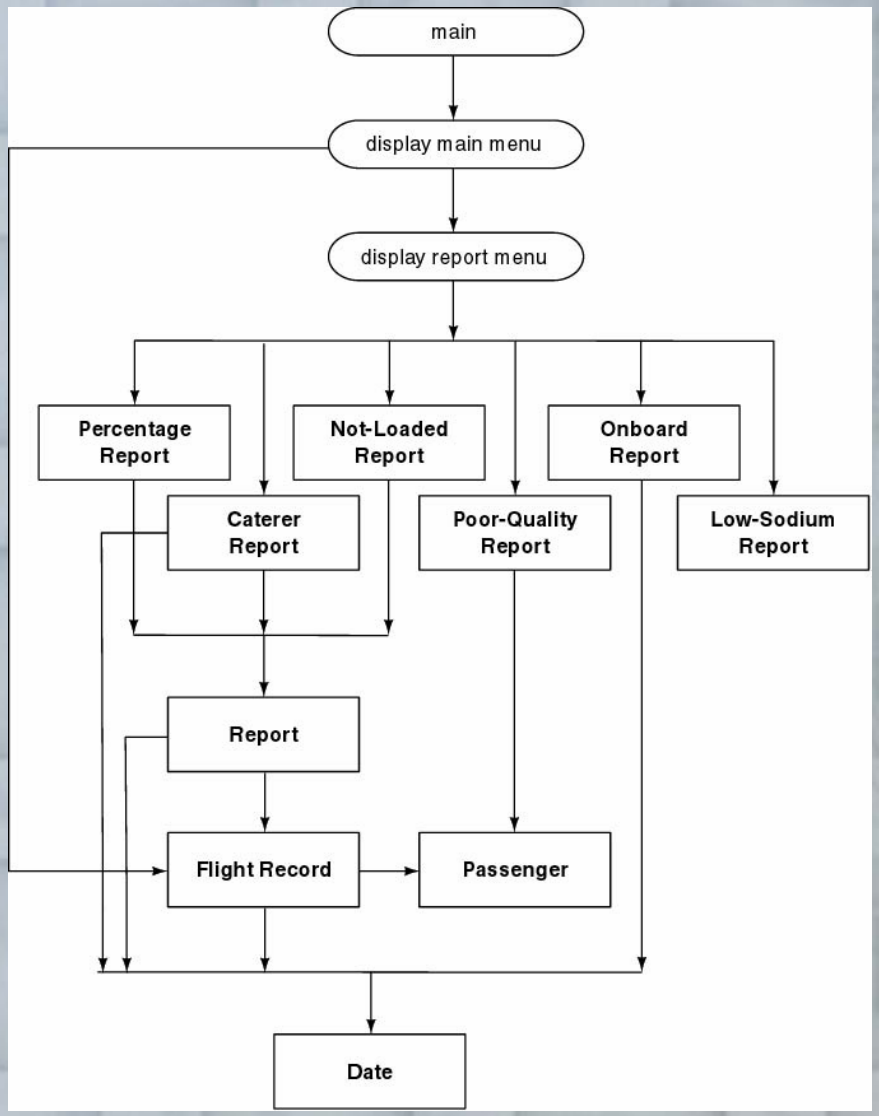

Date
month : short
month name : string
day : short
year : short
str. date : string
get month ()
set month ()
get month name ()
set month name ()
get day ()
set day ()
get year ()
set year ()
get str. date ()
set str. date ()
is leap year ()
days in month ()
valid date ()
break up date ()
Flight Record
passenger ID : string
reservation ID : string
flight number : string
flight date : string
seat number : string
meal type : m-type
meal loaded : boolean
on board : boolean
perc meal quality : int
get passenger ID ()
set passenger ID ()
get reservation ID ()
set reservation ID ()
get flight number ()
set flight number ()
get flight date ()
set flight date ()
get seat number ()
set seat number ()
get meal type ()
set meal type ()
get meal loaded ()
set meal loaded ()
get on board ()
set on board ()
get perc meal quality ()
set perc meal quality ()
insert ()
write ()
read ()
get reservation ()
check in passenger ()
scan special meals ()
scan postcard ()
check reservation id ()
check flight number ()
check seat number ()
already exists ()
1
*
Passenger
passenger ID : string
first name : string
middle initial : char
last name : string
suffix : string
address 1 : string
address 2 : string
city : string
state/province : string
postal code : string
country : string
get passenger ID ()
get first name ()
get middle initial ()
get last name ()
get suffix ()
get address 1 ()
get address 2 ()
get city ()
get state ()
get postal code ()
get country ()
insert ()
write ()
read ()
get passenger ()
get description ()
*
1
Report
from date : string
to date : string
get from date ()
get to date ()
set from date ()
set to date ()
print ()
virtual
get qualifications ()
qualifies for report ()
print record ()


Report
from date : Date
to date : Date
get from date ()
get to date ()
set from date ()
set to date ()
print ()
virtual
get qualifications ()
qualifies for report ()
print record ()
Percentage Report
loaded as specified,
passenger on board,
total encountered,
not loaded : int[13]
print ()
qualifies for report ()
print record ()
Low-Sodium Report
qualifies for report ()
Poor-Quality Report
qualifies for report ()
print record ()
Not-Loaded Report
print ()
qualifies for report ()
print record ()
already encountered ()
not loaded > once ()
Onboard Report
flight number : string
print ()
get qualifications ()
qualifies for report ()
Caterer Report
flight number : string
special meal tally :
int[13]
print ()
get qualifications ()
qualifies for report ()

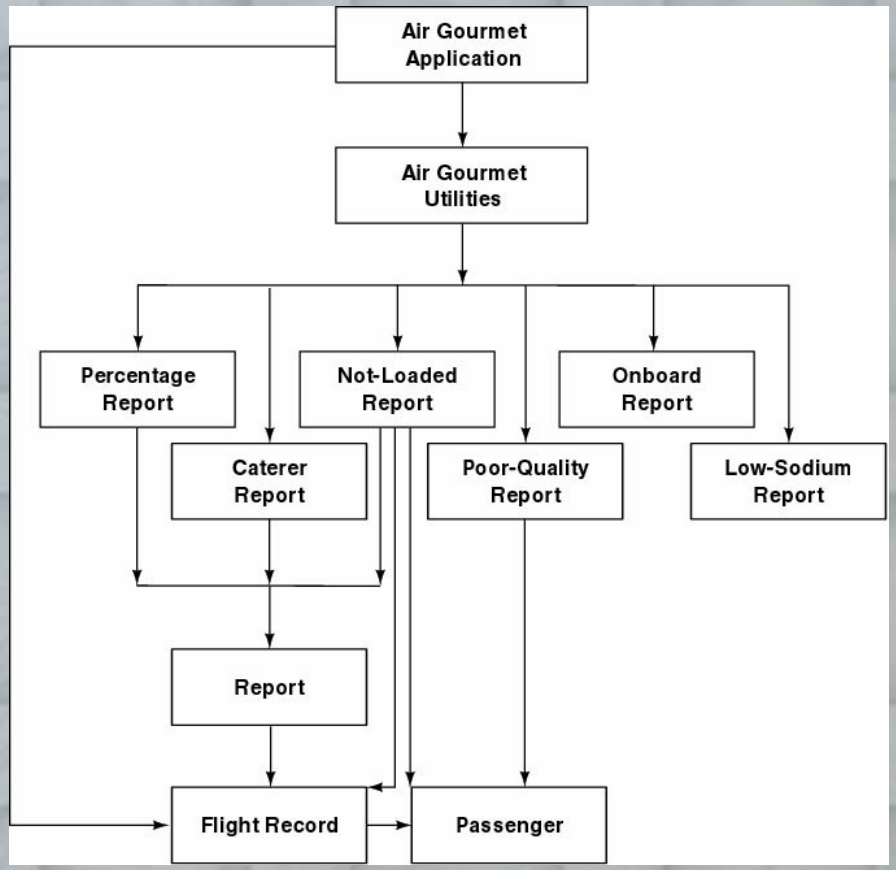

Air Gourmet Application
Air Gourmet Utilities
get character ()
read string ()
clear screen ()
press enter ()
display main menu ()
display report menu ()
Flight Record
passenger ID : string
reservation ID : string
flight number : string
flight date : Date
seat number : string
meal type : m-type
meal loaded : boolean
on board : boolean
perc meal quality : int
get passenger ID ()
set passenger ID ()
get reservation ID ()
set reservation ID ()
get flight number ()
set flight number ()
get flight date ()
set flight date ()
get seat number ()
set seat number ()
get meal type ()
set meal type ()
get meal loaded ()
set meal loaded ()
get on board ()
set on board ()
get perc meal quality ()
set perc meal quality ()
insert ()
write ()
read ()
get reservation ()
check in passenger ()
scan special meals ()
scan postcard ()
check reservation id ()
check flight number ()
check seat number ()
already exists ()
1
*
Passenger
passenger ID : string
first name : string
middle initial : char
last name : string
suffix : string
address 1 : string
address 2 : string
city : string
state/province : string
postal code : string
country : string
get passenger ID ()
get first name ()
get middle initial ()
get last name ()
get suffix ()
get address 1 ()
get address 2 ()
get city ()
get state ()
get postal code ()
get country ()
insert ()
write ()
read ()
get passenger ()
get description ()
*
1
Report
from date : Date
to date : Date
get from date ()
get to date ()
set from date ()
set to date ()
print ()
abstract
get qualifications ()
qualifies for report ()
print record ()


Report
from date : Date
to date : Date
get from date ()
get to date ()
set from date ()
set to date ()
print ()
virtual
get qualifications ()
qualifies for report ()
print record ()
Percentage Report
loaded as specified,
passenger on board,
total encountered,
not loaded : int[13]
print ()
qualifies for report ()
print record ()
Low-Sodium Report
qualifies for report ()
Poor-Quality Report
qualifies for report ()
print record ()
Not-Loaded Report
print ()
qualifies for report ()
print record ()
already encountered ()
not loaded > once ()
Onboard Report
flight number : string
print ()
get qualifications ()
qualifies for report ()
Caterer Report
flight number : string
special meal tally :
int[13]
print ()
get qualifications ()
qualifies for report ()

Step 3. Client–Object Relations



C++ version



Java version

Step 4. Detailed Design

- The detailed design can be found in
  - See the implementation in C++
  - See the implementation in Java)