Design Case Study -1

Overview

- Extracting the entity classes
- The elevator problem
- Functional modeling: The elevator problem case study
- Entity class modeling: The elevator problem case study
- Dynamic modeling: The elevator problem case study

Overview (contd)

- Dynamic modeling: The elevator problem case study
- The test workflow: The elevator problem case study
- Extracting the boundary and control classes

Object-Oriented Analysis

- OOA is a semiformal analysis technique for the object-oriented paradigm
  - There are over 60 equivalent techniques
  - Today, the Unified Process is the only viable alternative
- During this workflow
  - The classes are extracted
- Remark
  - The Unified Process assumes knowledge of class extraction

11.4 The Analysis Workflow

- The analysis workflow has two aims
  - Obtain a deeper understanding of the requirements
  - Describe them in a way that will result in a maintainable design and implementation

The Analysis Workflow (contd)

- There are three types of classes:
- Entity classes
- Boundary classes
- Control classes

The Analysis Workflow (contd)

- Entity class
  - Models long-lived information

- Examples:
  - **Account Class**
  - **Investment Class**

The Analysis Workflow (contd)

- Boundary class
  - Models the interaction between the product and the environment
  - A boundary class is generally associated with input or output

- Examples:
  - **Investments Report Class**
  - **Mortgages Report Class**

The Analysis Workflow (contd)

- Control class
  - Models complex computations and algorithms

- Example:
  - **Estimate Funds for Week Class**

UML Notation for These Three Class Types

- Stereotypes (extensions of UML)

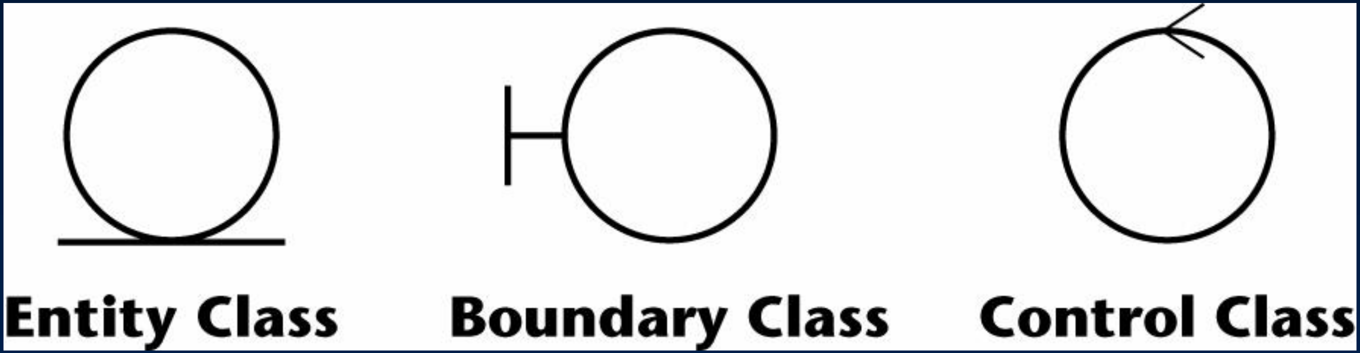


Figure 11.1

11.5 Extracting the Entity Classes

- Perform the following three steps incrementally and iteratively
  - Functional modeling
    - » Present scenarios of all the use cases (a *scenario* is an instance of a use case)
  - Class modeling
    - » Determine the entity classes and their attributes
    - » Determine the interrelationships and interactions between the entity classes
    - » Present this information in the form of a *class diagram*
  - Dynamic modeling
    - » Determine the operations performed by or to each entity class
    - » Present this information in the form of a *statechart*

11.6 The Elevator Problem

A product is to be installed to control n elevators in a building with m floors. The problem concerns the logic required to move elevators between floors according to the following constraints:

1. Each elevator has a set of m buttons, one for each floor. These illuminate when pressed and cause the elevator to visit the corresponding floor. The illumination is canceled when the corresponding floor is visited by the elevator

2. Each floor, except the first and the top floor, has two buttons, one to request an up-elevator, one to request a down-elevator. These buttons illuminate when pressed. The illumination is canceled when an elevator visits the floor, then moves in the desired direction

3. If an elevator has no requests, it remains at its current floor with its doors closed

11.7 Functional Modeling: The Elevator Problem Case Study

- A use case describes the interaction between
  - The product, and
  - The actors (external users)

Use Cases

- For the elevator problem, there are only two possible use cases
  - `Press an Elevator Button`, and
  - `Press a Floor Button`

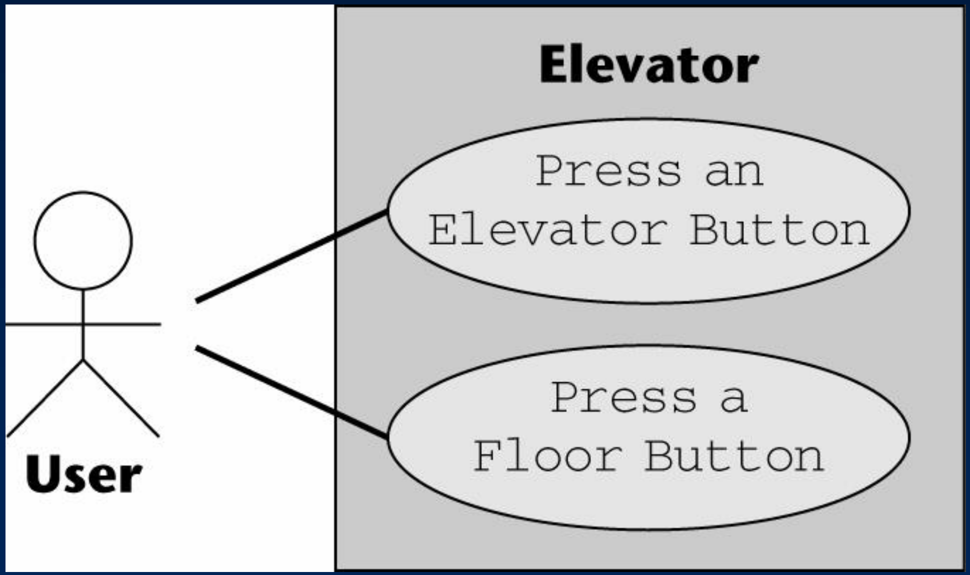


Figure 11.2

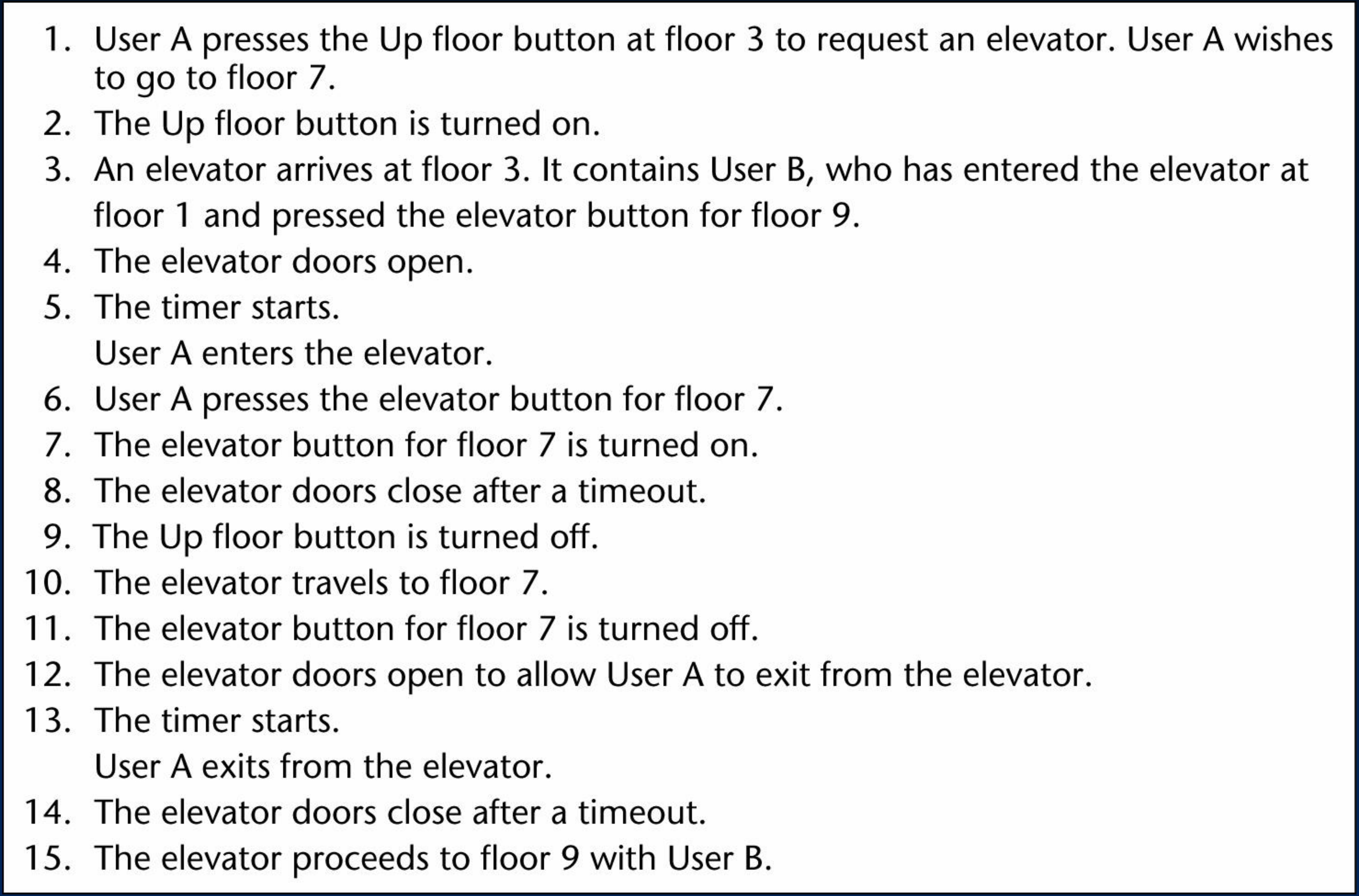
Scenarios

- A use case provides a generic description of the overall functionality

- A scenario is an instance of a use case

- Sufficient scenarios need to be studied to get a comprehensive insight into the target product being modeled

Normal Scenario: Elevator Problem



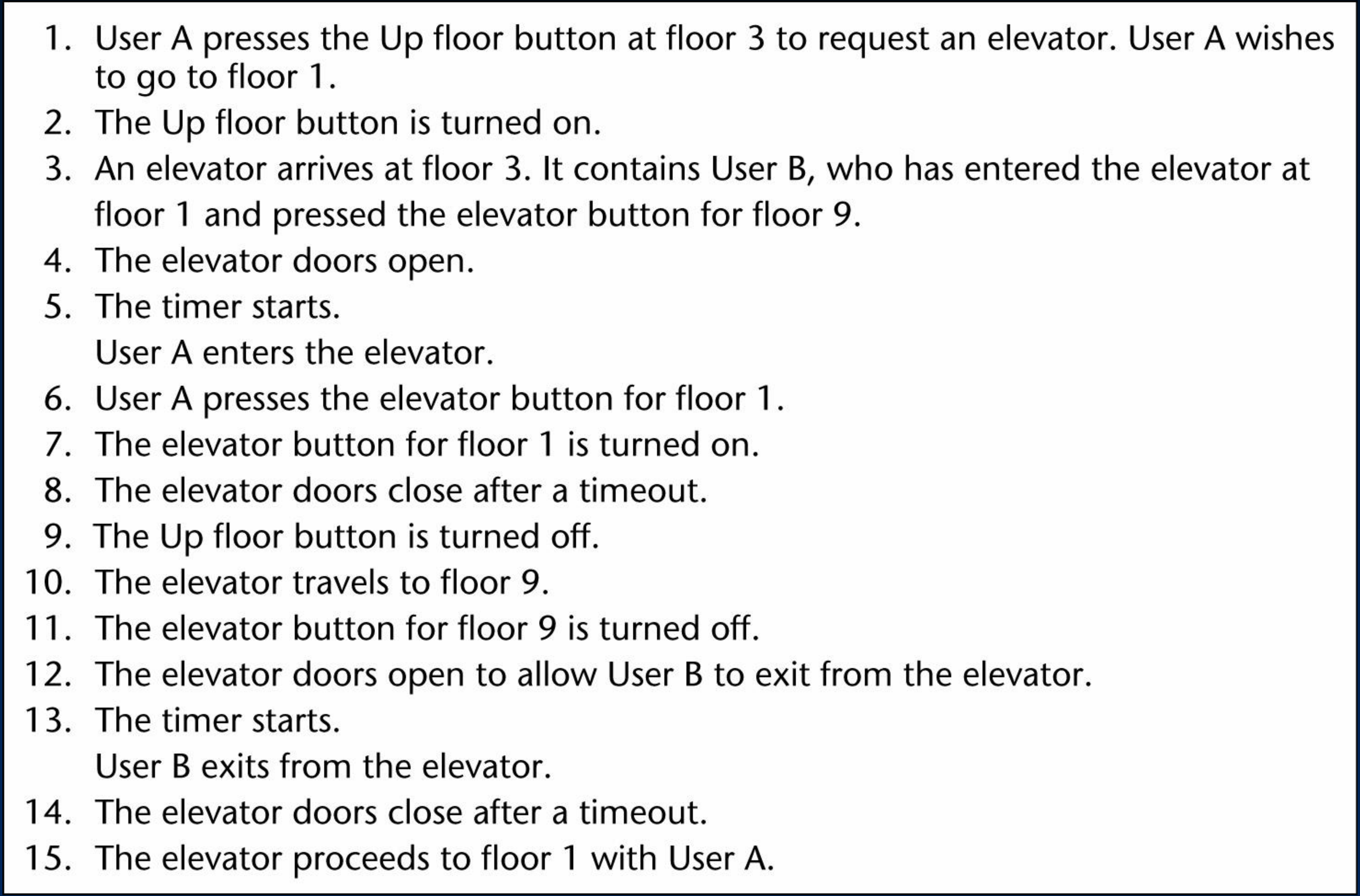
1. User A presses the Up floor button at floor 3 to request an elevator. User A wishes to go to floor 7.
2. The Up floor button is turned on.
3. An elevator arrives at floor 3. It contains User B, who has entered the elevator at floor 1 and pressed the elevator button for floor 9.
4. The elevator doors open.
5. The timer starts.

   User A enters the elevator.
6. User A presses the elevator button for floor 7.
7. The elevator button for floor 7 is turned on.
8. The elevator doors close after a timeout.
9. The Up floor button is turned off.
10. The elevator travels to floor 7.
11. The elevator button for floor 7 is turned off.
12. The elevator doors open to allow User A to exit from the elevator.
13. The timer starts.

    User A exits from the elevator.
14. The elevator doors close after a timeout.
15. The elevator proceeds to floor 9 with User B.

Figure 11.3

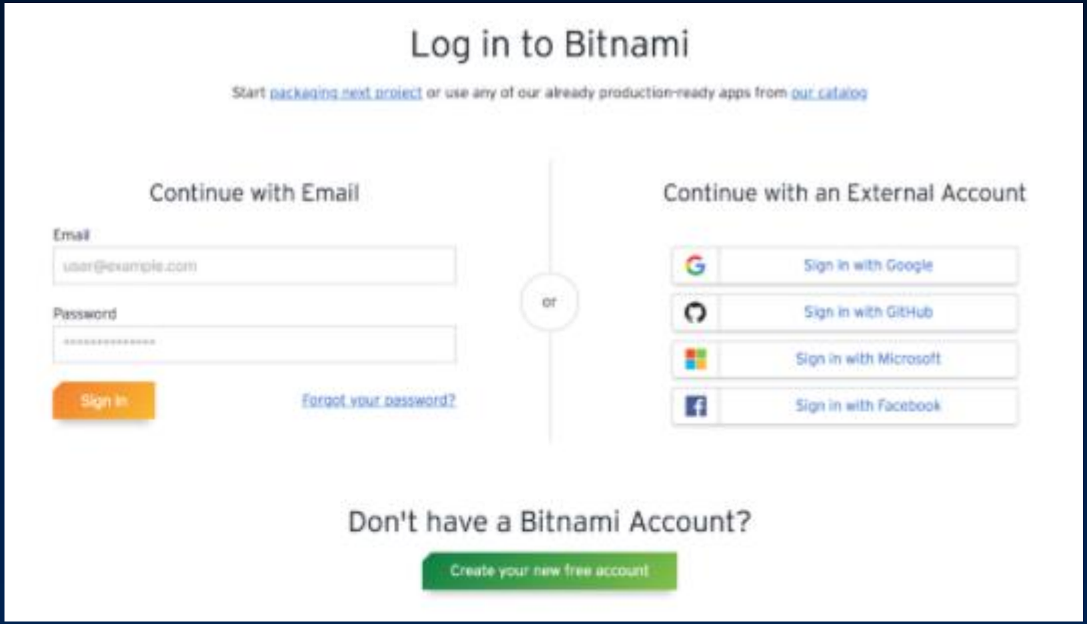
Exception Scenario: Elevator Problem



1. User A presses the Up floor button at floor 3 to request an elevator. User A wishes to go to floor 1.
2. The Up floor button is turned on.
3. An elevator arrives at floor 3. It contains User B, who has entered the elevator at floor 1 and pressed the elevator button for floor 9.
4. The elevator doors open.
5. The timer starts.
   User A enters the elevator.
6. User A presses the elevator button for floor 1.
7. The elevator button for floor 1 is turned on.
8. The elevator doors close after a timeout.
9. The Up floor button is turned off.
10. The elevator travels to floor 9.
11. The elevator button for floor 9 is turned off.
12. The elevator doors open to allow User B to exit from the elevator.
13. The timer starts.
    User B exits from the elevator.
14. The elevator doors close after a timeout.
15. The elevator proceeds to floor 1 with User A.

Figure 11.4

Agile: Prototype



Log in to Bitnami

Start packaging next project or use any of our already production-ready apps from our catalog

Continue with Email

Email: user@example.com

Password: ***************

Continue with an External Account

Don't have a Bitnami Account?

11.8 Entity Class Modeling : The Elevator Problem Case Study

- Extract classes and their attributes
  - Represent them using a UML diagram

- One alternative: Deduce the classes from use cases and their scenarios
  - Possible danger: Often there are many scenarios, and hence
  - Too many candidate classes

- Other alternatives:
  - CRC cards (if you have domain knowledge)
  - Noun extraction

11.8.1 Noun Extraction

- A two-stage process

- Stage 1. Concise problem definition
  - Describe the software product in single paragraph
  - Buttons in elevators and on the floors control the movement of n elevators in a building with m floors. Buttons illuminate when pressed to request the elevator to stop at a specific floor; the illumination is canceled when the request has been satisfied. When an elevator has no requests, it remains at its current floor with its doors closed

Noun Extraction (contd)

- Stage 2. Identify the nouns
  - Identify the nouns in the informal strategy
  - Buttons in elevators and on the floors control the movement of n elevators in a building with m floors. Buttons illuminate when pressed to request the elevator to stop at a specific floor; the illumination is canceled when the request has been satisfied. When an elevator has no requests, it remains at its current floor with its doors closed

- Use the nouns as candidate classes

Noun Extraction (contd)

- Nouns
  - button, elevator, floor, movement, building, illumination, request, door
  - floor, building, door are outside the problem boundary — exclude
  - movement, illumination, request are abstract nouns — exclude (they may become attributes)
- Candidate classes:
  - **Elevator Class** and **Button Class**
- Subclasses:
  - **Elevator Button Class** and **Floor Button Class**

First Iteration of Class Diagram

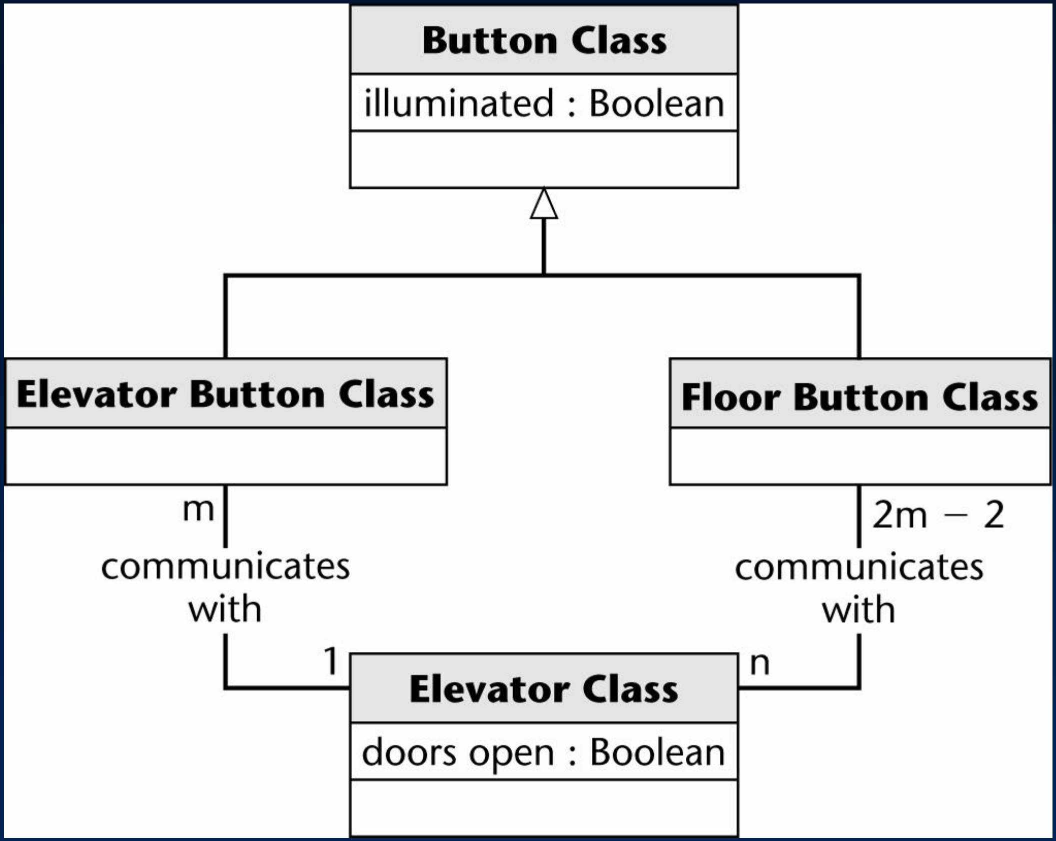


Figure 11.5

- Problem
  - Buttons do not communicate directly with elevators
  - We need an additional class: **Elevator Controller Class**

Second Iteration of Class Diagram

- All relationships are now 1-to-n
  - This makes design and implementation easier

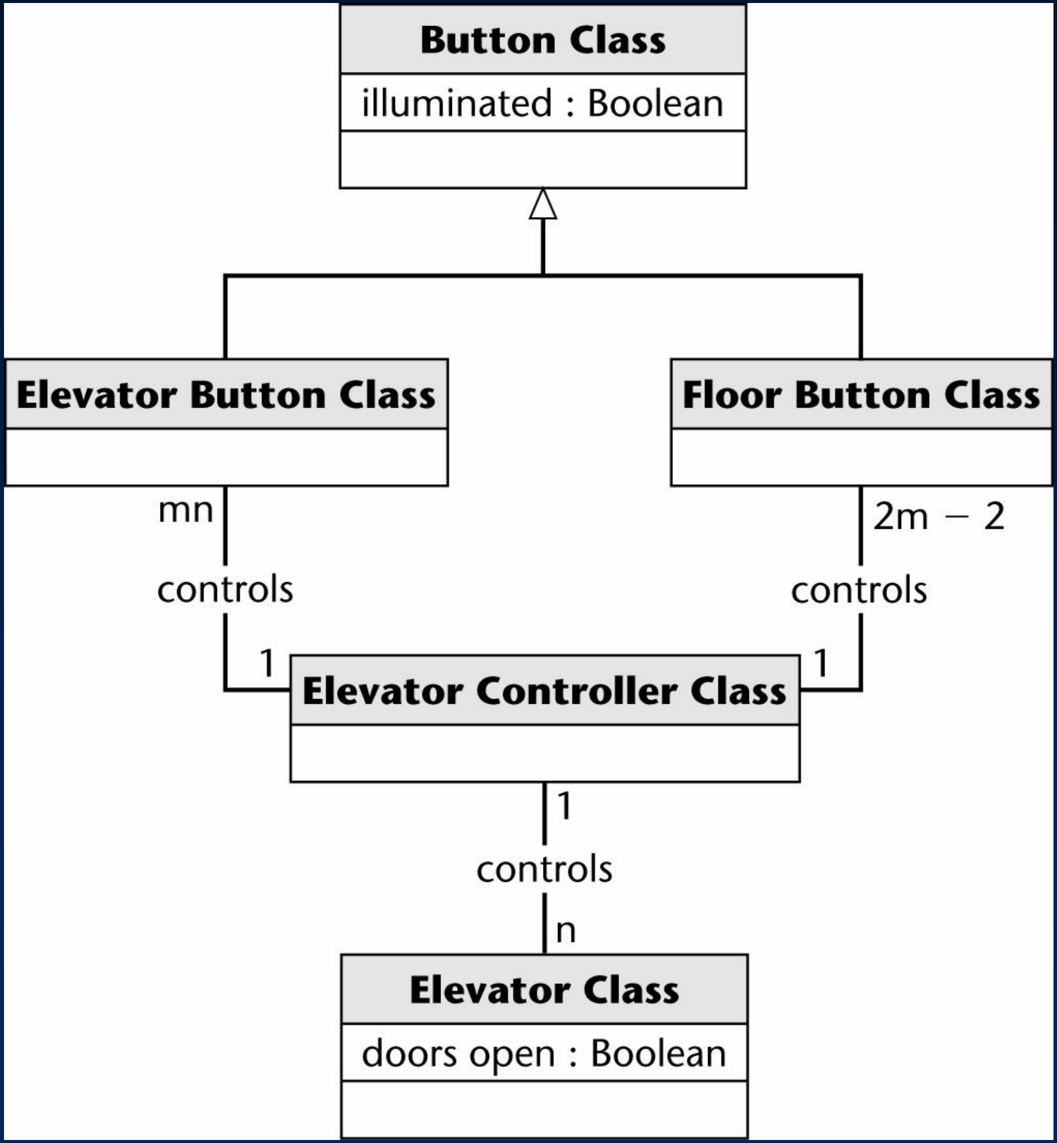


Figure 11.6

11.8.2 CRC Cards

- Used since 1989 for OOA

- For each class, fill in a card showing
  - Name of Class
  - Functionality (Responsibility)
  - List of classes it invokes (Communication)

- Now CRC cards are automated (CASE tool component)

CRC Cards (contd)

- Strength
  - When acted out by team members, CRC cards are a powerful tool for highlighting missing or incorrect items
- Weakness
  - If CRC cards are used to identify entity classes, domain expertise is needed

11.9 Dynamic Modeling: The Elevator Problem Case Study

- Produce a UML statechart

- State, event, and predicate are distributed over the statechart

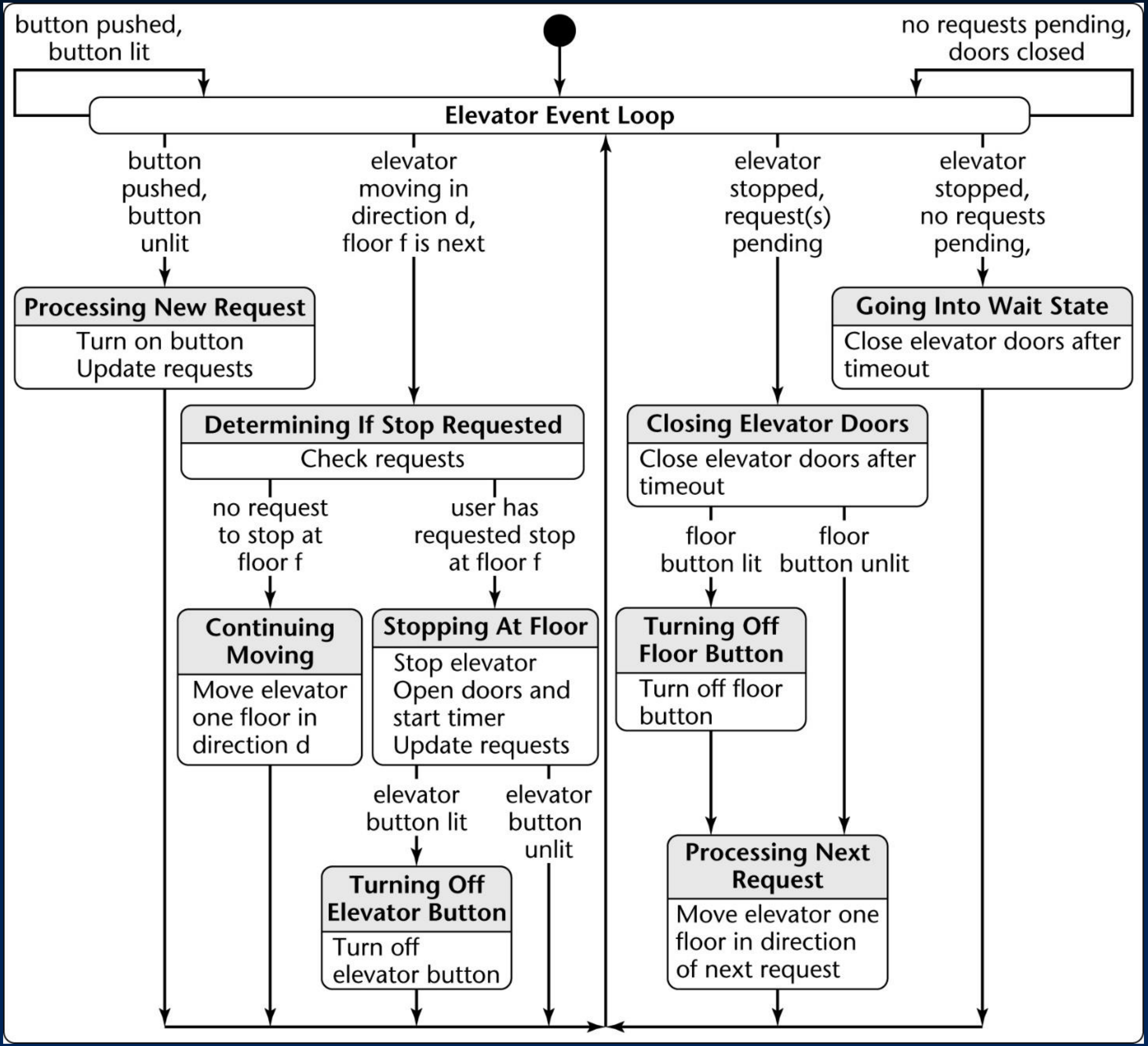

button pushed, button lit
no requests pending, doors closed
Elevator Event Loop
button pushed, button unlit
elevator moving in direction d, floor f is next
elevator stopped, request(s) pending
elevator stopped, no requests pending,
Processing New Request
Turn on button
Update requests
Going Into Wait State
Close elevator doors after timeout
Determining If Stop Requested
Check requests
Closing Elevator Doors
Close elevator doors after timeout
no request to stop at floor f
user has requested stop at floor f
floor button lit
floor button unlit
Continuing Moving
Move elevator one floor in direction d
Stopping At Floor
Stop elevator
Open doors and start timer
Update requests
Turning Off Floor Button
Turn off floor button
elevator button lit
elevator button unlit
Turning Off Elevator Button
Turn off elevator button
Processing Next Request
Move elevator one floor in direction of next request

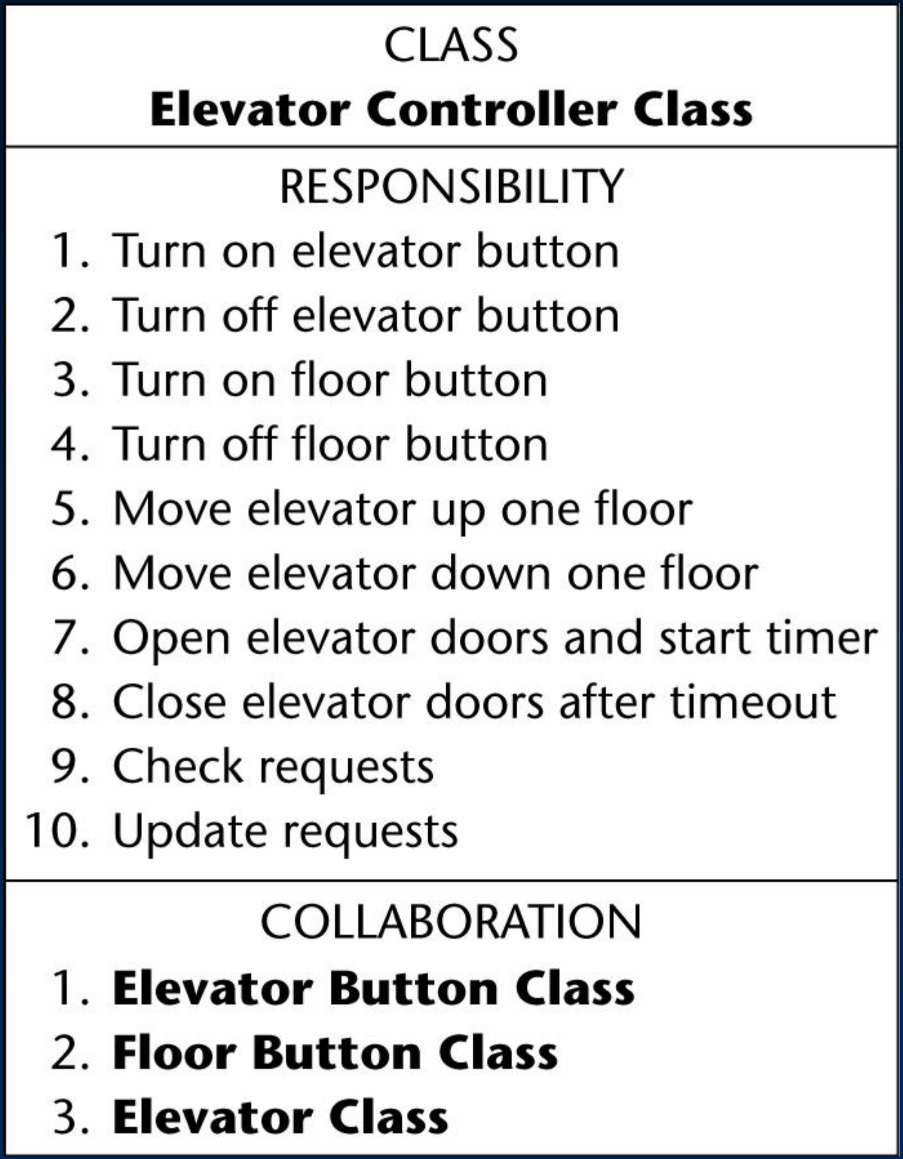
Figure 11.7

Dynamic Modeling: Elevator Problem (contd)

- This UML statechart is equivalent to the state transition diagram of Figures 11.15 through 11.17
- This is shown by considering specific scenarios
- In fact, a statechart is constructed by modeling the events of the scenarios

11.10 The Test Workflow: Elevator Problem

- CRC cards are an excellent testing technique



CLASS

Elevator Controller Class

RESPONSIBILITY

1. Turn on elevator button
2. Turn off elevator button
3. Turn on floor button
4. Turn off floor button
5. Move elevator up one floor
6. Move elevator down one floor
7. Open elevator doors and start timer
8. Close elevator doors after timeout
9. Check requests
10. Update requests

COLLABORATION

1. **Elevator Button Class**
2. **Floor Button Class**
3. **Elevator Class**

Figure 11.8

CRC Cards

- Consider responsibility
  - `1.    Turn on elevator button`

- This is totally inappropriate for the object-oriented paradigm
  - Responsibility-driven design has been ignored
  - Information hiding has been ignored

- Responsibility

  `1.    Turn on elevator button`

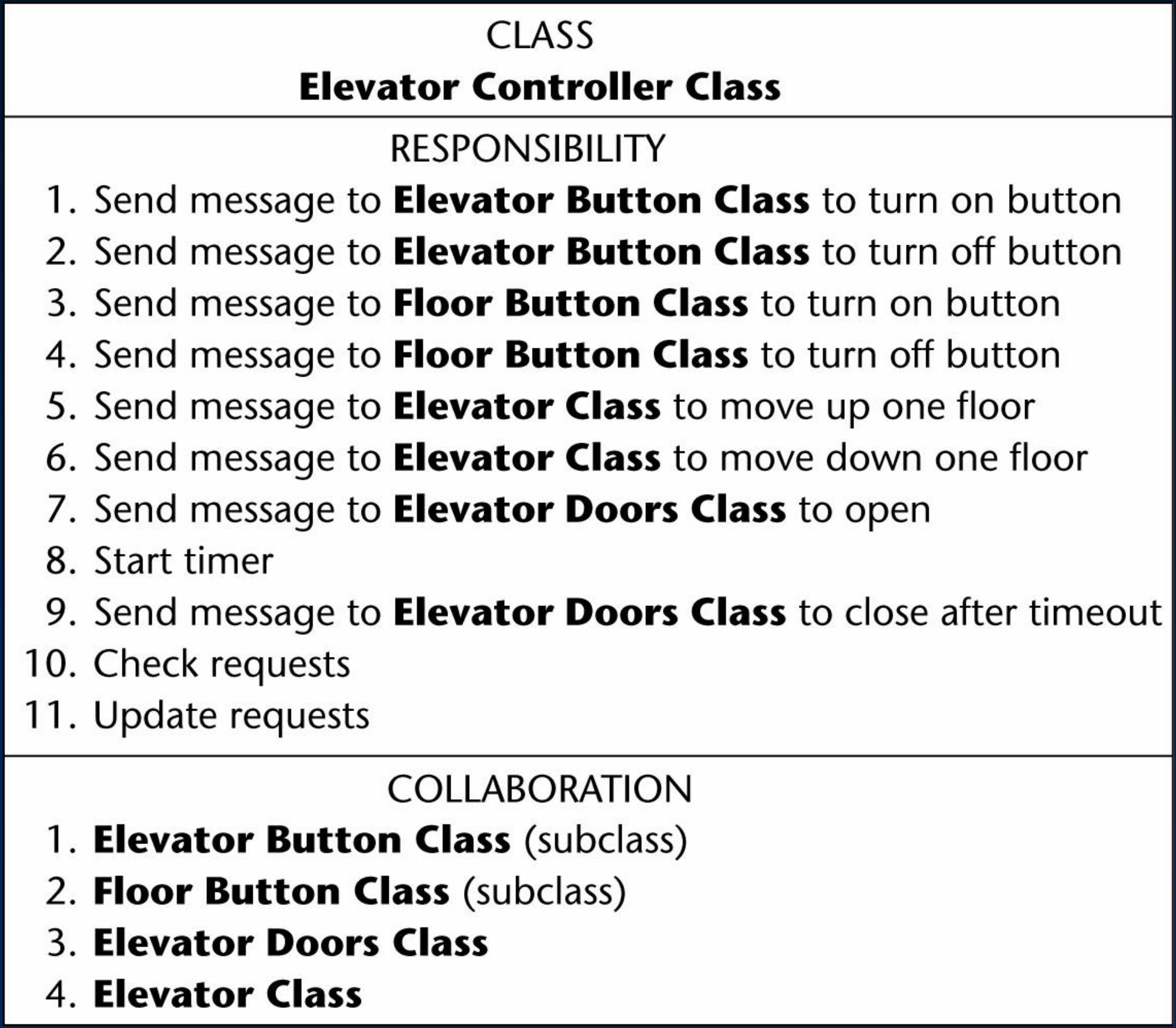
  should be

  `1.    Send message to` **`Elevator Button Class`** `to turn itself on`

CRC Cards (contd)

- Also, a class has been overlooked
- The elevator doors have a *state* that changes during execution (class characteristic)
  - Add class **`Elevator Doors Class`**
  - Safety considerations
- Modify the CRC card

Second Iteration of the CRC Card



CLASS

Elevator Controller Class

RESPONSIBILITY

1. Send message to **Elevator Button Class** to turn on button
2. Send message to **Elevator Button Class** to turn off button
3. Send message to **Floor Button Class** to turn on button
4. Send message to **Floor Button Class** to turn off button
5. Send message to **Elevator Class** to move up one floor
6. Send message to **Elevator Class** to move down one floor
7. Send message to **Elevator Doors Class** to open
8. Start timer
9. Send message to **Elevator Doors Class** to close after timeout
10. Check requests
11. Update requests

COLLABORATION

1. **Elevator Button Class** (subclass)
2. **Floor Button Class** (subclass)
3. **Elevator Doors Class**
4. **Elevator Class**

Figure 11.9

CRC Cards (contd)

- Having modified the class diagram, reconsider the
  - Use-case diagram (no change)
  - Class diagram (see the next slide)
  - Statecharts
  - Scenarios (see the slide after the next slide)

Third Iteration of Class Diagram

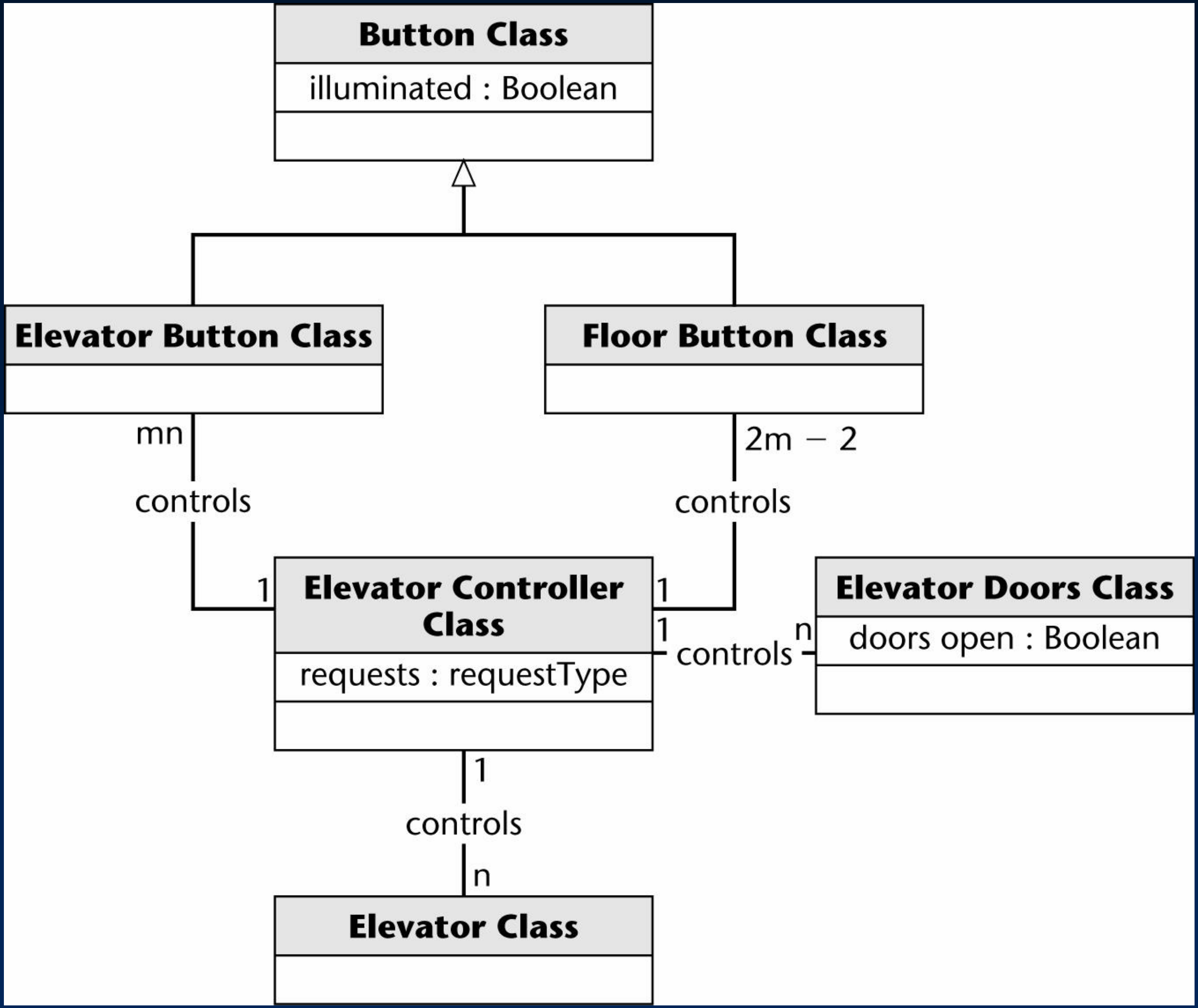
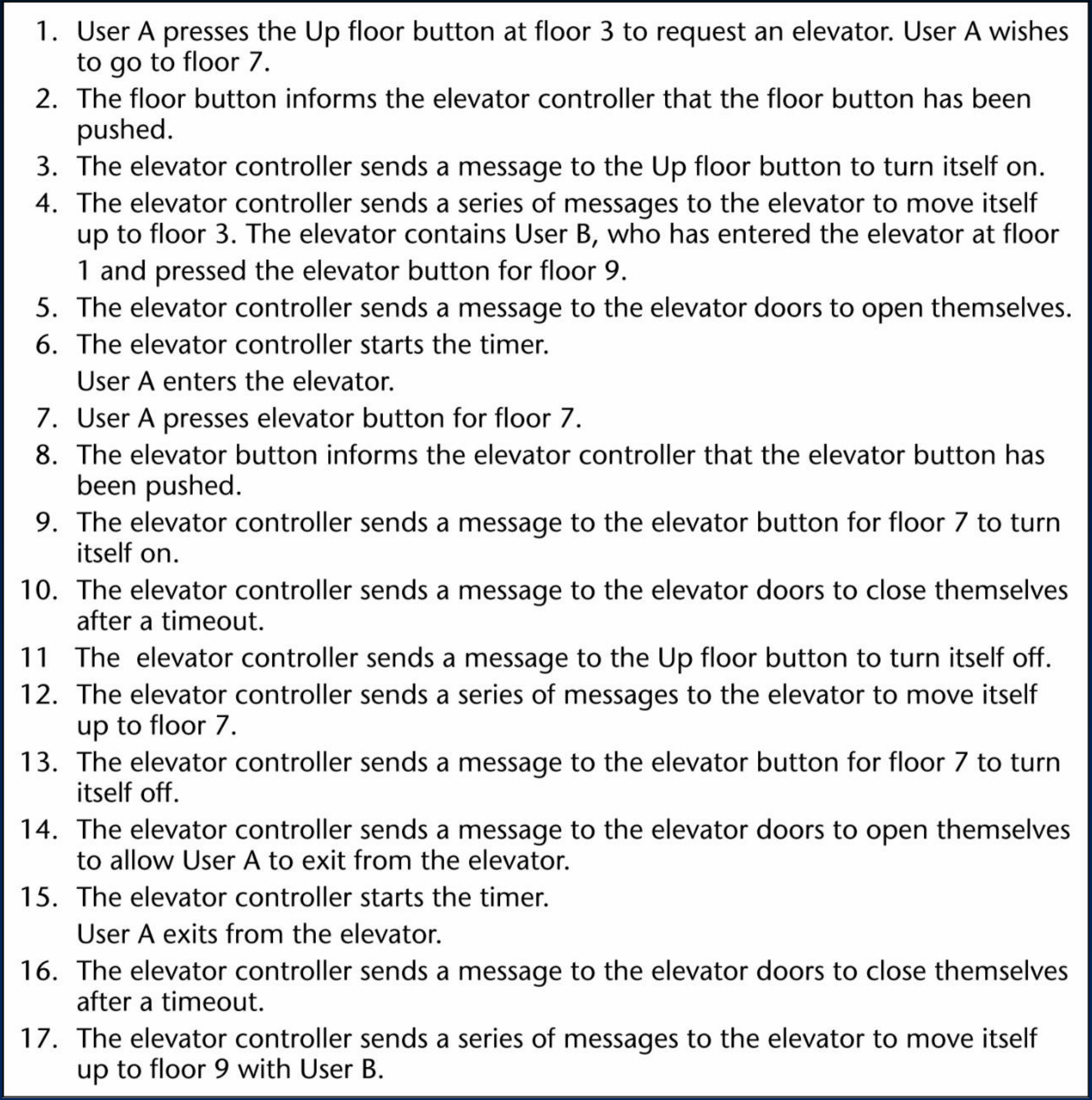


Figure 11.10

Second Iteration of the Normal Scenario:



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 elevator doors to open themselves.
6. The elevator controller starts the timer.

   User A enters the elevator.
7. User A presses elevator button for floor 7.
8. The elevator button informs the elevator controller that the elevator button has been pushed.
9. The elevator controller sends a message to the elevator button for floor 7 to turn itself on.
10. The elevator controller sends a message to the elevator doors to close themselves after a timeout.

11 The elevator controller sends a message to the Up floor button to turn itself off.

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.

Figure 11.11

The Analysis Workflow: Elevator Problem (contd)

- The analysis workflow is now fine
- We should rather say:
  - The analysis workflow is fine *for now*
- We may need to return to the analysis workflow during the design workflow

11.11 Extracting the Boundary and Control Classes

- Each
  - Input screen,
  - Output screen, and
  - Report

  is modeled by its own boundary class

- Each nontrivial computation is modeled by a control class