Software Engineering

Zhang Shuang

zhangs@swc.neu.edu.cn

東北大學

Northeastern University

OBJECT-ORIENTED ANALYSIS (continued)

Chapter 5 Object-Oriented Analysis

- Use-Case Modeling
- Class Modeling
- Dynamic Modeling
- Testing during OOA
- Challenges of OOA

Objects

- **Can you give the definition of class & object in your own words? Give some instances of classes & objects, please.**

Objects

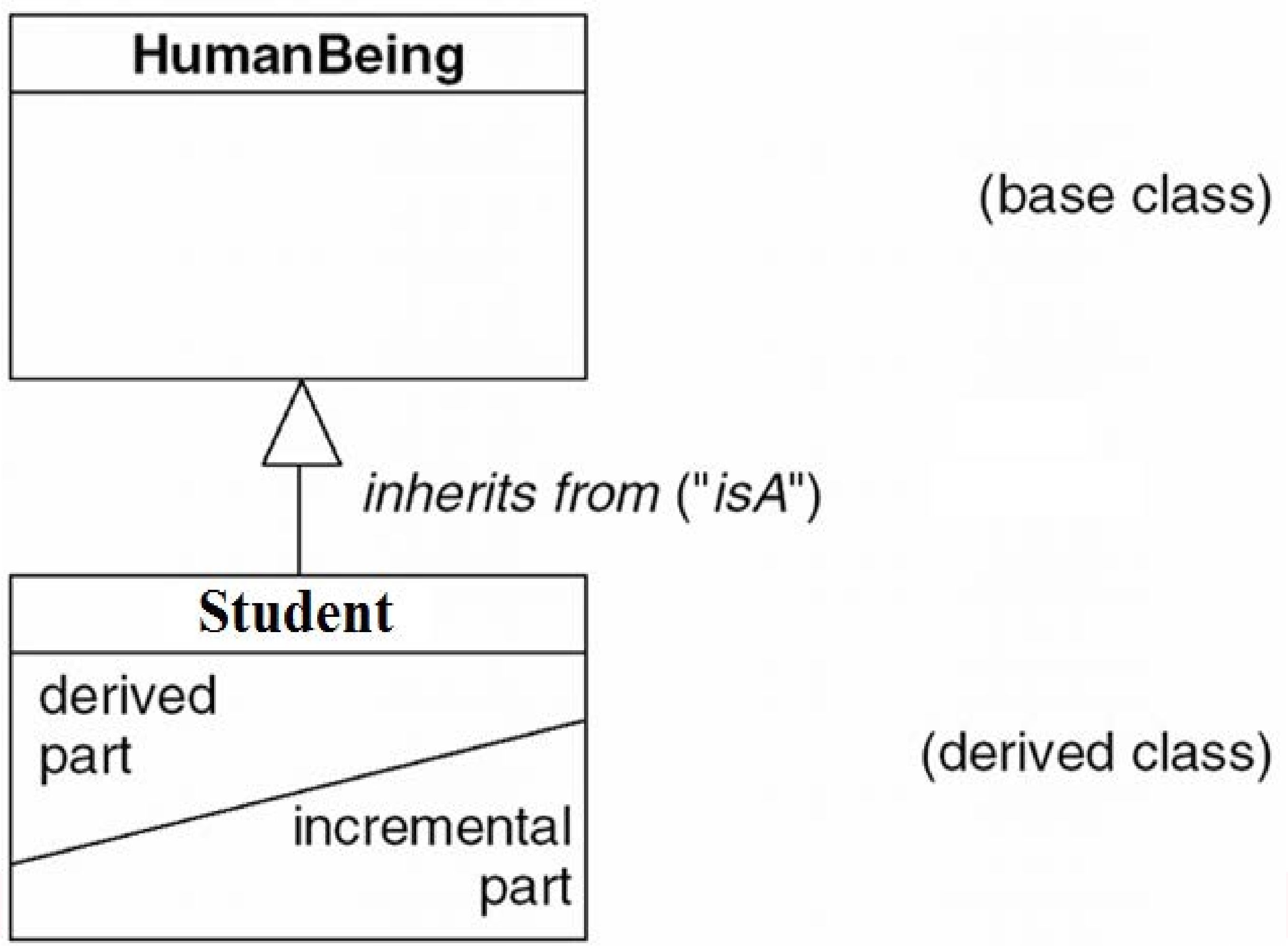
- **Class: abstract data type that supports *inheritance.***
- **Objects are instantiations of classes.**

Inheritance

- **Define *HumanBeing* to be a *class***
  - **A *HumanBeing* has *attributes*, e.g., name, ID, and so on.**
  - **Assign values to attributes when describing object.**
- **Define *Student* to be a *subclass* of *HumanBeing***
  - **A *Student* has all attributes of a *HumanBeing*, plus attributes of his/her own (e.g., School, StudentNo).**
  - **A *Student* inherits all attributes of *HumanBeing*.**

Inheritance

- **UML notation ---- Inheritance is represented by a large open triangle.**



Java Implementation

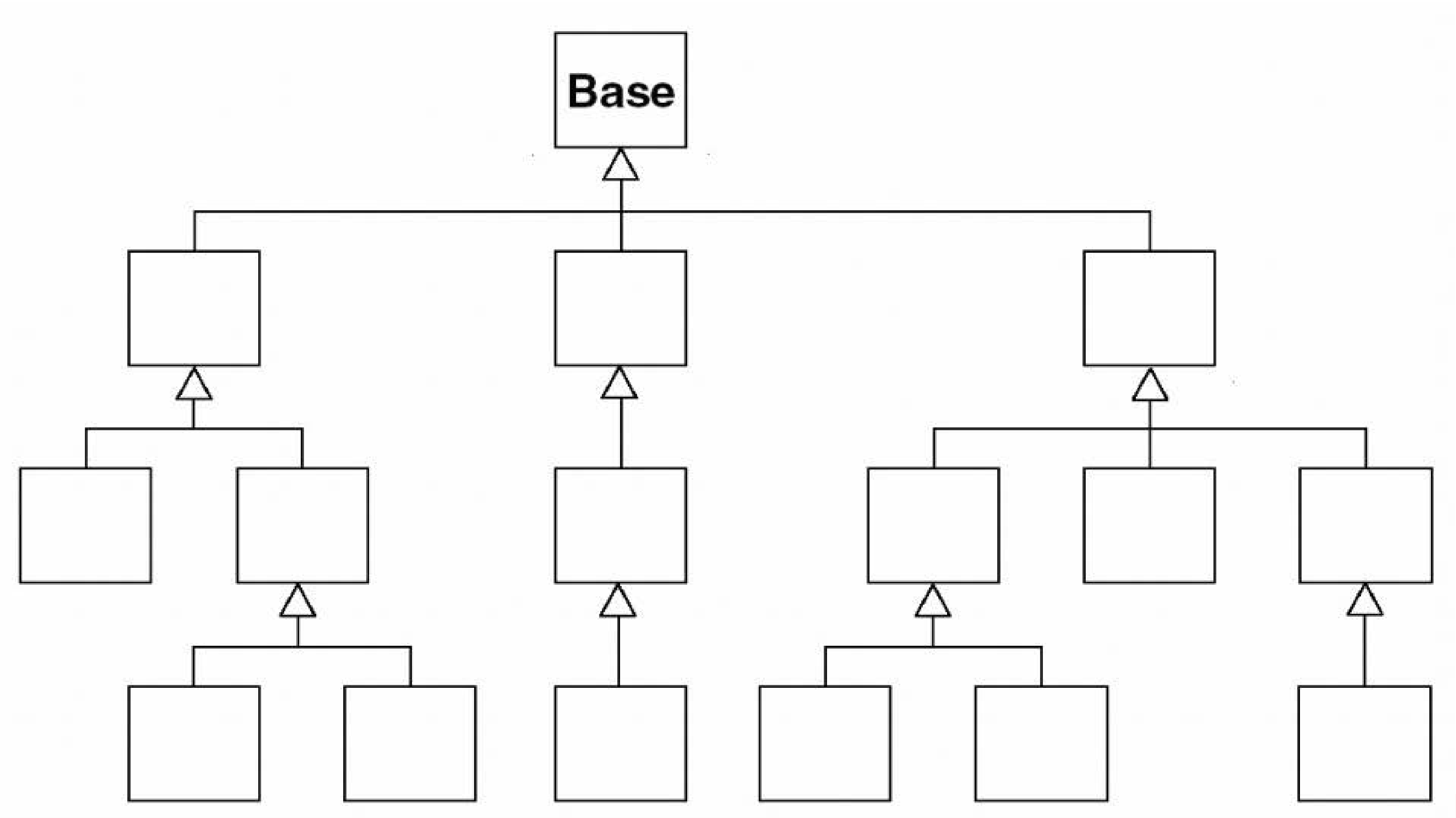
- The property of inheritance is an essential feature of object-oriented languages such as Java, Smalltalk, C++ (but not C, Fortran)

```
class Humanbeing
{
     String ID;
     String Name;
     Date Birthday;
    // public declarations of operations on HumanBeing
}

class Student extends HumanBeing
{
     String StudentNo;
     String StudentName;
     String School;
    // public declarations of operations on Student
}
```

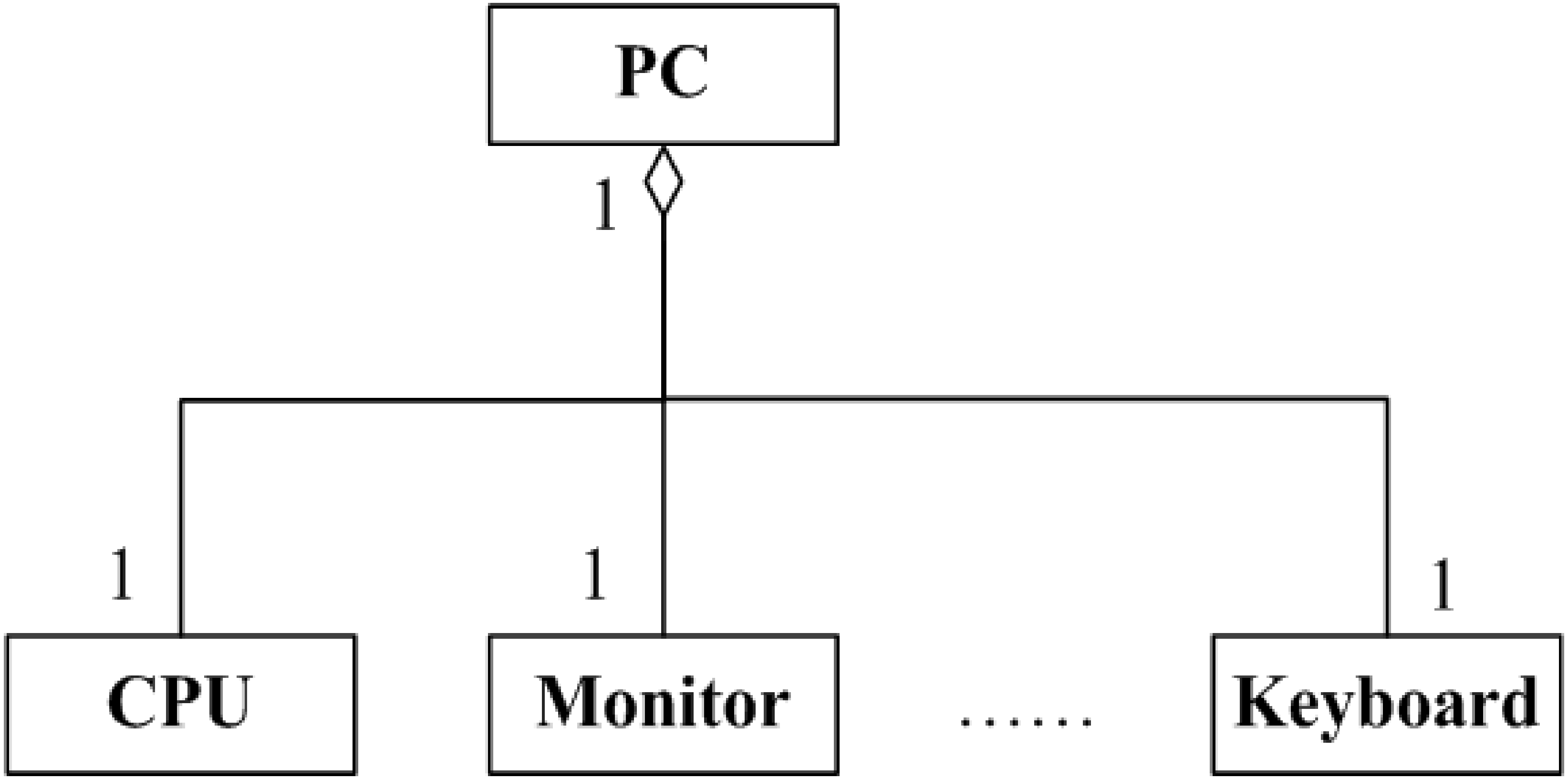
Inheritance

- **Thinking: Fragile base-class problem**



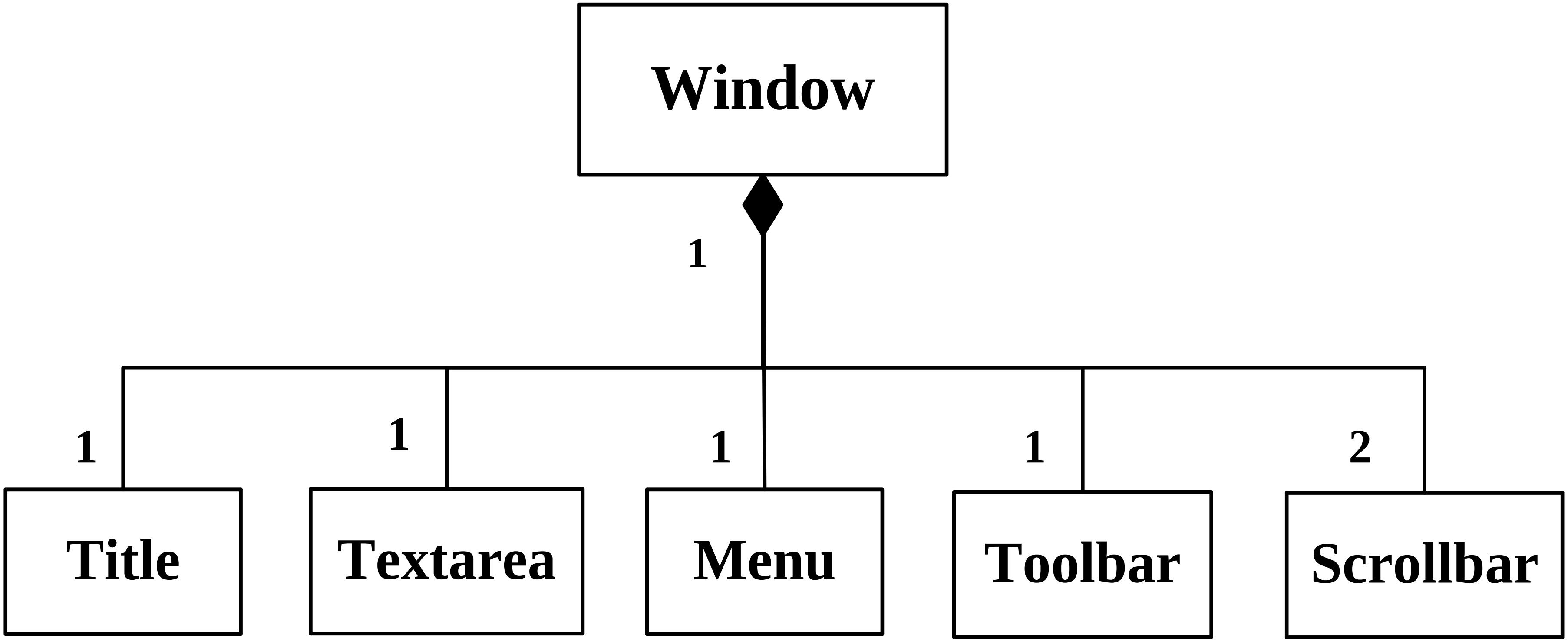
Aggregation

- **UML Notation**



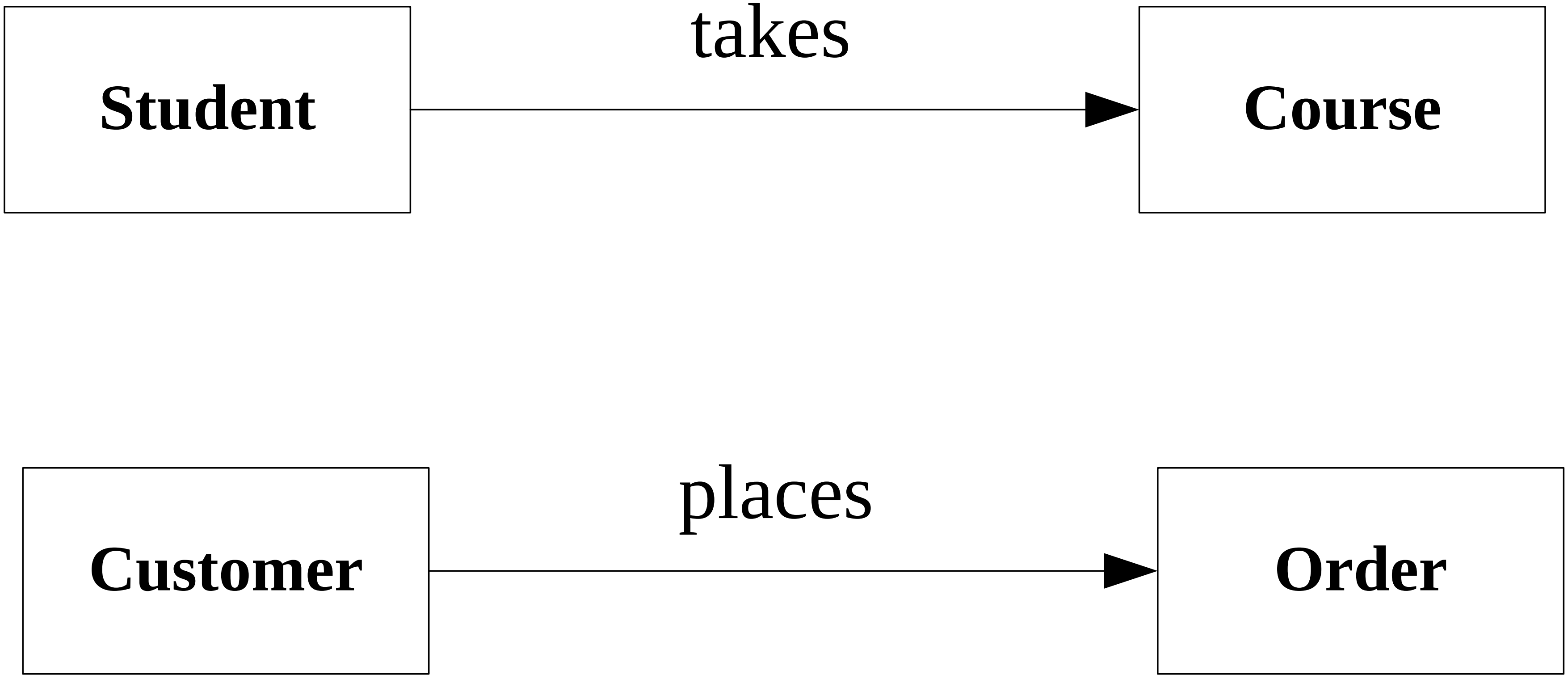
Composition

- **UML Notation**



Association

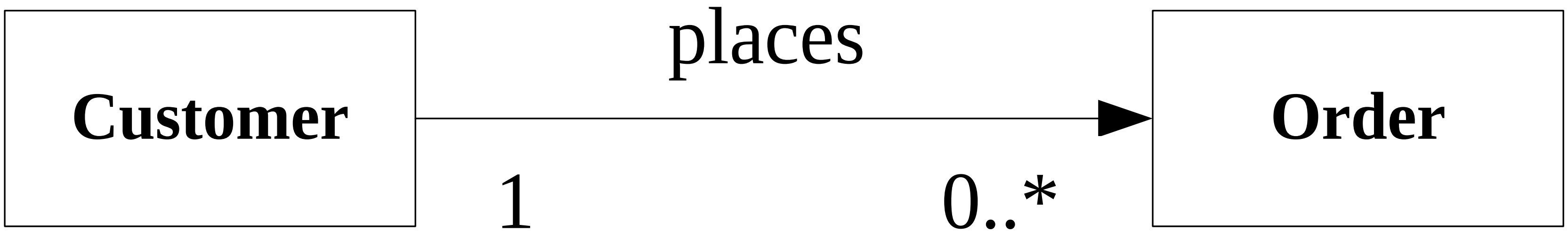
- **UML Notation**



Multiplicity

- **Every class engaged in a relationship should have a multiplicity. (*except inheritance*)**
- **If the multiplicity is 1, indicate that it is 1; in order that reader know that multiplicity has been considered.**
- **UML Multiplicity Indicators**

Multiplicity



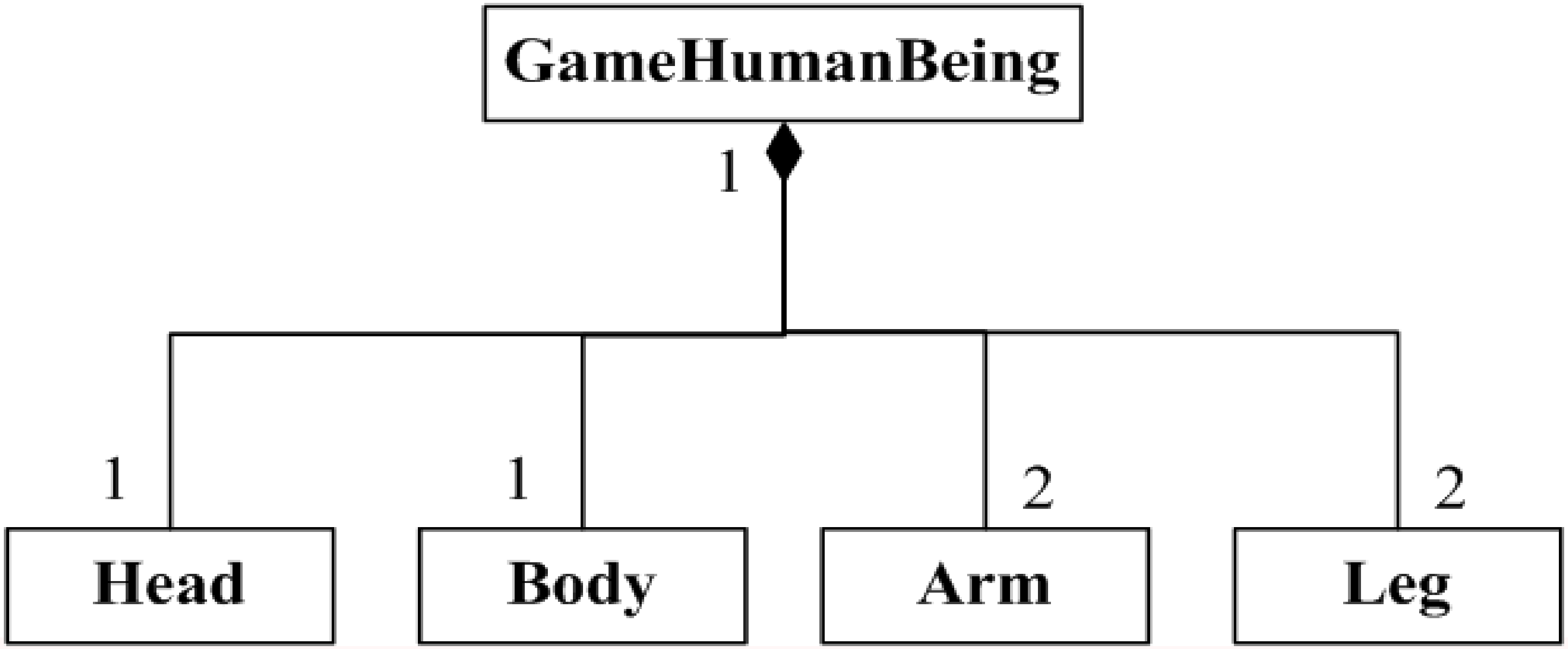
Customer

places

1

0..*

Order



GameHumanBeing

1

1

1

2

2

Head

Body

Arm

Leg

2. Class Modeling

- **What is a class diagram?**
  - **A class diagram shows the entity classes, their attributes and their relationships to other entity classes in the target system.**
  - **It is the static view of a system.**
  - **It supports the functional requirements of a system.**
  - **No processing or workflow in class diagram.**

Two Approaches to Class Modeling

- **Noun extraction**
  - **Always works**
- **CRC cards ---- Class-Responsibility-Collaboration**
  - **Need to have domain expertise**
  - **For testing class diagram**

Noun Extraction

- **Stage 1. Concise Problem Definition ---- Define product briefly and concisely.**

Buttons in elevators and on the floors control movement of n elevators in a building with m floors. Buttons illuminate when pressed to request the elevator to stop at a specific floor; 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.

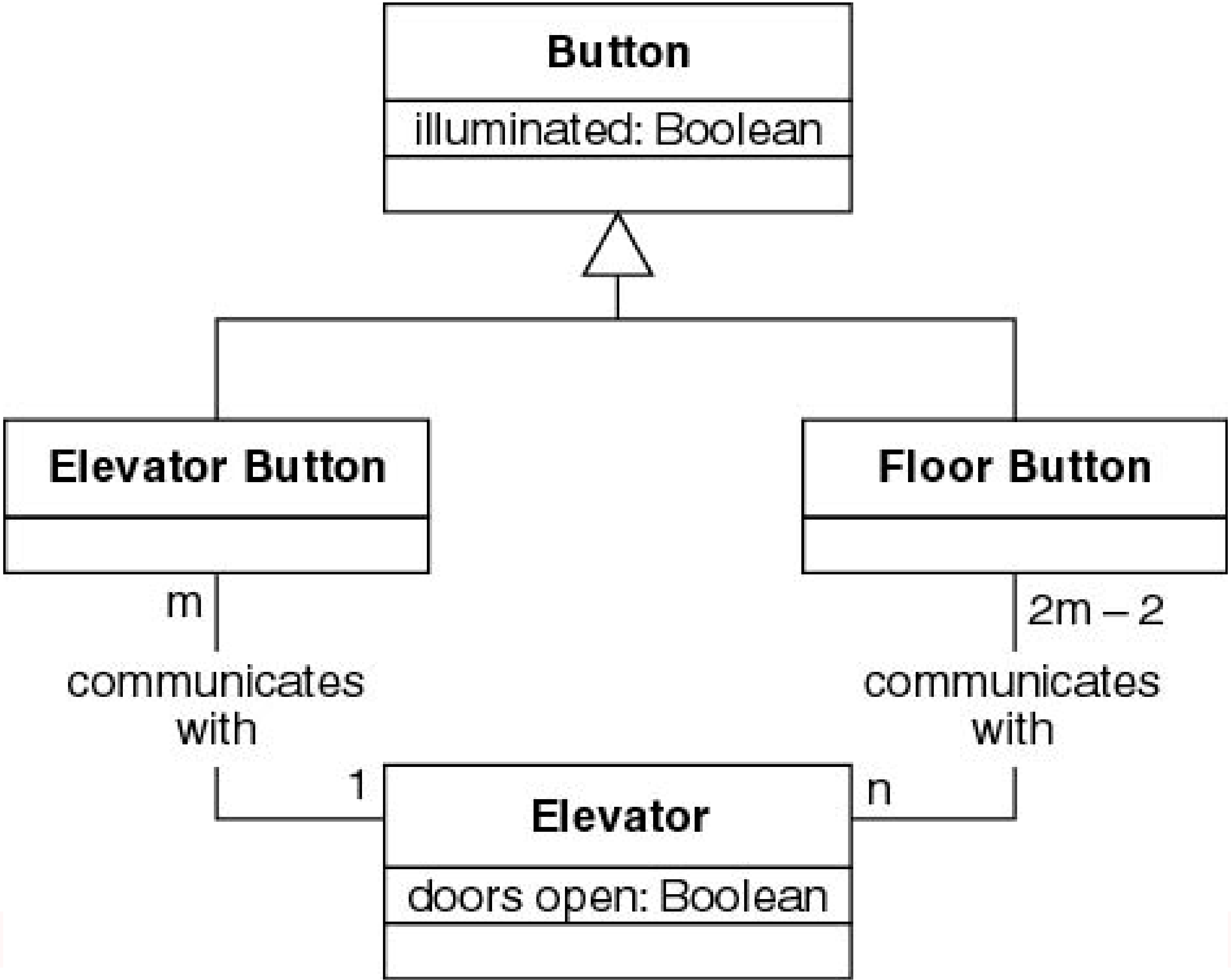
Noun Extraction (contd)

- **Stage 2. Identify nouns in informal strategy. Use the nouns as candidate entity classes.**
- **Nouns**
  - **button, elevator, floor, movement, building, illumination, door**

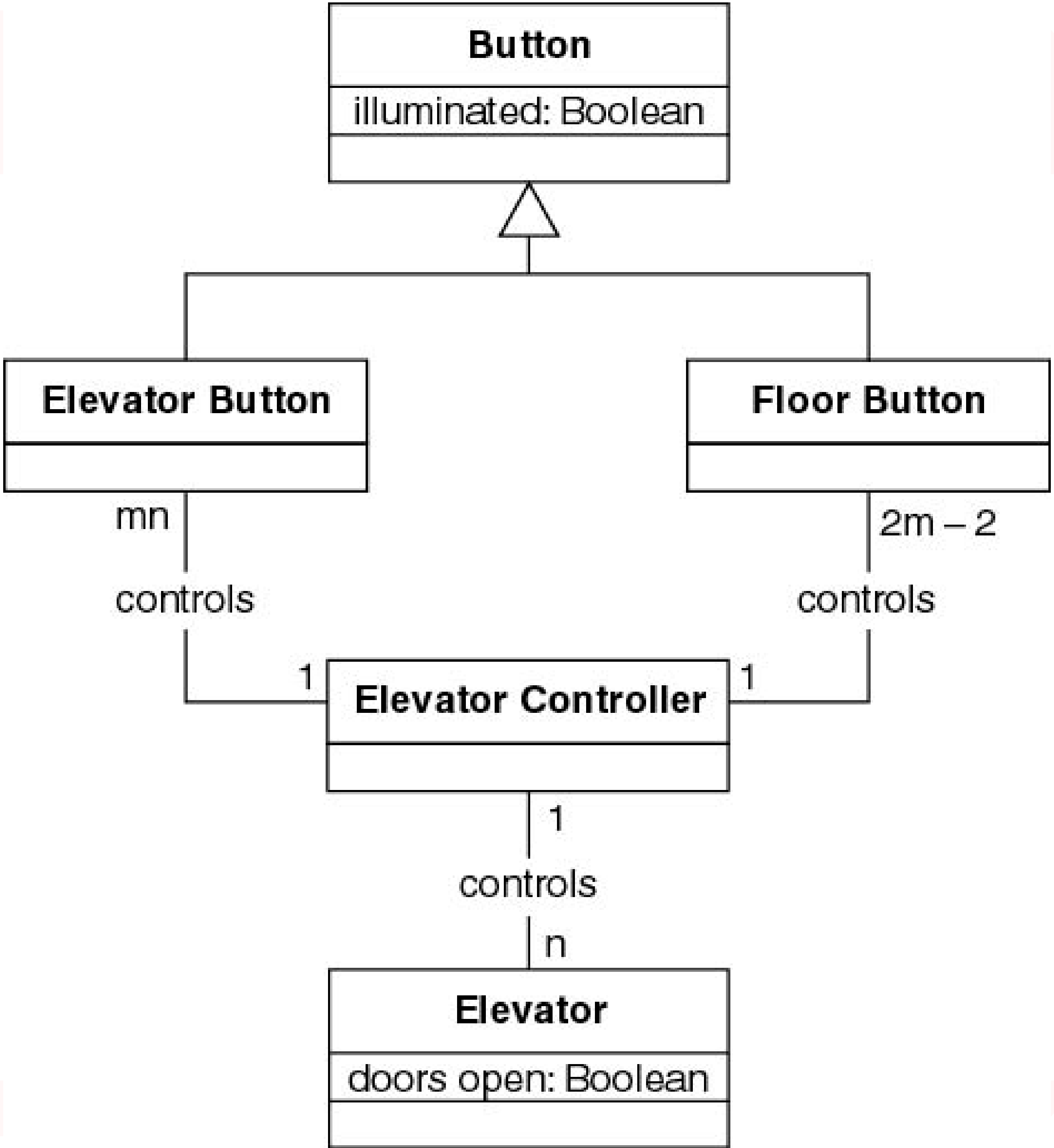
Noun Extraction (contd)

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

First Iteration of Class Diagram

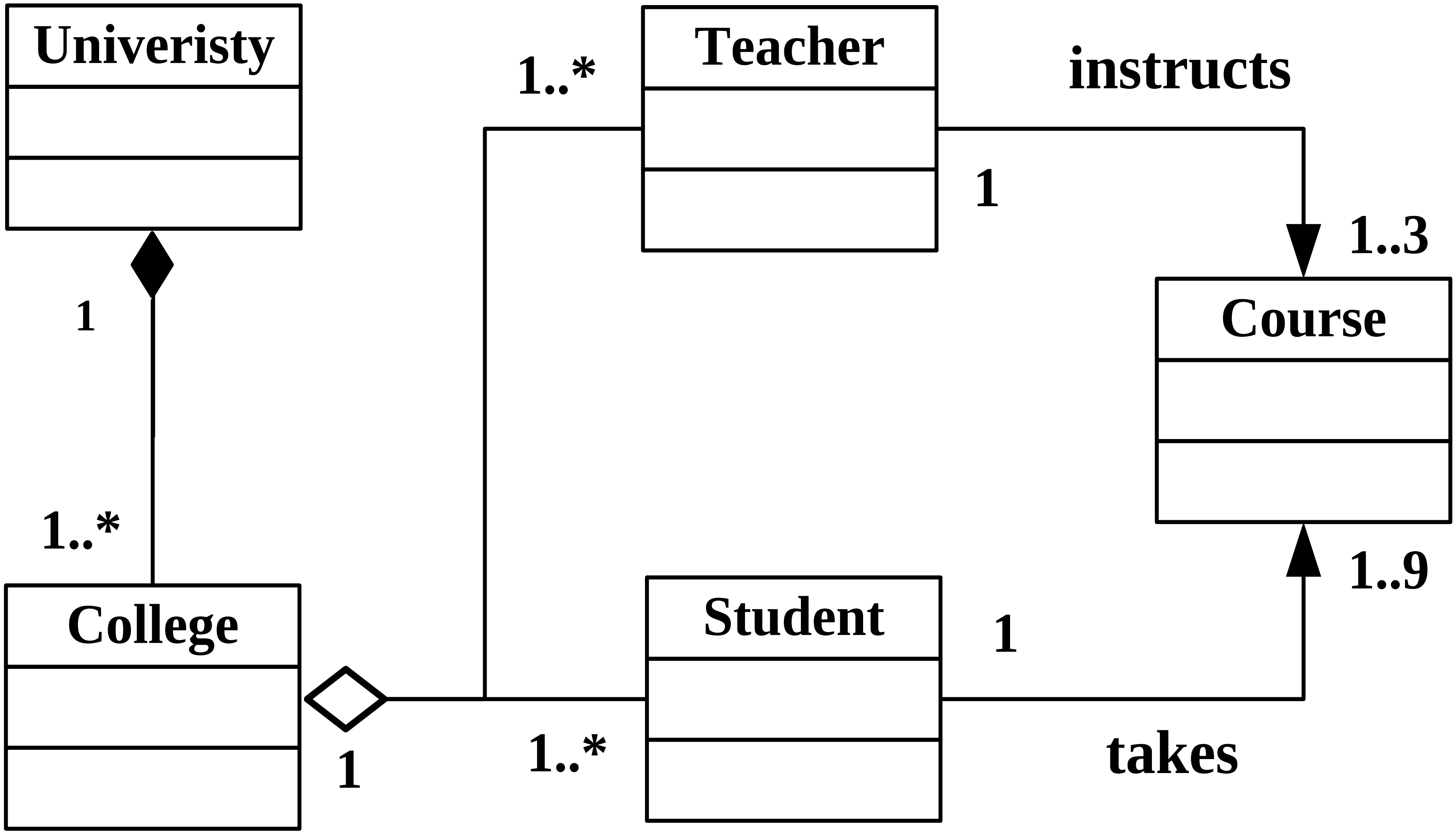


Second Iteration of Class Diagram



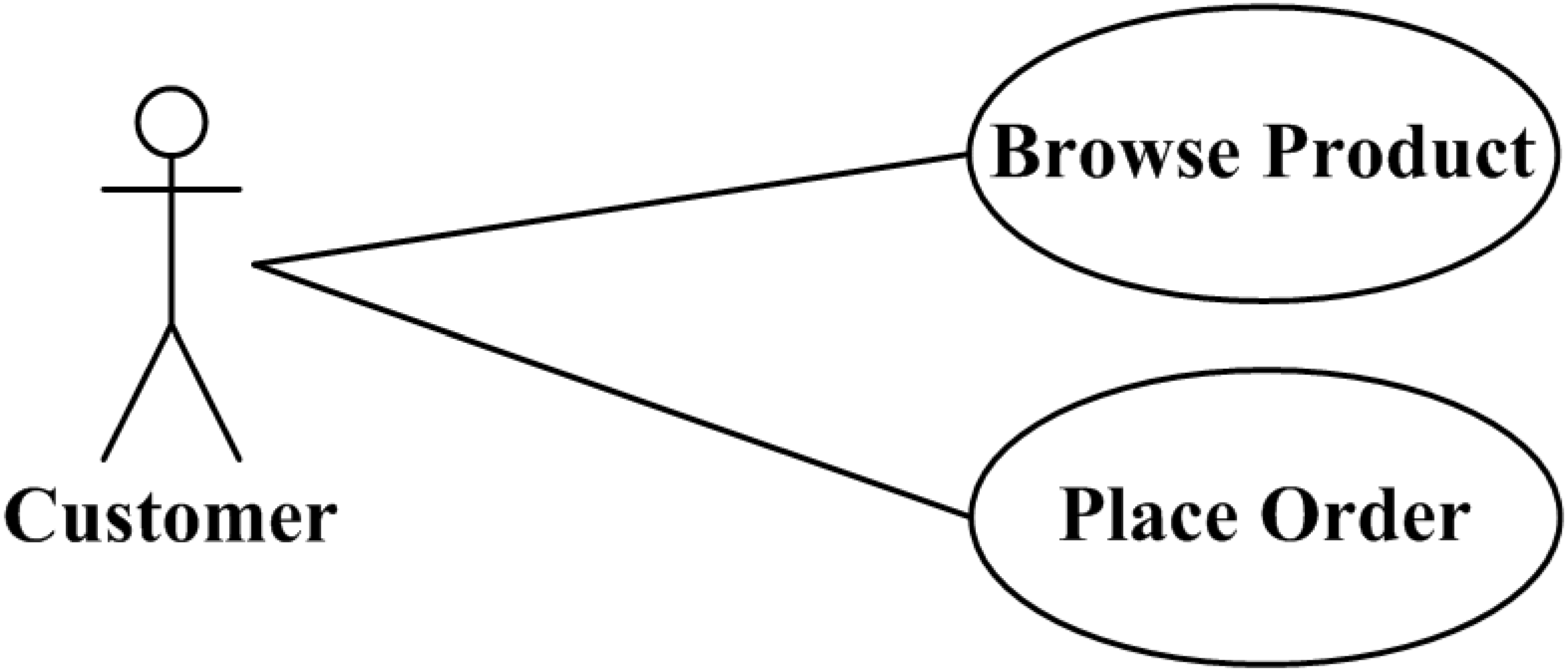
Exercise 1

- **OOA: get class diagram for University System.**

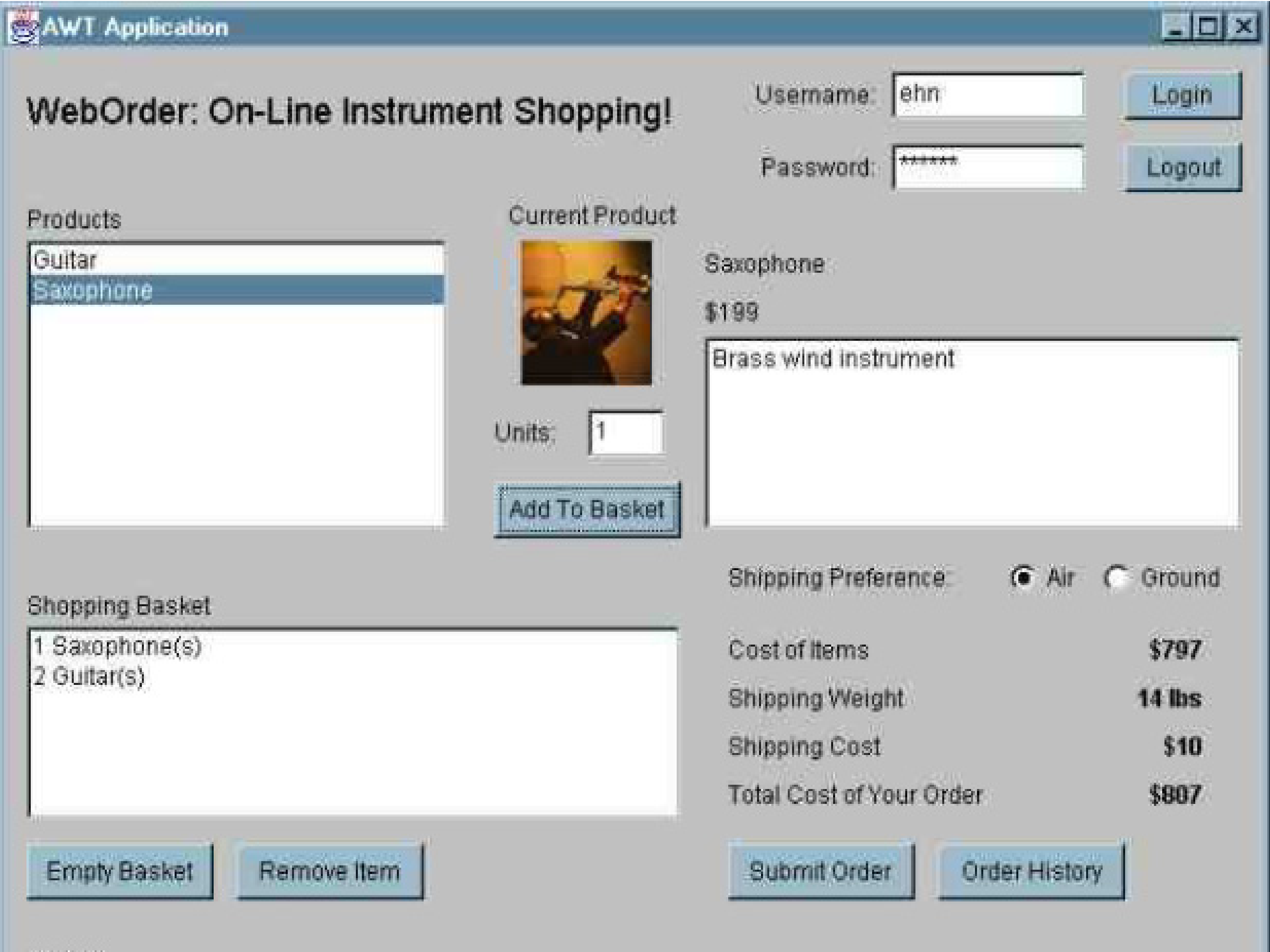


Exercise 2

- Case 2: On-line Shop



- Shopping basket won’t be persisted for the user once he logs out.



WebOrder: On-Line Instrument Shopping!

Username: ehn

Password: ******

Products

Guitar
Saxophone

Current Product

Saxophone

$199

Brass wind instrument

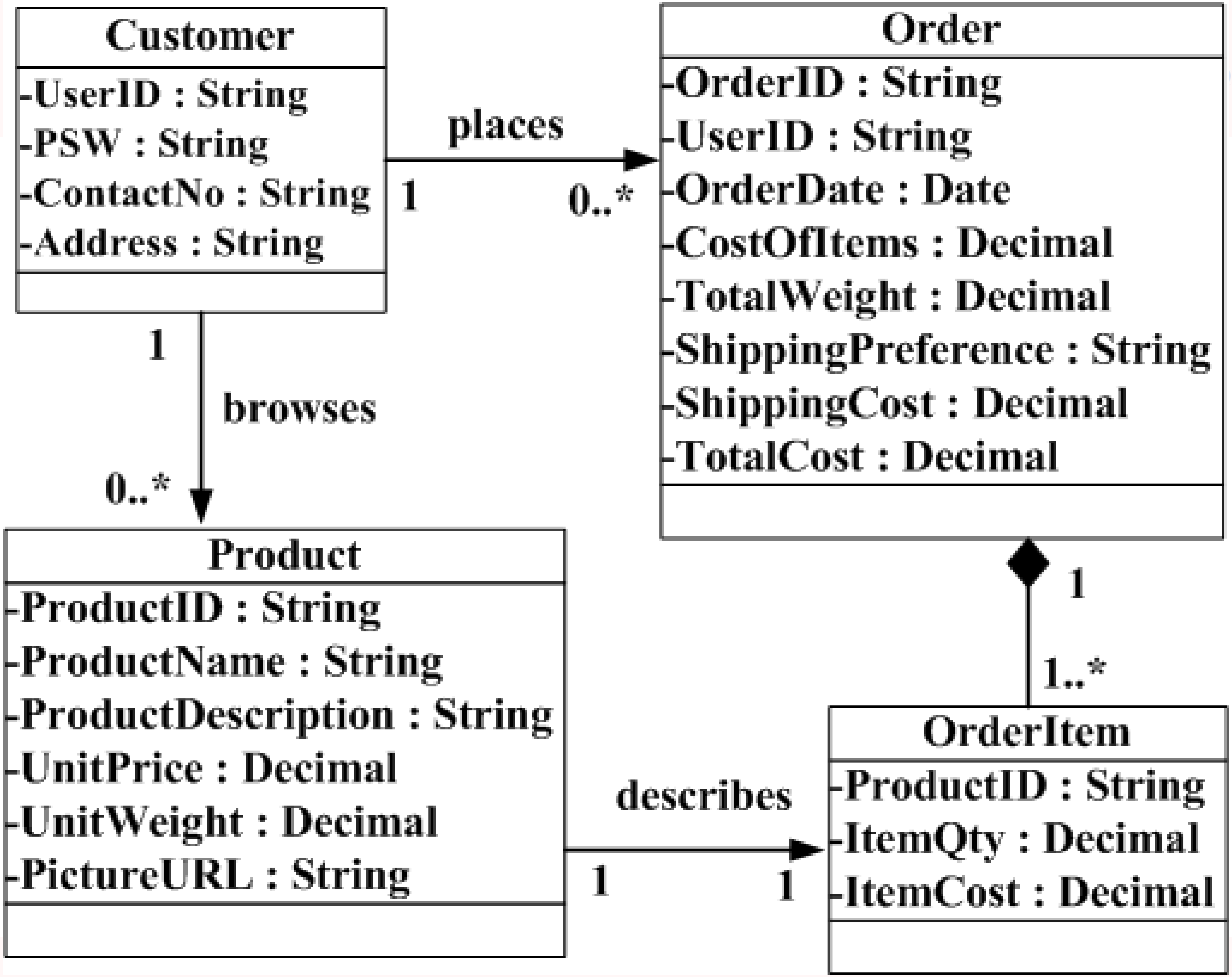
Units: 1

Shipping Preference: ● Air ○ Ground

Shopping Basket

1 Saxophone(s)
2 Guitar(s)

Cost of Items	$797
Shipping Weight	14 lbs
Shipping Cost	$10
Total Cost of Your Order	$807


Customer
-UserID : String
-PSW : String
-ContactNo : String
-Address : String
places
1
0..*
Order
-OrderID : String
-UserID : String
-OrderDate : Date
-CostOfItems : Decimal
-TotalWeight : Decimal
-ShippingPreference : String
-ShippingCost : Decimal
-TotalCost : Decimal
1
browses
0..*
Product
-ProductID : String
-ProductName : String
-ProductDescription : String
-UnitPrice : Decimal
-UnitWeight : Decimal
-PictureURL : String
1
1..*
describes
1
1
OrderItem
-ProductID : String
-ItemQty : Decimal
-ItemCost : Decimal

CRC Cards

- **CRC ----**

  Class-Responsibility-Collaboration

3. Dynamic Modeling

- Produce UML state diagram.
- State diagram is replenishment of class description. It depicts all the states that a class' instance may experience and the causing events.
- An event may be a message from another object, or meeting some conditions.
- An event may be an action which causing state change.

3. Dynamic Modeling

- The state's change is called transition.
- One state diagram is for one class.
- Not all classes need state diagram.
- Some classes have clear states changing according to conditions and events.

3. Dynamic Modeling

- States, events, guards/conditions are distributed over state diagram
- UML “guards” are in brackets [ ]
- Initial state ●
- End state ◉

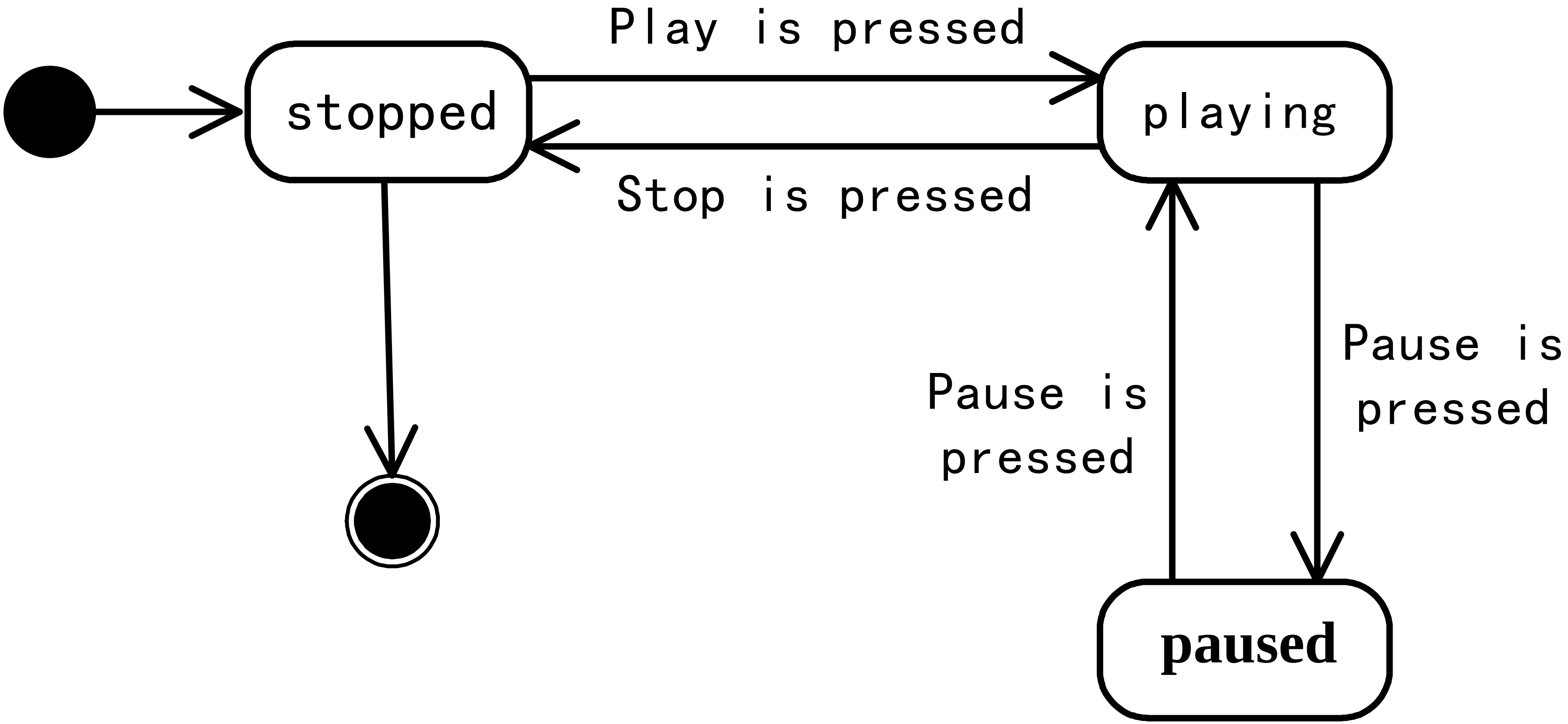
3. Dynamic Modeling

- **Exercise 1**

  Draw a UML state diagram to model the control program for a portable CD player. Include three states: *stopped*, *playing*, and *paused*. Also, include three events possible in any state: *pause_is_pressed*, *stop_is_pressed*, *play_is_pressed*.

3. Dynamic Modeling

- **State Diagram for *CD player***



3. Dynamic Modeling

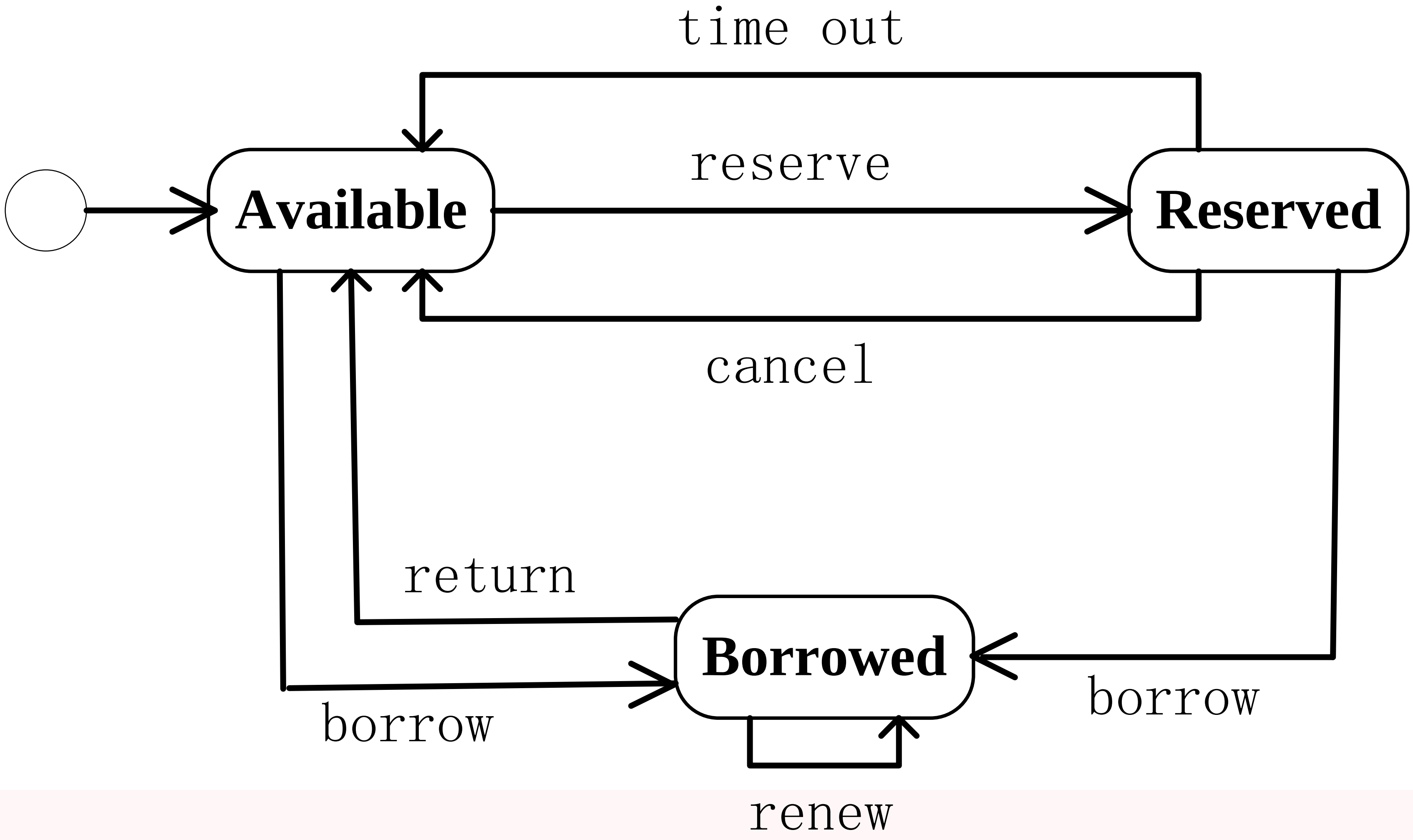
- **Exercise 2**

Draw a UML state diagram to model the Library Mgmt. System for a *book*.

- **A book may have three states: *available*, *borrowed*, and *reserved*.**
- **Possible events: *borrow*, *return*, *reserve*, *cancel reservation*, *reservation times out* and *renew*.**

3. Dynamic Modeling

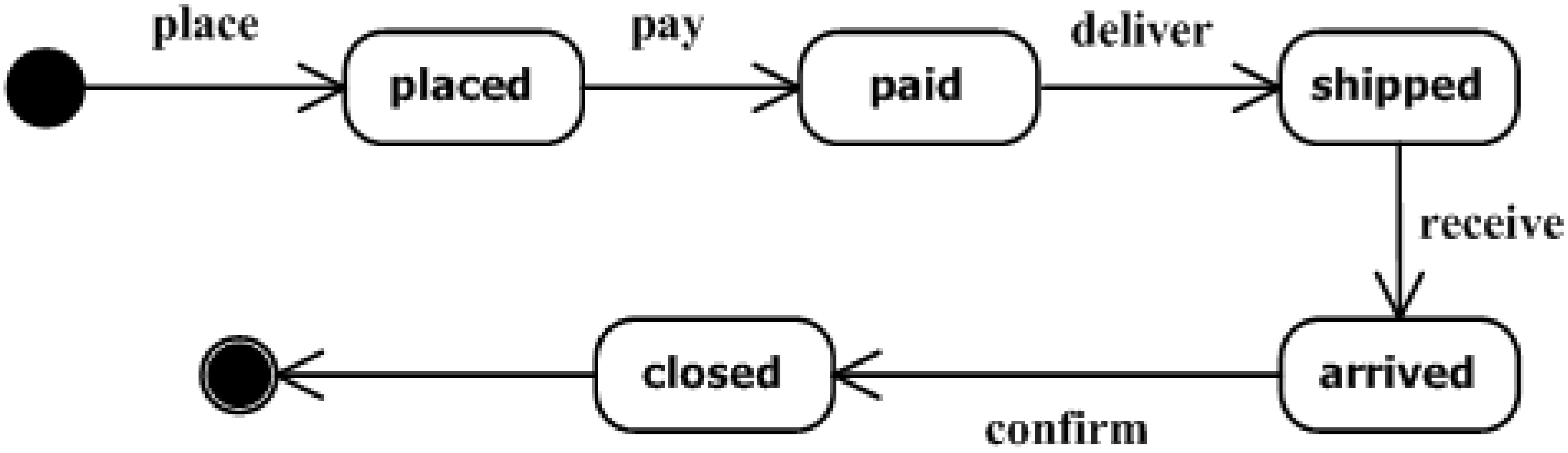
- State Diagram for ***Book*** in Library Mgmt. Sys.



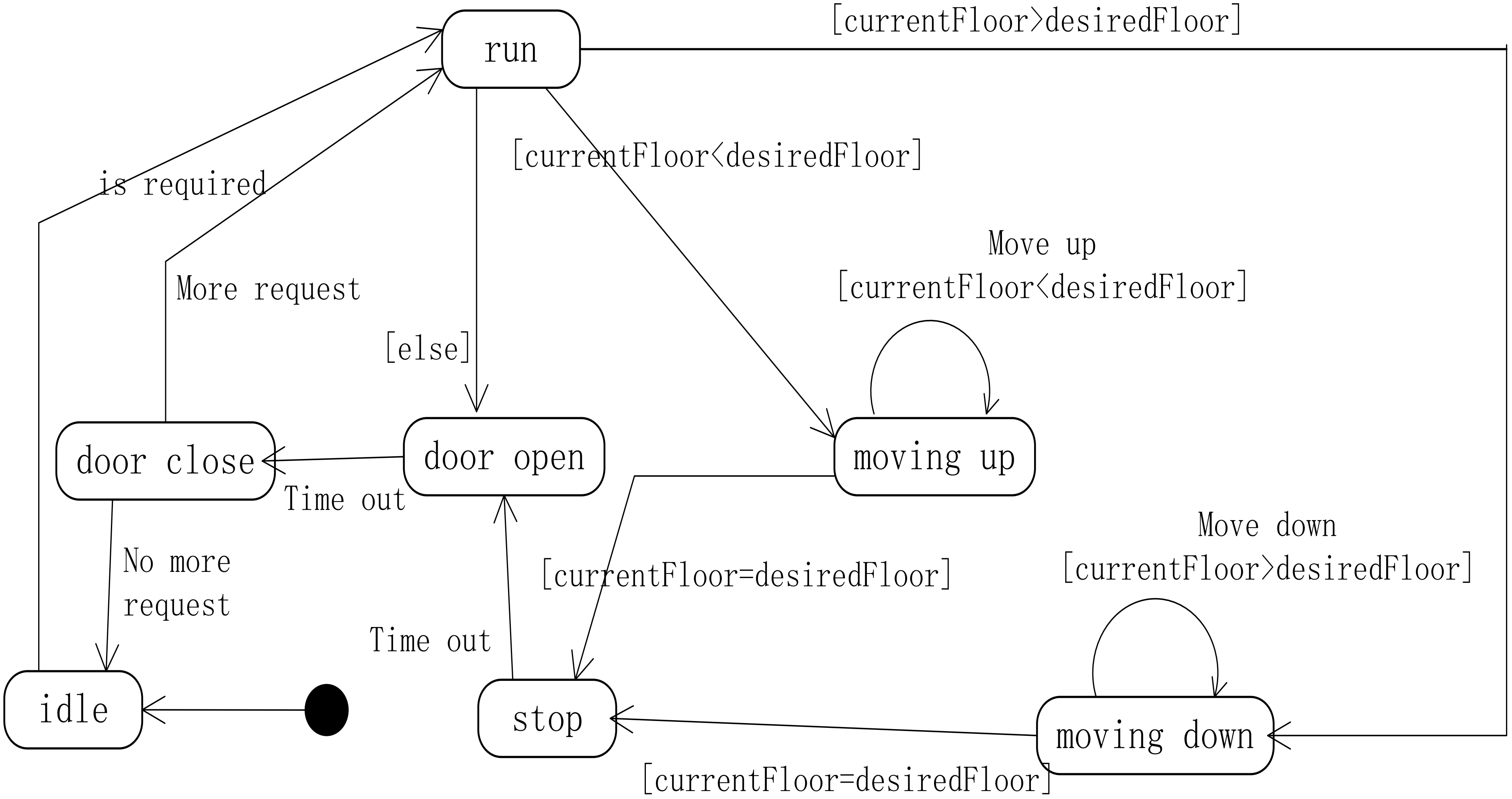
3. Dynamic Modeling

- **Exercise 3 ---- Order in an eCommerce system**

An order may experience the states of *placed*, *paid*, *shipped*, *arrived*, *closed*.



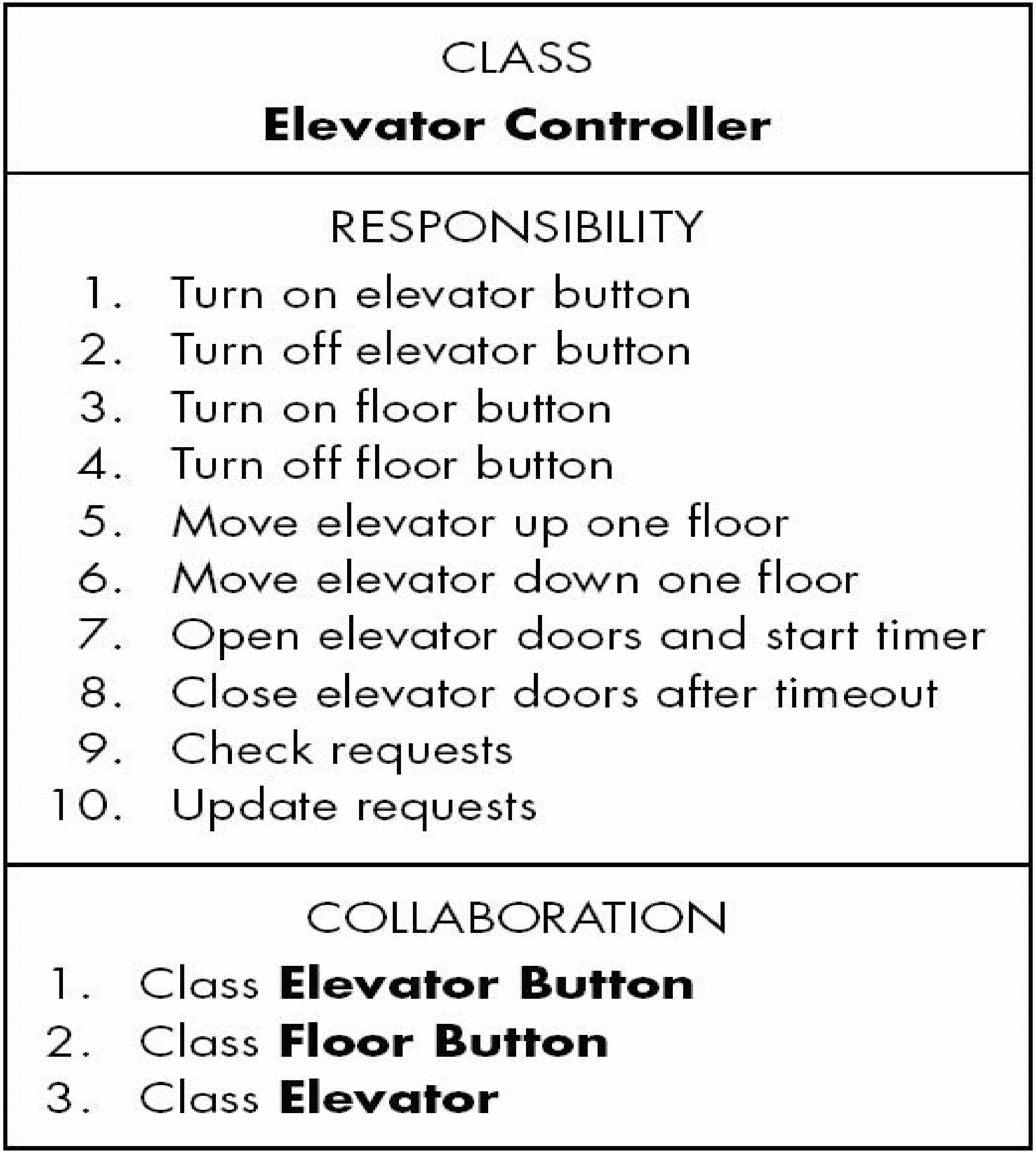
3. Dynamic Modeling
—— State Diagram of class Elevator



[currentFloor>desiredFloor]
run
[currentFloor<desiredFloor]
is required
More request
Move up
[currentFloor<desiredFloor]
[else]
door close
door open
moving up
Time out
Move down
[currentFloor>desiredFloor]
No more
request
[currentFloor=desiredFloor]
Time out
idle
stop
moving down
[currentFloor=desiredFloor]

4. Testing

- **CRC cards are an excellent testing technique.**



CLASS

Elevator Controller

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

4. Testing during OOA ---- CRC Cards

- **Consider responsibility**

  1. Turn on elevator button

- **Totally unacceptable for object-oriented paradigm**
- **Information hiding ignored**
- **Responsibility-driven design ignored**
- **Responsibility**
  - ***1. Turn on elevator button***

    should be

  - ***1. Send message to ElevatorButton to turn itself on***

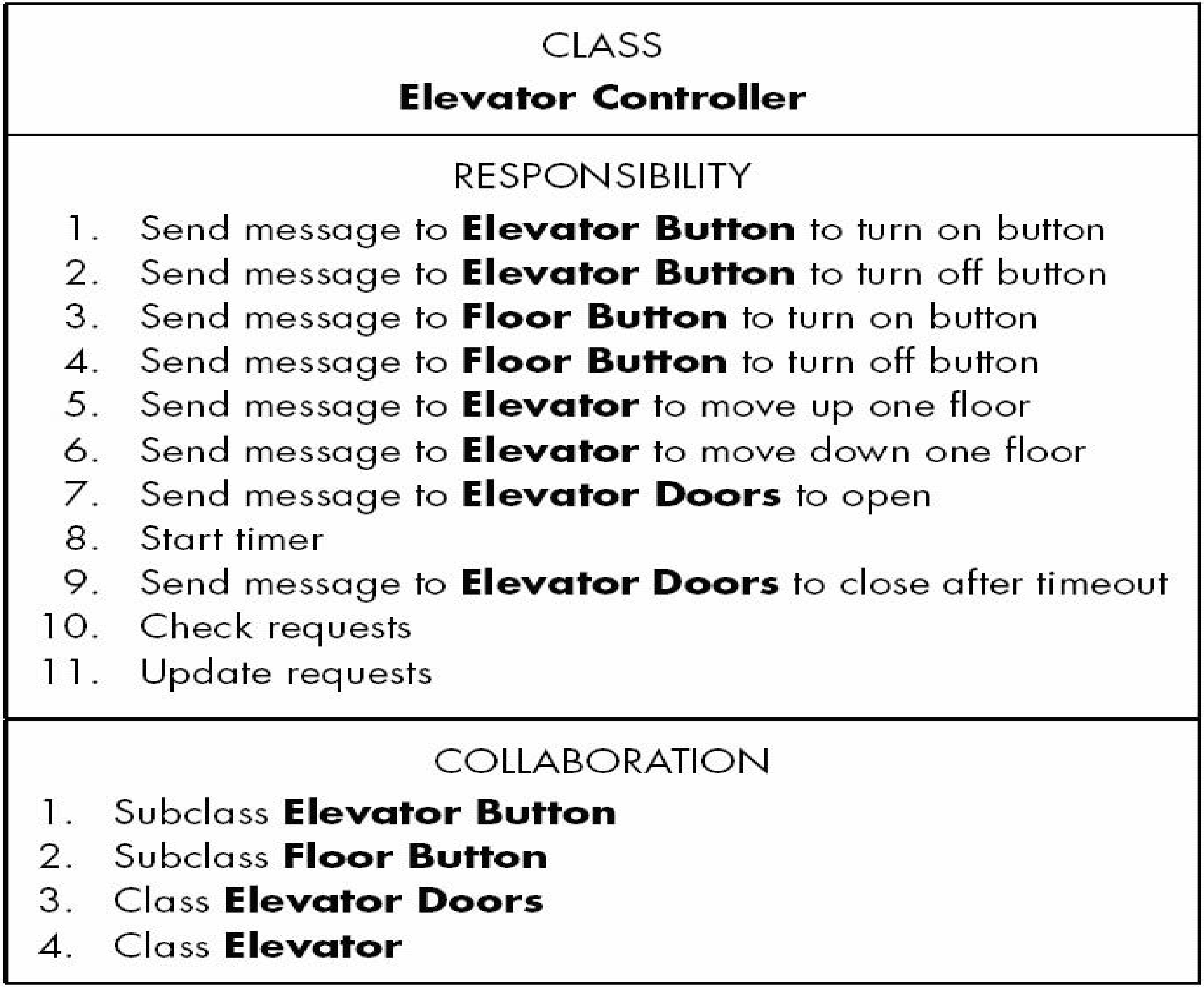
4. Testing during OOA ---- CRC Cards

- **A class has been overlooked**
  - **Elevator *doors* have a *state* that changes during execution (class characteristic)**
  - **Add class *ElevatorDoors***
- **If a component in question possesses a state that will be changed during execution of the implementation, it probably should be modeled as a class.**

4. Testing during OOA ---- CRC Cards

- **Reconsider class model**
- **Then reconsider dynamic model, use-case model**

Second Iteration of CRC Card



CLASS

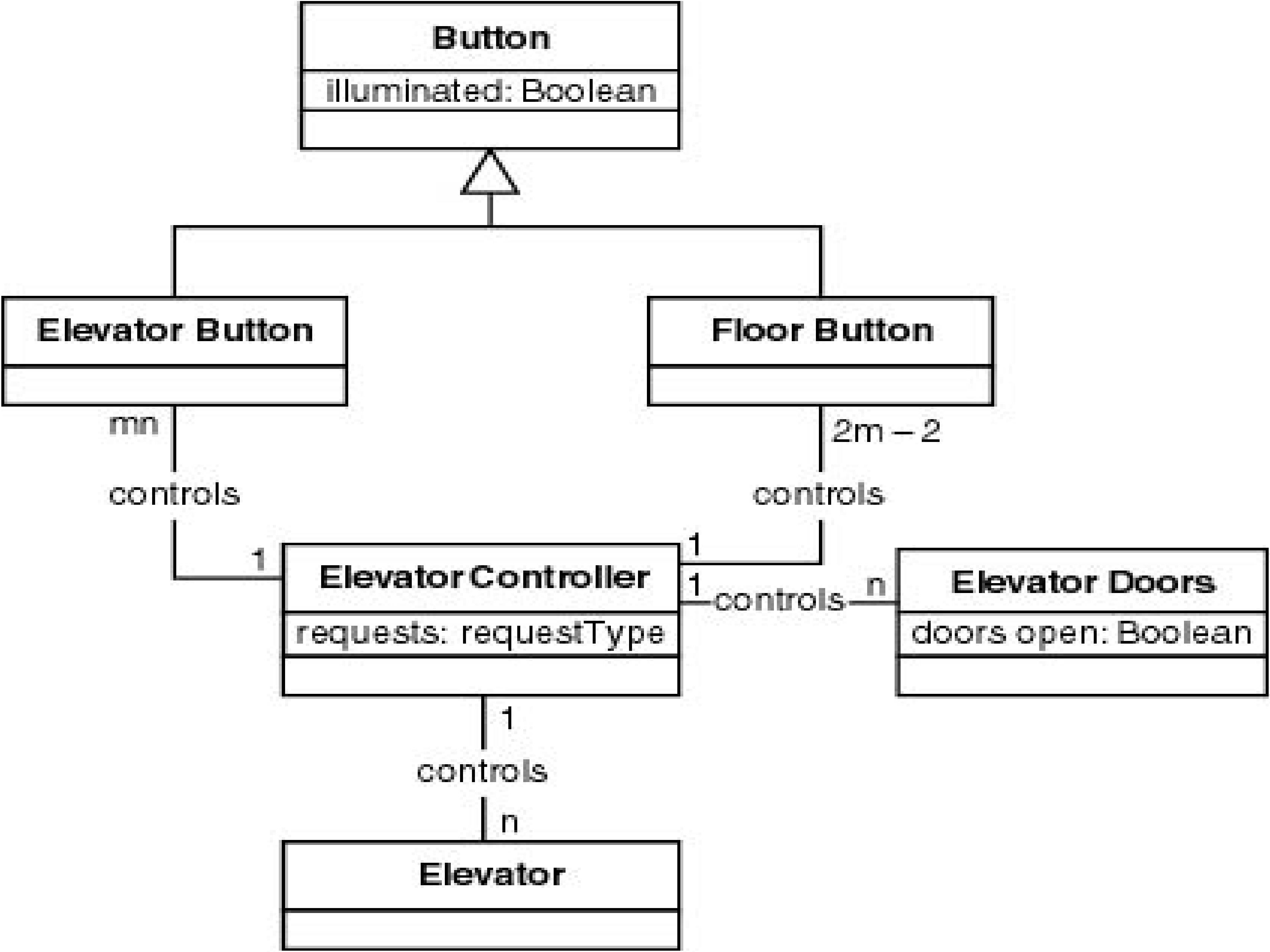
Elevator Controller

RESPONSIBILITY

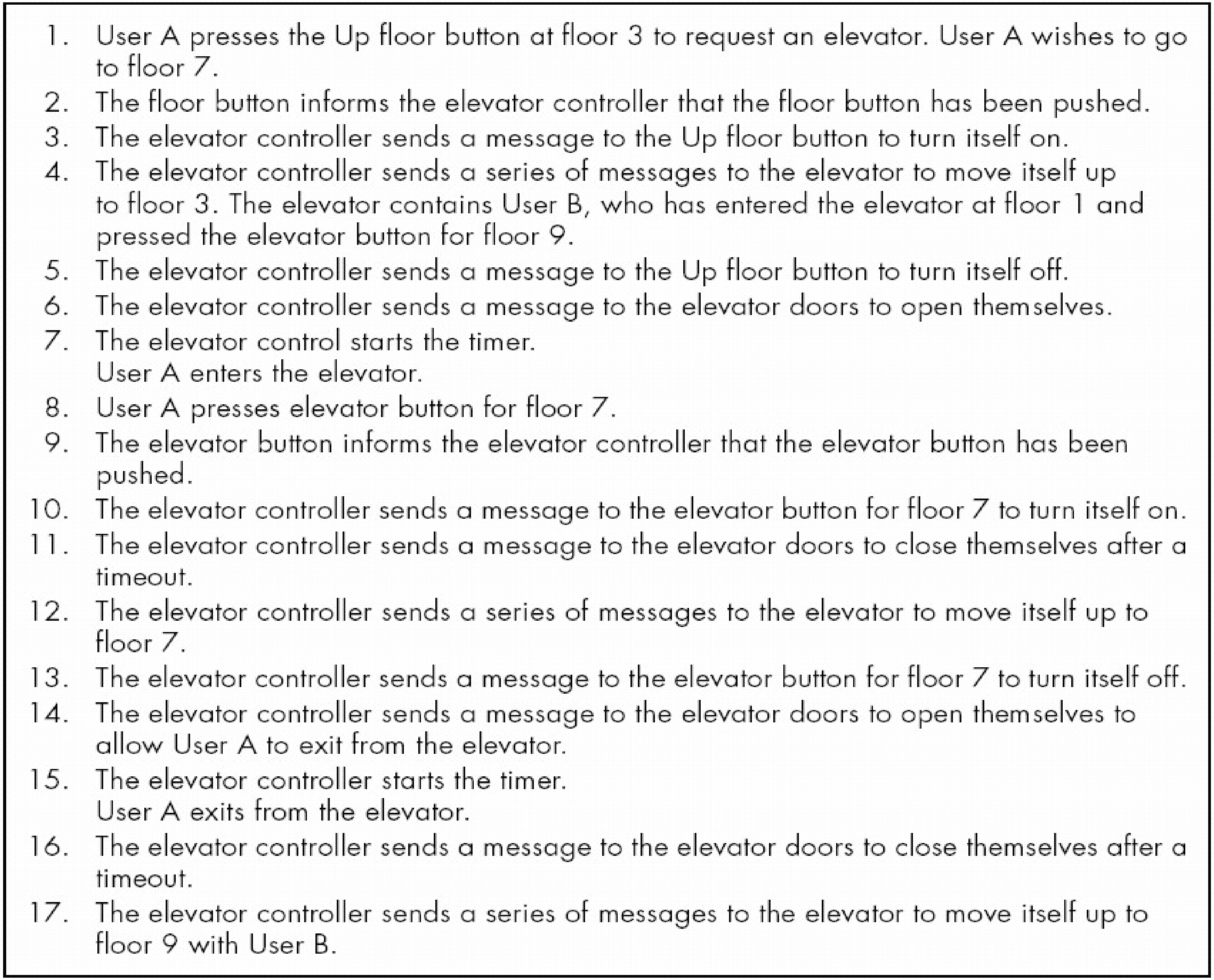
1. Send message to **Elevator Button** to turn on button
2. Send message to **Elevator Button** to turn off button
3. Send message to **Floor Button** to turn on button
4. Send message to **Floor Button** to turn off button
5. Send message to **Elevator** to move up one floor
6. Send message to **Elevator**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

COLLABORATION

1. Subclass **Elevator Button**
2. Subclass **Floor Button**
3. Class **Elevator Doors**
4. Class **Elevator**


Button
illuminated: Boolean
Elevator Button
Floor Button
mn
controls
1
2m – 2
controls
1
Elevator Controller
requests: requestType
1
controls
n
Elevator Doors
doors open: Boolean
1
controls
n
Elevator

Second Iteration of 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 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.

Elevator Problem: OOA (contd)

- **All three models are now fine.**
- **We should rather say:**
  - **All three models are fine *for now***
- **We may need to return to the object-oriented analysis phase during the object-oriented design phase.**

Why Is All This Iteration Needed?

- Iteration is an intrinsic property of all software production
  - Especially for medium- and large-scale products
  - Iteration is expected in the object-oriented paradigm

5. Challenges of the OOA Phase

- **Do not consider *boundary* and *control* classes for OOA.**

Thank You !