Lessons 20, 23 & 24: Object-Oriented Design

October 2, 5, 7, 2015

Data and Actions

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

* Object-Oriented Design

- Classes are extracted during the object-oriented analysis workflow, and
  - Designed during the design workflow

- Accordingly
  - Classical **architectural** design corresponds to part of the object-oriented _____workflow

  - Classical **detailed** design corresponds to part of the object-oriented _____workflow

*Object-Oriented Design

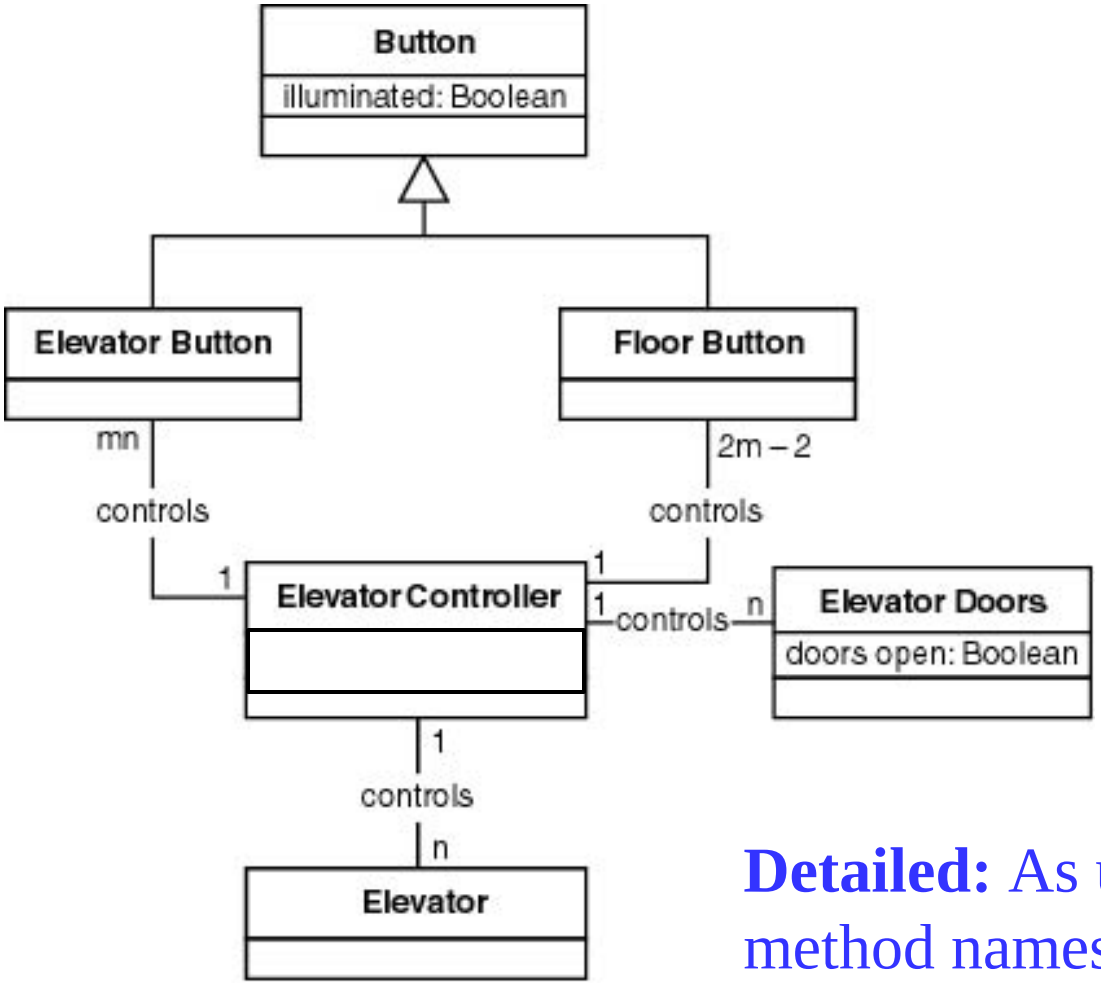
- **Goal:** Design product in terms of objects identified during OOA

- OOD consists of two key steps:
  1. Complete the detailed class diagram
  2. Perform the detailed design

*Object-Oriented Design

- Completing the detailed class diagram consists of:
  1. Determining the formats of the attributes
     a. Formats of attributes generally deduced directly from analysis artifacts
  2. Determining the methods and assigning them to the relevant classes
     a. Methods determined by constructing interaction diagrams for each scenario. Two types:
        i. Collaboration diagrams (Spatial) and
        ii. Sequence Diagrams (Temporal)
     b. Methods assigned to classes by designing the product in terms of clients of objects

*Determining Class' Attributes



Detailed: As used here means that all method names, and (as a by-product) attributes ID'd by designer, are represented on the diagram

Interaction Diagrams

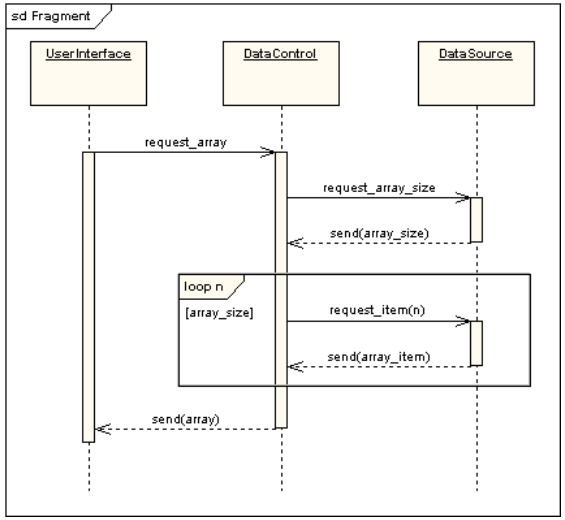
- Construct interaction diagrams for *each* scenario; focus is on objects and the *messages passed* between them.
  - "Passing a message" between objects really means having one object invoke a public method of another object.

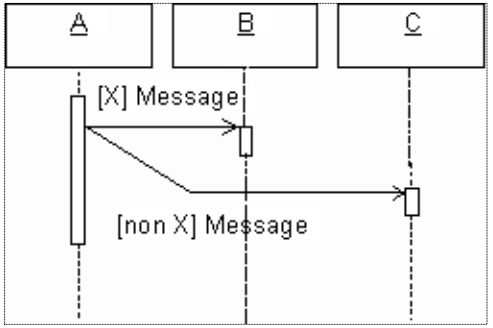
*Collaboration Diagrams: *spatial* representation

- Focus is on the *relationships* between objects
  - Structure becomes clear
  - Timing info is obscure, messages numbered to indicate sending order

*Sequence Diagrams: *temporal* representation

- Focus is on message broadcast chronology of interactions between objects.





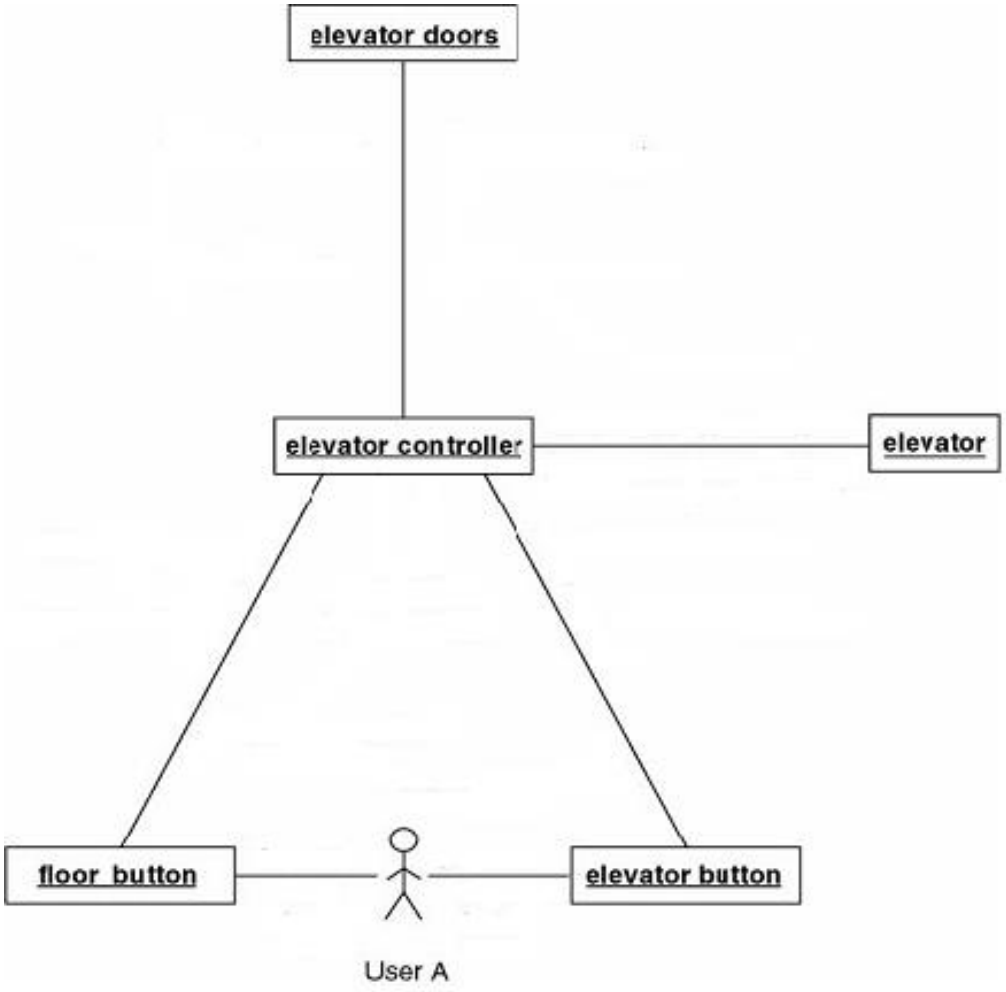
Use Scenarios to Create Interaction Diagrams

Example of a Normal Use 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.

Collaboration Diagram

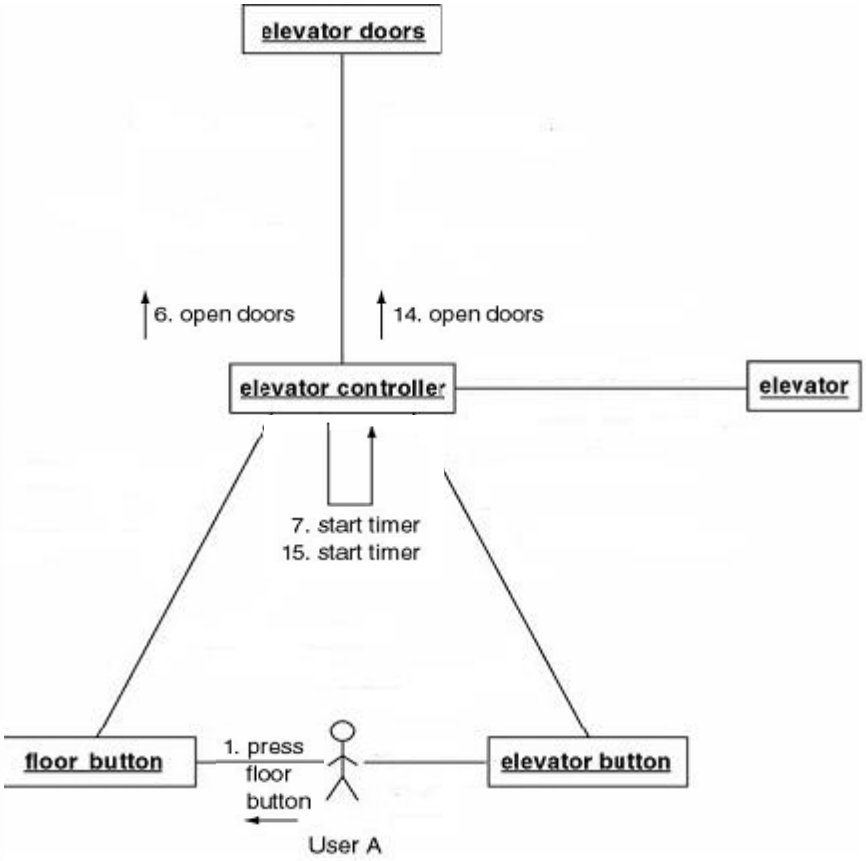
Start with diagram of objects and actors



*Collaboration Diagram

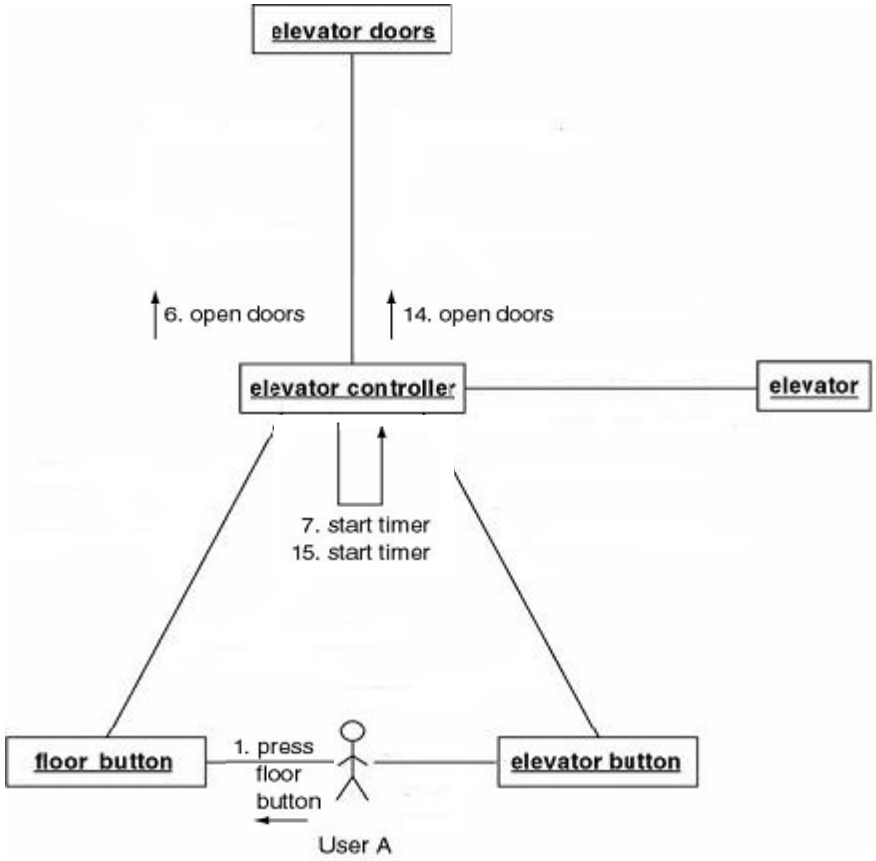
Use scenario to identify messages passed among objects

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.

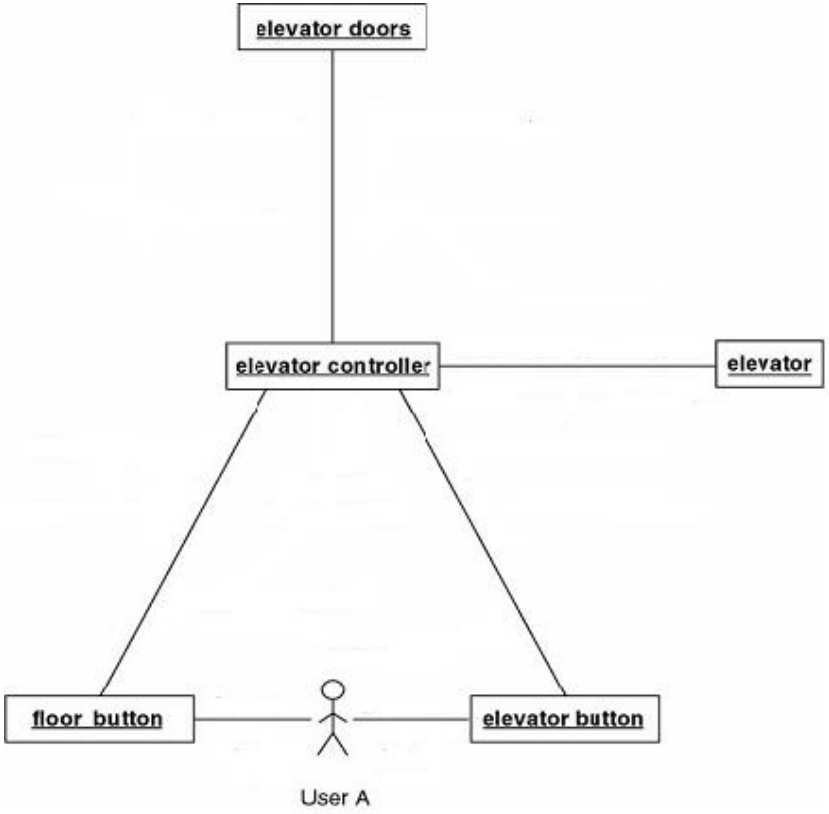


*ICE- Complete Elevator Collaboration Diagram

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.



*Equivalent Sequence Diagram



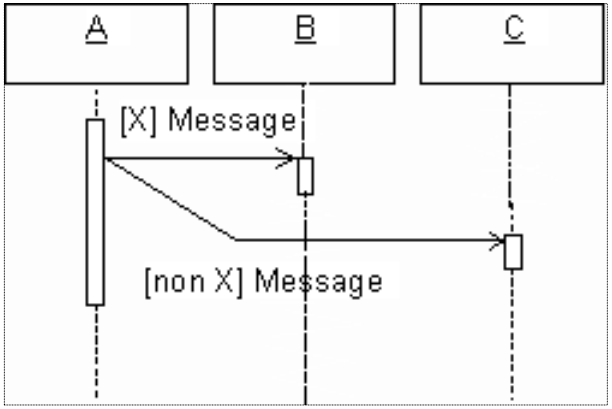
Homework- Interaction Diagrams

Produce an interaction diagram (your choice of sequence or collaboration) for the Apartment Automatic Garage Door
Scenario 1: "User swipes card and drives car into garage."

Scenario 1
1. Driver A swipes card. 2. If door is open, GarageDoorController resets timer. 3. If door is closed, GarageDoorController sends msg to Garage Door to open. 4. When door is fully opened, DoorOpenSensor notifies GarageDoorController. 5. GarageDoorController resets timer. 6. Timer times out; GarageDoorController notified. 7. GarageDoorController sends msg to GarageDoor to close. 8. When door is fully closed, DoorClosedSensor notifies Garage DoorController.

*Assigning Methods to Classes

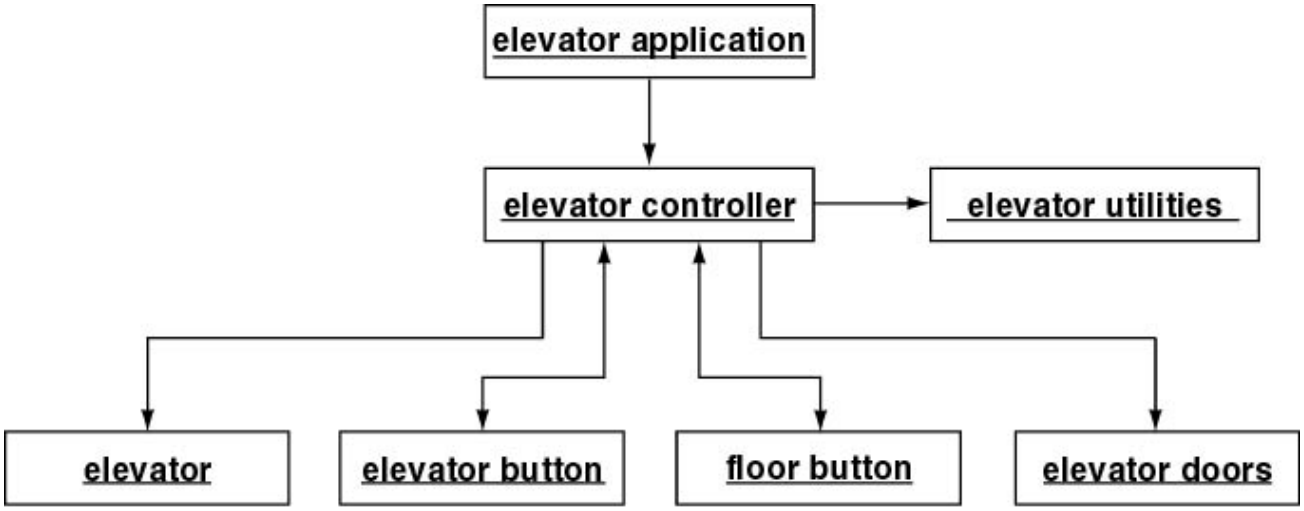
- A well-structured Interaction Diagram gives very specific information on what responsibilities a class should have.
- Main question, do we assign an action to a class or to a client of that class?



  - Examples
    - `close doors` assigned to?
    - `move one floor down` assigned to?

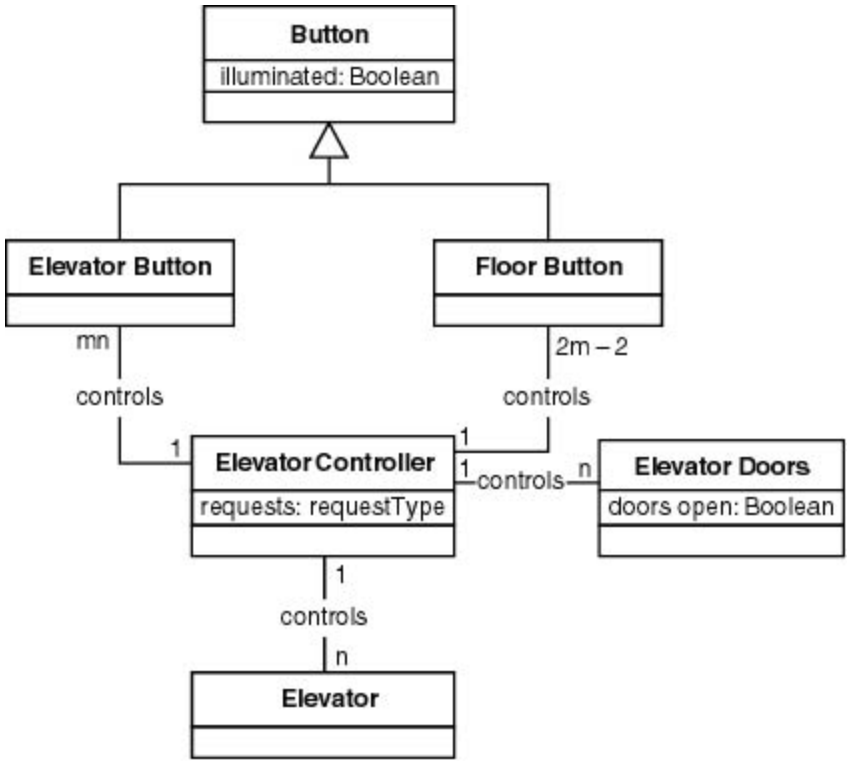
*Determine Clients of Objects

- Design Product in Terms of Clients of Objects to determine which objects should create other objects
  - Draw a single arrow from each object to a client of that object
  - Objects that are not clients of any object have to be initiated, probably upon system launch, Additional methods may be needed.



ICE- Detailed Class Diagrams

ICE: Complete the **Detailed Class Diagram** for the Elevator, showing all method names and attributes identified by the designer for each class.



Homework- Detailed Class Diagrams

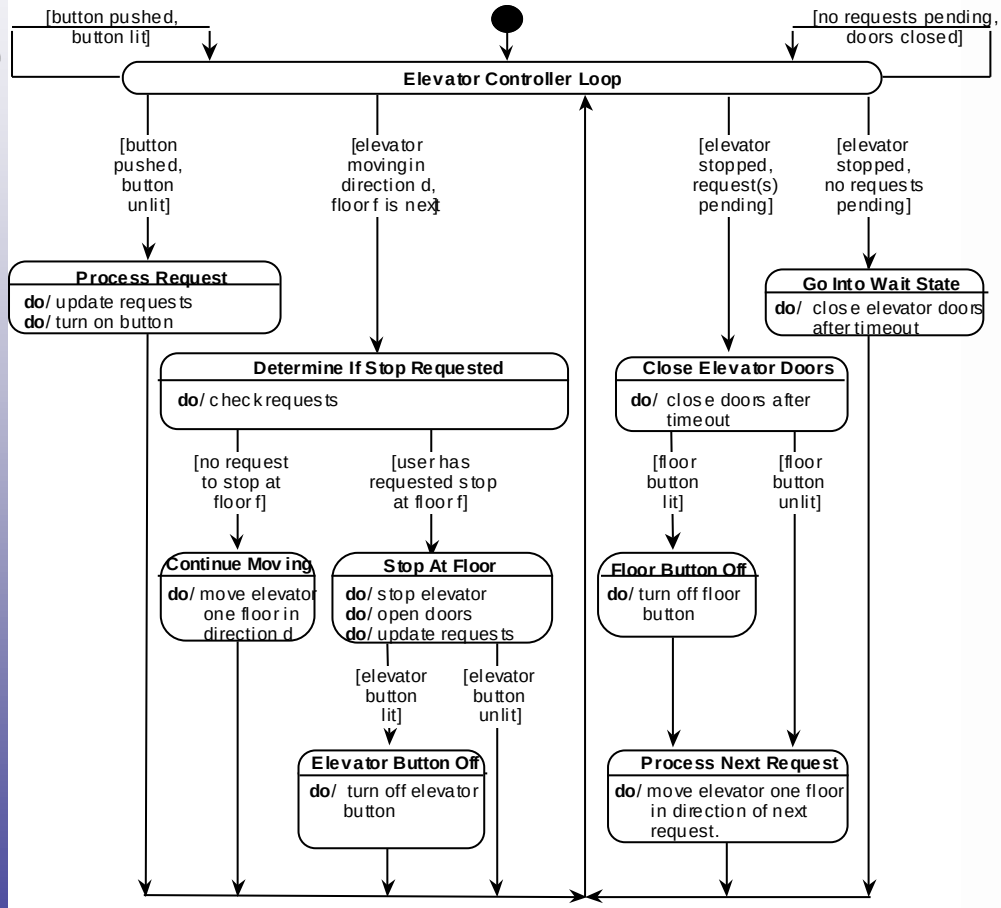
Homework: Produce a **Detailed Class Diagram** for the Apartment Automatic Garage Door, showing all method names and attributes identified by the designer.

Scenario 1	
1.	Driver A swipes card.
2.	If door is open, GarageDoorController resets timer.
3.	If door is closed, GarageDoorController sends msg to Garage Door to open.
4.	When door is fully opened, DoorOpenSensor notifies GarageDoorController.
5.	GarageDoorController resets timer.
6.	DoorCloseTimer times out; GarageDoorController notified.
7.	GarageDoorController sends msg to GarageDoor to close.
8.	When door is fully closed, DoorClosed Sensor notifies GarageDoorController

*Step 2 of OOD: Perform Detailed Design

- Detailed design of method elevator event loop, draws heavily from prior UML diagrams. Design Team:
  - Should not do too much:
    - Detailed design should not become complete code
  - Should not do too little:
    - It is essential for detailed design to fully indicate how all functionality is met.
- So, what should detailed design's output be?

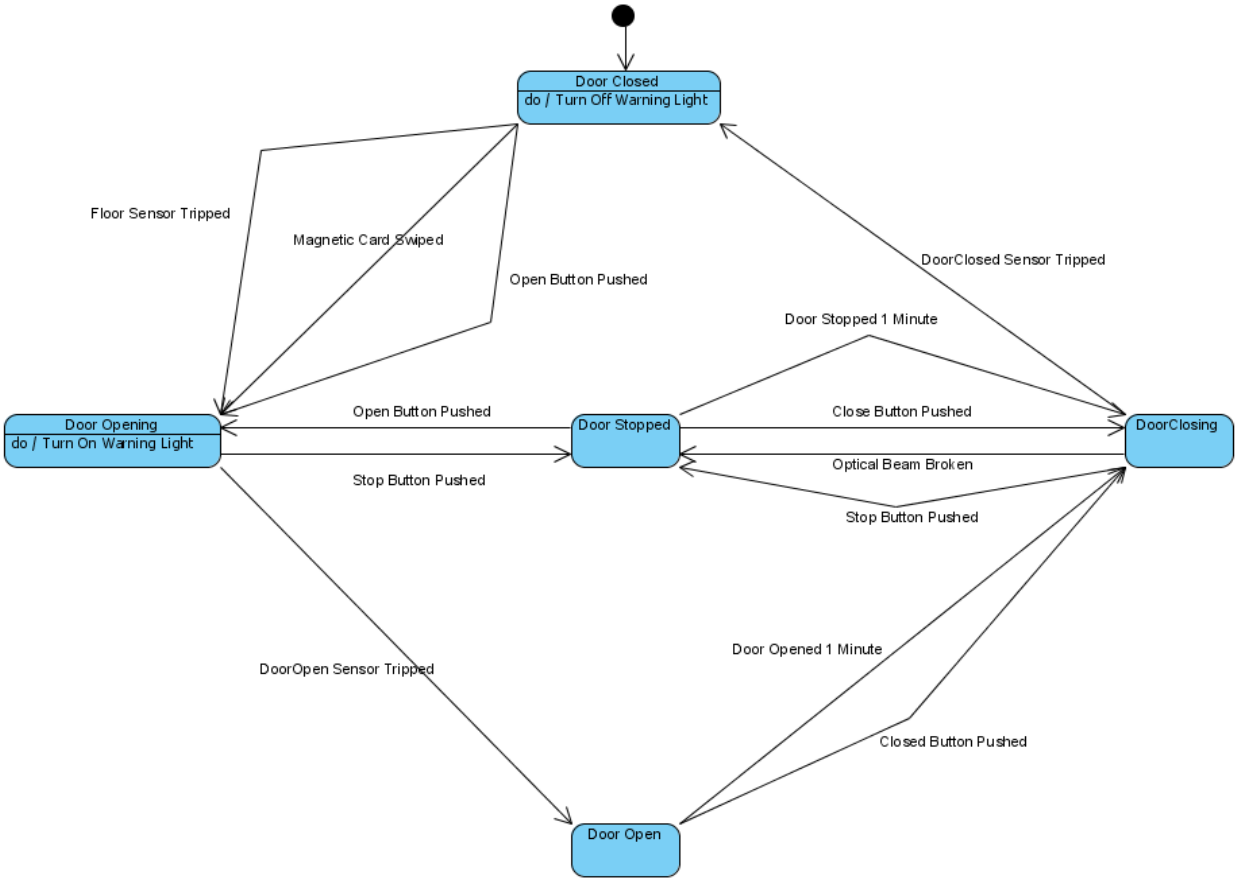
*Step 2 of OOD: Perform Detailed Design




```
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;
  }
}
```

ICE/Homework- Detailed Design

ICE/Homework: Produce a **Detailed Design** for method garageDoorControlLoop in class GarageDoorController for the Apartment Automatic Garage Door, using Java-like pseudo-code.



Real-Time Design Techniques

- Difficulties associated with real-time systems
  - Inputs come from the real world
    - Software has no control over the timing of the 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?

* Design Testing

- Design reviews are essential
  - A client representative is not usually present (except in DoD procurements)
- Types of faults uncovered:

*Testing During the Design Workflow

- Goal of Design Testing- answer the following:
  - Is the specification accurately and completely incorporated into the design?
  - Is the design itself correct?
    - No logic faults
    - Interface for each class fully defined.
- Design reviews. Purpose is to show that Design correctly reflects the Specification. How can this be shown?

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
    - This should lead to fewer faults

*Case Tools for OOD

- UpperCASE tools
  - Examples of UpperCASE tools:
    - Software through Pictures,
    - Visual Paradigm
    - Rational Rose
  - Built around data dictionary
  - Screen, report generators
  - What automated support could be provided by a case tool that represents OOD using UML?

*Metrics for OOD

- What Metrics can be used to describe various aspects of the Design?
- One is McCabe's Cyclomatic complexity. Measure based on # of control flow decisions, can apply to pseudo-code.
  - the number of binary decisions plus 1,
  - a metric of design quality, the lower the M the better
- What other aspects of a Design can be Measured?

Recap: Relationships Between UML Diagrams