Lessons 18, 19, 22, & 23: Object-Oriented Analysis

September 21, 23, 28, 30, 2015

The Analysis Workflow

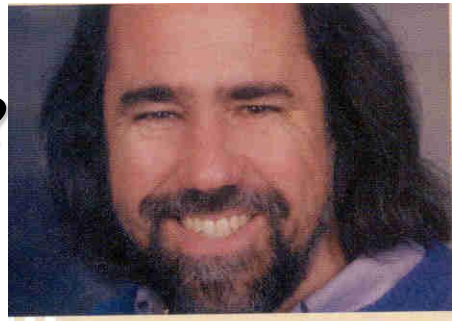
- The analysis workflow has two aims:

Object-Oriented Analysis (OOA) (Schach Ch 13)

- OOA Provides Counterpart to "Structured" Portions of the Classical Specification Phase: specifically, ER diagrams, DFDs, and FSMs.
- OOA: A *Semi-formal* Specification Technique, Loosely based on the Above.

*Many Different OO Methods

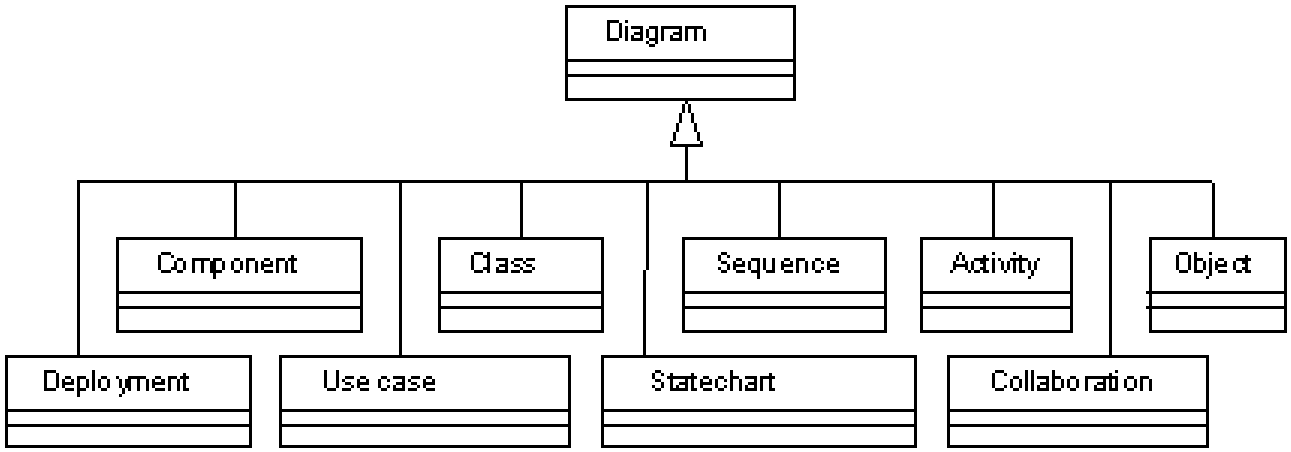
- Initially, Many Different "Methods" Emerged (Booch, OMT, Shlaer-Mellor, Coad-Yourdon) — all essentially equivalent.
- So, Just what *is* a "Method" in this context?



Grady Booch

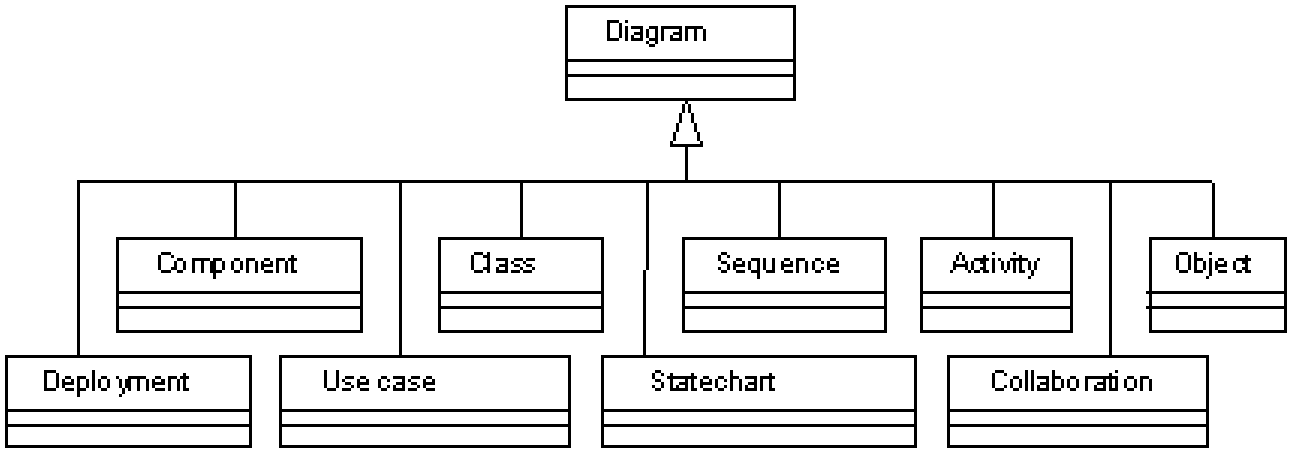
- In 1994, Booch, Rumbaugh and Jacobson combined their Methods into the **Unified Process,** which uses **UML** (Unified Modeling Language) to record results
- UML Emerged as a *defacto* standard; uses a Common Notation for representing OOA & OOD.

*Different Types of Diagrams Defined by UML



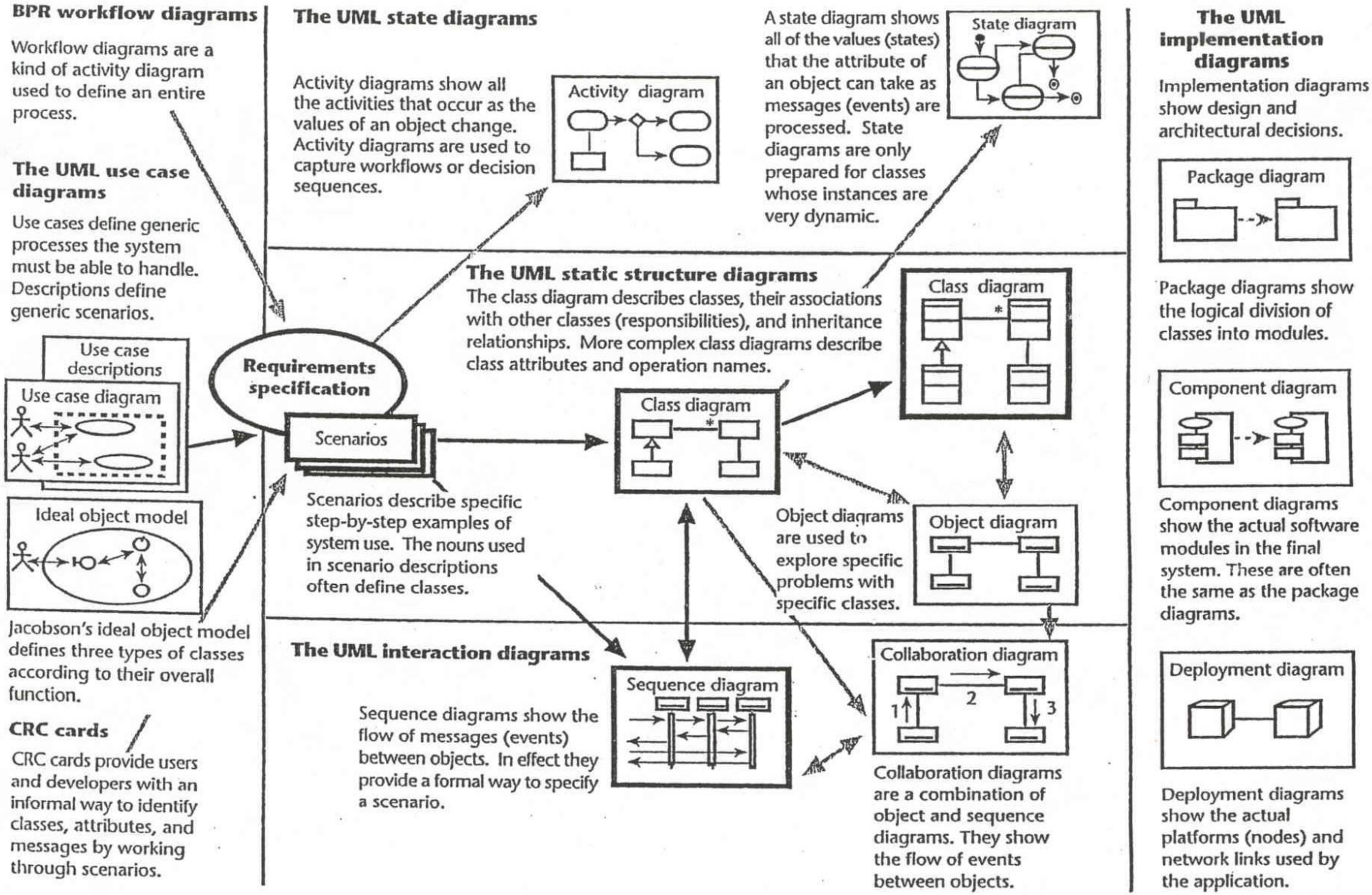
- **Use case diagrams**
- **Class diagrams**
- **Interaction diagrams:**
- **Statechart diagrams:**

*UML Diagrams (cont.)



- Additional Diagrams (we won't use these in this course):

Relationships Between UML Diagrams



*OOA Consists of Three Basic Steps:

1. **Functional modeling** (Mostly Action-Oriented)

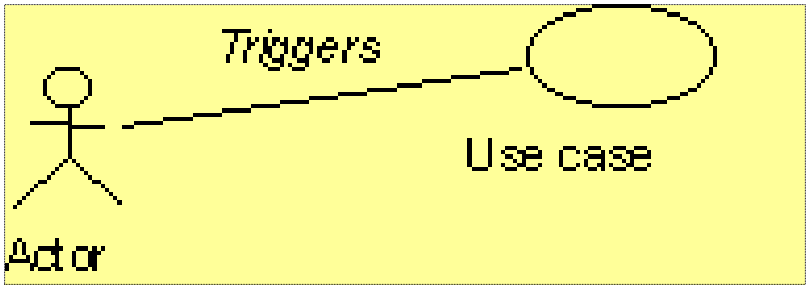
2. **Class Modeling** ("Object Modeling") (Purely Data Oriented)

3. **Dynamic Modeling** (Purely Action-Oriented)

- Note: OOA is Iterative, above Steps Repeatedly Revisited

*Functional Modeling (Step 1 of OOA)

- Use Cases Model *Intended* Behavior of System, without concern for how the Behavior will be *Implemented*.
- A Use Case Carries out Tangible Work of Value from the Perspective of an *Actor*. Examples:
  - Calculate a Result,
  - Generate a New Object, or
  - Change the State of another Object



- UML Notation Allows Visualization of a Use Case Apart from its Realization and in Context with other Use Cases.
- An Actor's role is to trigger (communicate with) a use case.

* Elevator Problem: OOA

Step I of OOA: Functional Modeling

- *Use Case*: Generic Description of Overall Functionality
- *Scenario*: Instance of a Use Case
  - Consider Typical Scenarios of Activities of each Class
  - *Goal:* Obtain Insight into Product Behavior

- Example: Consider an elevator control system that controls a bank of elevators in a high-rise.
  - What Actors and Use-Cases are relevant to the system?
  - As a starting point, think of various scenarios of use.

.

*Scenarios in Terms of UML

- A Scenario is an *Instance* of a Use Case, Explores Behavior Likely to go on "Behind the Scene". Considers:
  - **Normal** uses of System:
    - User Wants to use Elevator to go from Floor 3 to Floor 7, Presses "Up" Button. Elevator is currently empty.
  - **Abnormal** uses of System:
    - User "A" Wants to go from Floor 3 to Floor 7, but Presses Down Button. Elevator Already Contains User "B" who Entered at Floor 9 and is Going to Floor 1.

~*Normal Scenario Go from Floor 3 to Floor 7, Press "Up" Button

1. User presses **Up** floor button at floor 3 to request elevator. User wishes to go to floor 7.
2. **Up** floor button is turned on.
3. Elevator arrives at floor 3, **Up** floor button is turned off.
4. Elevator doors open. Timer starts. User enters elevator.
5. User presses elevator button for **floor 7**.
6. **Floor 7** elevator button is turned on.
7. Elevator doors close after timeout.
8. Elevator travels to **floor 7**.
9. **Floor 7** elevator button is turned off.
10. Elevator doors open to allow User to exit elevator.
11. Timer starts. User exits.
12. Elevator doors close after timeout.

~*ICE: Abnormal Scenario

- User "A" Wants to go from Floor 3 to Floor 7, but Presses Down Button. Elevator Already Contains User "B" who Entered at Floor 9 and is Going to Floor 1. User "A" enters “Down” elevator when it stops. Develop the Scenario.

*The Role of Use-Cases and Scenarios

- The scope of use cases and scenarios goes far beyond solely defining system requirements; also allows:

- Use-Cases and Scenarios are used, albeit for different purposes, during each workflow of object-oriented software development.

Specification/Analysis **Design** **Integration**

*Aside: Walkthroughs

Technique Used to Uncover Application's Desired Behavior

- Pretend You Already Have a Working Application, Walk Through the Various Uses of the System
- Walkthroughs Help To Uncover All *Intended* Uses of a System
- **Question:** When do you stop walking through scenarios?

~Example: Apartment Automatic Garage Door

Description:

The door of the garage in an apartment building basement is opened from the outside by means of a magnetic card. From the inside it is opened by a floor sensor tripped by an exiting car. There is also a manual unit with open, close, and stop buttons inside the garage. Sensors in the door frame are tripped when the door reaches the fully closed and fully opened positions. An optical sensor mounted on the door frame ensures that the doorway is free from obstacles, stopping the door from closing if the light beam is broken. When the door has been either fully opened or stopped for 1 minute, it automatically starts moving down, provided there is no obstacle.

Homework: Define the use cases for the Apartment Automatic Garage Door and develop the following scenarios for the above system.

Scenario 1: User swipes card and drives car into garage.

Scenario 2: Door begins to close one minute after car enters the garage, when another car attempts to exit the garage as the door is coming down.

*Step II of OOA: Class Modeling

Goal: Extract classes /Attributes, & represent Relationships (including Inheritance) between Classes (*Similar to ER*).

- Various Approaches:
  - Deduce Classes from Use Cases and their Scenarios
  - Noun extraction
  - CRC cards

Class Modeling

- There are three types of classes:

  - Entity classes
  - Boundary classes
  - Control classes

Noun Extraction Approach to Class Modeling

- For Developers Without Domain Experience
- Consists of Five Stages from highly to less Abstract:
  - Stage 1: Concise Definition
  - Stage 2: Include Constraints
  - Stage 3: Identify Nouns
  - Stage 4: Exclude Outliers
  - Stage 5: Identify Classes

*Stage 1 of Noun Extraction: **Concise Definition**

Stage 1 of Noun Extraction: **Concise** Definition

- Define Product as Concisely as Possible (in Single Sentence if you can!)

```
Buttons in elevators and on floors
are used to control motion of n
elevators in building with m floors.
```

*Stage 2 of Noun Extraction: **Include Constraints**

- Incorporate Constraints into Stage 1
- Express Result in a Single Paragraph (preferably)

```
Elevators have floor buttons and
elevator buttons that control movement
of n elevators in building with m
floors.  Buttons illuminate when
pressed by user to request elevator to
stop at specific floor; illumination is
canceled when request has been
satisfied.  If elevator has no
requests, it remains at its current
floor with its doors closed.
```

*Stage 3 of Noun Extraction: **Identify Nouns**

- Identify Nouns in Informal Strategy for use as Candidate Classes:
  - Nouns from stage 3:

```
Elevators have floor buttons and
elevator buttons that control movement
of n elevators in building with m
floors.  Buttons illuminate when
pressed by user to request elevator to
stop at specific floor; illumination is
canceled when request has been
satisfied.  If elevator has no
requests, it remains at its current
floor with its doors closed.
```

*Stage 4 of Noun Extraction: **Exclude Outliers**

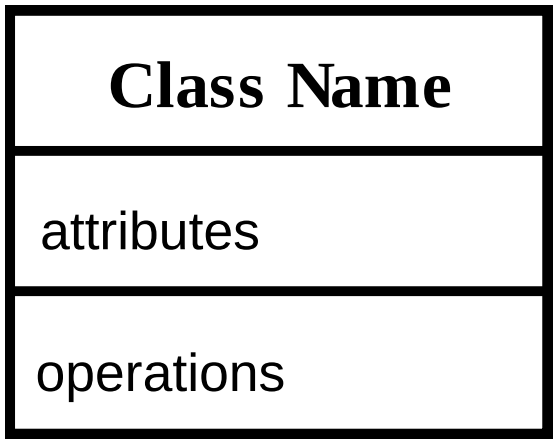
- Exclude nouns that are outside problem boundary, and abstract nouns (abstract nouns may become attributes)

Elevators have floor buttons and elevator buttons that control movement of n elevators in building with m floors. Buttons illuminate when pressed by user to request elevator to stop at specific floor; illumination is canceled when request has been satisfied. If elevator has no requests, it remains at its current floor with its doors closed.

*Stage 5 of Noun Extraction: **Identify Classes**

- Candidate classes: **Elevator**, and **Button**
- Subclasses: **Elevator Button** and **Floor Button**

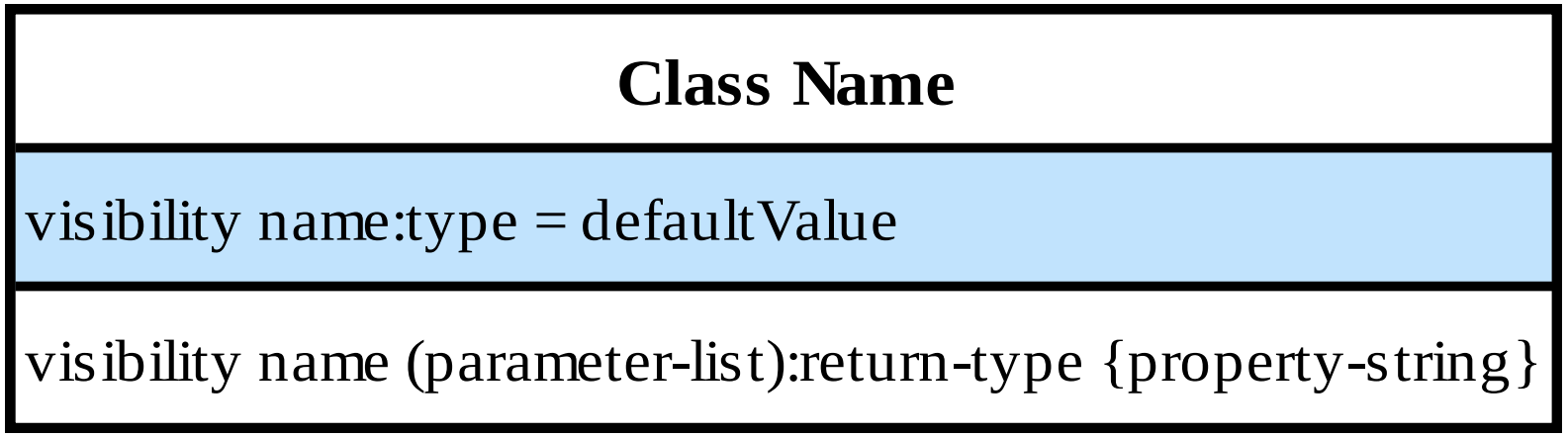
Aside: UML Class Diagram- Basic



attributes: capture characteristics of entity being modeled

operations: capture behavior of entity being modeled

UML Class Diagram- Expanded



UML attribute syntax:

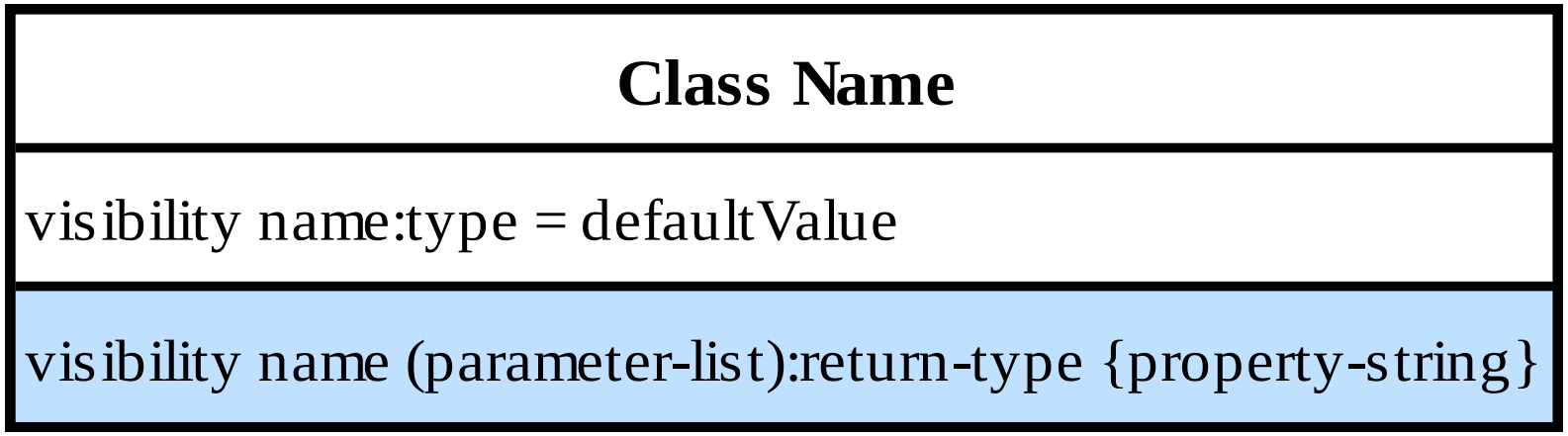
visibility: + (public), - (private), or # (protected)

name: name of the attribute

type: data type of the attribute

defaultValue: an initial value for the attribute

UML Class Diagram- Expanded



UML operation syntax:

visibility: + (public), - (private), or # (protected)

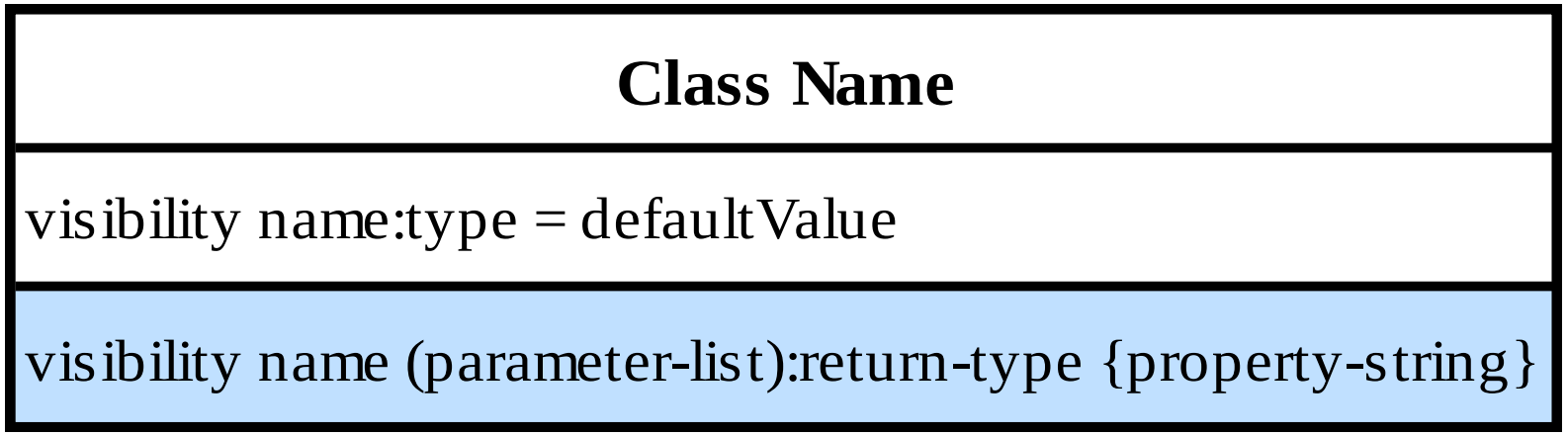
name: name of the operation

parameter-list: comma-separated list of function parameters

return-type: data type of the result of the operation

property-string: property values that apply to the operation

UML Class Diagram- Expanded



parameter-list syntax: direction name: type = defaultValue

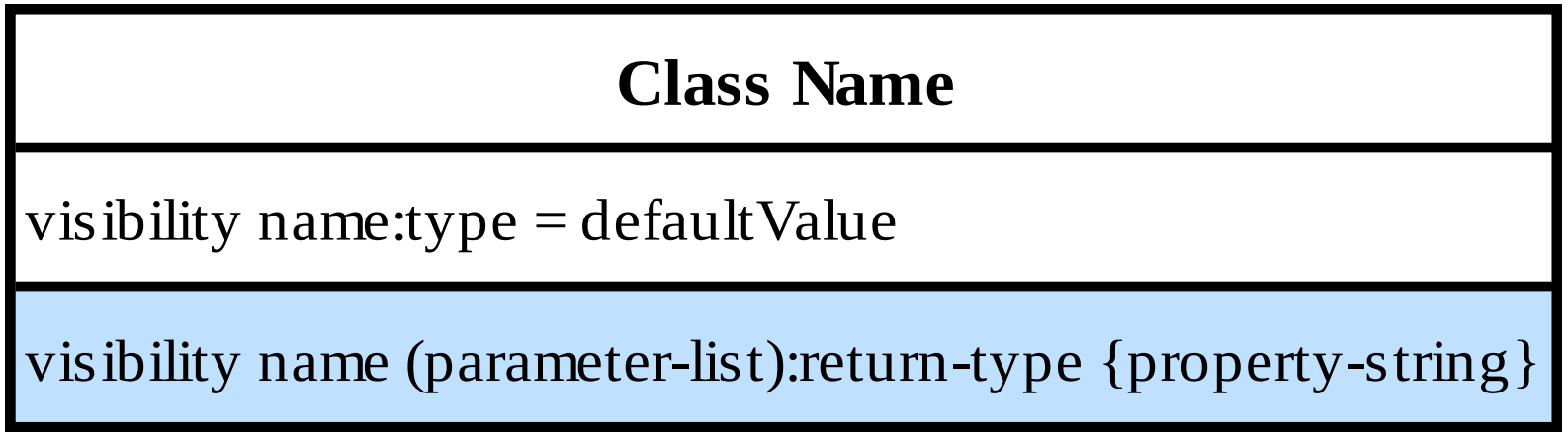
direction: used to show whether parameter is used for input (in), output (out), or both (inout)

name: name of the parameter

type: data type of the parameter

defaultValue: value to be used for the parameter if no argument is provided

UML Class Diagram- Expanded



property-string:

query: indicates that the operation doesn't modify any data in the class

UML notation for Is-A, Has-A, Association

- Inheritance (Is-A) represented by a large open triangle

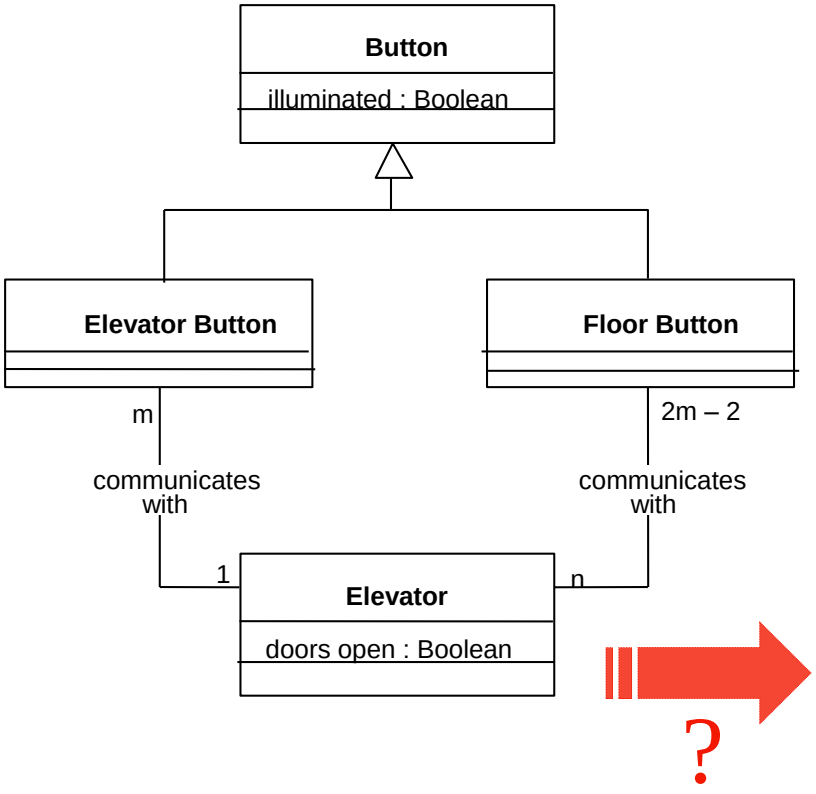
*UML Notation (cont.)

- Aggregation (Has-A)

- Association (Similar to ER relationships)

*First Iteration of Elevator System Class Diagram

*Second Iteration Of Class Diagram



Example: Modify Class Diagram by Adding a Controller Class to determine which elevator is sent to a Floor Button Request.

Point-> OOA is an intentionally iterative process.

~Example: Apartment Automatic Garage Door

Description:

The door of the garage in an apartment building basement is opened from the outside by means of a magnetic card. From the inside it is opened by a floor sensor tripped by an exiting car. There is also a manual unit with open, close, and stop buttons inside the garage. Sensors in the door frame are tripped when the door reaches the fully closed and fully opened positions. An optical sensor mounted on the door frame ensures that the doorway is free from obstacles, stopping the door from closing if the light beam is broken. When the door has been either fully opened or stopped for 1 minute, it automatically starts moving down, provided there is no obstacle.

HOMEWORK- Using the above informal strategy with constraints (Noun Extraction Stage 2) description of our Apartment Automatic Garage Door, complete the Noun Extraction process to create an initial Class Diagram.

- **Stage 3 (Formalize the Strategy) - Identify the nouns**
- **Stage 4 - Remove abstract nouns and those outside the problem boundary**
- **Stage 5 step 1 - Produce the first cut of a UML class diagram**
- **Stage 5 step 2- See if all the scenarios are met.**

*CRC Card Approach to Class Modeling

Developed by RebeccaWirfs-Brock: For each class, fill in card showing:

- Name of **C**lass
- Public Behavior class is responsible for (**R**esponsibility)
- List of Classes it Uses or is Used by (**C**ollaboration)

- *Strengths*: Teams can use to "Act Out" Scenarios,
  - Useful for highlighting Missing/Incorrect Components
  - Inexpensive, Erasable, Physical
  - An Excellent way to complete the OOA Class Modeling Process
- *Weakness*: Most Useful if Designer has Domain Expertise

*CRC Card for Class Elevator Controller

- Walk Through Use-Case Modeling Scenarios, Show How CRC Cards Handle User's Needs
  - Note both Uses and Used By areas of Collaborators Section
  - Class meets responsibilities either through its methods, or by invoking methods of one of its "Uses" collaborators

CLASS **Elevator Controller**
RESPONSIBILITY Turn on elevator button Turn off elevator button Turn on/off floor button Open elevator doors Close elevator doors Move elevator to floor(int)
COLLABORATION Uses Class **Elevator Button** Class **Floor Button** Class **Elevator** Used By (none)

*Testing During the OOA Phase

- CRC Card Reviews are a useful OOA Testing Technique; Revisit **Elevator Controller** Class's 1st Responsibility:

  "Turn on elevator button" From an OOA standpoint, do you see any problem with this?

CLASS **Elevator Controller**
RESPONSIBILITY Turn on elevator button Turn off elevator button Turn on/off floor button Open elevator doors Close elevator doors Move elevator one floor up Move elevator one floor down
COLLABORATION Uses Class **Elevator Button** Class **Floor Button** Class **Elevator** Used By (none)

*Another Problem with Elevator CRC Cards

- Another Problem: a Class has Been Overlooked.
- Concept Of *State* is Important. If an Item's State Changes During Execution, Item Likely Should be a Class. Are there any items that we haven't discussed whose state changes during execution?

- OOA is an *Iterative* Process: May Also Need to Return to Object-Oriented Analysis Phase during Object-Oriented Design Phase

Homework- Apartment Automatic Garage Door

- **Homework**: Develop CRC cards for Apartment Automatic Garage Door Example. Also, do a scenario walkthrough to root out missing responsibility

Description:

The door of the garage in an apartment building basement is opened from the outside by means of a magnetic card. From the inside it is opened by a floor sensor tripped by an exiting car. There is also a manual unit with open, close, and stop buttons inside the garage. Sensors in the door frame are tripped when the door reaches the fully closed and fully opened positions. An optical sensor mounted on the door frame ensures that the doorway is free from obstacles, stopping the door from closing if the light beam is broken. When the door has been either fully opened or stopped for 1 minute, it automatically starts moving down, provided there is no obstacle.

*Is All This Iteration Really Needed?

- All software development models include iteration.
  - Waterfall Model
  - Rapid Prototyping Model
  - Incremental Model, Iterative and Incremental Model, and
  - Spiral Model

- Is iteration *Intrinsic* or *Extrinsic* to the Software Development Problem?

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

Extracting the Boundary and Control Classes

- For an Elevator simulation (as opposed to actual Elevator system), could add boundary classes to simulate Floor and Elevator controls/displays
  - Boundary classes for *FloorControls* and *ElevatorControls* used to provide GUI to simulate Floor and Elevator controls
- Control class *DoorSafetyControls* could be used to perform checks/calculations to ensure safe operation of the elevator doors during:
  - Personnel entry and exit (infrared sensor)
  - Fire or other emergencies (prevent door from opening on floor where fire has been detected)

Step III of OOA: Dynamic Modeling

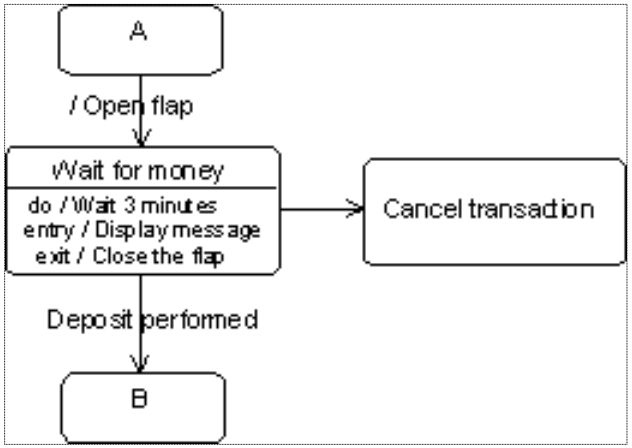
Goal: Produce a UML State Diagram for *Each* Class

- A UML State Diagram is Similar to a FSM, but is Semi-Formal
  - State, Actions, Predicate distributed over UML State Diagram
  - UML "guards" (Predicates) are in brackets
- State Diagram Shows which State is entered given a Starting State, and State Transitions that occur if Predicates Hold.
- State Diagrams help Determine Actions Performed by or to each Class. An *Action-Oriented* Step of OOA

Note: UML Reserved Word **do** indicates an Action that is to be Carried Out.

*Ex. A delay sequence in an ATM Deposit Scenario

- When the flap that accepts deposits is opened, the system tells users that they have three minutes to make their deposit.
  - If the deposit is not performed within the given time, the automatic transition towards the Cancellation state is triggered at the end of the delay activity.
  - Conversely, if deposit performed within 3 minutes, the delay activity is interrupted by the triggering of transition towards state B.
- In both cases, the flap is closed by the exit action of the delay state.



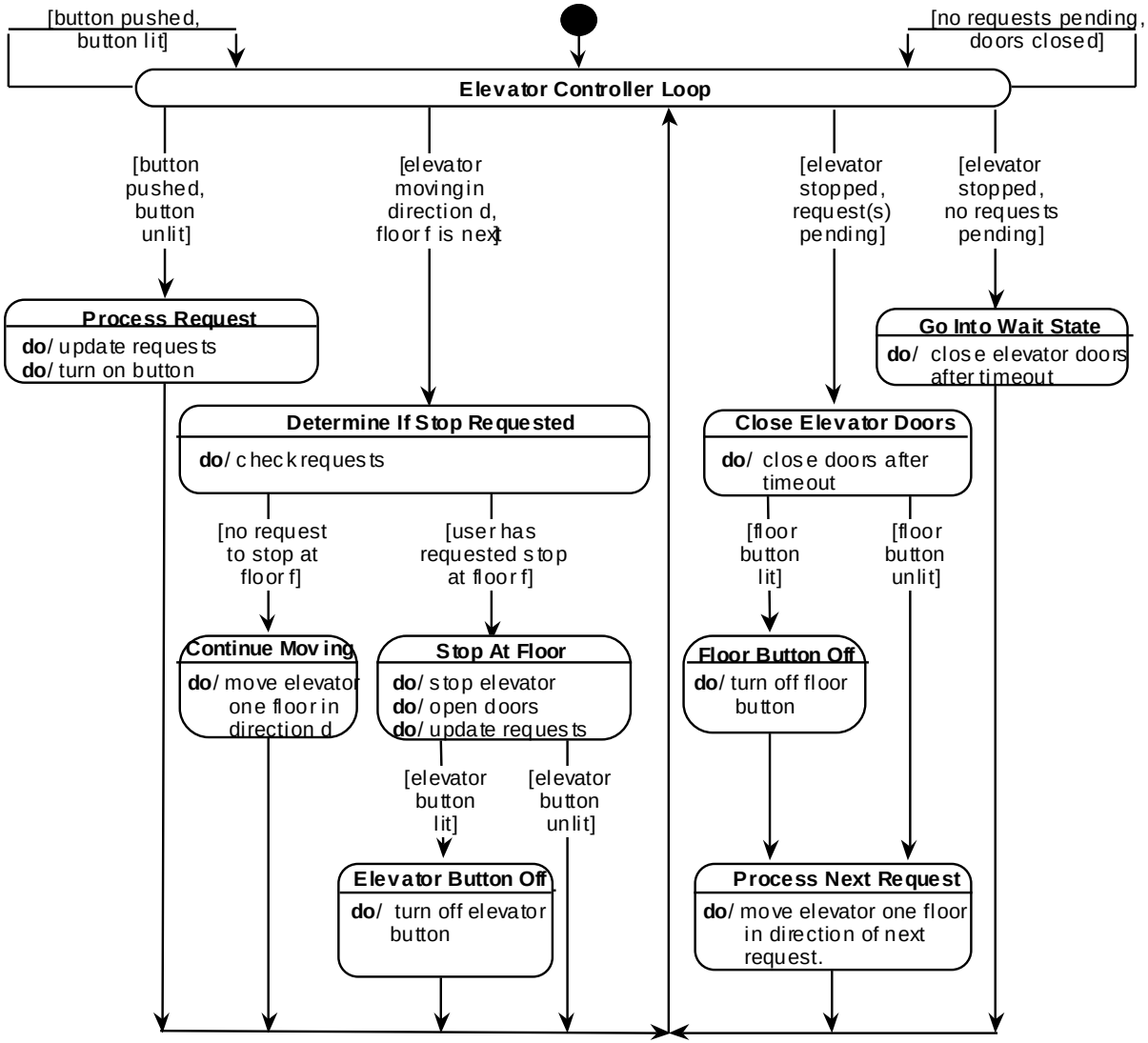
- Six Places for specifying Ops:

*The role of UML Guards and Operations

- A **guard** is a Boolean condition used to validate the triggering of an event occurrence.
- Guards support the determinism of a state machine, even when many transitions can be triggered by the same event.
  - When the event takes place, guards, which must be mutually exclusive, are evaluated, and then a transition is validated and triggered.

Example, guards make it possible to start the air conditioning *or* simply open the windows, according to season.

*UML Statechart Diagram



- **Homework:** Produce a UML Statechart for Apartment Automatic Garage Door Controller.

Transitioning To The OOD Phase

- Once Specification Contract is Approved (Signed), the following is Delivered to the Design Team:
  - Specification Document,
  - Use Case Scenarios,
  - UML Use-Case, Class, and State Diagrams,
  - CRC Cards

- Object-Oriented Design uses the above as the beginning of high level design.

Recap: Relationships Between UML Diagrams