INF 111 / CSE 121: Software Tools and Methods

Lecture Notes for Summer Quarter 2008

Michele Rousseau

Lecture Notes 7 - UML

Announcements

- **Quiz #3- Thursday – What will it cover?**
  - All readings assigned since the last quiz
  - Plus the readings not covered on the last quiz:
  - Through slide 54 in today's lecture
    - Configuration Management
    - Only through the first break!
- **Readings on UML**
  - Another book on UML:
    - McLaughlin, Pollice & West (2006). Head First Object-Oriented Analysis & Design. O'Reilly, 2006.
- **Assignments**
  - #1 has been graded
  - Assignment #3 will be posted later this week

Last Lecture

- **Quiz #2**
- **Configuration Management**

Today's Lecture

- **Finish up**
  - Configuration Management
    - Version Control
- **Modeling**
  - OOAD
    - UML
      - Class Diagrams
      - Use Case Diagrams
      - Sequence Diagrams

CASE tools for configuration management

- **CM processes are often standardised**
  - procedures are pre-defined
- **Lots of Docs and Data to be managed**
- **Tools make it possible**
- **Can be…**
  - Individual tools
  - Workbenches
  - Environments

CM workbenches

- **Open workbenches**
  - Tools for each stage in the CM process are integrated
    - Includes organizational procedures
- **Integrated workbenches**
  - Provide whole-process, integrated support
    - More tightly integrated tools → easier to use.
    - Less flexible in the tools used
      - Have to use tools built in

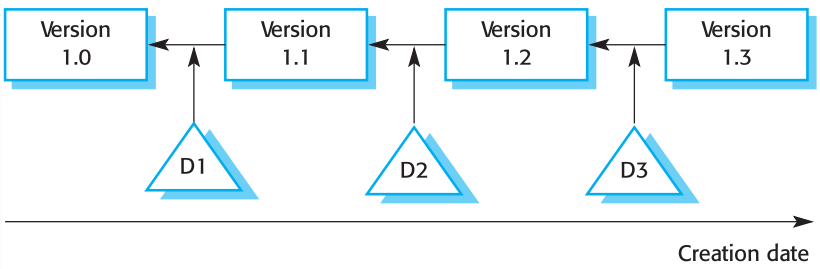
Change management tools

- **Change management is a procedural process**
  - it can be modelled & integrated with a version management system.
- **Change management tools**
  - Form editor
    - supports processing the CRFs
  - Workflow system
    - define who does what
    - automates information transfer;
  - Change database
    - manages change proposals
    - linked to a VM system
  - Change reporting system
    - generates management reports (CR status)

Version management tools

- **Version and release identification**
  - assigns identifiers automatically for each new version
- **Storage management.**
  - Stores the differences between versions (the delta)
    - rather than all the version code.
- **Change history recording**
  - Record reasons for version creation.
- **Independent development**
  - Only one version at a time may be checked out for change.
  - Parallel working on different versions.
- **Project support**
  - Manages groups of files associated with a project
    - rather than just single files.

Delta-based versioning



Moving on to OOAD

- **Object Oriented Analysis & Design (OOAD) using…**
  - UML – Part
    - Overview
    - More details in discussion

Brooks on Invisibility of Software

"Software is ***invisible*** and ***unvisualizable***. Geometric abstractions are powerful tools."

"As soon as we attempt to diagram software structure, we find it to constitute one, but several general directed graphs, superimposed on upon another. The several graphs may represent the flow of control, the flow of data, patterns of dependency, time sequence, name-space relationships. These are usually not even planar, much less hierarchical. Indeed***, one of the ways of establishing conceptual control over such structure is to enforce link cutting until one or more of the graphs becomes hierarchical***."

What is UML?

Unified Modeling Language (UML)

Let's break it down:

Unified

- **In 1994,**
  - Two important methodologists Rumbaugh and Booch decided to unify their approaches in 1994
- **In 1995, another methodologist, Jacobson, joined the team**
  - His work focused on use cases
- **In 1997,**
  - the Object Management Group (OMG) started to standardize UML

Models

Models are *abstract representations*

- Contain essential characteristics and omit non-essential details
- Models can be representations of the world
  - Domain models
  - Requirements
- Models can be representations of software
  - Specifications
  - Design
  - Systems

Why make models?

- Systems are complex and hard to understand
  - **The world, organizations, relationships, software**
- Models can make certain aspects more clearly visible than in the real system
- What can you do with models?
  - **Express your ideas and *communicate* with other engineers**
  - ***Reason* about the system**
    - **detect errors**
    - **predict qualities**
  - ***Generate parts of the real system***
    - **Code**
    - **Schemas**

Can ***reverse engineer*** a system to make a model

What constitutes a good model?

- A model should…
  - **Provide** *abstraction*
  - **Render the problem in a format amenable to reasoning**
  - **use a standard notation**
  - *be understandable* **by clients and users**
  - **lead software engineers to have insights about the system**
  - **make the problem solvable computationally**
    - **Be good enough**
  - **Be a tool for** *communication*

Architecture not usually a good example…

…BUT

Imagine the building the Biltmore

- 175,000 Sq. Ft. (that's 4 acres)
- 250 rooms
- 35 bd/43 ba.
- 65 fireplaces
- 125,000 acre lot (that's 195 sq mi)

Where would *you* begin?

- **A model helps reduce complexity**
  - Eliminates details that are not necessary at the time
  - Allows you to divide an conquer large tasks

What constitutes a good model?

A model should…

- Abstract away unnecessary details
- Provide a means to reason about the system
- Use a standard notation
- Be understandable by clients and users
- Lead software engineers to have insights about the system
- Be a tool for communication

Remember: It's only a model

There will always be:

- Phenomena in the application domain that are not in the model (abstraction)
- Details in the application that are not in the model (abstraction)
  - **Just what you need**
- A model is never perfect
  - **"If the map and the terrain disagree, believe the terrain"**

Modeling Languages

- Natural language
  - **Extremely expressive and flexible**
  - **Very poor at capturing the semantics of the model**
  - **Better used for elicitation, and to annotate models for communication**
- Semi-formal notation
  - **Captures structure and some semantics**
  - **Can perform (some) automated reasoning, consistency checking, animation, etc.**
  - **Mostly visual - for rapid communication with a variety of stakeholders**

Examples: diagrams, tables, structured English, etc.

Modeling Languages (2)

- Formal notation
  - **very precise semantics, extensive reasoning possible**
  - **Can automate reasoning, consistency checking, completeness checking, simulation, etc..**
  - **Every detailed models)**

Unified Modeling Language (UML)

UML is a …

- *semi-formal graphical (visual) modeling language*
- *Object Modeling Language (OMD)*
- *A way to communicate details…*
  - *Code*
  - *Architecture*

- ***Uml is descriptive → tries not to be prescriptive***

… essentially
it is a set of diagrams used to model the system

3 Common Way to Use UML

- ***Sketch*** *- Quick Communication*
- ***Blueprint*** *– Complete Specification*
- ***Programming Language***

UML as a Sketch

- **Helps** communicate **some aspect of the system**
  - Forward & reverse engineering
- **"Rough out" issues in the code**
- **Not all of the code – just parts that you are working on immediately**
  - Selective communication → NOT complete specification
- **Short discussion with a team**
  - (10 min – 1 day)
- **Quick and Collaborative**
- **Informal**

UML as a Blueprint

- **Complete specification**
  - Forward & reverse engineering
- **Detailed design**
  - All design decisions laid out
  - Simplifies programming
- **Usually done by senior developer**
- **More formally documented**
- **CASE tools**
  - Forward Engineering
    - Support diagramming
    - Repository to store information
  - Reverse Engineering
    - Read source code → Generate Diagrams

UML as a Programming Language

- **UML Diagrams compiled into exe code**
  - Automatic code generation
  - Sophisticated tool support

Types of UML Diagrams

Structure (6 types)

- Class diagrams
- Object diagram
- Package diagram
- Composite structure diagram
- Component diagram
- Deployment Diagram

Behavior (4 types)

- Activity diagram
- Use Case diagram
- State machine diagram
- Interaction diagrams
  - **Sequence diagram**
  - **Communication diagram**
  - **Interaction overview diagram**
  - **Timing diagram**

If the appropriate diagram is not part of UML
use it anyways

UML & the S/W Process(Requirements)

- Use Cases
  - Describe how people interact with the system
- Class Diagram
  - Drawn from a conceptual perspective
  - Can build up a rigorous vocab of the domain
- Activity Diagram
  - Shows the workflow of the org.
    - Shows how s/w and human activities interact
  - Context for Use Cases
  - Details of complex Use Cases

UML & the S/W Process(Requirements)

- State Diagram
  - Shows states and events that change the state
    - Can be useful with interesting life cycles

Communication is key
Customers may not be familiar with S/W techniques

Break the rules is it enhances Communication

UML & the S/W Process (Design)

- **Class Diagrams**
  - From a software perspective
    - Show classes & how they interrelate
- **Sequence Diagrams**
  - For Common Scenarios
    - Pick most significant scenarios from Use Cases
    - Use CRC cards or sequence diagrams to determine how the software should behave
      - Class, Responsibilities, Collaborators (CRC) cards are index cards used to represent
        - » the responsibilities of classes
        - » interaction between the classes

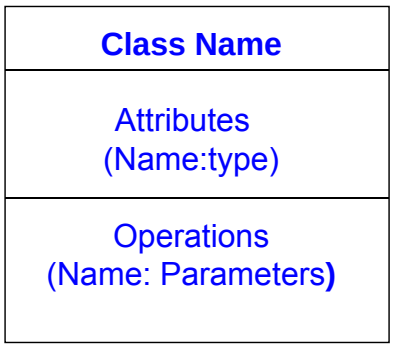
UML & the S/W Process (Design)

- **Package Diagrams**
  - Show large-scale organization of the system
- **State Diagrams**
  - Used for classes with complex lifecycles
- **Deployment Diagrams**
  - Show the physical layout of the software

All of these can be used for design

Class Diagrams

"A ***Class Diagram*** describes the types of objects in the system and the various kinds of *static relationships* that exist among them"



Makes it easier to see the big picture

- Know what a class does at a glance

Attributes and Operations

- **Attributes →**
  - Describes a property as a line of text within the class box
  - Attribute name corresponds to the name of a field in a programming language
  - ***Visibility Marker →***
    - **Denotes whether an attribute is…**
      - **Public (+) or Private (-)**
- **Operations →**
  - Actions that a class knows to carry out
  - *Corresponds* to methods on a class

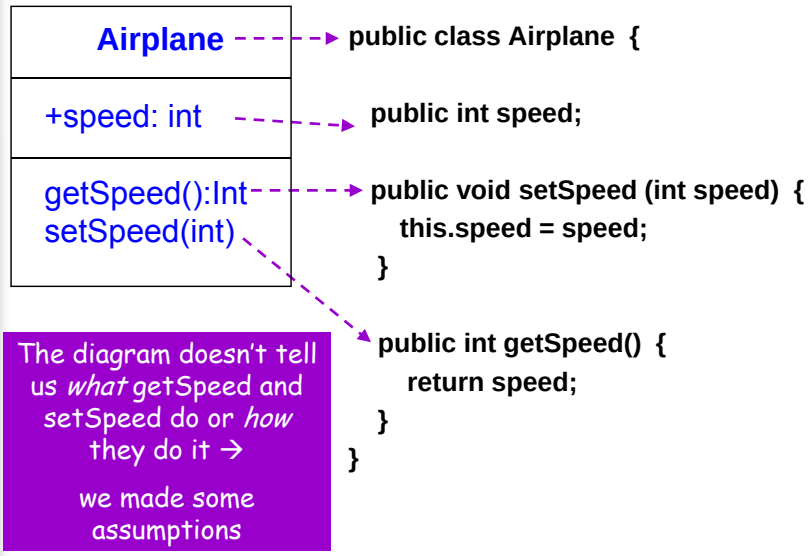
Attributes and Operations (2)

Operation & Method are *not* the same thing

An Operation is the procedure declaration
A Method is the body of a procedure

- **Associations →**
  - Describe the relationship between two classes

Example of a Class

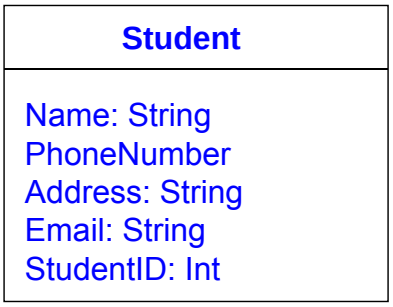


The diagram doesn't tell us *what* getSpeed and setSpeed do or *how* they do it →

we made some assumptions

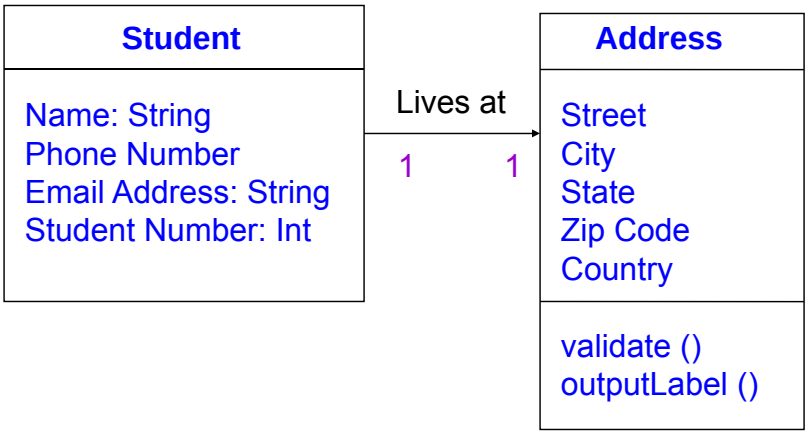
Example 2 of a Class

Different level of abstraction



Associations -- Notation

Associations



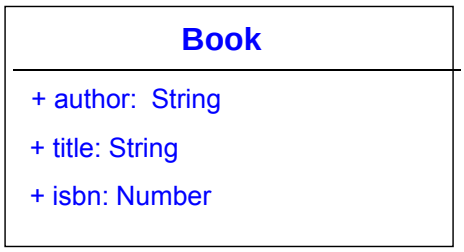
Properties → Attributes & Associations

- **Properties →**
  - A structural feature of a class (fields in a class)
  - Can be represented 2 ways: *Attributes* or *Associations*

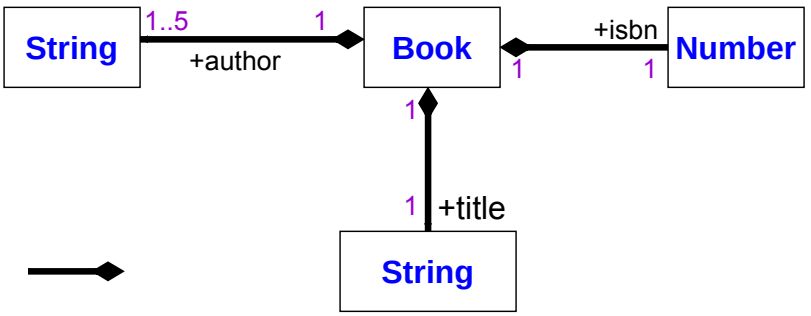
Attributes and Associations →
Different notations for the same thing

Example: Properties as Attributes

Simple Example



Properties as Associations



Attributes & Associations

- **Same properties → different notations**
- **When do you use which?**
  - Attributes for more simple properties (such as Booleans or Dates)
  - Associations for more significant properties (such as Orders or Customers)
- **Associations show more – such as multiplicities (covered in discussion)**

Some Basic Concepts

- **Generalization (AKA Inheritance)**
  - For all instances of a superclass…
    - The subclass *inherits…*
      - Attributes
      - Operations
      - Associations (we'll talk about these later)
    - Whatever is true for the superclass is true for the subclass

Inheritance lets you build classes based on other classes without having to duplicate or repeat code.

Example of Generalization

Airplane is the superclass for Jet.

Jet is the subclass

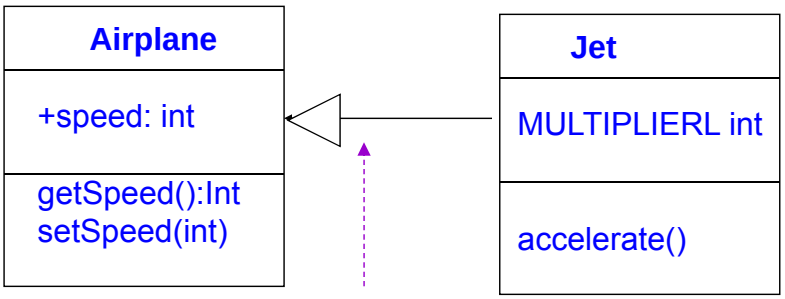
```
Public class Jet extends Airplane {
    private static final int MULTIPLIER =2;

   public Jet () {
        super();
    }
   public void setSpeed (int speed) {
      super.setSpeed(speed *
     MULTIPLIER)
     }
   public void ()  {
        super.setSpeed (getSpeed() *2);
   }
}
```

Jet extends from the Airplane class – that means it inherits all of Airplane's behavior

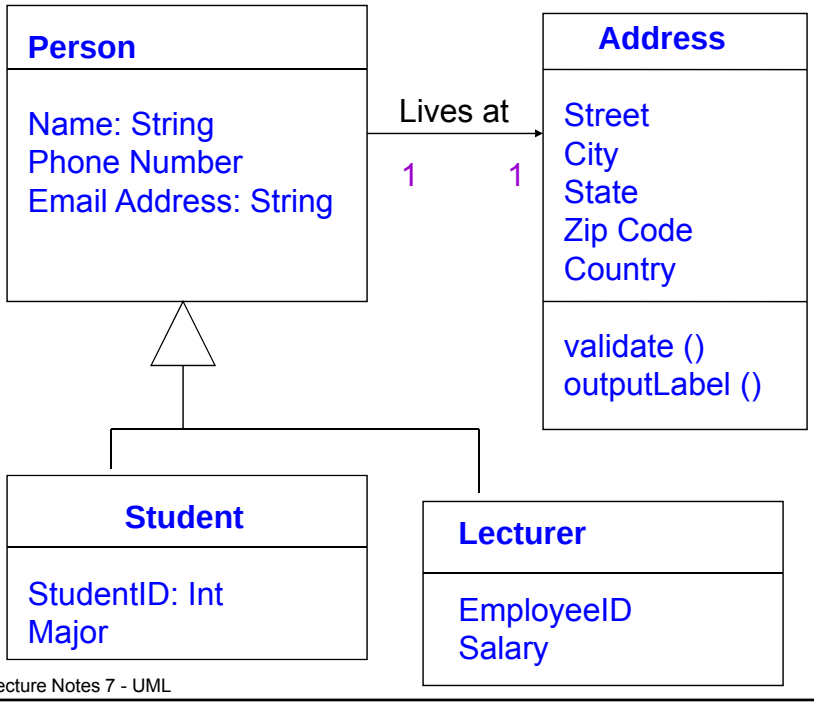
Jet can modify the behavior of the superclass' methods → it can also just call on them.

Note: getSpeed is not in here because Jet is not modifying it
→ You can still call getSpeed on Jet

Generalization notation



Generalization is notated using a big open arrow



Polymorphism

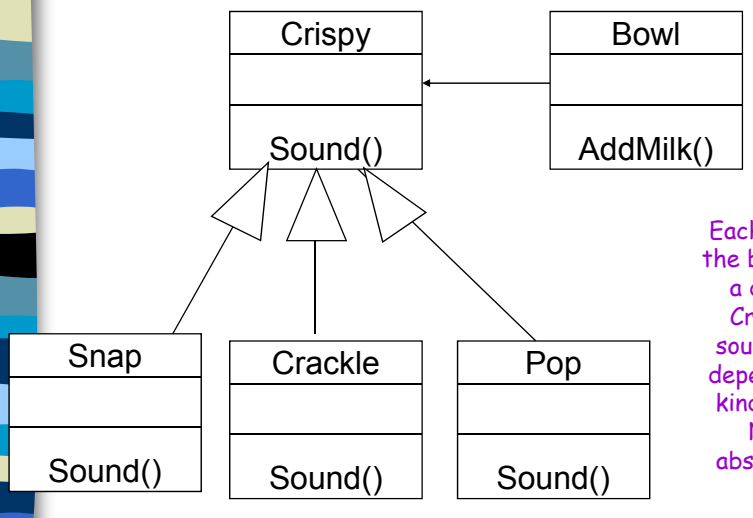
- **if an operation is applied to an object and there are several alternative classes that have the operation defined**

  → then the object to which the operation is applied always determines the operation that is executed.

IN OTHER WORDS..

- **Allows you to process objects differently depending on their data type or class**
  - Redefine methods for a derived class

Polymorphism (Example)



Each pointer in the bowl selects a different Crispy. The sound of each depends on the kind of Crispy Not the abstract type

Class Diagrams

- **Association**

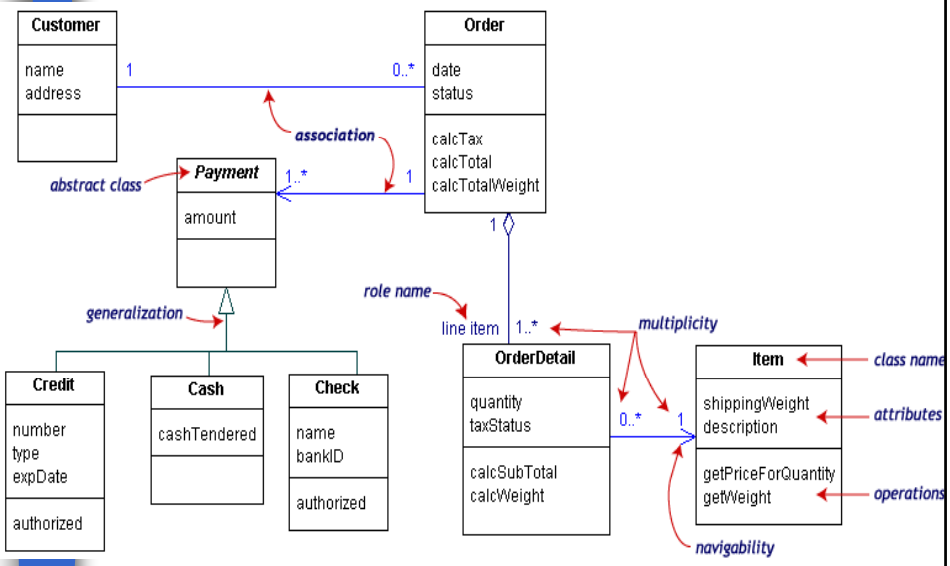
  There is an ***association*** between two classes if an instance of one class must know about the other in order to perform its work.
  - A relationship between instances of the two classes.
  - In a diagram, an association is represented by a link connecting two classes.
  - may have a *role name* to clarify the nature of the association
  - A *navigability arrow* on an association indicates which direction the association can be traversed or queried.
    - no navigability arrows are bi-directional.

Class Diagrams (2)

- **Aggregation**
  - An association in which one class belongs to a collection.
  - In a diagram, an aggregation is represented with a diamond end pointing to the part containing the whole.
    - **"is a part of"**
- **Generalization**
  - An inheritance link indicating one class is a superclass of the other
    - "is a" or "is like a"
  - A generalization is represented with a triangle pointing to the superclass.

Class Diagrams provide a static model view of the system
Describes the Structure

Class Diagrams



Take a break!

- **Get some Coffee**
- **Wakey-Wakey**

When we return…

- **Modeling**
  - More on UML

Moving on

- **UML**
  - Use Case Diagrams
  - Sequence Diagrams

Types of UML Diagrams

Structure .	Behavior .
(6 types)	(4 types)
Class diagrams	Activity diagram
Object diagram	Use Case diagram
Package diagram	State machine diagram
Composite structure diagram	Interaction diagrams
Component diagram	**Sequence diagram**
Deployment Diagram	**Communication diagram**
	Interaction overview diagram
	Timing diagram

If the appropriate diagram is not part of UML
use it anyways

What is a Scenario

- **A Scenario is an example of what happens when someone interacts with the system**
- **Describes the system from an *external* viewpoint**
- **EXAMPLE Scenario – Medical Clinic:**
  - "A patient calls the clinic to make an appointment for a yearly checkup. The receptionist finds the nearest empty time slot in the appointment book and schedules the appointment for that time slot. "

What is a Use Case?

- A use case is a reason to use the system
- Again - describes the system from an *external* viewpoint
  "provides an outsider's view"
- A way of formalizing scenarios
- A summary of scenarios for a single task or goal
- Treat system as a black box
  - Don't incorporate design decisions
    - Applies unnecessary constraints at design

Use Case Diagrams describe the dynamic behavior of the system

Use Case Basics

- **Actors**
  - who or what initiates the events involved in that task
  - roles that people/objects/systems (anything *external* to the system) play
  - Represented as stick figures
- **Use Case – some system function (a summary of related scenarios)**
  - Represented as an oval
- **Communication (or Communication Association)**
  - A Connection between the actor and the use case
  - Represented as a line

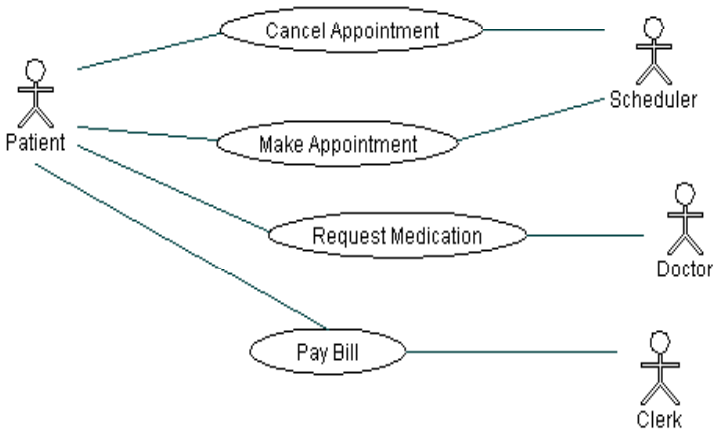
Use Case Diagrams

- A collection of actors, use cases, and their associations

Use case diagrams are helpful in three areas

- **Determining features (requirements)**
  - New use cases often generate new requirements.
    - Can happen during design and system analysis
- **Communicating with clients**
  - Simple notation makes them easy to understand
- **Generating test cases**
  - The collection of scenarios for a use case may suggest a suite of test cases for those scenarios

Use Case Diagram – Medical Clinic



Expanding Use Cases

- **A simple use case diagram can be expanded to display more information**
- **Use Cases can be developed iteratively and incrementally**
- **System boundaries**
  - separates the system from the external actors
  - Represented as a rectangle
- **Generalizations**
  - shows that one use case is simply a special kind of another
  - Represented with an open triangle

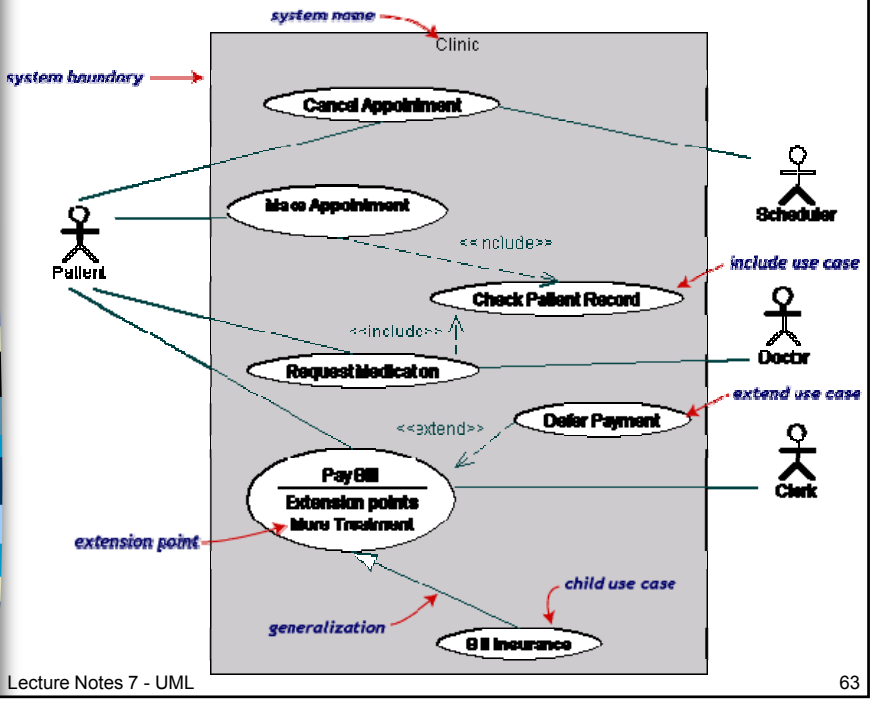
Use Cases: Includes & Extends

- **Includes**
  - A relationship in which one use case (the base use case) includes the functionality of another use case
  - Promotes reuse
  - Should be used when the inclusion case is common in two or more use cases

Both use similar notation, but are very different.
Represented with a dashed line and <<includes>> or <<extends>>

Includes & Extends

- **Extends:**
  - specifies that one use case (extension) extends the behavior of another use case (base).
  - reveals details about a system or application that are typically hidden in a use case
  - the extension use case is not meaningful on its own
  - Describes behavior sequences that can change the base case
  - Each behavior sequence can be inserted into the base use case at a different point, called an extension point
  - When do you use it
    - A part of a use case that is optional system behavior
    - A subflow is executed only under certain conditions
    - A set of behavior segments that may be inserted in a base use case



Use-Case Templates

- **Extended format of a use-case**
- **Provides consistent documentation for each use-case**
- **Clarifies what you are describing**
- **Many templates**
  - You will be provided one for your homework

Sequence Diagrams

- **One type of Interaction Diagram**
- **Represent one scenario**
- **Describe the dynamic behavior of the system**
- **Details how operations are carried out**
  - What messages are sent when
- **Organized according to time**
- **Objects listed from left to right**
  - According to when they take part in the message sequence

Sequence Diagrams (2)

- **Good at**
  - describing the behavior of several objects within a single use case
  - Showing collaborations between objects
- **Not good at precise definition of the behavior**

Sequence Diagrams:Basic Elements

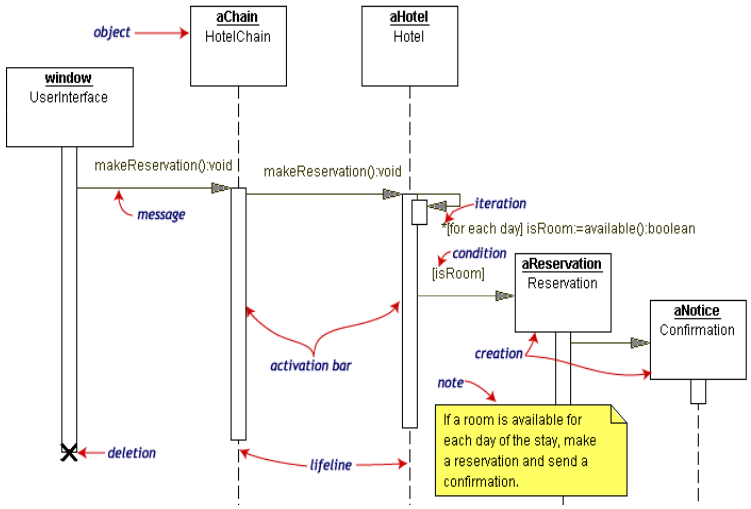
- Objects
  - Lifelines: time goes from top to bottom
    - represents the time that an object exists
  - Activation bar: represents the duration of execution of the message
    - **(if the object is active)**
  - May be several instances of one class
- Messages
  - Analogous to method calls in a program
  - Can have parameters
  - Represented by an arrow between activation bars

Sequence Diagrams:Basic Elements

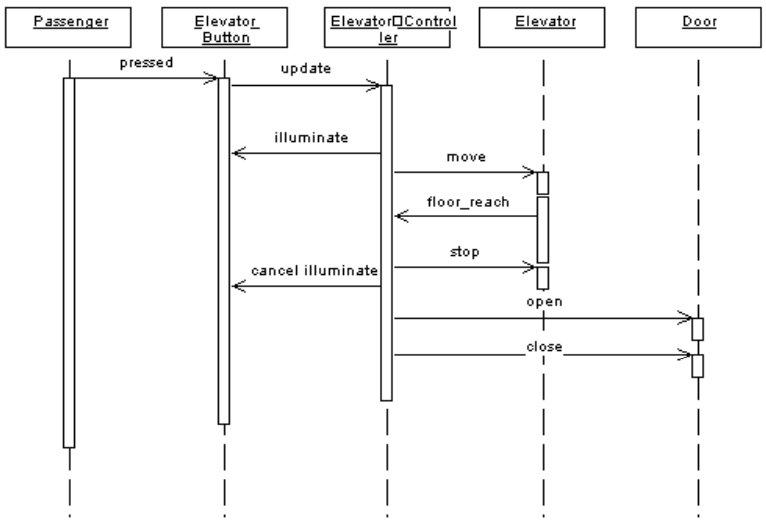
- Special messages
  - New — shown by position of object
  - Delete — shown with a big X
  - Return messages -- represented by a dashed arrow
  - Self-calls – when an object calls itself
- **A note is used to clarify details**
  - Represented with a dog-eared rectangle

(Notes can be put into any kind of UML diagram)

Sequence Diagram Example: Hotel Reservation



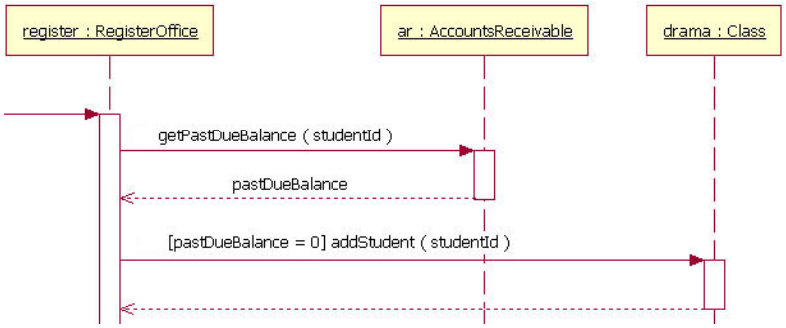
Elevator Example: Sequence Diagram



Sequence Diagram for Serving Elevator Button

Guards

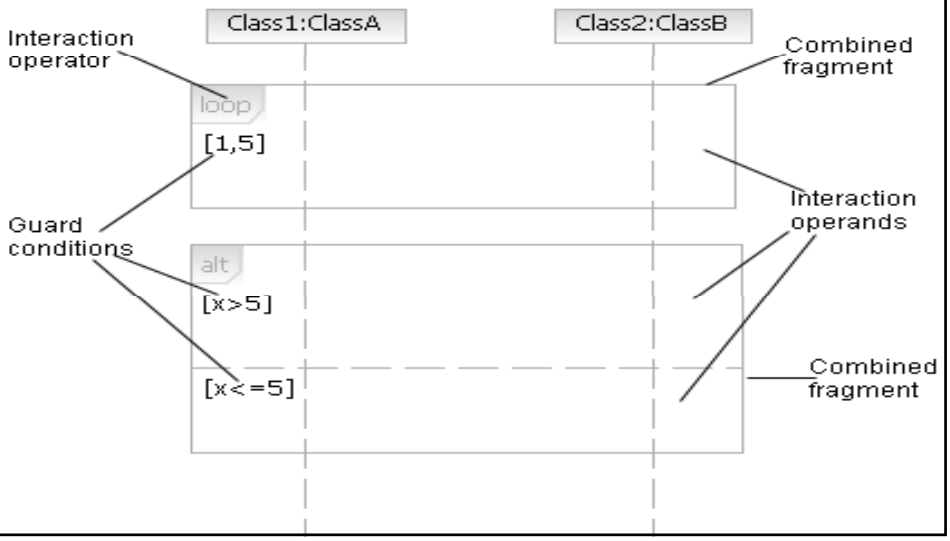
- **When a condition must be met before a message is sent**
- **Represented by brackets on the message line [*guard*]**



Frames

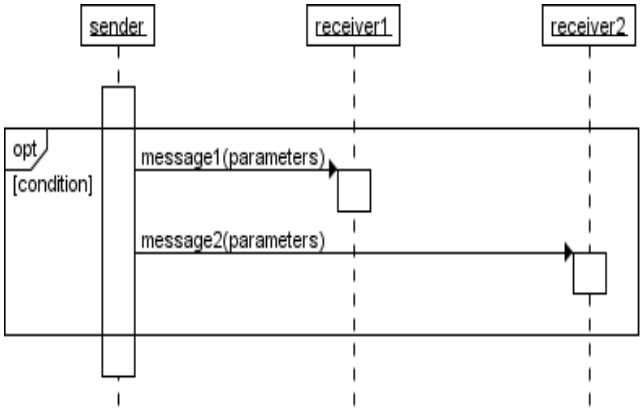
- Encloses a region of a sequence diagram
- Guard specifies condition
  - **Allows you to specify several interactions within a guard**
- Can be divided into one or more fragments
- Keyword specifies the type of frame
- Keywords:
  - **opt -Optional fragment that executes if guard is true**
  - **alt -Alternative fragment for mutual exclusive choice between two or more message sequences**
    - **Eg. If → then → Else**
  - **loop -Loop fragment while guard is true**
  - **par -Fragments that execute in parallel**
  - **region -Critical region within which only one thread can run**

Frames/Combined Fragments

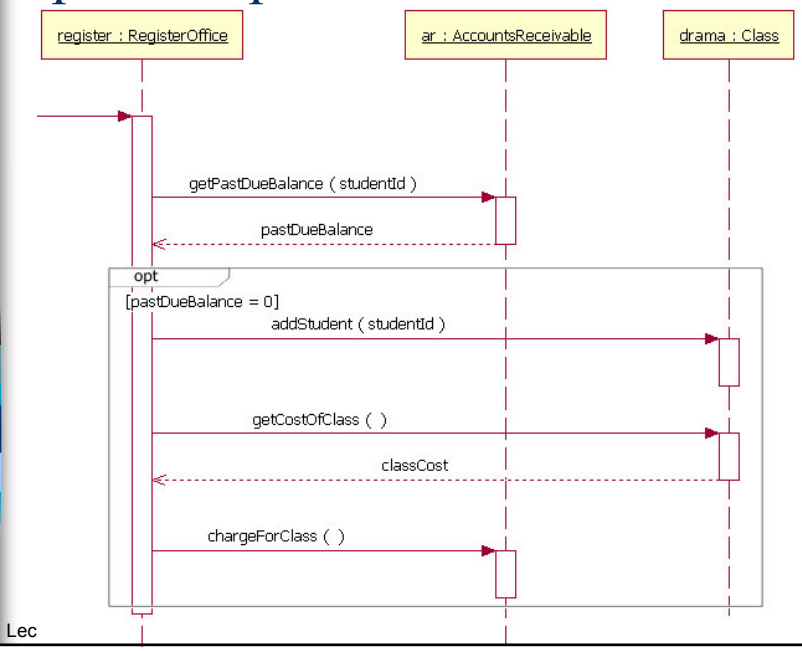


Frame: Option

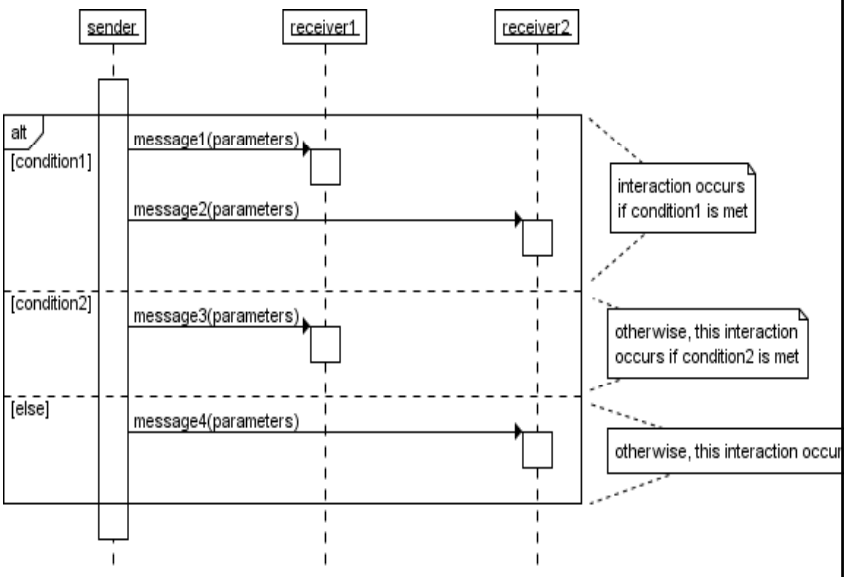
- **Like a typical guard expanded**



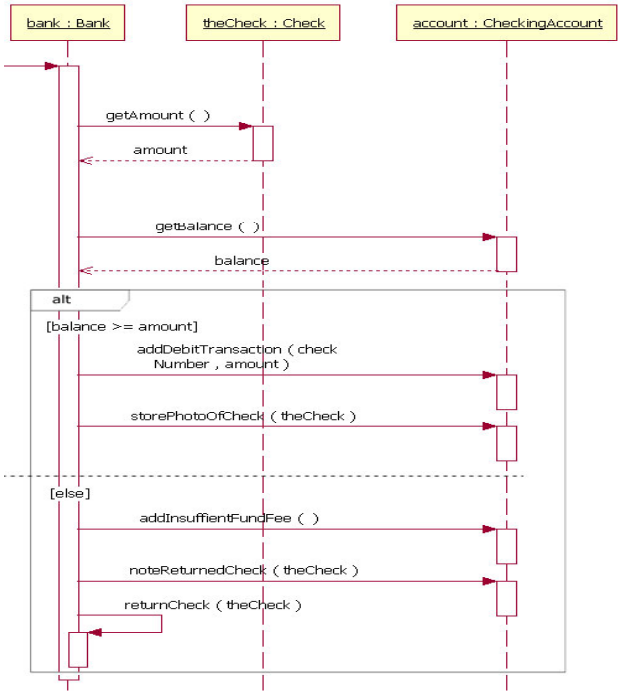
Opt Example



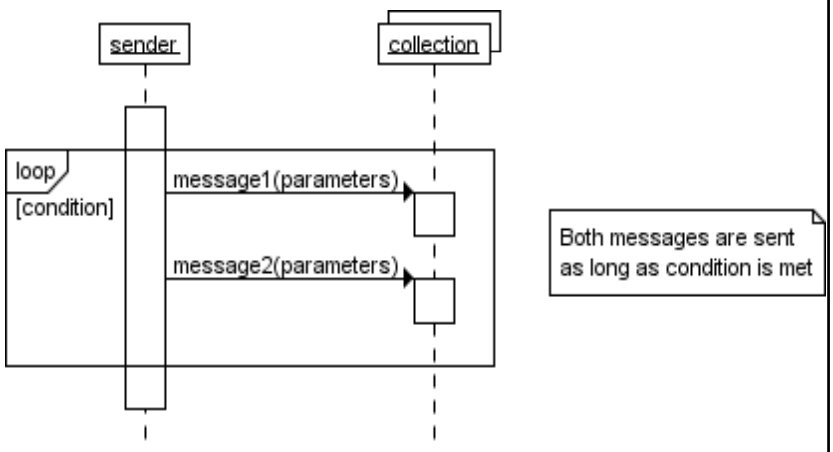
Frame: Alt



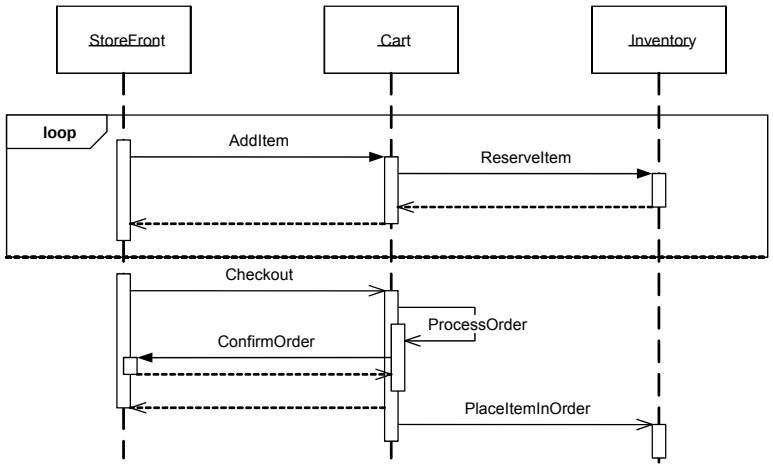
Alt Example



Frame: Loop

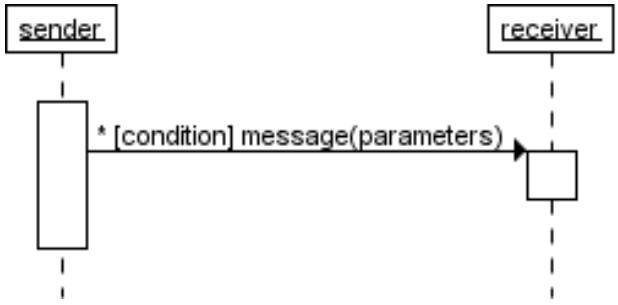


Loop Example



What if you only have 1 msg to loop?

- **Use the "*" symbol**
- **As long as the condition holds the message is sent**



Synchronous & Asynchronous Calls

- Synchronous
  - **Some methods must finish before another can start**
- Asynchronous
  - **Some methods can continue executing while others run**

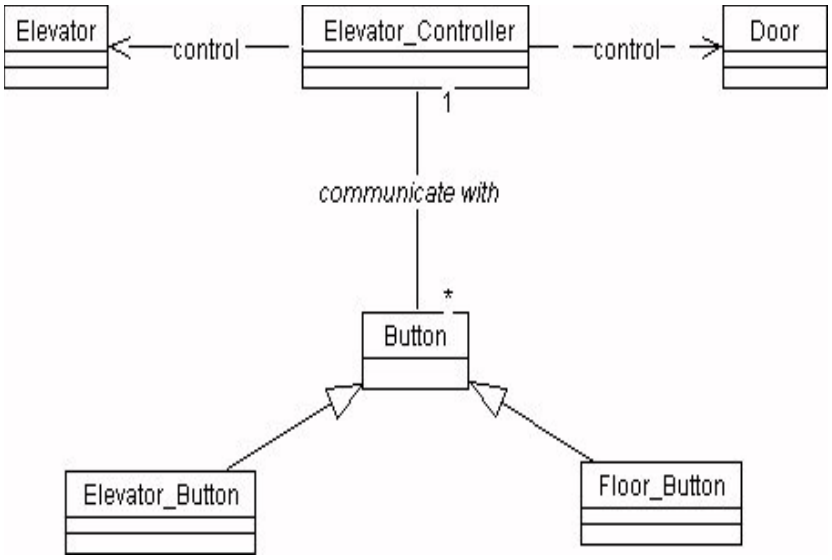
Putting them together

- **Class Diagrams**
- **Scenarios**
- **Use Cases**
- **Sequence Diagrams**

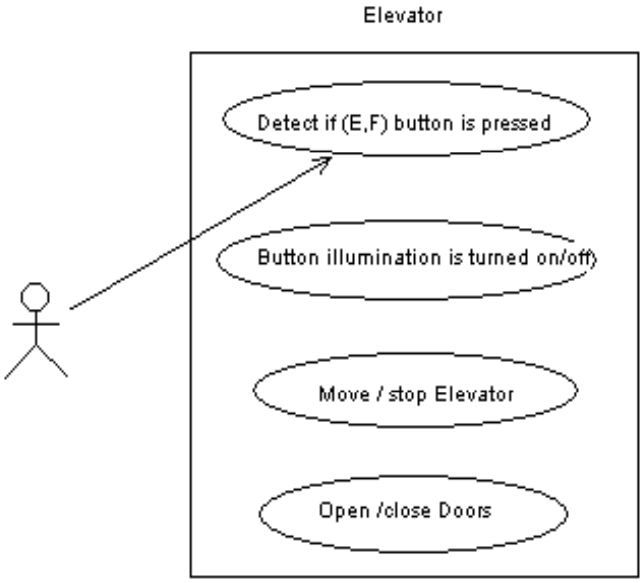
- **How do they all work together**

UML is iterative & Incremental

Elevator Example: Basic Class Diagram



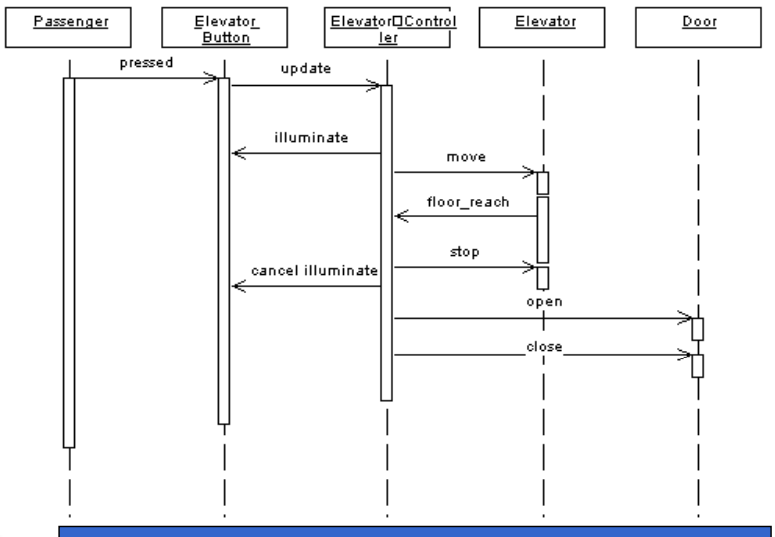
Elevator Example: Use Case



Elevator Example: Scenario

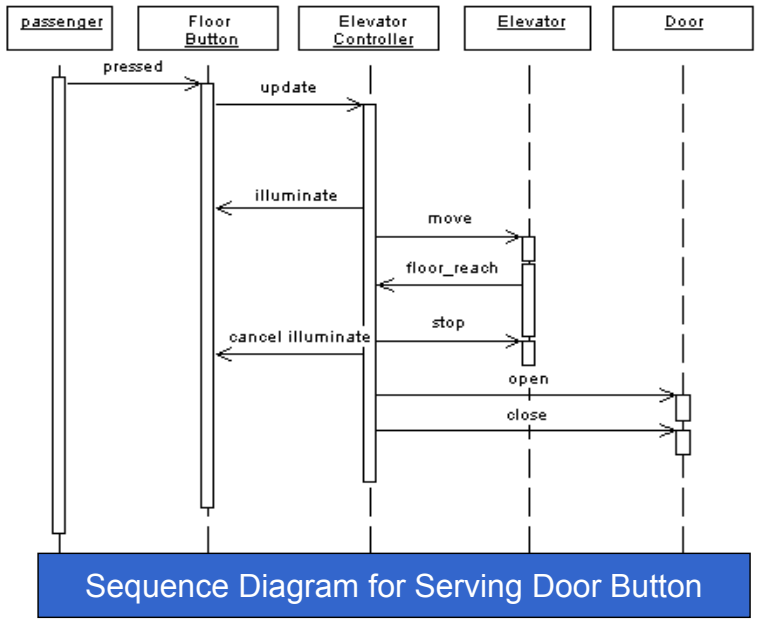
- Passenger pressed floor button
- Elevator system detects floor button pressed
- Elevator moves to the floor
- Elevator doors open
- Passenger gets in and presses elevator button
- Elevator doors close
- Elevator moves to required floor
- Elevator doors open
- Passenger gets out
- Elevator doors close

Elevator Example: Sequence Diagram



Sequence Diagram for Serving Elevator Button

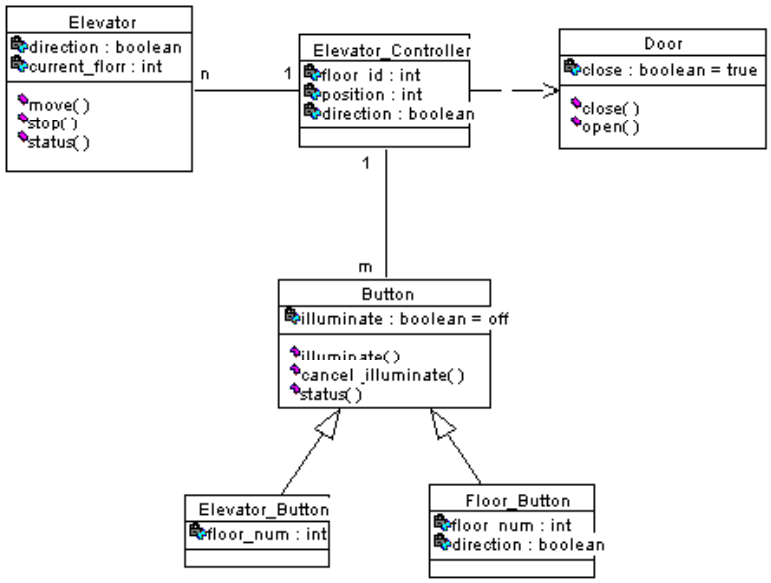
Elevator Example: Sequence Diagram



Sequence Diagram for Serving Door Button

Le 87

Elevator Example: Revising the Class Diagram



Le 88