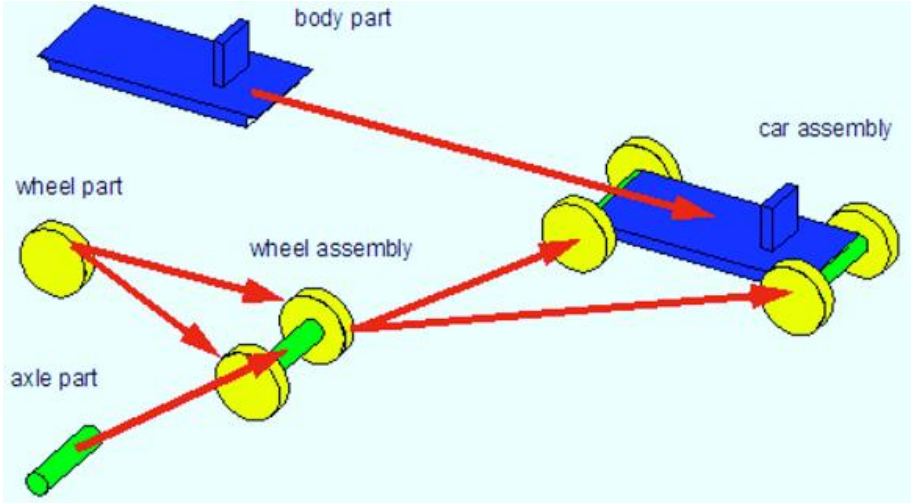
Class Model

Lecture 12
18 June 2026

Slides created by
Prof Amir Tomer
tomera@cs.technion.ac.il



Picture Source: http://www.techsoft3d.com/developers/technical-documentation/siemens-parasolid/

Topics for Today

- Class Model

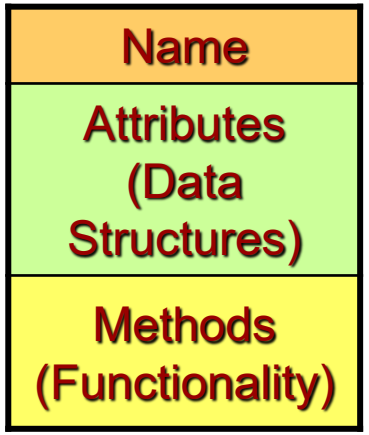
Software Elements: Objects and Classes

Objects

- Basic elements of software
- Every object manages its information via internal functionality (methods)
- Objects exist in memory while the program is running
  - Can create and destroy them dynamically
  - Constructor, Destructor
- Every object has at least one handle or pointer to it in memory

Classes

- Forms out of which objects are created
- Contain three elements

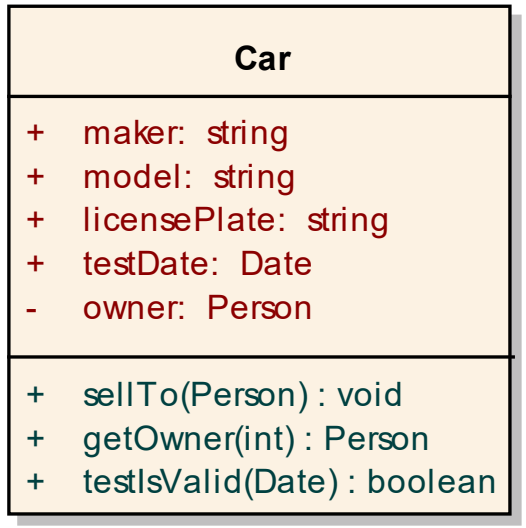


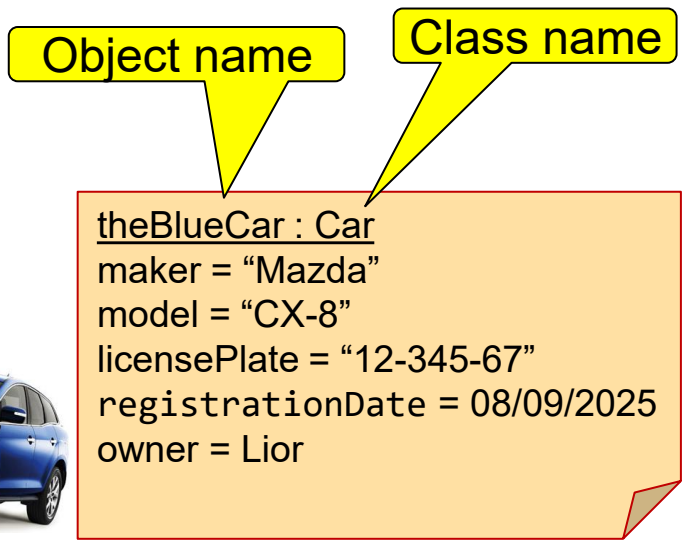
- Defined in code by the programmer
- Objects are instances of classes

Creating and Operating Objects

Create a new object	`theBlueCar = new Car()`
Initialize car details	`theBlueCar.make = "Mazda"` `theBlueCar.model = "CX-8"`
Licensing and registration	`theBlueCar.licenseplate = "12-345-67"` `theBlueCar.registrationDate = 08/09/2025`
Selling	`theBlueCar.sellTo(Lior)`

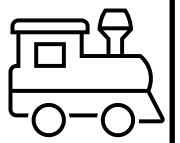
	`Function sellTo (X){`
	`  owner = X;`
	`}`





Candidates for Objects in Software

Objects that represent physical entities



- Door, Engine, Workstation
- Attributes:
  - Parameters and data about the entity
  - Input/output
- Methods: Physical functionality
- Serve as a stand-in or interface to physical object

Objects that represent logical entities

- Process, Service
- Attributes:
  - Parameters and data about the entity
  - Input/Output
  - Methods:
    - Operations the entity can do

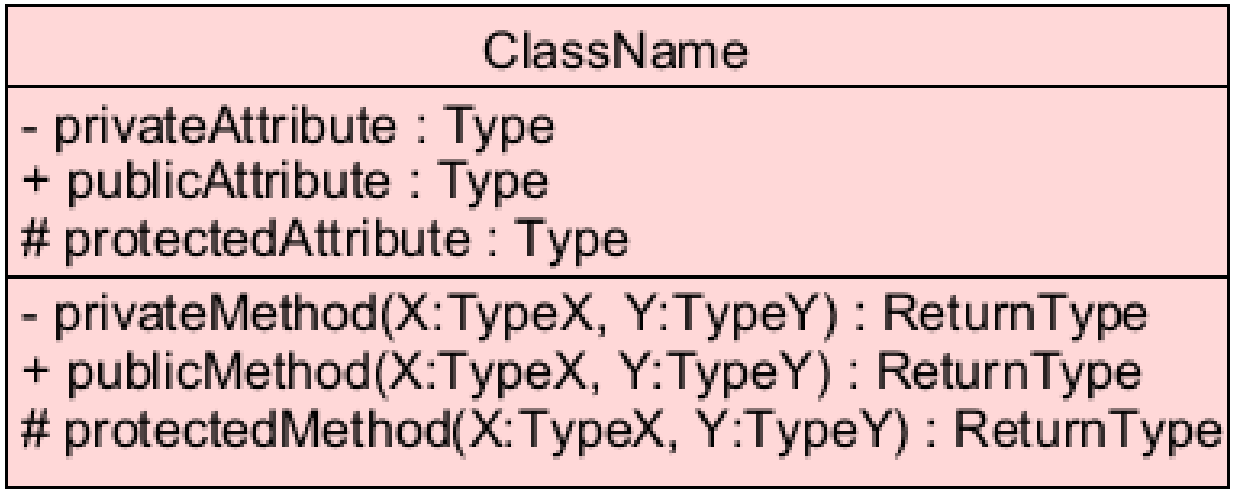
Objects that represent data entities

- Databases, data stores, lists, queues
- Attributes:
  - Data elements managed by the object
- Methods:
  - Data operations (store, retrieve, modify)

Class Diagram - Syntax

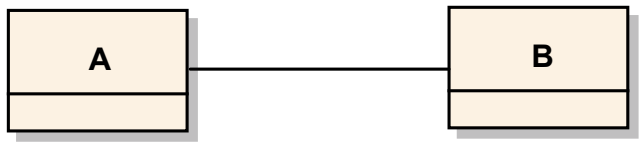
Class

- Name
- Attributes (variables)
  - Private (-): Can only be accessed from within the class
  - Public (+): Can be accessed also externally
  - Protected (#): Can be accessed within the package or by sub-classes
- Methods (Functions)
  - Private (-): Can only be called from within the class
  - Public (+): Can be called also externally
  - Protected (#): Can be called within the package or by sub-classes

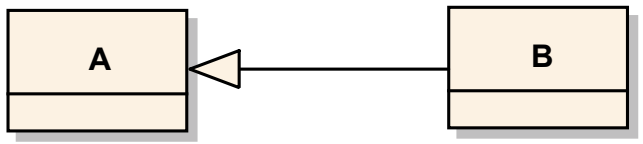


Class Diagram - Syntax

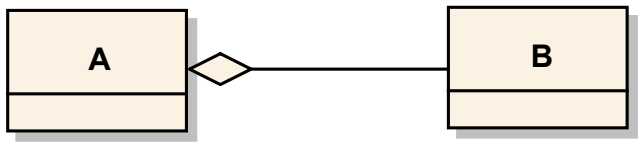
Class diagrams are based on the principles of semantic networks



Association



Inheritance



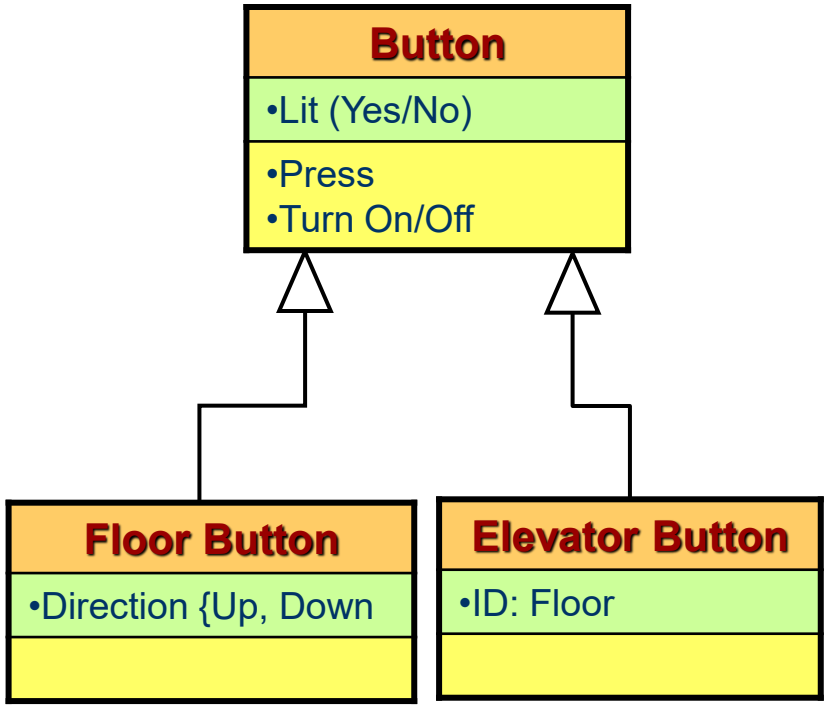
Aggregation

Inheritance and Generalization

Inheritance creates hierarchical relationships

Inheritance is "B is-an A"

- Class B inherits from Class A
- B has all of A's attributes
- B has all of A's methods



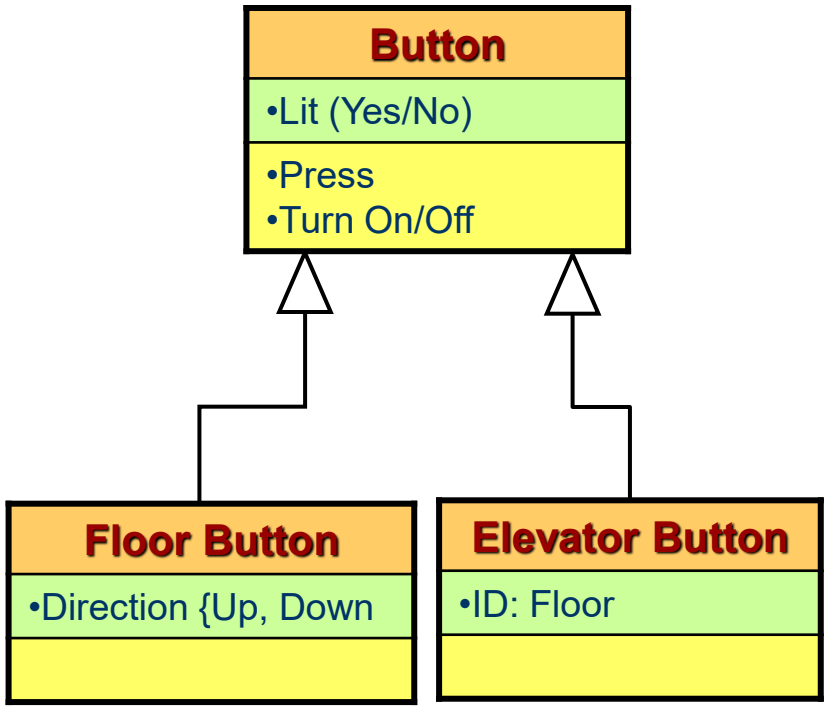
B is a "sub-class" of A

- B can do more than A

Inheritance and Generalization

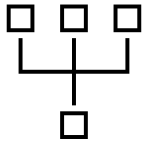
Abstract classes

- Missing elements to make it concrete/instantiable
- A class you can't make instances of
- All instances are of subclasses (e.g. vehicle)



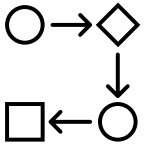
Problems with inheritance

Multiple Inheritance



- One class inherits from multiple classes
- Problem:
  - Might lead to contradictions in attributes or methods
- Solution:
  - Most programming languages don't allow multiple inheritance (force tree-like structure)

Deep Inheritance



- A ◁ – B ◁ – C ◁ – D ◁ – … ◁ – X
- Problem:
  - Keeping track of connections (maintainability)
- Solution:
  - Break the chain where connection is weak

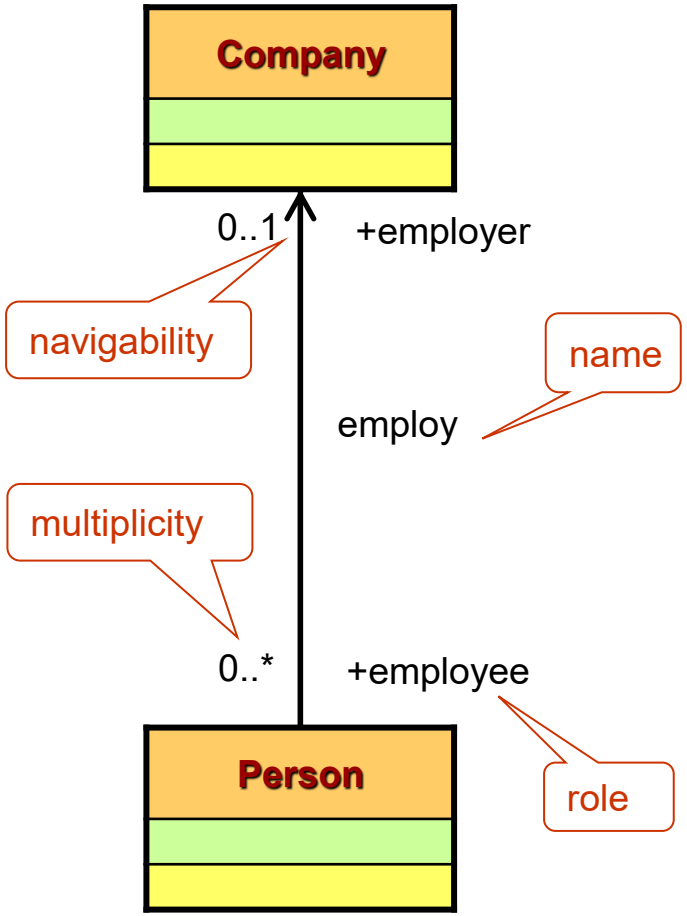
Problems with inheritance

False Inheritance

- Example: A square is a type of rectangle, so Rectangle ◁ — Square
- Problem:
  - Rectangle has two attributes (len, wid)
  - Square has one attribute (edge)
- Solution:
  - Define inheritance based on shared properties (attributes, methods) not just conceptual similarity

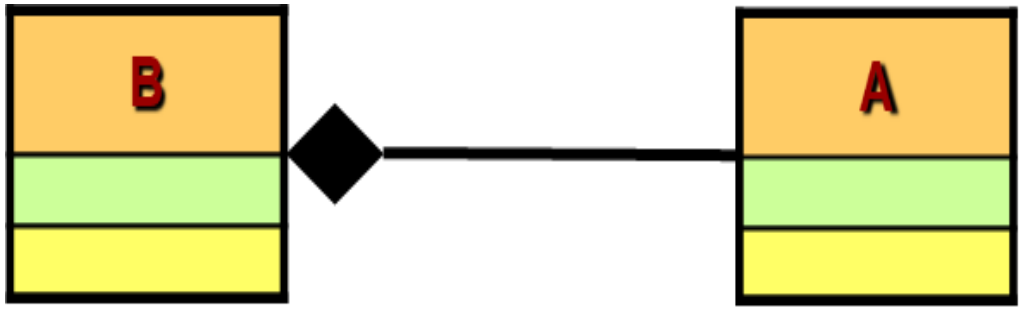
Association

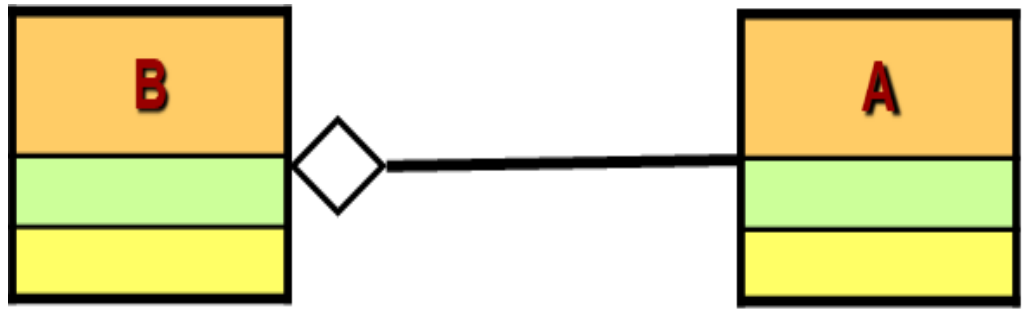
- Relationship between classes that instances of them “know” each other
  - Knowing works via references and pointers
  - Properties specific association
- Name: Can be read both ways
  - Company employs person, Person employed by company
- Role
  - Company is the employer, Person is the employee
- Multiplicity
  - Company employs 0 or more Persons, Person is employed by 0 or 1 Companies
- Navigability
  - Person knows who is its Company, Company does not know its Persons



Aggregation

- A special kind of association (knowing): B has-a A





Composite Aggregation

- A is an integral part of B and only of B
- A exists only due to B
- AKA: Whole-part aggregation, Non-shared aggregation

Shared Aggregation

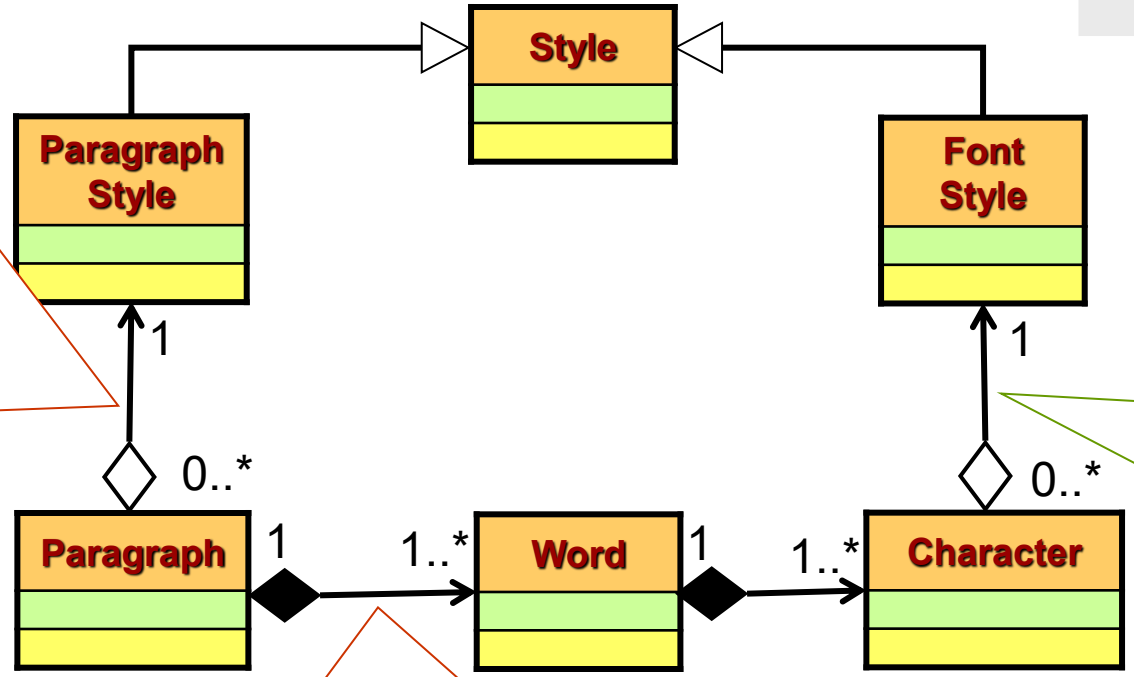
- A is part of B, but
- A can exist without B
- A can be shared between other objects

Aggregation Example

1. A Numbered Title

   This is the first *paragraph* of this document. It contains 17 *words* and 80 non-blank characters.

Shared aggregation:
- A paragraph has one Paragraph Style
- Paragraph style can apply to many paragraphs
- Paragraph style is an independent entity and can exist without any paragraphs
- Deleting a paragraph doesn't delete the style



Navigation:
- Character knows its style
- Style doesn't know characters it applies to

Composite aggregation:
- Paragraph has at least one word
- Each word in the paragraph belongs to only one paragraph
- Words exist only within a paragraph
- Deleting a paragraph deletes all the words within

Software level functional analysis

As with functional analysis before, must do software functional analysis (objects, classes)

To start, software elements are classes in the PDOM → Then assign methods and functionality

- May add classes later as needed

Sources of functionality

- Software architecture
- Implement processes found in the sequence diagrams

When assigning functionality, preserve "Specialization" and "Independence"

- Tight cohesion: What is the common ground among data and methods in the class?
- Weak coupling: How much is a class dependent on others?

System level functional analysis

Reminder

- We did the following at the system architecture level

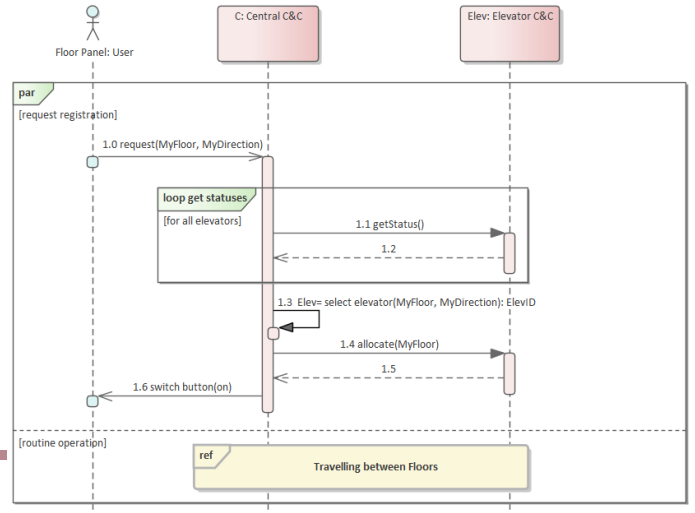
1. Find functionality in use cases

SUC-1	Call Elevator
Actors and Goals	Passenger: To receive an elevator car available for travel
Stakeholders and Interests	None
Pre-conditions	• User is on a floor in the building with an elevator door • System is operational (post-condition of UC: Start-up)
Post-conditions	An elevator car is at the user's floor with the door open (destination floor)
Trigger	**Passenger** pushes the up or down button on the floor
Main Success Sequence (MSS)	1. The system records the button press 2. The button lights up 3. The system finds a car traveling in the desired direction 4. The system assigns a stop for the car 5. The elevator arrives at the floor 6. The door opens 7. The floor button turns off

2. Break down into components

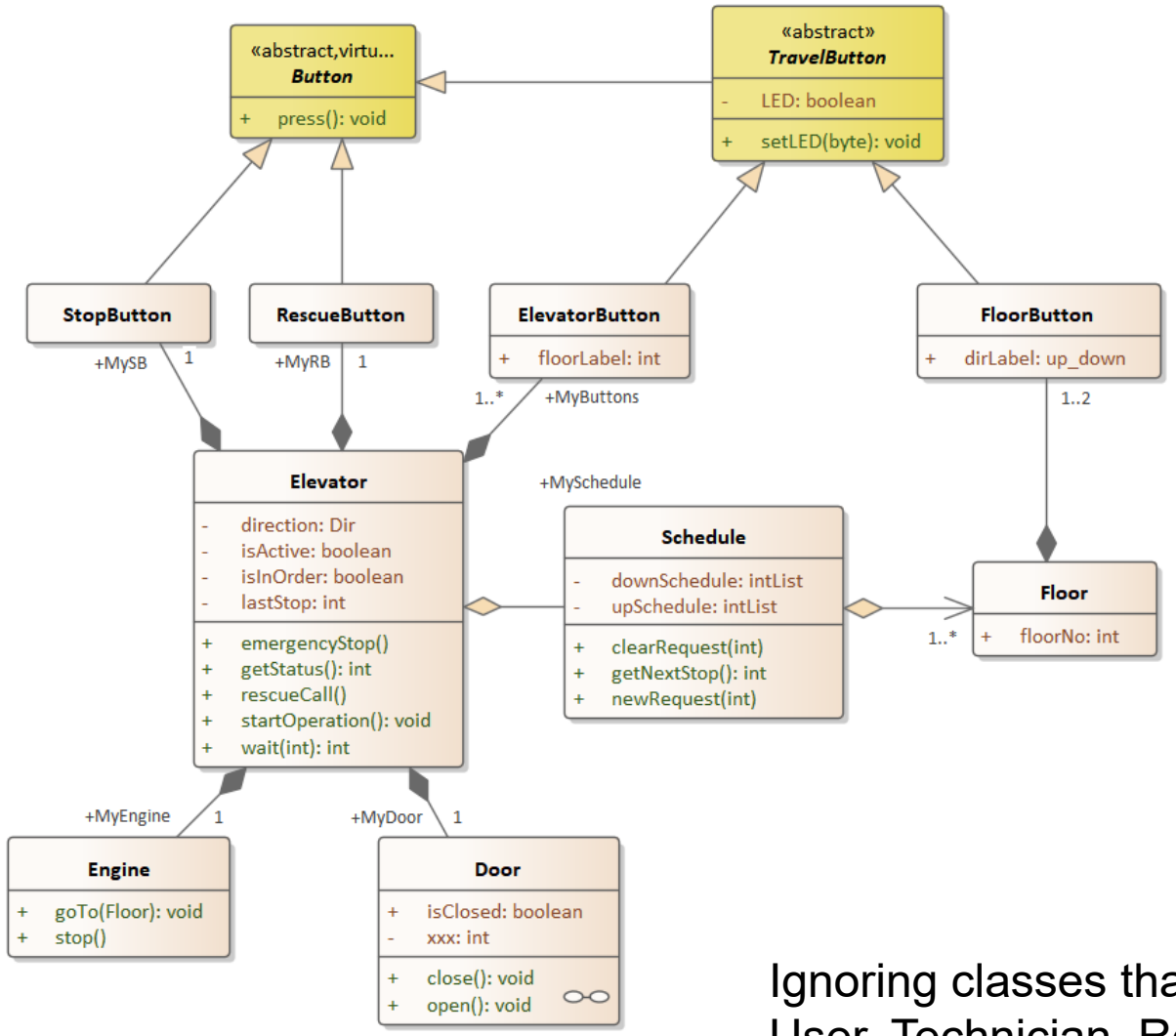
Whole system operations

System Command and Control	Central C&C	1. Receive user requests from per-floor up/down buttons 2. Assigning calls to elevator cars 3. Sending calls to elevator cars 4. Identify emergency events and take care of rescue operations
System Maintenance	Maintenance	5. Built in Test (BIT) 6. Support technician tests 7. Support technician repair tasks
Configuration Management	Administration	8. Start up 9. Shut down 10. Sabbath mode



3. Implement processes using sequence diagrams

Elevator system – Base class diagram from PDOM

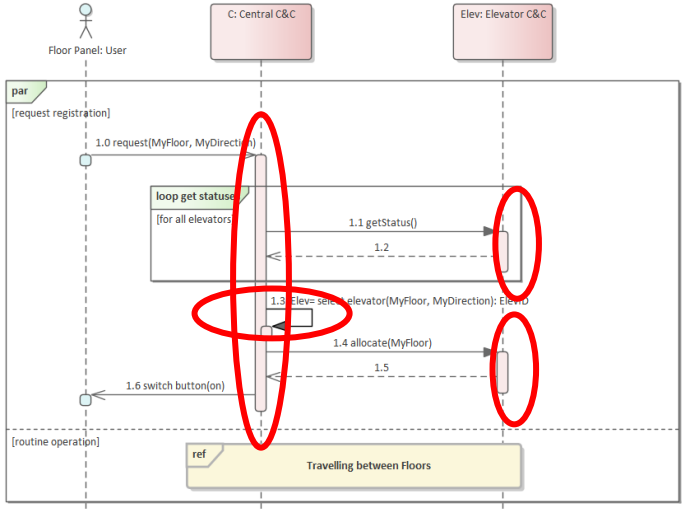


Ignoring classes that represent people:
User, Technician, Rescuer, Maintenance Engineer

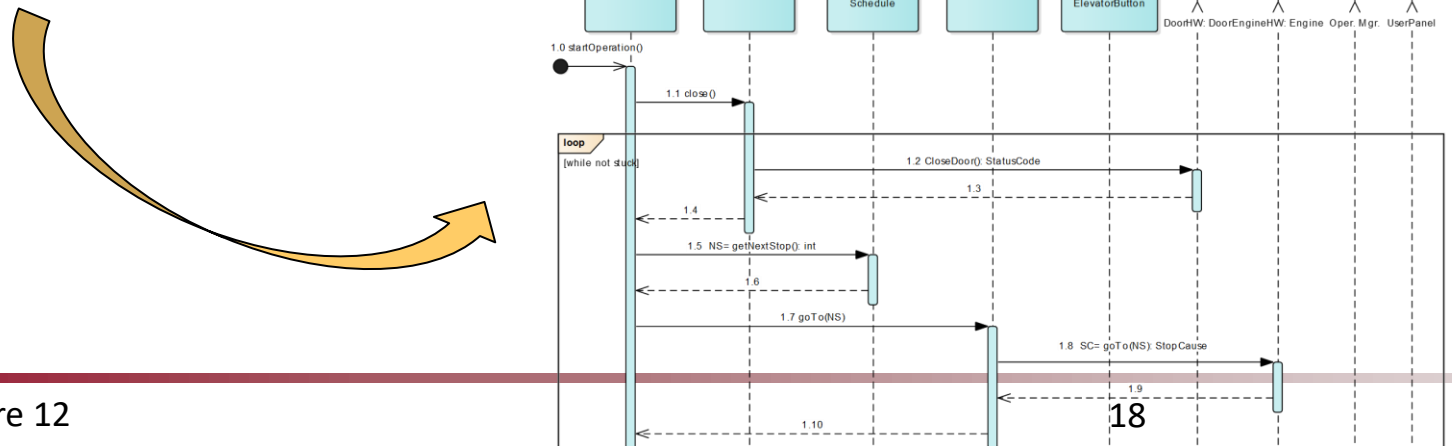
Software level functional analysis

- Perform the steps above on the level below

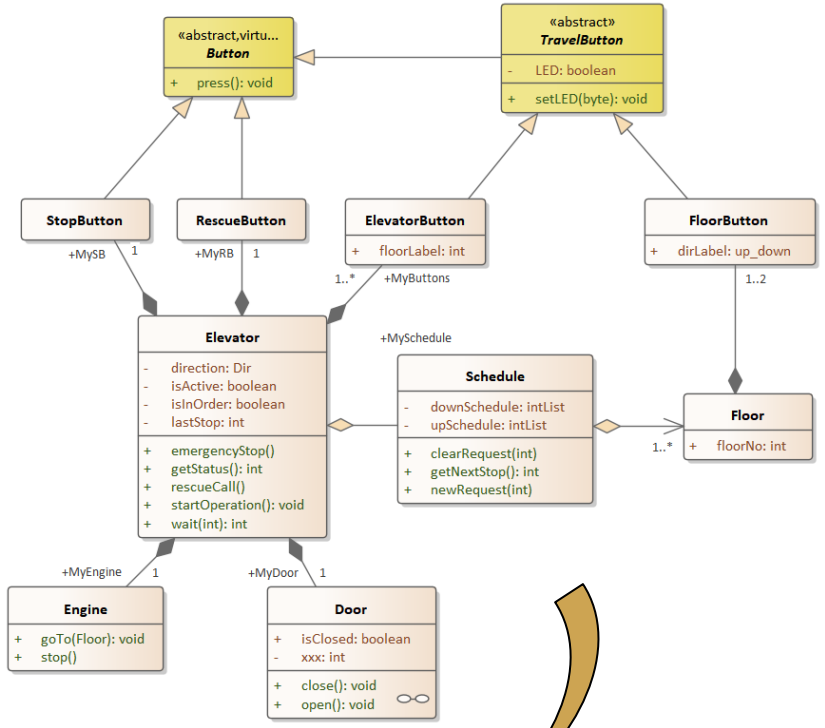
1. Find functionality in seq. diagrams



See next slide



2. Find classes, attributes, and methods

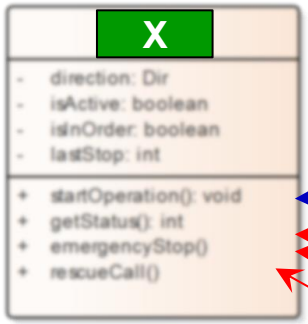


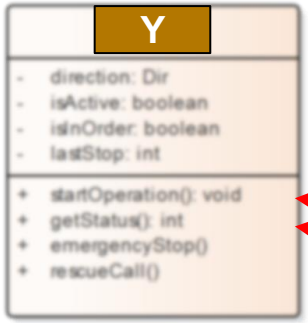
3. Implement functionality via interactions

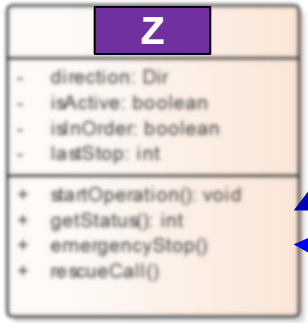
Assigning component functionality to classes

Build A = X+Z
Build B = X+Y+NewClass

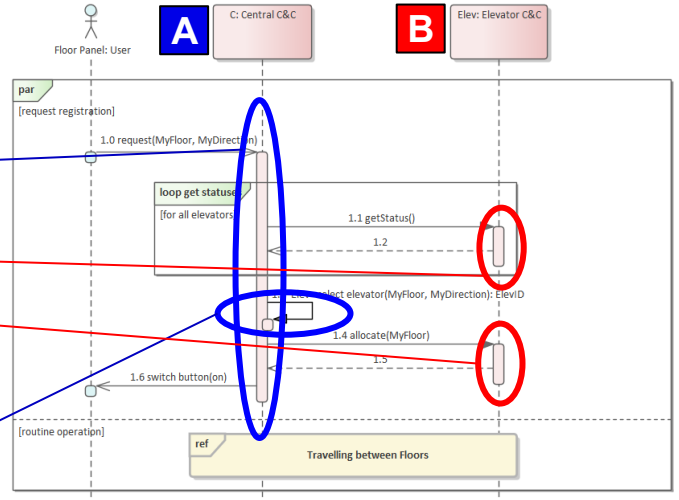
Classes initially identified in the PDOM

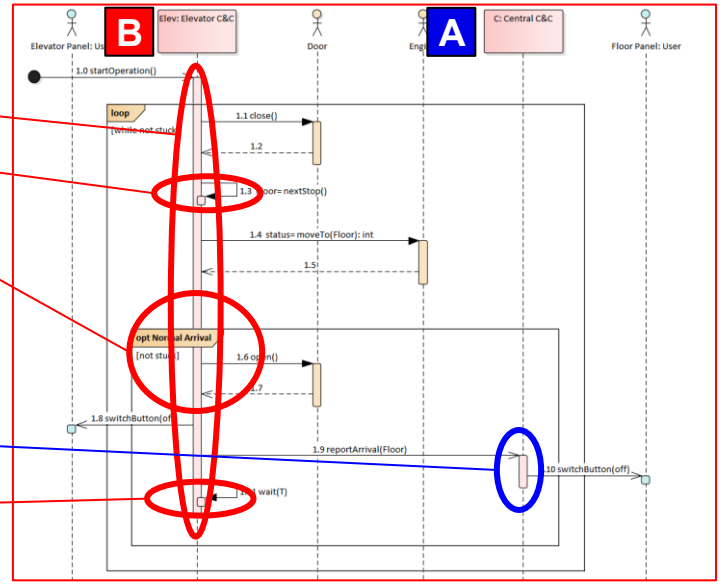






NewClass

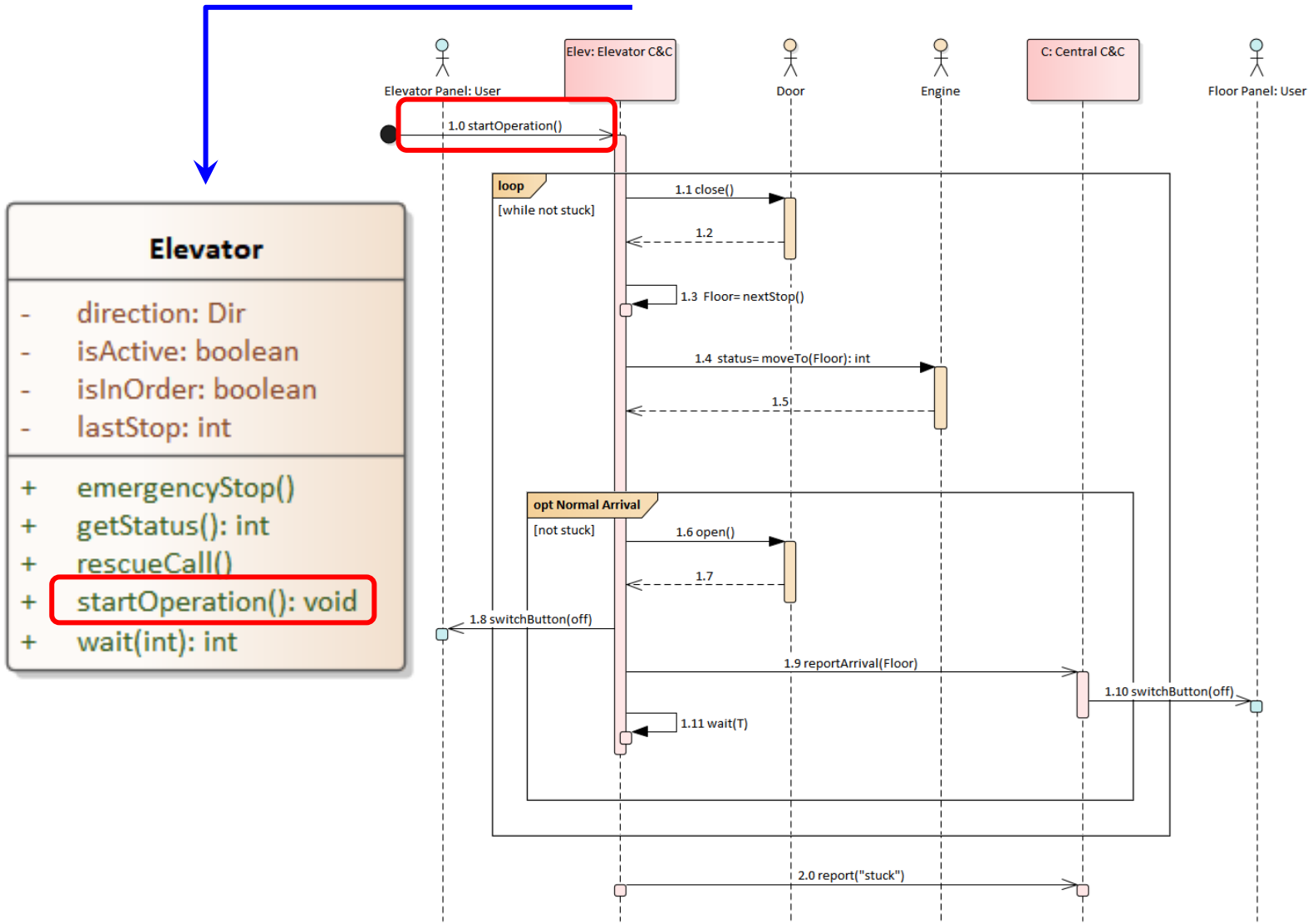




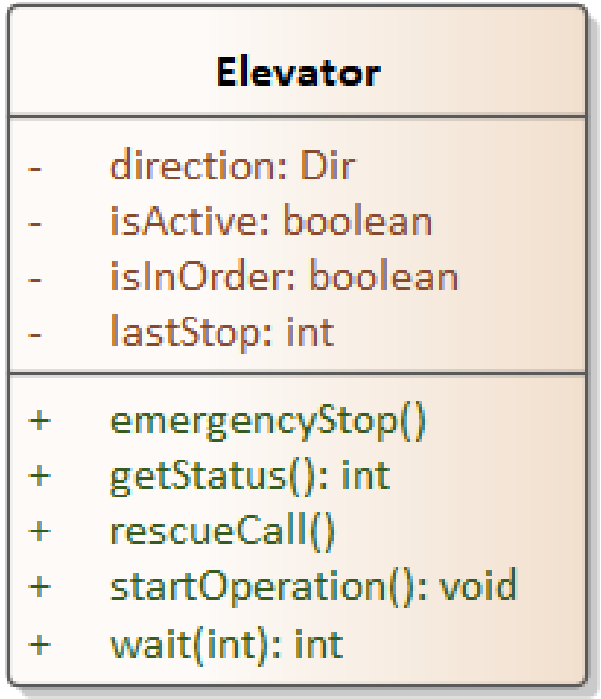
Sequence diagrams with components A and B

Example: Regular Elevator Operations

ref
Travelling between Floors



Elevator System: Elevator Class



In addition to these attributes, the pointers to other classes that are derived from associations and aggregations

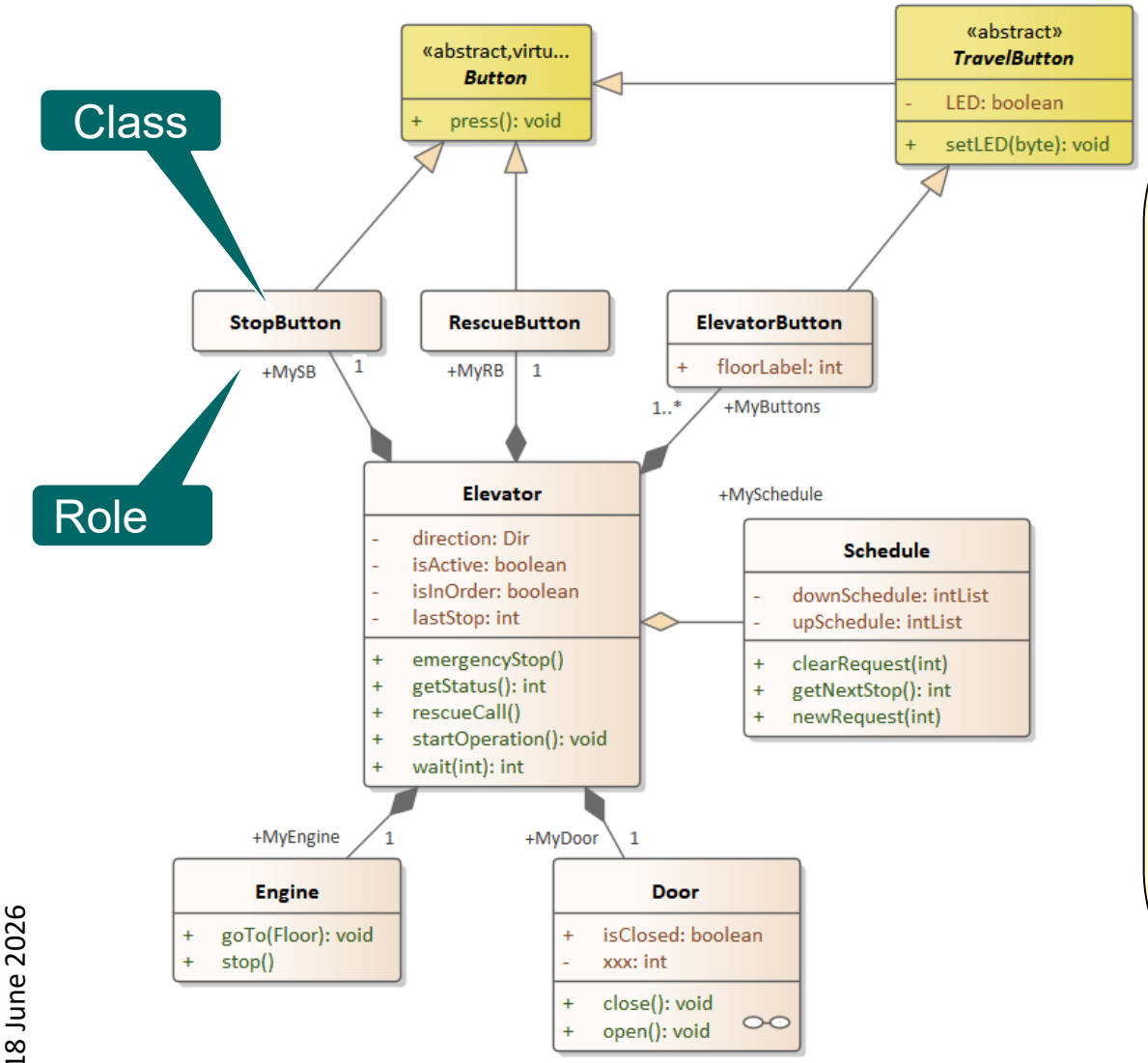
In addition to these attributes and methods, there are inherited attributes and methods

Tracking functional requirements to class diagram

- Classes in the class diagram must fulfill all system functionality

- Every functional requirement (FR) must point to one or more classes that fulfill it
  - Participate in an OR
    - E.g. “If the button wasn’t previously lit, it turns on after being pressed” → Button
  - Provide data structures for DR
    - E.g. “Every floor has two buttons” → Floor
- Each class in the class diagram must refer to the FR related to it

Automatic Static Code from Class Diagram – Attributes




```
public class Elevator {

    private Dir direction;
    private boolean isActive;
    private boolean isInOrder;
    private int lastStop;
    public ElevatorButton MyButtons;
    public Door MyDoor;
    public Schedule MySchedule;
    public RescueButton MyRB;
    public Floor ServedFloors;
    public Engine MyEngine;
    public StopButton MySB;
```

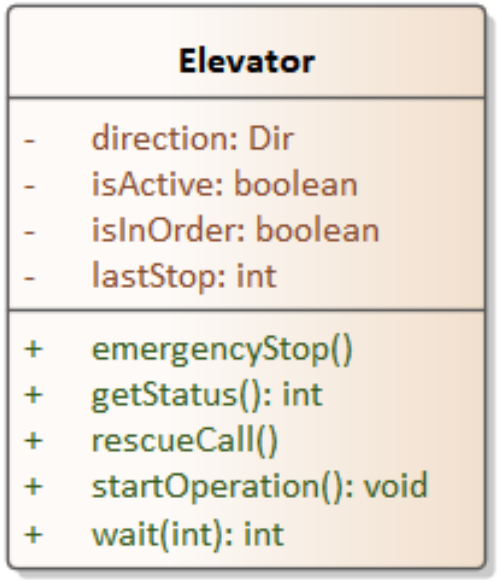
Explicitly defined attributes

Attributes derived from associations

Class

Role

Automatic Static Code from Class Diagram – Methods



Constructor

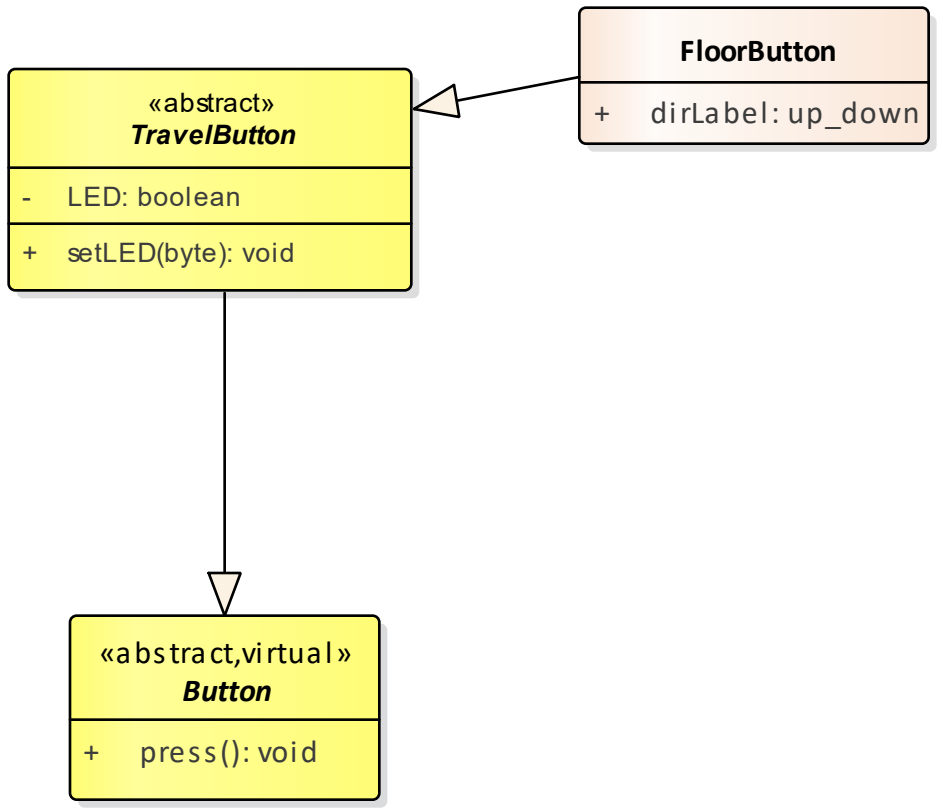
Destructor

Explicitly defined methods

```
public class Elevator {

    public Elevator(){ }
    public void finalize() throws Throwable { }
    public emergencyStop(){}
      public int getStatus(){
            return 0;
    }
    public rescueCall(){}
    public void startOperation(){ }
    public int wait(int Sec){}
```

Automatic Static Code from Class Diagram – Method Inheritance




```
public class FloorButton extends TravelButton {
        public up_down dirLabel;
        public FloorButton(){ }
        public void finalize() throws Throwable {
                super.finalize();
        }
}
```

```
public abstract class TravelButton extends Button {
        private boolean LED;
        public TravelButton(){ }
        public void finalize() throws Throwable {
                super.finalize();
        }
        public void setLED(byte on_off){ }
}
```

```
public abstract class Button {
        public Button(){ }
        public void finalize() throws Throwable { }
        public void press(){ }
}
```

In Class Assignment

- Create a class diagram for the base classes in ePark
  - Use the entities defined in the PDOM as classes
  - Use the methods shown in the sequence diagrams as a starting point
    - If you foresee other necessary methods, add them!
  - Add class attributes (think!)
  - Add getter and setter operations for one of the attributes at methods

Conclusion

- Class Model