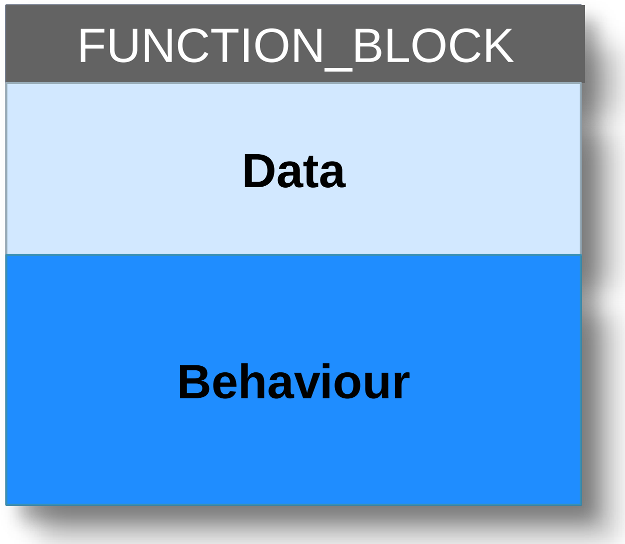
OOP Extensions for CODESYS

Agenda

Agenda

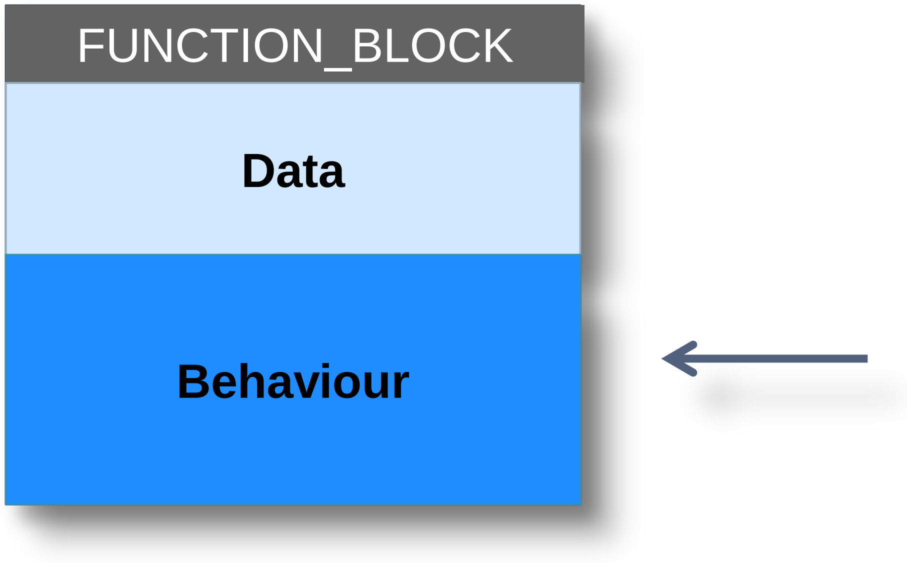
Class

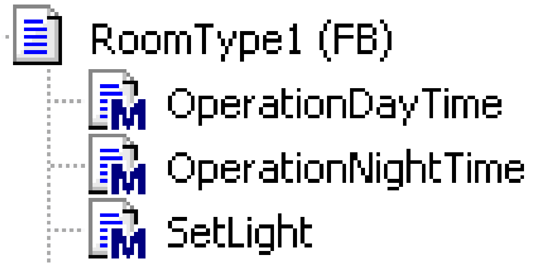
- The IEC 61131-3 already contains a simple class concept, the function block



Class

- A traditional function block contains only one routine



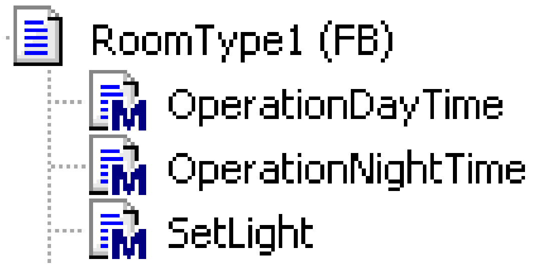


Now METHODs are available

Theory

Method

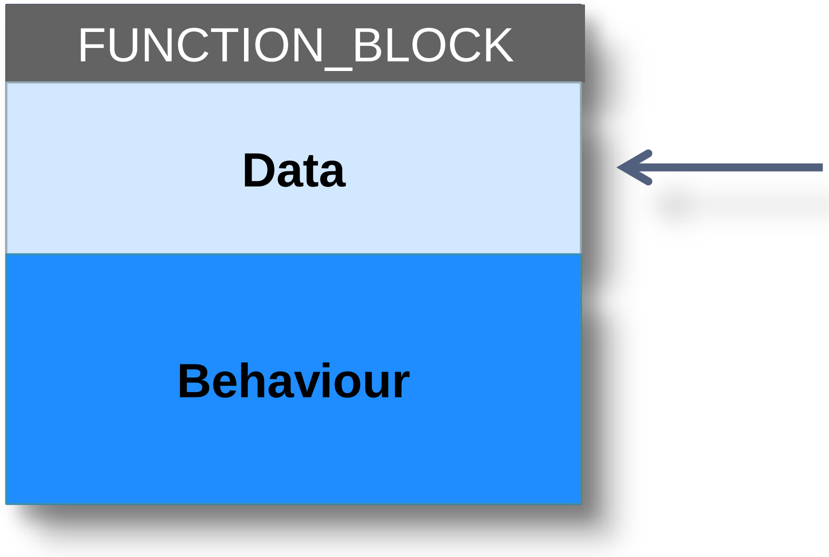
- can be used to describe a sequence of instructions
- a method is not an independent POU
- it can be regarded as a function within an instance of the respective functions block



```
METHOD SetLight : BOOL
VAR_INPUT
        xValue  : BOOL;
END_VAR
```

Class

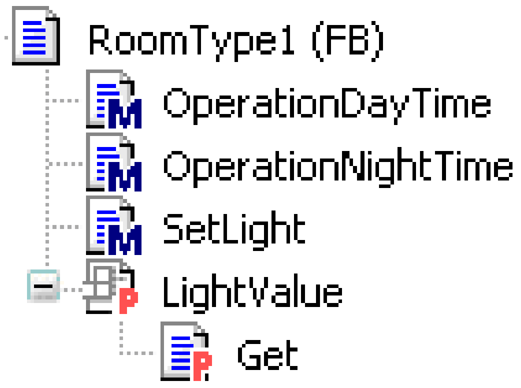
- A traditional function block contains data



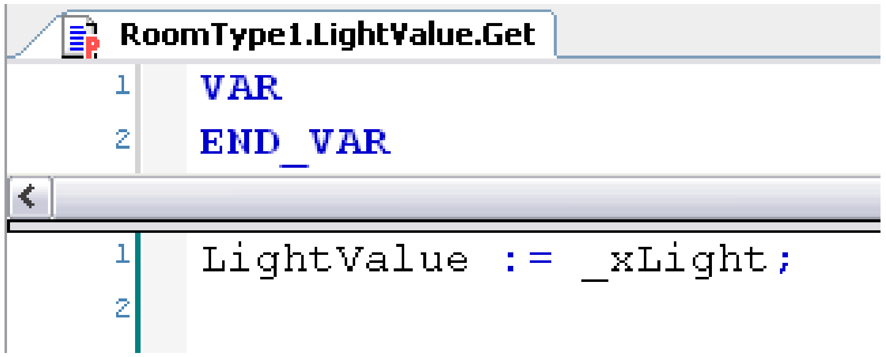
Now Properties are available

Property

- consists of a pair of “accessor methods“ (get, set)
- a property can have additional local variables
- But no additional inputs and – in contrast to a FUNCTION or METHOD – no additional outputs



```
RoomType1 (FB)
    OperationDayTime
    OperationNightTime
    SetLight
    LightValue
        Get
```



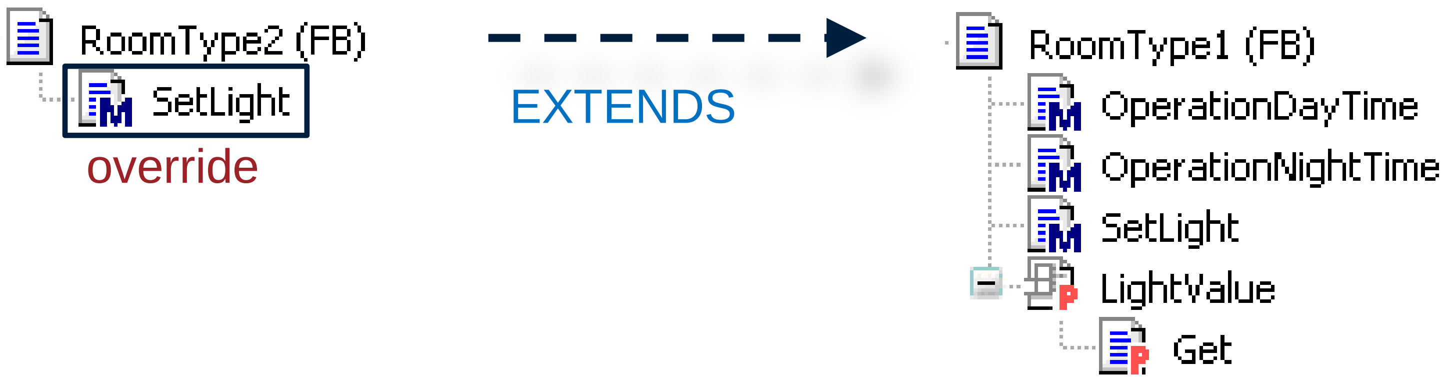
RoomType1.LightValue.Get

```
VAR
END_VAR
```

```
LightValue := _xLight;
```

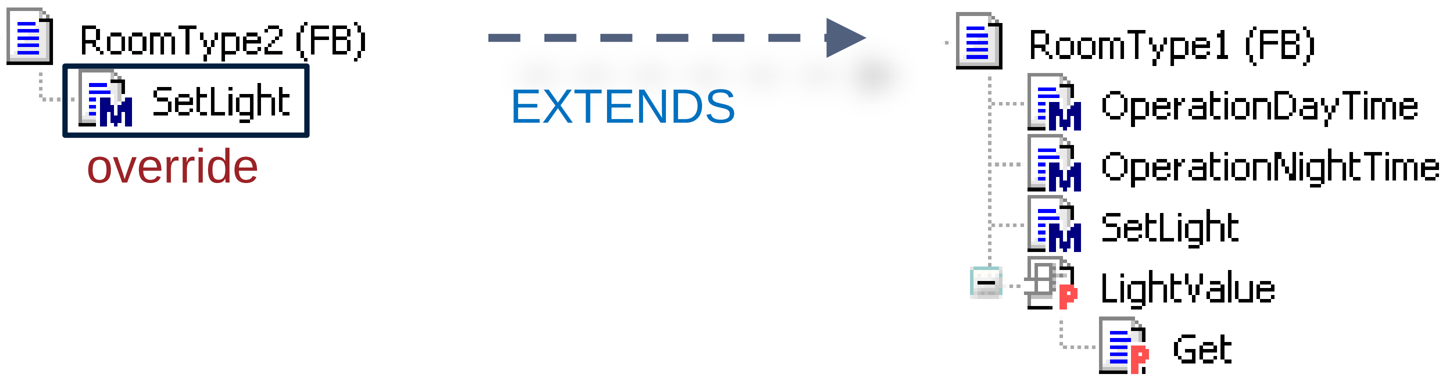
Inheritance / Extends

- A new FB may be constructed by inheritance. This means:
  the new function block inherits all variables and methods of the old function block
- EXTENDS in a function block turn it into the subclass of another function block
- The new FB may define additional variables and additional methods and it may override methods of its parent



Pointer SUPER and THIS

- SUPER offers access to the methods of the base class implementation, e.g. SUPER^.SetLight(TRUE);
- THIS points to its own FB instance.
  It may only be used in methods and in the associated FB implementation



Exercise 1

Introduction

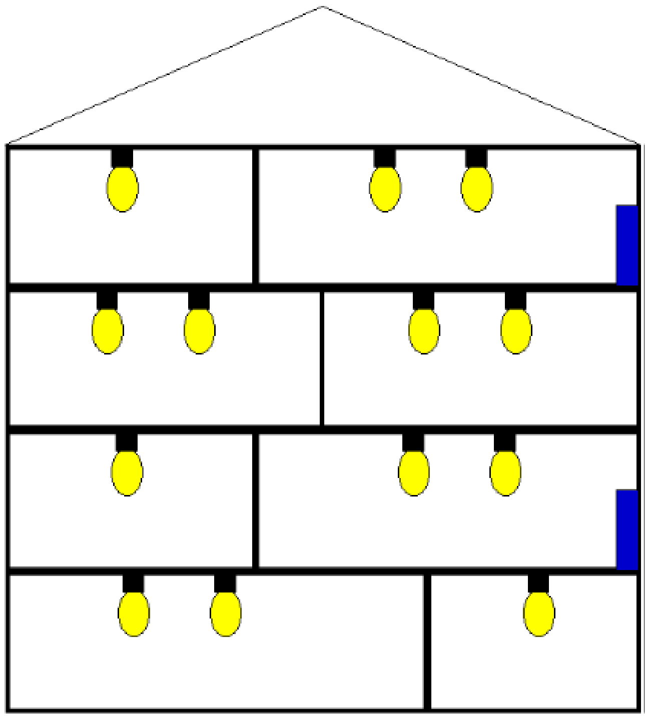
- In the exercise we will go throw the OOP extensions for CODESYS. The scenario is to control a house with different type of rooms.

Operation daytime

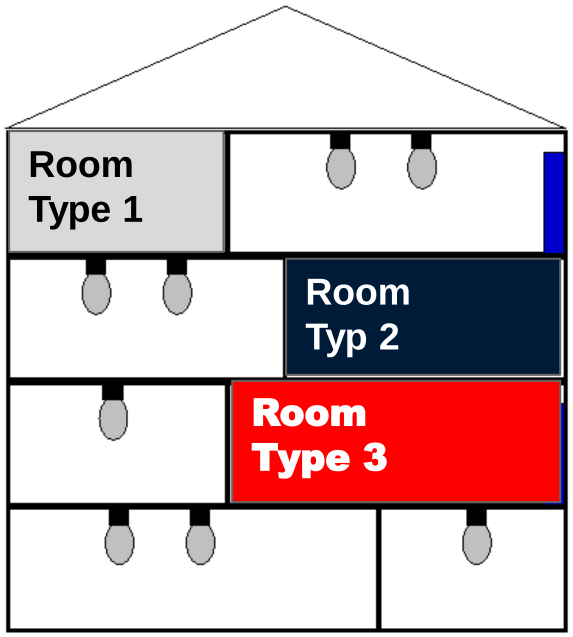
Operation nighttime

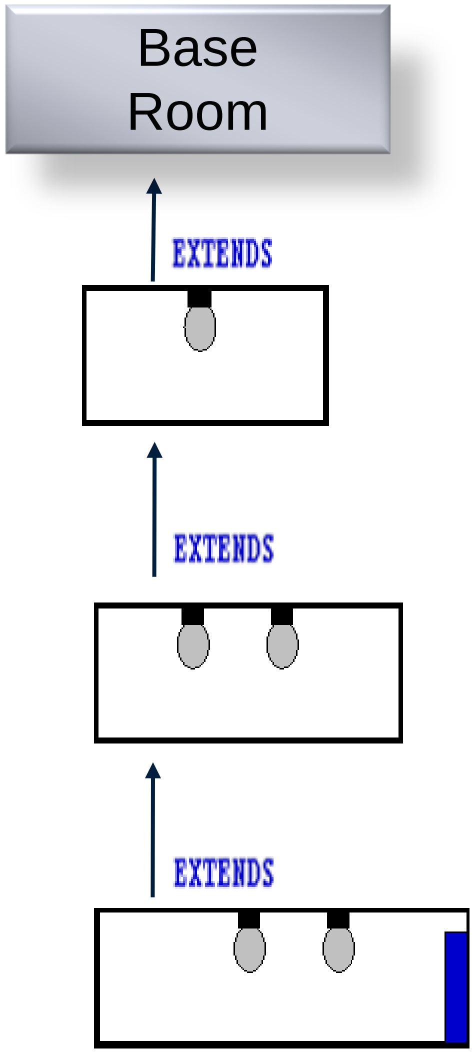
Operation daytime

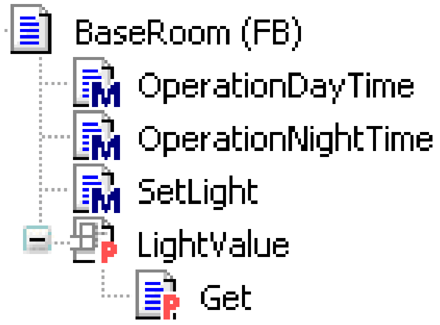
Operation nighttime

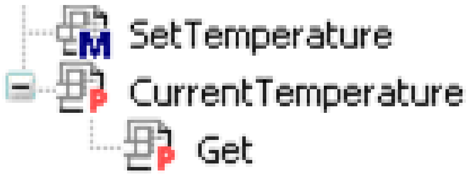


Exercise 1


Room Type 1
Room Typ 2
Room Type 3

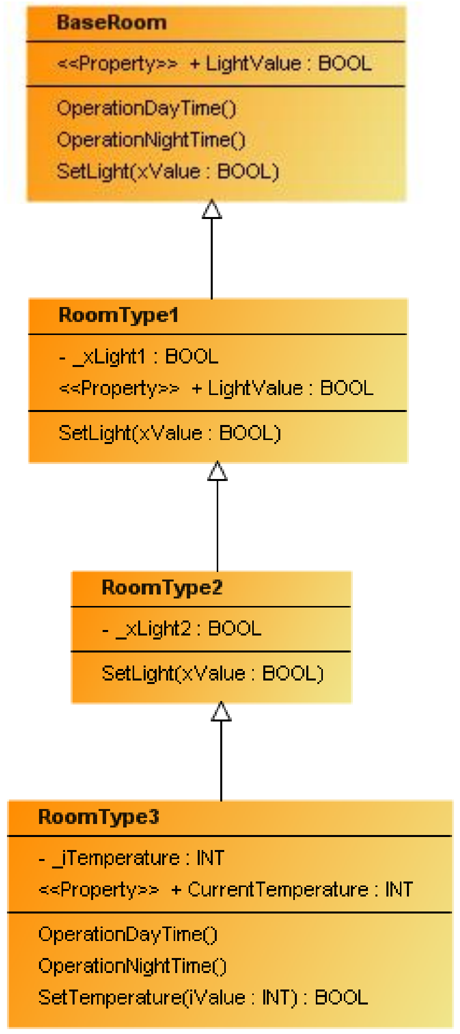

Base Room
EXTENDS
EXTENDS
EXTENDS


BaseRoom (FB)
OperationDayTime
OperationNightTime
SetLight
LightValue
Get

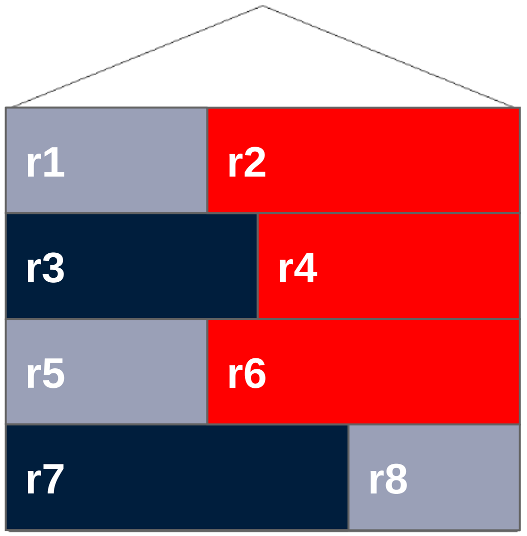

SetTemperature
CurrentTemperature
Get

Exercise 1

Please create a project that fits to the class diagram



Exercise 2



```
rm1  :  RoomType1;
rm2  :  RoomType1;
rm3  :  RoomType1;
rm4  :  RoomType2;
rm5  :  RoomType2;
rm6  :  RoomType2;
rm7  :  RoomType3;
rm8  :  RoomType3;

apRoom : ARRAY[1..8] OF POINTER TO BaseRoom
      := [
          ADR(rm1),
          ADR(rm2),
          ADR(rm3),
          ADR(rm4),
          ADR(rm5),
          ADR(rm6),
          ADR(rm7),
          ADR(rm8)
          ];
```

Exercise 2

```
apRoom : ARRAY[1..8] OF POINTER TO BaseRoom
    := [
        ADR(rm1),
        ADR(rm2),
        ADR(rm3),
        ADR(rm4),
        ADR(rm5),
        ADR(rm6),
        ADR(rm7),
        ADR(rm8)
        ];
```

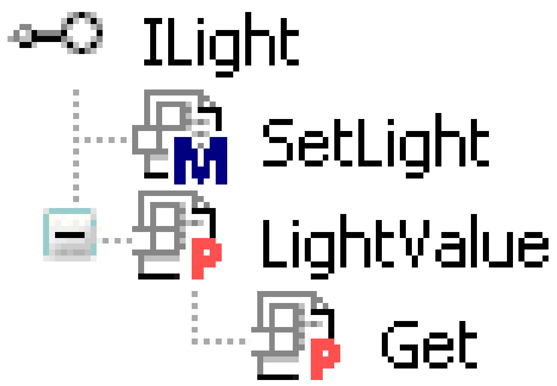
Polymorphism

Handle different objects with same base type

```
FOR i := 1 TO 8 DO
    IF xDayTime THEN
        apRoom[i]^.OperationDayTime();
    ELSE
        apRoom[i]^.OperationNightTime();
    END_IF
END_FOR
```

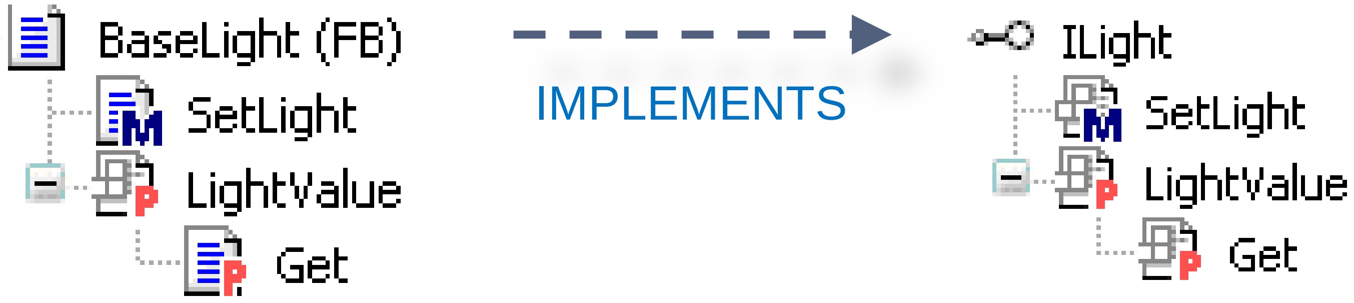
Interface

- An interface is similar to a function block with subordinated method prototypes
- The interface does not have local variables or an implementation part
- Only input, output and InOut variables are allowed

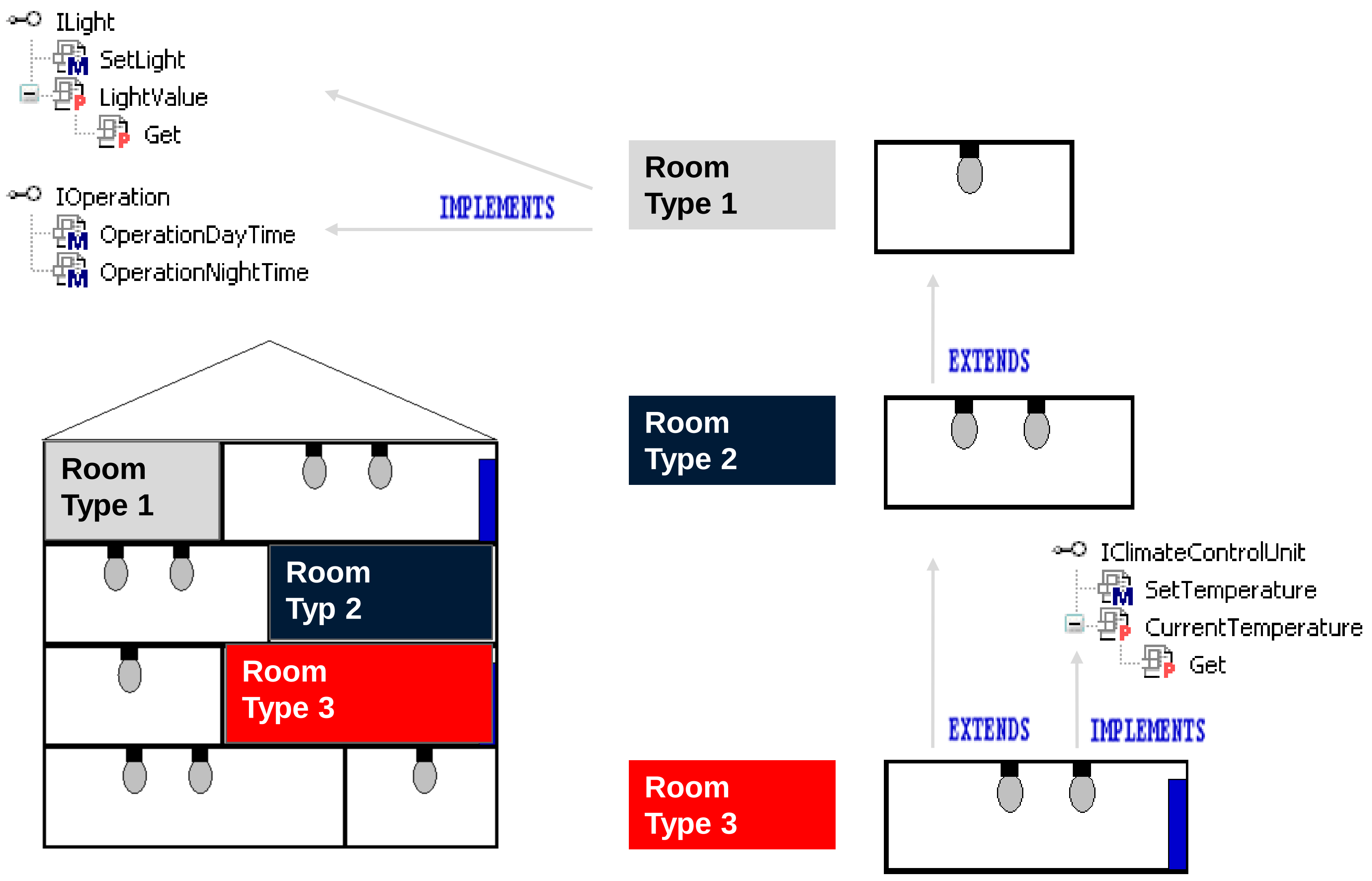


Implements

- The keyword IMPLEMENTS in a function block turns it into the subclass of one or more interfaces
- An IMPLEMENTS declaration requires the function block to have at least all the methods which the named interfaces with the same parameter and result types



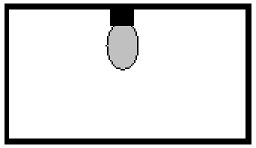
Exercise 2

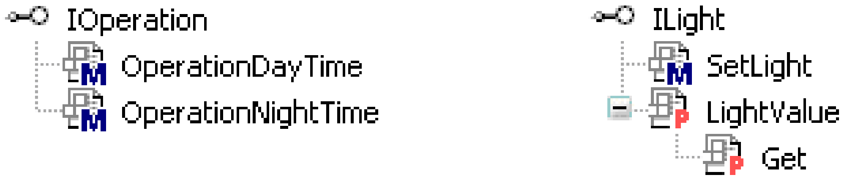

ILight
SetLight
LightValue
Get
IOperation
OperationDayTime
OperationNightTime
IMPLEMENTS
Room Type 1
EXTENDS
Room Type 2
IClimateControlUnit
SetTemperature
CurrentTemperature
Get
EXTENDS
IMPLEMENTS
Room Type 3
Room Type 1
Room Typ 2
Room Type 3

Exercise 2

```
FUNCTION_BLOCK RoomType1 IMPLEMENTS IOperation, ILight
```

Room Type 1



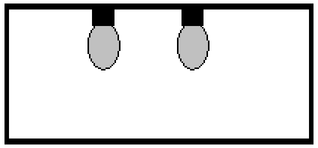


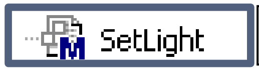
IOperation
- OperationDayTime
- OperationNightTime

ILight
- SetLight
- LightValue
  - Get

```
FUNCTION_BLOCK RoomType2 EXTENDS RoomType1
```

Room Type 2

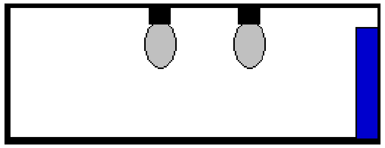


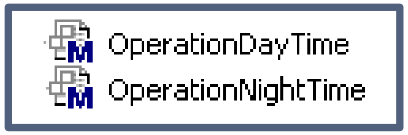


SetLight **overwriting**

```
FUNCTION_BLOCK RoomType3 EXTENDS RoomType2 IMPLEMENTS IClimateControlUnit
```

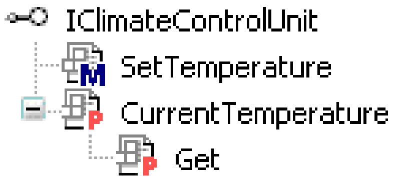
Room Type 3





OperationDayTime
OperationNightTime

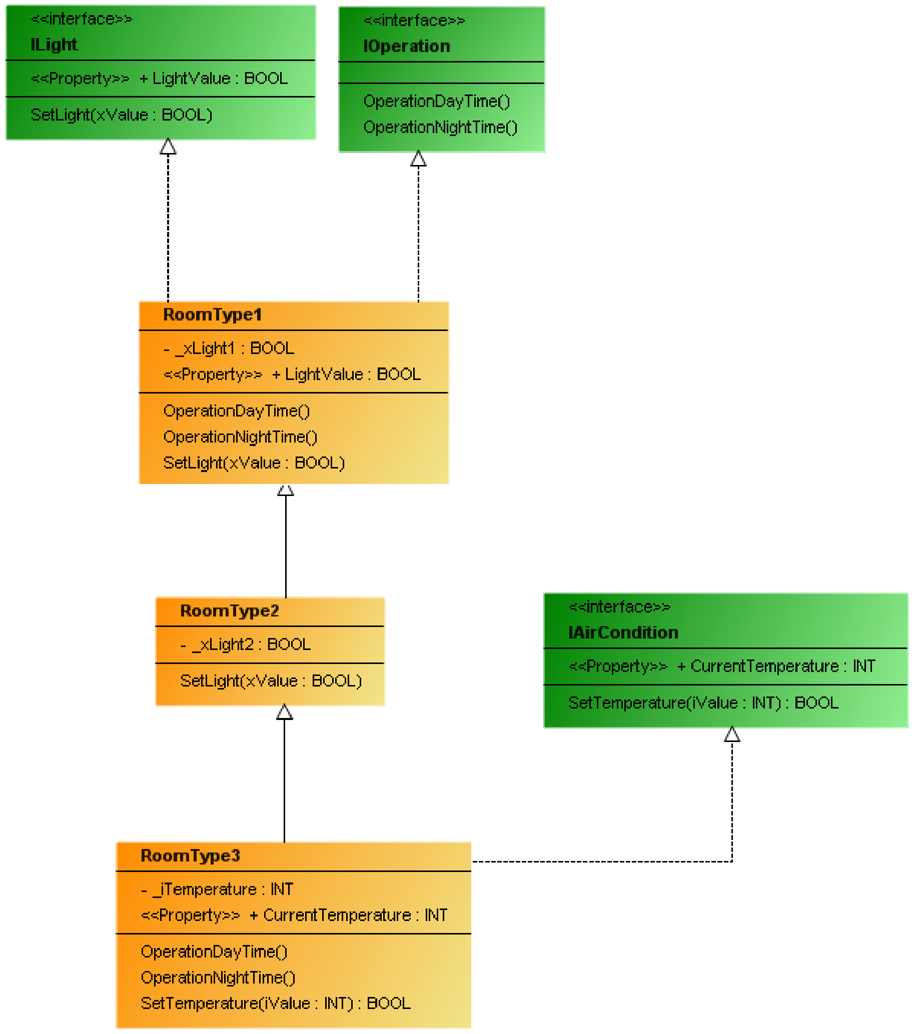
overwriting



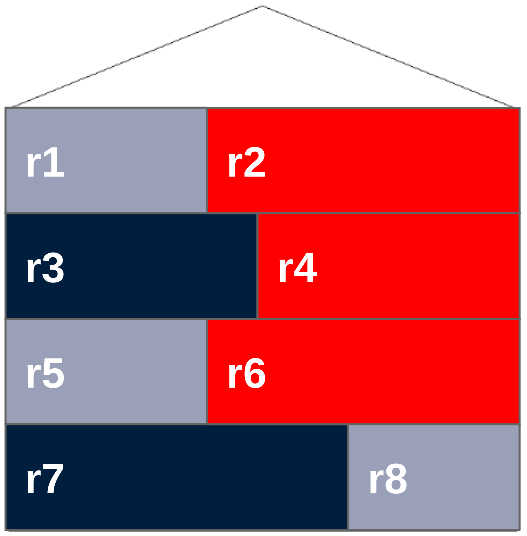
IClimateControlUnit
- SetTemperature
- CurrentTemperature
  - Get

Exercise 2

Please create a project that fits to the class diagram



Exercise 2



```
Room1 : RoomType1;
Room2 : RoomType3;
Room3 : RoomType2;
Room4 : RoomType3;
Room5 : RoomType1;
Room6 : RoomType3;
Room7 : RoomType2;
Room8 : RoomType1;

aitfOperation : ARRAY[1..8] OF IOperation
    := [
            Room1,
            Room2,
            Room3,
            Room4,
            Room5,
            Room6,
            Room7,
            Room8
       ];
```

Exercise 2

```
aitfOperation : ARRAY[1..8] OF IOperation
    := [
            Room1,
            Room2,
            Room3,
            Room4,
            Room5,
            Room6,
            Room7,
            Room8
        ];
```

Polymorphism

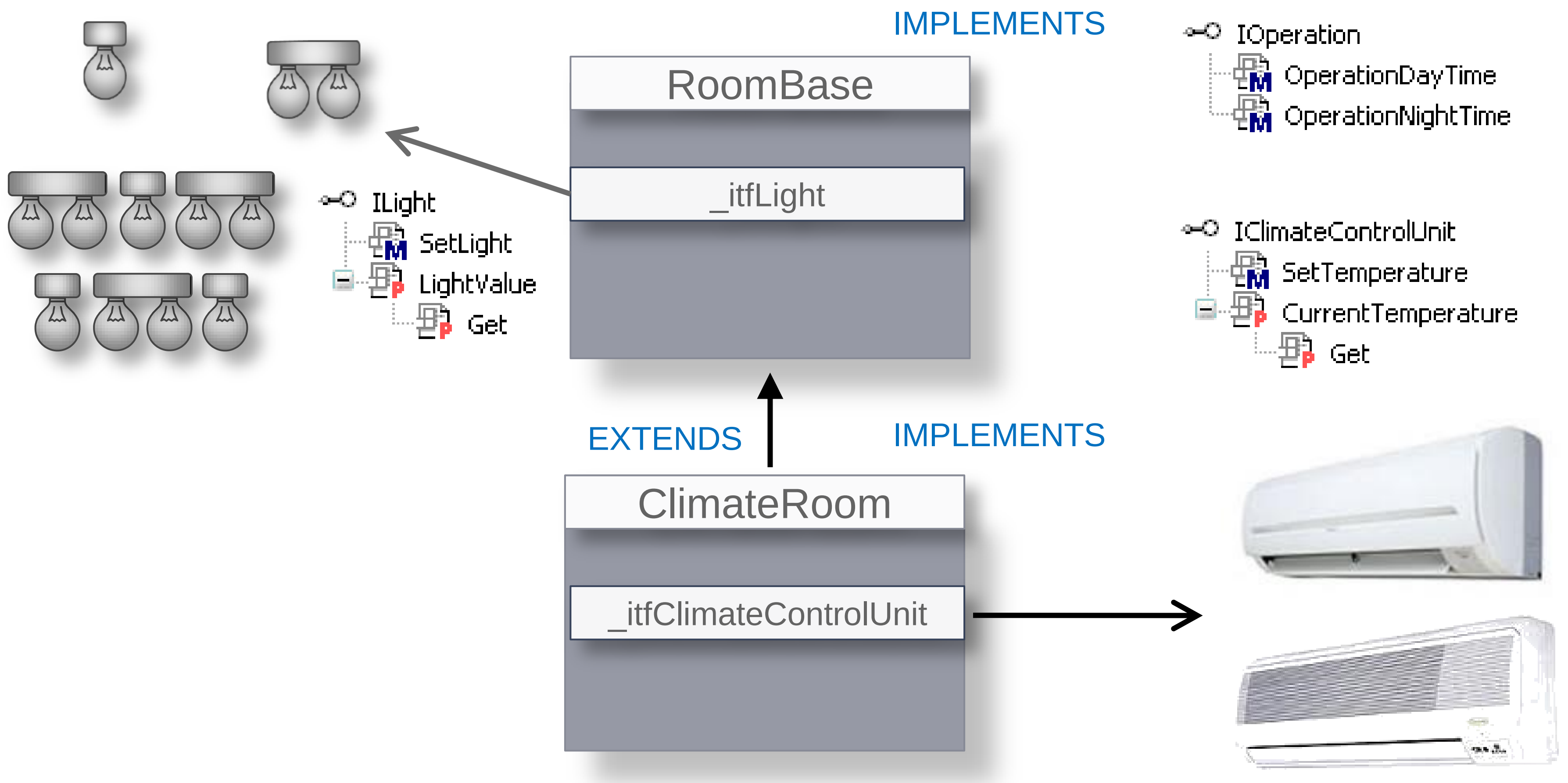
```
FOR ui := 1 TO 8 DO
    IF xDayTime THEN
        aitfOperation[ui].OperationDayTime();
    ELSE
        aitfOperation[ui].OperationNightTime();
    END_IF
END_FOR
```

Handle different objects with same interface

Exercise 3

We change the design

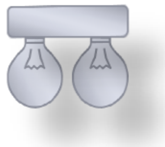
- The setup of the room is configurable too

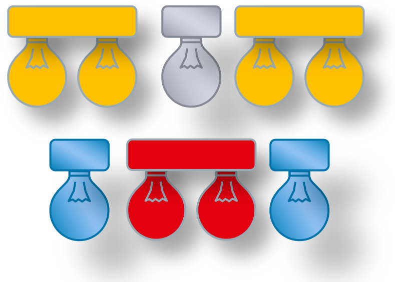


Exercise 3

We change the design

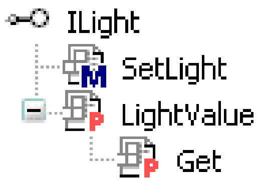
- There are different kind of lights available
  - SingleLight
  - DuoLight
  - ArrayLight
    It consist of other lights and is configurable

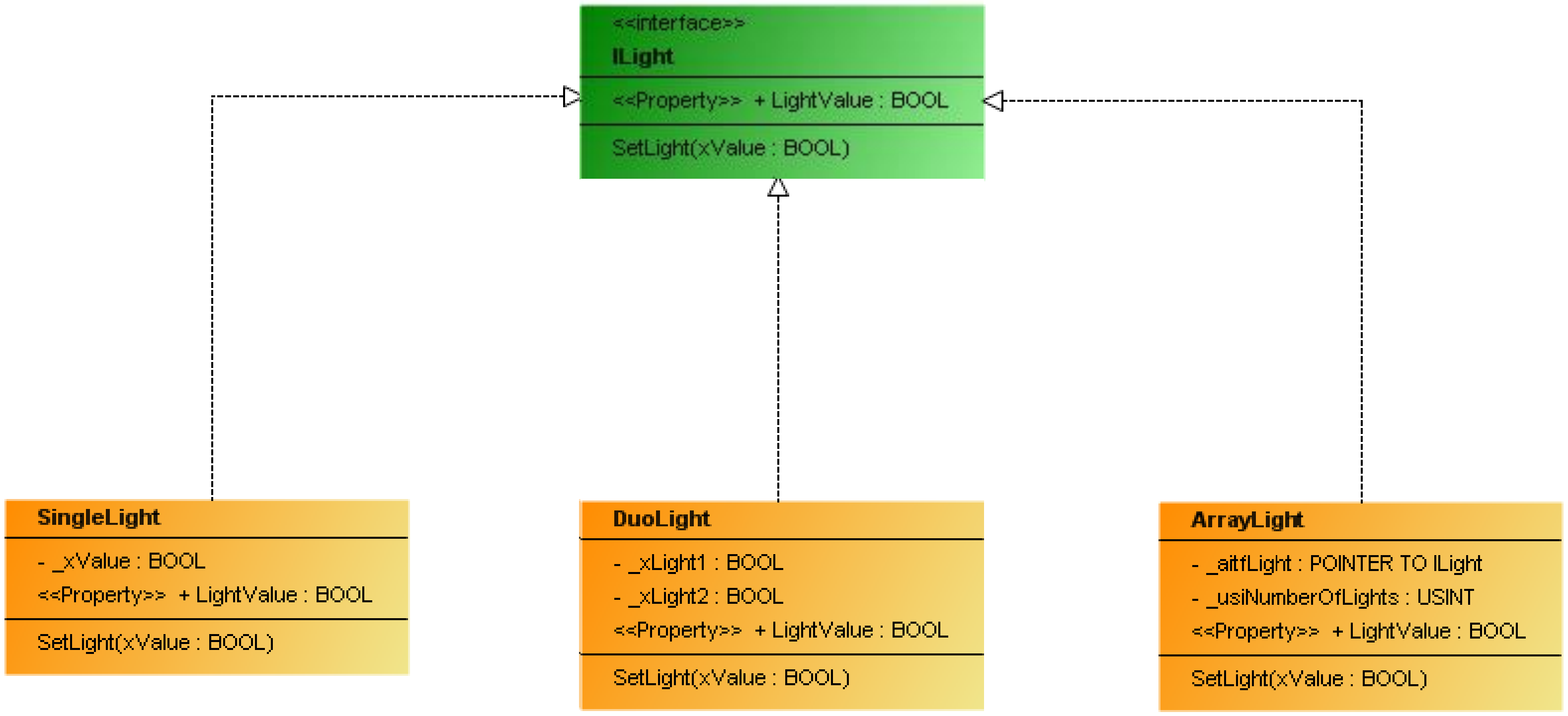




Exercise 3

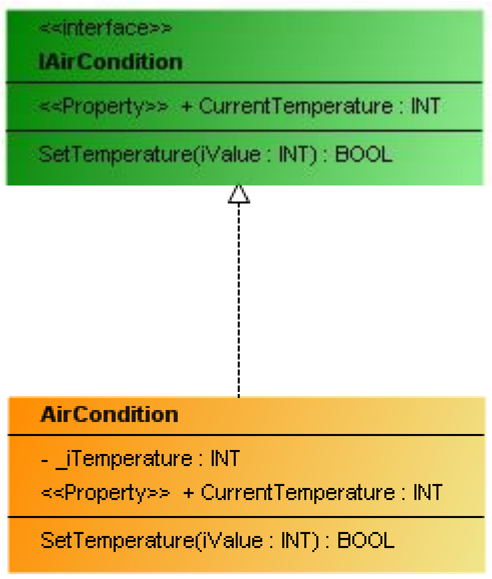
class diagram lights



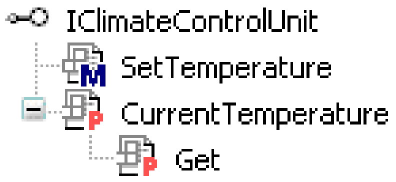


Exercise 3

class diagram air condition

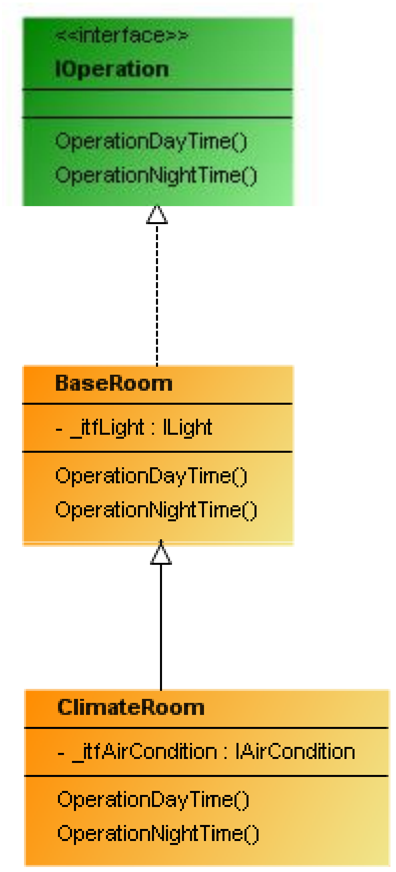


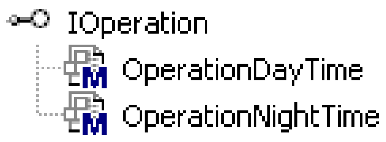
<<interface>>
IAirCondition
<<Property>> + CurrentTemperature : INT
SetTemperature(iValue : INT) : BOOL
AirCondition
- _iTemperature : INT
<<Property>> + CurrentTemperature : INT
SetTemperature(iValue : INT) : BOOL



Exercise 3

class diagram rooms





Summary

- Real life objects contain state and behavior.
- A software object's behavior is exposed through methods .
- Hiding internal data from the outside world, and accessing it only through publicly exposed methods is known as data encapsulation .
- A blueprint for a software object is called a class .
- Common behavior can be defined in a superclass and inherited into a subclass using the extends keyword.
- A collection of methods with no implementation is called interface .

Inspiring Automation Solutions

Thank you for your attention.