Design Patterns & Software Architecture: Labs

Course period: 2018-2019
Insititute: Sofware College, Northeastern University, Shenyang
Instructor: dr. Joost Schalken-Pinkster

Overall lab instructions:

- Please form groups of three students each to work on the labs of this course.
- You are not allowed to switch groups during the course, so when groups are formed, they are final.
- Please go the Canvas course site and register your student group (the Lab Groups can be viewed in the tab on the People page of the course).
- Each group should make their own solution to the labs, copying of other people's work is not allowed.
- Due to filing regulations, which require an archival copy of lab report for each student, so I need you to do a group submission (pdf lab report + zip file containing the source) and a personal submission (pdf lab report).
- Only the group submission will be graded, but I need the personal submission before I can give a grade!

- For each lab assignment, make a group pdf lab report, with the name NEU-DPSA-2018-2019-LAB-REPORT-***<groupnumber>***-***<lab-no>***.pdf
- For each lab assignment, make a group pdf lab report, with the name NEU-DPSA-2018-2019-LAB-PERSONAL-REPORT-***<studentnumber>***-***<lab-no>***.pdf
- For the lab report:
  - Make sure your group lab report file contains all the students' English names and student numbers.
  - Make sure your personal lab report file contains only your English name and student number.
- For each lab assignment, make a zip file, with the name NEU-DPSA-2018-2019-LAB-SOURCES-***<groupnumber>***-***<lab-no>***.pdf
- For the source code zip:
  - Make sure to remove all binaries (.jar, .class) which you built before making the source zip file(to reduce space)
  - Use a reasonable layout for your project (e.g. source code in src/ or src/main/java)
  - Make sure your project builds with out problems (e.g. use Maven or Ant)

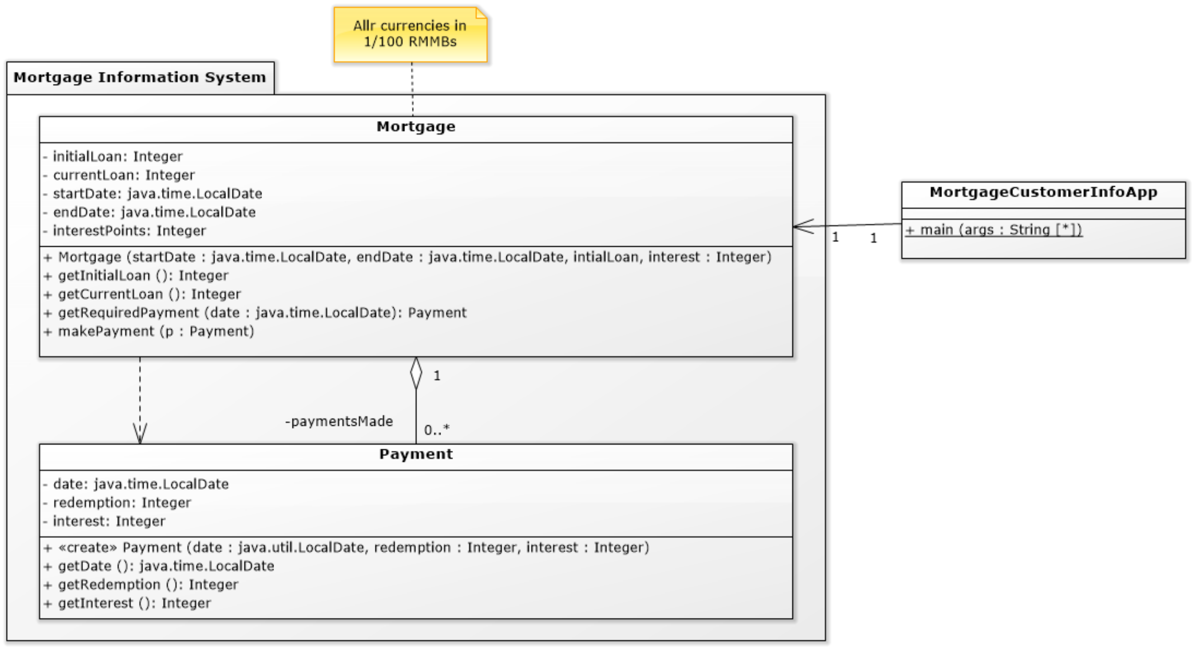
- Please hand in lab solutions on or before Thursday December 27th 2018 (no later than 22:00u).
- Late submissions cannot be graded…

Lab assignment 1: Mortgages

Suppose we want to update an application for a bank. The bank has a successful app, the MortgageCustomerInfoApp, that allows a customer to query what the outstanding balance on a loan is and to request what payment need to be made (by providing a date, of which only month and year are relevant, since all payments are due on the first of the month). A payment consist of interest (which is for the bank to pay for borrowing) and a repayment (to decrease the loan).

The MortgageCustomerInfoApp is however talking to an outdated backend, the Mortgage Information System. The Mortgage Information System is only capable of handling linear mortgages (which means the repayment is the initialLoan devided by the loan period, i.e. endDate-startDate). The interest is computed by multiplying currentLoan times interestPoints (which are 1/100 of % of interest per year) * 1/12. These calculations are performed in getRequiredPayment().

Suppose we want to upgrade the Mortgage Information System to accept different loan forms (e.g. amortized loans or loans without repayments –i.e. the repayment = 0--). Please use the Strategy pattern to add different ways of paying for a Mortgage.



Allr currencies in 1/100 RMMBs
Mortgage Information System
Mortgage
- initialLoan: Integer
- currentLoan: Integer
- startDate: java.time.LocalDate
- endDate: java.time.LocalDate
- interestPoints: Integer
+ Mortgage (startDate : java.time.LocalDate, endDate : java.time.LocalDate, intialLoan, interest : Integer)
+ getInitialLoan (): Integer
+ getCurrentLoan (): Integer
+ getRequiredPayment (date : java.time.LocalDate): Payment
+ makePayment (p : Payment)
MortgageCustomerInfoApp
+ main (args : String [*])
1
1
1
-paymentsMade
0..*
Payment
- date: java.time.LocalDate
- redemption: Integer
- interest: Integer
+ «create» Payment (date : java.util.LocalDate, redemption : Integer, interest : Integer)
+ getDate (): java.time.LocalDate
+ getRedemption (): Integer
+ getInterest (): Integer

What you need to implement:

You need to create a Java classes/interfaces called 'Mortgage', 'Payment', 'MortgageStrategy', 'LinearMortgageStrategy', another '….MortgageStrategy' and a simple 'MortgageCustomerInfoApp', that can display (using System.out.println) the mortgage information.

What you need to hand in:

1. The source code of the solution.
2. A lab report of the solution (both a group report and a personal report)
3. Place a UML class diagram of the source code in your lab report document.
4. Explain in less than 200 words (less is better) the working of your source code in code in your lab report document.

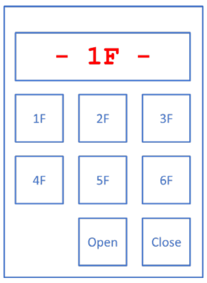
Extra work:

1. Make a build script (Maven, Ant, Gradle) for your lab.
2. Create JUnit tests for your classes

3. Add an Observer pattern to Mortgage, so applications can notice when payments have been made.
4. Create GUI instead of System.out.println system

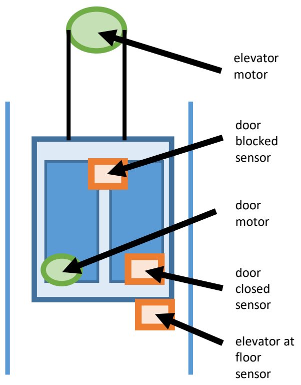
Lab assignment 2: Elevator control system

Let's design a software control software system for the cart of an elevator. The elevator cart has a control panel consisting of buttons for each floor (1 to 6). In addition the control panel has a button "Open" and a button "Close" and a display to show the current floor.

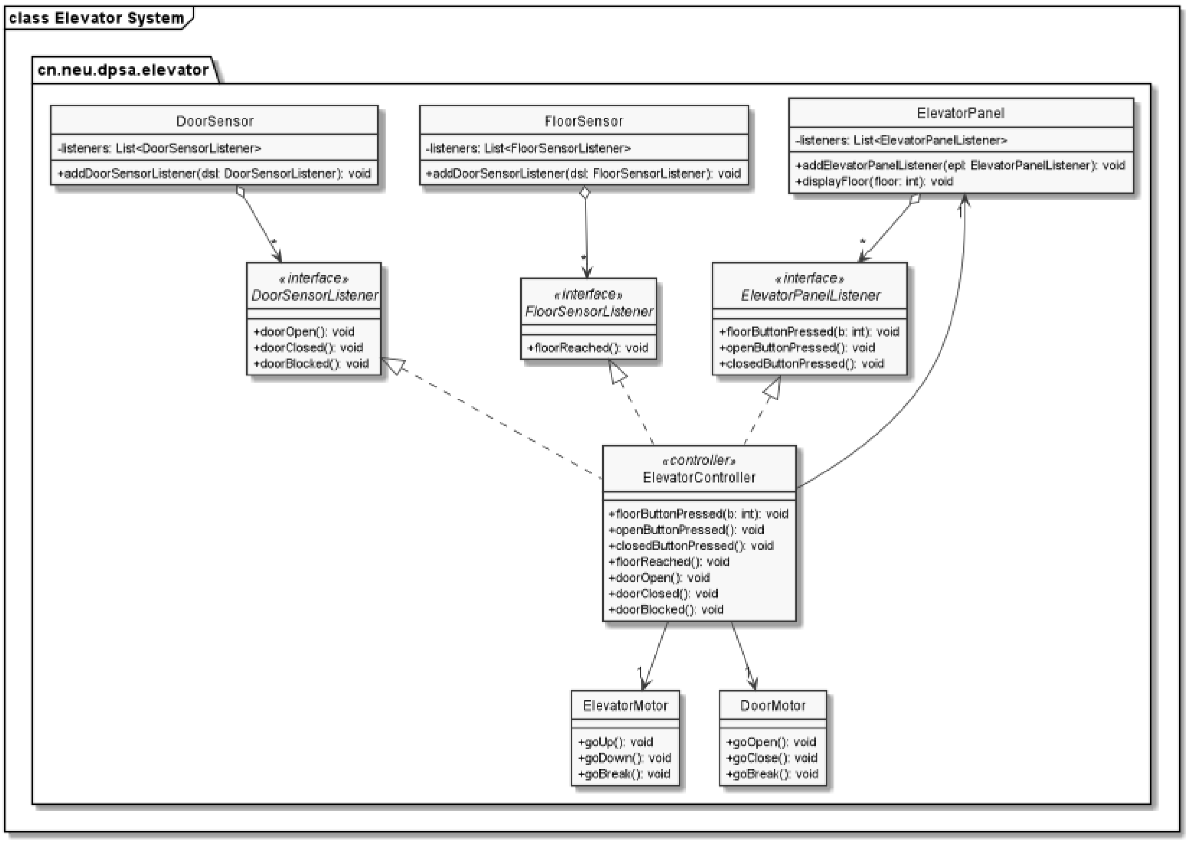


In addition, the control software receives input from a door status sensor (is the door closed or not), a door blocked sensor (which detects if something is blocking the closing of the door) and a floor sensor (which detects --with light-- if the elevator cart is currently at a floor or between floors).

Response to the inputs, the Elevator drives the motors of the elevator cart (go up, go down, brake) and drives the motors of the elevator door (open, close, brake).



As simple class diagram of the system is as follows:

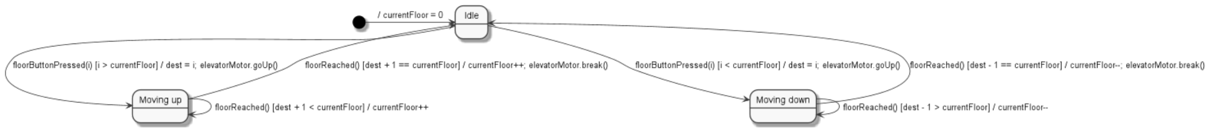


class Elevator System
cn.neu.dpsa.elevator
DoorSensor
-listeners: List<DoorSensorListener>
+addDoorSensorListener(dsl: DoorSensorListener): void
FloorSensor
-listeners: List<FloorSensorListener>
+addDoorSensorListener(dsl: FloorSensorListener): void
ElevatorPanel
-listeners: List<ElevatorPanelListener>
+addElevatorPanelListener(epl: ElevatorPanelListener): void
+displayFloor(floor: int): void
*
*
*
1
«interface»
DoorSensorListener
+doorOpen(): void
+doorClosed(): void
+doorBlocked(): void
«interface»
FloorSensorListener
+floorReached(): void
«interface»
ElevatorPanelListener
+floorButtonPressed(b: int): void
+openButtonPressed(): void
+closedButtonPressed(): void
«controller»
ElevatorController
+floorButtonPressed(b: int): void
+openButtonPressed(): void
+closedButtonPressed(): void
+floorReached(): void
+doorOpen(): void
+doorClosed(): void
+doorBlocked(): void
1
1
ElevatorMotor
+goUp(): void
+goDown(): void
+goBreak(): void
DoorMotor
+goOpen(): void
+goClose(): void
+goBreak(): void

Simple Elevator Cart control software

Assume that everytime the lift control software is started, the lift cart is located at the ground floor (1F). If the lift motor makes the lift cart go up, each time the lift cart gets a signal that a floor is reached, the lift cart should increase the current floor at which the lift is. In the same way the if the lift motor makes the lift go down, each time the lift cart gets a signal that a floor is reached, the lift cart should decrease the current floor. If the destination is reached, the system should brake and go idle.

A (simplified) State machine diagram of this can be seen below:


/ currentFloor = 0
Idle
floorButtonPressed(i) [i > currentFloor] / dest = i; elevatorMotor.goUp()
floorReached() [dest + 1 == currentFloor] / currentFloor++; elevatorMotor.break()
floorButtonPressed(i) [i < currentFloor] / dest = i; elevatorMotor.goUp()
floorReached() [dest - 1 == currentFloor] / currentFloor--; elevatorMotor.break()
Moving up
floorReached() [dest + 1 < currentFloor] / currentFloor++
Moving down
floorReached() [dest - 1 > currentFloor] / currentFloor--

(if you do not know State machine diagram, please visit https://www.lucidchart.com/pages/uml-state-machine-diagram)

If the lift cart is moving, no open or close button should function. If the lift cart is idle, you can open and close the door, but if it is blocked, you should break the door motor for 5 seconds.

Please be flexible with the behaviour, for example the system should not need to be fully redone when you want to add a maintenance operations mode or a failure operations mode.

Please implement this code using suitable design patterns.

What you need to hand in:

1. The source code of the solution.
2. A lab report of the solution (both a group report and a personal report)
3. Place a UML class diagram of the source code in your lab report document.
4. Explain in less than 500 words (less is better) the working of your source code in code in your lab report document.

Extra work:

1. Make a build script (Maven, Ant, Gradle) for your lab.
2. Create JUnit tests for your classes
3. Create GUI instead of System.out.println system
4. Create state machine diagrams for all behaviour.