CSC 7003 : Basics of Software Engineering

J Paul Gibson, D311

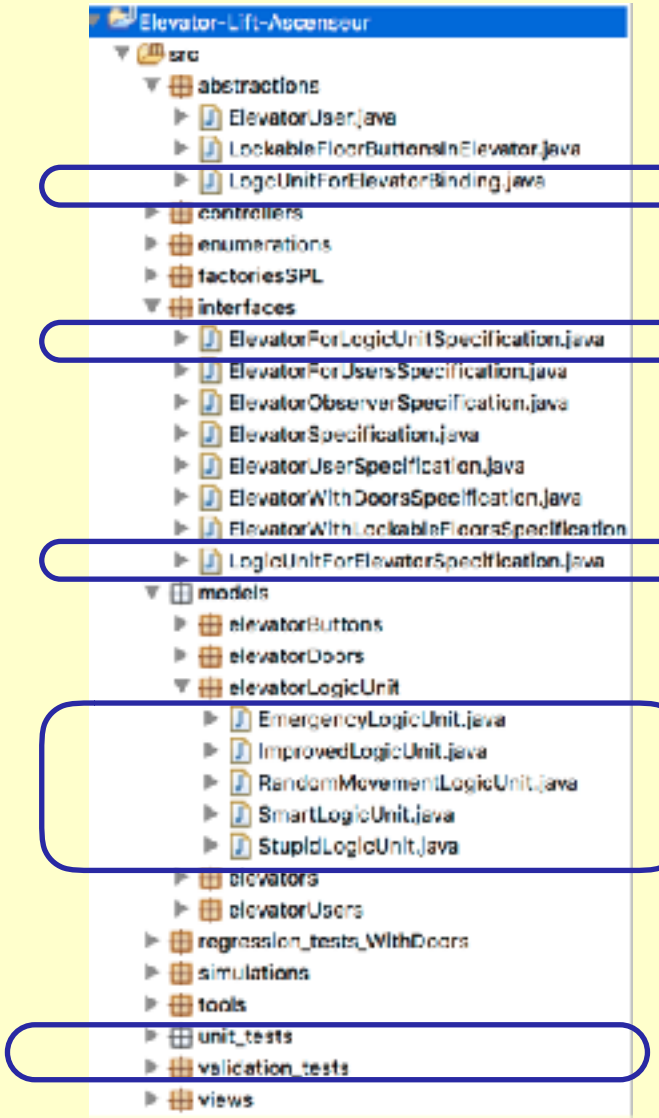
paul.gibson@telecom-sudparis.eu

http://www-public.telecom-sudparis.eu/~gibson/Teaching/CSC7003/

Design

/~gibson/Teaching/CSC7003/L4-Design-SampleSolution.pdf

We will focus on the design of the lift controller

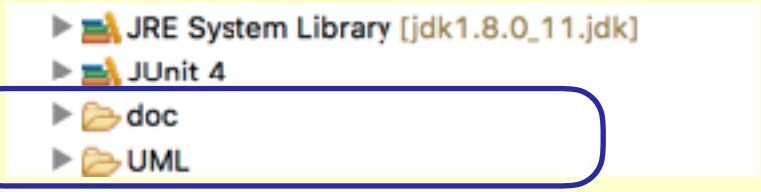


Requirements - specified in interface and tests

Design - following a SPL approach using factories

Implementation - 5 different pluggable logic units

Documentation - javadocs + UML



We will focus on the design of the lift controller

Code

```
package enumerations;

 * This logic type must be kept coherent with all the logic unit options
public enum LogicType {

    /**
     * Controller type for {@link models.elevatorLogicUnit.RandomMovementLogicUnit}
     */
    RANDOM,

    /**
     * Controller type for {@link models.elevatorLogicUnit.StupidLogicUnit}
     */
    STUPID,

    /**
     * Controller type for {@link models.elevatorLogicUnit.ImprovedLogicUnit}
     */
    IMPROVED,

    /**
     * Controller type for {@link models.elevatorLogicUnit.EmergencyLogicUnit}
     */
    EMERGENCY,

    /**
     * Controller type for {@link models.elevatorLogicUnit.SmartLogicUnit}
     */
    SMART

}
```

We will focus on the design of the lift controller

javadocs

enumerations.LogicType

This logic type must be kept coherent with all the logic unit options specified as subclasses of abstractions.LogicUnitForElevatorBinding:

- RANDOM - models.elevatorLogicUnit.RandomMovementLogicUnit
- STUPID - models.elevatorLogicUnit.StupidLogicUnit
- IMPROVED - models.elevatorLogicUnit.ImprovedLogicUnit
- EMERGENCY - models.elevatorLogicUnit.EmergencyLogicUnit
- SMART - models.elevatorLogicUnit.SmartLogicUnit

It is also recommended that the factoriesSPL.LogicUnitFactory is kept up to date so that every time a new logic unit type is added then the factory knows of that type.

Author:
jpaulgibson

We will focus on the design of the lift controller

Code

```
package interfaces;

import enumerations.DirectionState;

/**
 * The interface defining the methods offered by the controller to the elevator
 * (see {@link interfaces.ElevatorSpecification})
 * @author J Paul Gibson
 */
public interface LogicUnitForElevatorSpecification  {

    /**
     * @return the direction in which the lift should next move
     */
    public DirectionState calculateDirection();

    /**
     * @return whether the lift should stop at the next floor after it has completed its move
     */
    public boolean stopAtNextFloor();

    /**
     * Ensure that the controller is controlling only a single elevator
     * @param elevator is the unique elevator to which the controller is now connected
     * @throws IllegalArgumentException if the elevator being connected is null
     */
    public void bindToElevator (ElevatorForLogicUnitSpecification elevator) throws IllegalArgumentException;

}
```

javadocs

Interfaces.LogicUnitForElevatorSpecification

The interface defining the methods offered by the controller to the elevator (see interfaces.ElevatorSpecification)

Author:
J Paul Gibson

We will focus on the design of the lift controller

Code

```
package interfaces;

import enumerations.DirectionState;

 * The interface offered by the elevator to the logic controller. <br>
public interface ElevatorForLogicUnitSpecification {

     * Permits elevator logic unit to see if a request to go down has been made at a specified floor
    public boolean downPressedAtFloor (int floor)throws IllegalArgumentException;

     * Permits elevator logic unit to see if a request to go up has been made at a specified floor
    public boolean upPressedAtFloor (int floor)throws IllegalArgumentException;

     * Permits elevator logic unit to see if there is a request to a particular floor
    public boolean floorPressedInElevator (int floor) throws IllegalArgumentException;

     * @param floor specifies the floor at which the request is being checked
    public boolean noMoreRequestsBelow(int floor)throws IllegalArgumentException;

     * @param floor specifies the floor at which the request is being checked
    public boolean noMoreRequestsAbove(int floor)throws IllegalArgumentException;

     * @return the current direction in which the elevator is moving
    public DirectionState currentDirection ();

     * @return the current floor of the elevator
    public int currentFloor ();

     * @return the number of the top floor
    public int topFloor ();

     * Update the lift logic unit component - intended to be done by an engineer
    public void installLogicUnit(LogicUnitForElevatorSpecification logic_unit) throws IllegalArgumentException;

}
```

We will focus on the design of the lift controller

javadocs

for the interface …

interfaces.ElevatorForLogicUnitSpecification

The interface offered by the elevator to the logic controller.
This is a subset of the complete interface of an elevator as specified by interfaces.ElevatorSpecification.
With this interface we hide the methods specific to the elevator users which should not be visible to the elevator logic unit.

Author:
J Paul Gibson
See Also:
interfaces.ElevatorForUsersSpecification

… and for each method, e.g.:

boolean interfaces.ElevatorForLogicUnitSpecification.noMoreRequestsBelow(int floor) throws IllegalArgumentExcepti

Parameters:
floor specifies the floor at which the request is being checked
Returns:
whether there are no requests for a floor (inside the elevator or at a floor) below the specified floor
Throws:
IllegalArgumentException - if the number of floors specified is not in range

We will focus on the design of the lift controller stupid logic unit

```
public class StupidLogicUnit extends LogicUnitForElevatorBinding{


    public StupidLogicUnit (ElevatorForLogicUnitSpecification elevator) throws IllegalArgumentException{

        super(elevator);
    }


    /**
     * If the elevator is on the bottom floor then set the direction to up<br>
     * If the elevator is on the top floor then set the direction to down<br>
     * Else If the elevator has been stopped then set off towards bottom<br>
     * Otherwise keep moving in same direction
     */
    public DirectionState calculateDirection() {

        if (elevator.currentFloor()==0) return DirectionState.UP;
        if (elevator.currentFloor()== elevator.topFloor()) return DirectionState.DOWN;
        if (elevator.currentDirection() == DirectionState.STAY) return DirectionState.DOWN;
        else return elevator.currentDirection();
    }

    /**
     * Always stop at the next floor
     */
    public boolean stopAtNextFloor(){
        return true;
    }

    public String toString (){

            return LogicType.STUPID.toString();
    }

}
```

models.elevatorLogicUnit.StupidLogicUnit

The stupid controller ensures that the elevator moves from bottom to top to bottom ... stopping at every floor. This corresponds to STUPID in the enumerations.LogicType

Author:
J Paul Gibson

We will focus on the design of the lift controller

improved (stupid) logic unit

```
public class ImprovedLogicUnit extends StupidLogicUnit{


    public ImprovedLogicUnit(ElevatorForLogicUnitSpecification elevator) {
        super(elevator);
    }

    /**
     * Only stop at floors where requests to stop have been made
     */
    public boolean stopAtNextFloor(){
        int floor = elevator.currentFloor();
        DirectionState dir = elevator.currentDirection();

        if (elevator.floorPressedInElevator(floor)) return true;
        if (elevator.downPressedAtFloor(floor) && dir == DirectionState.DOWN) return true;
        if (elevator.upPressedAtFloor(floor) && dir == DirectionState.UP) return true;

        return false;
    }

    public String toString (){

        return LogicType.IMPROVED.toString();
    }
}
```

models.elevatorLogicUnit.ImprovedLogicUnit

The improved stupid controller ensures that the elevator moves from bottom to top to bottom to top to ...; but its behaviour improves that of the StupidLogicUnit because it stops only at floors where requests are made. This corresponds to IMPROVED in the enumerations.LogicType

Author:
J Paul Gibson

We will focus on the design of the lift controller smart logic unit

```
 * The smart controller ensures that the elevator will change direction if there are
public class SmartLogicUnit extends LogicUnitForElevatorBinding{


    public SmartLogicUnit(ElevatorForLogicUnitSpecification elevator) {
        super(elevator);
    }

     * Only stop at floors where requests to stop have been made
    public boolean stopAtNextFloor(){

        int floor = elevator.currentFloor();
        DirectionState dir = elevator.currentDirection();

        if (elevator.floorPressedInElevator(floor)) return true;
        if (elevator.downPressedAtFloor(floor) && dir == DirectionState.DOWN) return true;
        if (elevator.upPressedAtFloor(floor)  && dir == DirectionState.UP) return true;

        return false;
    }
```

models.elevatorLogicUnit.SmartLogicUnit

The smart controller ensures that the elevator will change direction if there are no more requests in the current direction and their are requests in the opposite direction.
The elevator stops only at floors where requests to stop have been made.
This corresponds to SMART in the enumerations.LogicType

Author:
J Paul Gibson

We will focus on the design of the lift controller smart logic unit

```
public DirectionState calculateDirection() {

    int currentFloor = elevator.currentFloor();
    DirectionState currentDirection = elevator.currentDirection();
    boolean noMoreRequestsAboveCurrent = elevator.noMoreRequestsAbove(currentFloor);
    boolean noMoreRequestsBelowCurrent = elevator.noMoreRequestsBelow(currentFloor);
    boolean floorDownRequestAtCurrent = elevator.downPressedAtFloor(currentFloor);
    boolean floorUpRequestAtCurrent = elevator.upPressedAtFloor(currentFloor);

    if (currentDirection == DirectionState.STAY){

        if (noMoreRequestsAboveCurrent  && noMoreRequestsBelowCurrent) return DirectionState.STAY;
        else if (noMoreRequestsAboveCurrent) return DirectionState.DOWN;
        else return DirectionState.UP;
    }

    else if (currentDirection == DirectionState.UP){

        if (noMoreRequestsAboveCurrent  && noMoreRequestsBelowCurrent){

            if (!floorDownRequestAtCurrent && !floorUpRequestAtCurrent) return DirectionState.STAY;
            if (floorUpRequestAtCurrent) return DirectionState.UP;
            else return  DirectionState.DOWN;
        }

        else if (noMoreRequestsAboveCurrent) return DirectionState.DOWN;
        else return DirectionState.UP;
    }

    else if (currentDirection == DirectionState.DOWN){

        if (noMoreRequestsAboveCurrent  && noMoreRequestsBelowCurrent){
            if (!floorDownRequestAtCurrent && !floorUpRequestAtCurrent) return DirectionState.STAY;
            if (floorDownRequestAtCurrent) return DirectionState.DOWN;
            else return  DirectionState.UP;

        }
        else if (noMoreRequestsBelowCurrent) return DirectionState.UP;
        else return DirectionState.DOWN;
    }

       return DirectionState.STAY; // should not happen
}
```

models.elevatorLogicUnit.SmartLogicUnit

The smart controller ensures that the elevator will change direction if there are no more requests in the current direction and their are requests in the opposite direction.
The elevator stops only at floors where requests to stop have been made.
This corresponds to SMART in the enumerations.LogicType

Author:
J Paul Gibson

We will focus on the design of the lift controller

3 different types of test for each logic unit:

1.Unit test

3.Validation test (at console)

5.Animation simulation (GUI)

We will look at these types of test in more detail later in the module