Written 24-hour exam, Introductory Programming, December 2015

You have to implement the simulator described below in Java, preferably using NetBeans.
To help us see whose program we look at, your program folder must have your ITU mail name in the file name, e.g. Lift_sola.
Include all the files in one zip-file with the same name.
Include also the ITU Report Front page in the zip file.
Some changes have been made to the upload site. You can upload drafts several times, but when you click Final, you cannot change things anymore (sounds foolish to us, but we cannot do anything).

Lift simulator

A lift (elevator) moves up and down, picking up passengers on the way and letting them out where they want. There are various strategies for doing this and the simulator could help measuring how well a strategy works. In this exam you should only work with this strategy:

1. Move upwards, picking up and releasing passengers on the way until nothing more can be done in that direction.
2. Don't stop at a floor unless someone has pushed a button showing that they will out at this floor or in, moving up.
3. Then move downwards until nothing more can be done in that direction. And so on.
4. If nobody is waiting (i.e. no buttons pressed), remain where you are with doors open.

In this assignment you must program a simulator including a GUI (graphical user interface) as shown in the figure. We recommend that you start with only part of the system, enough for you to pass. Then build more as time allows. See more in the hints below.

Warning
On the web you can find several lift and elevator simulators. None of them do what this simulator has to do. Don't waste time trying to use them.

The lift

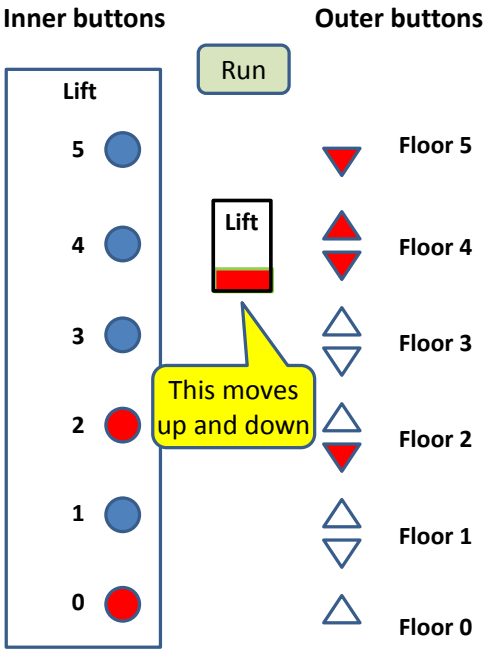
Inside the lift there is an *inner button* for each floor where the lift can stop. When a passenger pushes the button, the button will light up (red) and the lift will stop on that floor. When the lift stops on the floor, it turns off the button light.

The figure shows an example. The inner buttons on floor 0 and 2 are red, indicating that someone in the lift wants to get out at these floors.

On each floor there are two *outer buttons*: Up and Down. When a passenger wants to move upwards with the lift, he pushes the Up-button. The button lights up. The lift should stop there if it is moving upwards. When it has stopped on the way up, it should turn the Up button off, assuming that everybody who wanted to go up have entered. The Down button works in the same way. The bottom floor and top floor have only one button, of course.

In the figure someone wants to enter at floor 5 and move down. On floor 4 someone wants to move up and someone wants to move down. On floor 2 someone wants to move down.

The lift also has a weight sensor in the floor (bottom) of the lift. It tells the lift whether somebody is inside the lift. If there isn't anybody, it will not move according to the inner buttons, but turn all of them off. It will still check whether some outer buttons are on and move



in that direction. In the figure the lift floor is red, indicating that somebody is inside the lift. If nobody is in, it should be green.

The lift doesn't know about the individual passengers and where they want to go. It only knows which buttons are pushed and - through the weight sensor - whether somebody is inside the lift.

The simulator

As a user of the simulator, you can do what people would do in real life. You can click the buttons and they will light up. You cannot reset the buttons. Only the lift can do this.

You can click the floor sensor in the lift and it will toggle between red and green to indicate whether someone is in or not. In this way you simulate the real weight sensor.

When you click *Run*, the lift will close the doors (you cannot see the doors) and move according to the strategy. When it arrives at a floor, it will open the (invisible) doors to let people in and out.

Classes

You will probably need the following classes. There are some hints about the fields and methods. You may have to add methods on your own to make the program simpler.

Lift

Stage stage; // The Form where the simulator appears.
Pane root; // The panel where the graphical components reside.
Scene scene; // The area inside the Form. It must have a size that allows all the floors to show.
int direction; // +1 when the current move direction is up, -1 when down.
int currentFloor; // Where the lift is right now.
LiftBtn [ ] innerButtons; //Array of inner buttons.
LiftBtn [ ] upButtons; // Array of the outer buttons showing "up".
LiftBtn [ ] downButtons; // Array of the outer buttons showing "down".
FloorSensor floorSensor; // On when someone is in the floor.

Lift(Stage stg, int N) // Create a lift simulator with floors 0 to N. Create the buttons and the graphical components on the Pane root. When everything is created, call stage.show( ) and return.
Run( ) Find the next floor and move the lift there. Reset various buttons. Move the lift (the small rectangle) on the screen.

LiftBtn

boolean on; // When on, somebody has pushed the button and the light is red. Otherwise it is green.
Shape btn; // Reference to a JavaFX circle, rectangle or polygon on the screen.
LiftBtn(Shape btn) // The constructor. The caller has created the button Shape, e.g. a circle, and passes it on to LiftBtn.
Push( ) and Reset( ). The methods should not only set "on", but also update the Button color. There is no reason to refresh the screen. JavaFX does it automatically.

FloorSensor

Is similar to LiftBtn, but behaves differently. For instance, it toggles *on* when clicked and it may use different colors. You may use sub-classing.

LiftSim

The application class.
start(Stage primaryStage) This method creates a lift that puts all its visual components on primaryStage. When the start method returns, the system will show the lift. Ideally it should be possible to create and run several lifts at the same time.

Hints

At the beginning, implement only the inner buttons, the lift rectangle and a simple version of the Run method. Forget about the floor sensor and the outer buttons. When this works, you should pass the exam. Add more lift buttons and functionality as time allows to get higher marks.

The teacher's classes have roughly these lengths:

- Lift: around 160 lines. The constructor is 80 of these.
- LiftBtn: Around 30 lines.
- Application, start method: 3 lines.

The beginner version will save around 80 lines of the Lift class.

How to get started

1. With NetBeans, create a project of type JavaFX. Close other projects to avoid confusion.
2. The JavaFX project has a *start* method that contains code for creating a *scene*, a root *StackPane* and a button. **Try to run it.** It should show a Form with a Hello World button at the center. Try to click it. If it doesn't print anything on *out*, there is something wrong with your Java/Netbeans/JDK installation.
3. Now create the other classes as files. You can copy and paste from the text above.
4. Make the constructor for Lift. Move the code from the *start* method to the constructor and modify it so that it shows a Run button rather than a HelloWorld button.
5. Change the StackPane to a simple Pane. **Try to run it.** The simple Pane allows you to set the X-Y positions of the visual components. The default X-Y of the Button is 0, 0. So it should move to the top-left.
6. Let the Lift constructor create the inner LiftBtn's including the Circle Shapes. The Circle shapes must be added to the root panel in the same way as the Run button.
   Warning: NetBeans can suggest that you import a Circle file. Do so, but make sure you import a javaFX file, not a javaAWT. JavaAWT circles are invisible in a javaFX project.
7. Set the centerX and centerY position of the Circles. You can either set the position with setters (see below) or use a circle constructor that has centerX and centerY as parameters.
8. A LiftBtn is smart. It has a field which is a Shape, for instance a JavaFX Circle. When the user clicks the Circle, the event handler should *Push* the LiftBtn in order that the lift can stop at the right floor. But how does the event handler know which of the N+1 LiftBtn's it belongs to? If you set the event handler inside the LiftBtn constructor, it is simple. The event handler can address the LiftBtn with *this.* In the LiftBtn constructor it looks like this:

```
LiftBtn(Shape btn) {
    this.btn = btn; // Attach the btn to this LiftBtn.
    btn.setOnMouseClicked( e -> { // Attach the event handler.
       this.Push( ); // Push the lift button I am attached to.
    });
}
```

9. When creating a Circle or another Shape, you will need to set various fields/properties of the Shape. There are a lot to choose from. Unfortunately, few of them work for any Shape. The most important ones are:

   setFill(Color. ...) defines the inner color of most Shapes.
   setStroke(Color. ...) defines the border color of most Shapes.
   setCenterX(x), setCenterY(y), setRadius(r) define position and radius for a circle.
   setX(x), setY(y), setWidth(w), setHeight(h) define position and size of some Shapes.
   setTranslateX(), setTranslateY() define position for a Button.

10. There is no Triangle component. Use a Polygon Shape instead. Use the built-in documentation to find out how.
11. **Warning:** We have seen cases where a Shape is inserted correctly in the getChildren list, but doesn't show on the Form. It helped using setStroke(...) or for texts, making the text non-null.
12. **Warning:** Sometimes NetBeans behaves strangely. It may help to right-click the project icon and use Clean and Build.