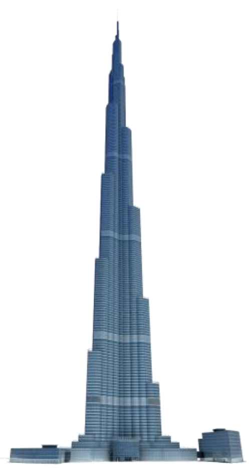
PARALLELISM AND CONCURRENCY
PRACTICAL ASSIGNMENT #2 WINTER TERM 2023
MONITORS

THE BURJ KHALIFA ELEVATOR



The Burj Khalifa building has a new monitor-controlled elevator that goes from ground level to the 148th floor (the observatory deck) in just a couple of snaps of a finger. It also goes down back to ground level, of course.

The monitor controls (synchronizes) the operations of the elevator but also the behaviour of the tourists that use it to have the most astonishing view of, … well the desert and its sandy surroundings.

Tourists: (repeatedly) wait on the ground floor until the elevator is there with its doors open and a green light flashes “elevator is ready to board for ascension”. Then, exactly 4 people can get in. When they reach the top floor, they wait until the doors are open again and a green light flashes “passengers can get out”. They leave the elevator and stay on the observatory enjoying the view for a while. When they are ready to descend, they wait until the elevator is there with its doors open and a green light flashes “elevator is ready to board for descent”. Then exactly 4 people can get in. When they reach the ground floor they wait until a green light flashes “passengers can get out” and then get out.

Elevator: (repeatedly) starts, empty, at the ground floor, opens its doors, flashes “elevator is ready to board for ascension” and waits until there are 4 tourists in. Then it makes the ascension and when it reaches the top floor it opens the doors, flashes “passengers can get out” and waits until its empty. Then flashes “elevator is ready for descent” and waits until 4 tourists are in. When it is full, it descends to the ground floor, opens the doors and flashes “passengers can get out”.

ElevatorMonitor monitor	
Tourist$_i$	**Elevator**

loop
    monitor.getInForAscenssion(this**)**
    monitor.getOut(this**)**
 	
    enjoyTheView()
 	
    monitor.getInForDescent(this**)**
    monitor.getOut(this**)**
	loop
	    monitor.loadForAscension()
	    ascend()
	    monitor.ascensionComplete()
	    monitor.unLoad()
	
	    monitor.loadForDescent()
	    descend()
	    monitor.descentComplete()
	    monitor.unLoad()

Notice that the enjoyTheView, ascend and descend operations are performed outside the monitor.

PART A (45%): UP and DOWN and that is all.

Design (pseudocode) a MESA-style ElevatorMonitor. The only available operations on **conditions** waitC and signalC. No notifyAll-like operations available. Follow the syntax shown in class. Starvation is not an issue (Tourists can overtake each other). Do not collapse the operations. All messages are printed inside the monitor.

Translate your monitor into Java using explicit locks and Condition objects. Use the project provided. You can only modify the “monitor” class. All other classes must never be modified. No marks will be awarded if the pseudocode is not correct. The translation must be faithful. Remember: no *All operations available.

Carefully analyse the sample images provided. Make sure your implemented monitor EXACTLY prints the same messages since the output of your program may be analysed by a trace analyser program.

In the document containing the pseudocode, specify, for each condition

a) What it is used to wait for
b) Who waits in it
c) Who signals it

Condition	What it is used to wait for	Who waits in it	Who signals it
…	…	…	…

The followin image depicts a full cycle of the elevator. Remenber that all messages are printed inside the monitor.

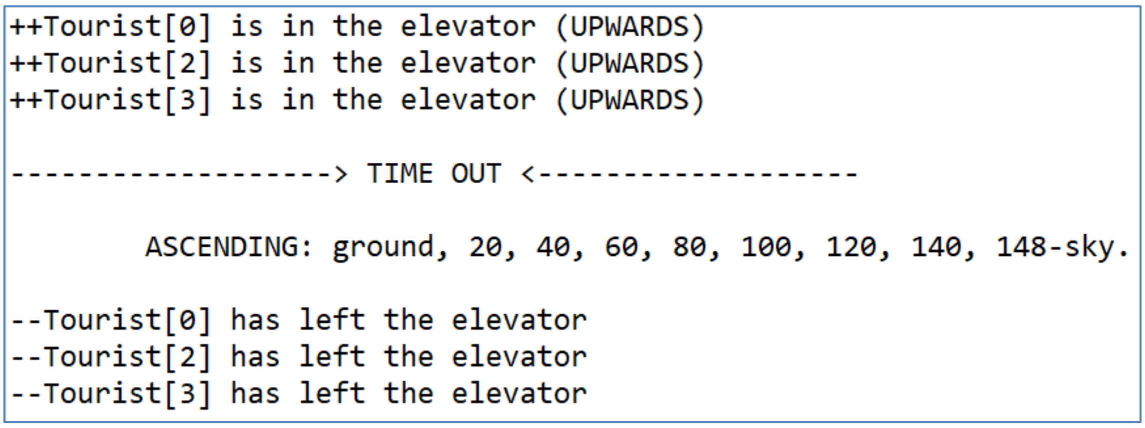
```
++Tourist[0] is in the elevator (UPWARDS)
++Tourist[1] is in the elevator (UPWARDS)
++Tourist[2] is in the elevator (UPWARDS)
++Tourist[4] is in the elevator (UPWARDS)

        ASCENDING: ground, 20, 40, 60, 80, 100, 120, 140, 148-sky.

--Tourist[4] has left the elevator
--Tourist[2] has left the elevator
--Tourist[0] has left the elevator
--Tourist[1] has left the elevator

++Tourist[0] is in the elevator (DOWNWARDS)
++Tourist[2] is in the elevator (DOWNWARDS)
++Tourist[4] is in the elevator (DOWNWARDS)
++Tourist[1] is in the elevator (DOWNWARDS)

        DESCENDING: 148-sky, 140, 120, 100, 80, 60, 40, 20, ground.

--Tourist[0] has left the elevator
--Tourist[2] has left the elevator
--Tourist[4] has left the elevator
--Tourist[1] has left the elevator
```

PART B (15%): ADDING a timer.

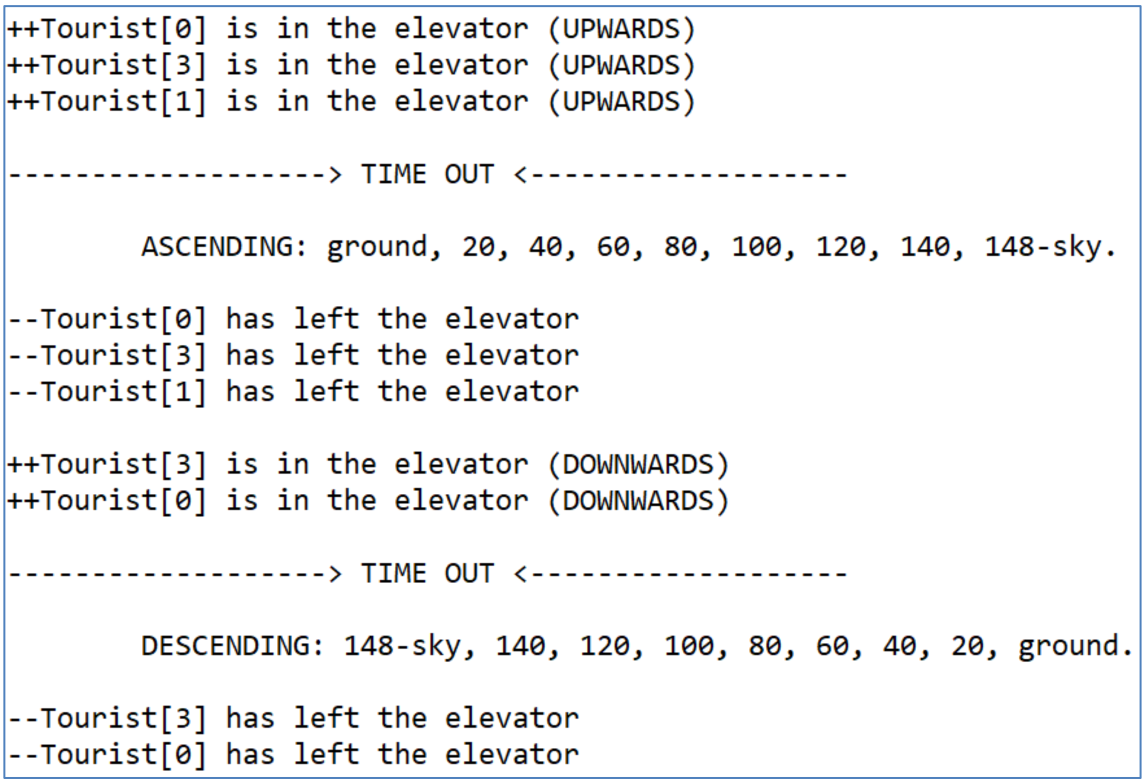
Modify your monitor so that it can give support to timers. **A timer is created and started every time the elevator is ready for ascension or descent. The timer is destroyed when the loading operation is about to terminate (return).**
When the timer goes off the elevator ascends (or descends) if it has at least one tourist inside. This way, the elevator makes rides even if it is not fully loaded. If the elevator is empty, the timer restarts itself (same time).

```java
public class Timer extends Thread {

    private ElevatorMonitor monitor;
    private long time;
    private boolean end = false;

    public Timer (ElevatorMonitor monitor, long time) {
        this.monitor = monitor;
        this.time = time;
    }

    public void destroy () {
        end = true;
    }

    public void run () {
        // Notice that timers do not necessarily run in an endless loop...
        boolean restart = true;
        while (restart && !end) {
            try {Thread.sleep(time);} catch(InterruptedException ie) {}
            if (!end) restart = monitor.timerGoesOff();
        }
    }
```

The following images depict the behaviour of the elevator when timers go off before four tourist were in the elevator.



```
++Tourist[0] is in the elevator (UPWARDS)
++Tourist[2] is in the elevator (UPWARDS)
++Tourist[3] is in the elevator (UPWARDS)

-------------------> TIME OUT <-------------------

        ASCENDING: ground, 20, 40, 60, 80, 100, 120, 140, 148-sky.

--Tourist[0] has left the elevator
--Tourist[2] has left the elevator
--Tourist[3] has left the elevator
```



```
++Tourist[0] is in the elevator (UPWARDS)
++Tourist[3] is in the elevator (UPWARDS)
++Tourist[1] is in the elevator (UPWARDS)

-------------------> TIME OUT <-------------------

        ASCENDING: ground, 20, 40, 60, 80, 100, 120, 140, 148-sky.

--Tourist[0] has left the elevator
--Tourist[3] has left the elevator
--Tourist[1] has left the elevator

++Tourist[3] is in the elevator (DOWNWARDS)
++Tourist[0] is in the elevator (DOWNWARDS)

-------------------> TIME OUT <-------------------

        DESCENDING: 148-sky, 140, 120, 100, 80, 60, 40, 20, ground.

--Tourist[3] has left the elevator
--Tourist[0] has left the elevator
```

For ascension, a timer is set to 2000ms. For descent it is set to 500ms.

Translate your monitor into Java using explicit locks and Condition objects. Use the project provided. You can only modify the "monitor" class. All other classes must never be modified. No marks will be awarded if the pseudocode is not correct. The translation must be faithful. Remember: no *All operations available.

PART C (40%): ADDING a safety measure.

Redo your pseudocode for part A, introducing the following safety measure that ONLY APPLIES FOR UPWARD RIDES (ascensions)

When the elevator is full (4 tourists in) it departs if, and only if, the total weight does not exceeds 300 kg. When the total weight exceeds 300 kg an alarm goes off and the elevator must be fully vacated because ascension is not safe. When tourists hear the alarm, they complain and get out.
When the elevator is finally empty, tourists (the same or others) may get in again.

In this part there's no timer

ElevatorMonitor monitor	
Tourist$_i$	**Elevator**

loop
 	
    …
	boolean safe
	
	loop
	
	    safe = false
	    while (!safe) {
	        safe = **monitor.loadForAscension()**
	        if (!safe) {
	            monitor.unLoad()
	        }
	    }
	    ascend()
	    monitor.ascensionComplete()
	    monitor.unLoad()
	
	    monitor.loadForDescent()
	    descend()
	    monitor.descentComplete()
	    monitor.unLoad()

Notice that now operation loadForAscension returns a boolean value

ElevatorMonitor monitor	
Tourist$_i$	**Elevator**

```
loop

  safe = false
  while (!safe) {
        safe = **monitor.getInForAscension(this)**
        if (!safe) {
              COMPLAIN
              **monitor.getOut(this)**
        }
  }
  **monitor.getOut(this)**

  enjoyTheView()

  **monitor.getInForDescent(t**his**)**
  **monitor.getOut(this)**
```
	…

Notice that now operation getInForAscension returns a boolean value.

If you have introduced new conditions, document them

Condition	What it is used to wait for	Who waits in it	Who signals it
…	…	…	…

Translate your monitor into Java using explicit locks and Condition objects. Use the project provided. You can only modify the "monitor" class. All other classes must never be modified. No marks will be awarded if the pseudocode is not correct. The translation must be faithful. Remember: no *All operations available.

```
++Tourist[5 w: 66] is in the elevator (UPWARDS)
++Tourist[6 w: 85] is in the elevator (UPWARDS)
++Tourist[7 w: 85] is in the elevator (UPWARDS)
++Tourist[8 w: 67] is in the elevator (UPWARDS)


EMERGENCY: weight EXCESS (303) Please, get out

**TOURIST[5] cries: "what the F*CK!. What's wrong with this junk?!"
--Tourist[5] hasleft the elevator
**TOURIST[7] cries: "what the F*CK!. What's wrong with this junk?!"
--Tourist[7] hasleft the elevator
**TOURIST[6] cries: "what the F*CK!. What's wrong with this junk?!"
--Tourist[6] hasleft the elevator
**TOURIST[8] cries: "what the F*CK!. What's wrong with this junk?!"
--Tourist[8] hasleft the elevator

++Tourist[11 w: 65] is in the elevator (UPWARDS)
++Tourist[13 w: 85] is in the elevator (UPWARDS)
++Tourist[12 w: 67] is in the elevator (UPWARDS)
++Tourist[14 w: 67] is in the elevator (UPWARDS)

        ASCENDING [w= 284]: ground, 20, 40, 60, 80, 100, 120, 140, 148-sky.
```

```
++Tourist[9 w: 66] is in the elevator (UPWARDS)
++Tourist[5 w: 66] is in the elevator (UPWARDS)
++Tourist[7 w: 85] is in the elevator (UPWARDS)
++Tourist[6 w: 85] is in the elevator (UPWARDS)


EMERGENCY: weight EXCESS (302) Please, get out

**TOURIST[9] cries: "what the F*CK!. What's wrong with this junk?!"
--Tourist[9] hasleft the elevator
**TOURIST[5] cries: "what the F*CK!. What's wrong with this junk?!"
**TOURIST[6] cries: "what the F*CK!. What's wrong with this junk?!"
**TOURIST[7] cries: "what the F*CK!. What's wrong with this junk?!"
--Tourist[5] hasleft the elevator
--Tourist[6] hasleft the elevator
--Tourist[7] hasleft the elevator
```

SUBMIT

A single Rar o Zip file containing two items:

- In a PDF file, the pseudocodes of the monitors designed. Use **bold** face and a different colour for the **waitC** and **signalC** operations. Remember to specify and document the usage of the conditions. This document must clearly state the names of the all the students in the group. In the monitor for part C, clearly highlight in green the differences with the monitor for part A. No scans or photos of handwriting allowed.

- A compressed folder containing the project with the translations of the monitors into Java (use the project provided)

Only one student makes the submission, on behalf of the whole group.

For submission deadline, check eCampus.