Sistemas Distribuídos

Coding Air Lift solution

António Rui Borges

List of students enrolled at lab class P1

N.Mec	*Nome*	*CC*	*Reg*	*T-G*
89290	ALEXANDRE FERREIRA ABREU	MIECT	O	1-7
88811	ANDRÉ MIGUEL SAMPAIO ALVES	MIECT	O	1-1
80316	CÁTIA ISABEL SOARES AZEVEDO	MIECT	O	1-2
90480	DANIEL VALA CORREIA	MIECT	O	1-7
88867	EDUARDO LUÍS FONSECA COELHO	MIECT	O	1-5
72783	EURICO OSÓRIO MARQUES DIAS	MIECT	O	1-3
88823	GONÇALO FILIPE FIGUEIREDO PERNA	MIECT	O	1-6
89140	JOÃO ANTÓNIO TAVARES TRINDADE	MIECT	O	1-4
79987	JOÃO GABRIEL PINTO DE MATOS E CASTRO ABRANTES	MIECT	O	1-2
88812	JOAQUIM MANUEL SANTOS RAMOS	MIECT	O	1-5
79671	JOSÉ CRISTIANO SANTOS MOREIRA	MIECT	O	1-8
84792	MARIANA ISABEL DA ROCHA PINTO	MIECT	O	1-1
80330	NUNO MIGUEL FÉLIX PEDRIDE	MIECT	O	1-8
71734	PAULO RICARDO CARDOSO PINHO	MIECT	O	1-9
88734	PEDRO MIGUEL SOARES VALÉRIO	MIECT	O	1-3
88861	PEDRO MIGUEL TEIXEIRA ALVES	MIECT	O	1-9
89147	RICARDO ANDRÉ POÇAS DE CARVALHO	MIECT	O	1-4
88802	RODRIGO DE LARMAND ALVIM LEAL ROSMANINHO	MIECT	G	1-6
76404	TIAGO JORGE GOMES ALMEIDA	MIECT	O	1-10
84957	TOMÁS GUERRA FREITAS	MIECT	D	1-10

Role of the General Repository of Information

The General Repository of Information works solely as the location where the visible internal state of the problem is stored. The *visible internal state* is defined by the set of variables whose value is printed in the logging file.

Whenever an entity (pilot, hostess or passenger) executes an operation that changes some of these variables, the fact must be reported by appending a new line group to the logging file. The report operation has to be *atomic*, that is, if two or more variables are changed, the report operation must be unique so that the line group reflects all the changes that have taken place.

No value should be retrieved from the General Repository of Information. One should remember that it does not belong properly to the solution of the problem. Its main role is in fact to provide the sole means available for visualization of the interaction evolution in time. Thus, it becomes quite useful for debugging.

Layout of the logging file - 1

```
                                    Airlift - Description of the internal state
 PT   HT   P00  P01  P02  P03  P04  P05  P06  P07  P08  P09  P10  P11  P12  P13  P14  P15  P16  P17  P18  P19  P20 InQ InF PTAL
#### #### #### #### #### #### #### #### #### #### #### #### #### #### #### #### #### #### #### #### #### #### ##  ##  ##

Flight #: boarding started.
#### #### #### #### #### #### #### #### #### #### #### #### #### #### #### #### #### #### #### #### #### #### ##  ##  ##
                                                                    . . .
#### #### #### #### #### #### #### #### #### #### #### #### #### #### #### #### #### #### #### #### #### #### ##  ##  ##

Flight #: passenger # checked.
#### #### #### #### #### #### #### #### #### #### #### #### #### #### #### #### #### #### #### #### #### #### ##  ##  ##
                                                                    . . .
#### #### #### #### #### #### #### #### #### #### #### #### #### #### #### #### #### #### #### #### #### #### ##  ##  ##

Flight #: departed with # passengers.
#### #### #### #### #### #### #### #### #### #### #### #### #### #### #### #### #### #### #### #### #### #### ##  ##  ##
                                                                    . . .
#### #### #### #### #### #### #### #### #### #### #### #### #### #### #### #### #### #### #### #### #### #### ##  ##  ##

Flight #: arrived.
#### #### #### #### #### #### #### #### #### #### #### #### #### #### #### #### #### #### #### #### #### #### ##  ##  ##
                                                                    . . .
#### #### #### #### #### #### #### #### #### #### #### #### #### #### #### #### #### #### #### #### #### #### ##  ##  ##

Flight #: returning.
#### #### #### #### #### #### #### #### #### #### #### #### #### #### #### #### #### #### #### #### #### #### ##  ##  ##
                                                                    . . .
#### #### #### #### #### #### #### #### #### #### #### #### #### #### #### #### #### #### #### #### #### #### ##  ##  ##

Airlift sum up:
Flight # transported # passengers
                  . . .
Flight # transported # passengers

Legend:
PT   - state of the pilot
HT   - state of the hostess
P##  - state of the passenger ##
InQ  - number of passengers presently forming a queue to board the plane
InF  - number of passengers in the plane
PTAL - number of passengers that have already arrived at their destination
```

Layout of the logging file - 2

- the logging file format must be strictly obeyed, no alterations to the supplied layout are allowed
- the *visible internal state* consists of
  - the current states of the pilot, the hostess and the passengers
  - the number of passengers presently forming a queue to board the plane
  - the number of passengers in the plane
  - the number of passengers that have already arrived at their destination
- it may, or may not, be part of the *visible internal state* the following quantities
  - the flight number
  - the number of passengers transported in each flight
- some of the reports require the appending of two data lines to the logging file, others just one line
- whether to print one or two lines depends on specific state transitions
- the full report printed at the end consists of n+1 lines, where n is the number of flights that took place (*who should print it?*)

Code organization

The code should be organized in a hierarchical and logical way through the use of packages. At least, four packages should be considered

- *main* – containing the data types which define the simulation parameters and the application launcher
- *entities* – containing the data types which define the different entities and their states
- *shared regions* – containing the data types associated with regions of thread communication and synchronization
- *common infrastructures* – containing the data types which implement different functionalities required to solve the problem.

The code should also be properly commented to produce useful documentation through the application of `javadoc`.

Synchronization

A choice has to be made of the synchronization devices to be used on the implementation of the shared regions

- *implicit monitors* – taking advantage of the implicit concurrent properties of Java (bear in mind that, since *condition variables* are restricted to a single one, associated to the monitor object, by Java concurrency model, it is necessary to turn the blocking conditions explicit by repetitive testing of normal variables)
- *explicit monitors* – using the data types `Lock`, `ReentrantLock` and `Condition` of the Java concurrency library (there is no limit to the number of *condition variables* that can be defined in this case)
- *semaphores* – either the Dijkstra semaphore, which was described in the lectures, or the data type `Semaphore` of the Java concurrency library (distinct semaphores are used here both to ensure mutual exclusion and to implement the synchronization points).

Home work - 1

Coding of the intervening entities data types

Each entity should be a derived data type of `Thread`. The method `run`, which represents the entity life cycle, has to be overridden. Also remember that each property that makes each instantiated entity unique, must have associated `set` and `get` public methods. Entity states should be coded as distinct non-instantiated data types where each state value is public and constant.

Coding of the shared regions of information

All public operations called on the shared regions of information should be executed in mutual exclusion. Care should be taken to avoid deadlocks and indefinite postponement situations. Also remember that whenever the operation execution should result in changes of some of the variables of the visible internal state, the fact has to be reported to the *general repository of information* within the call.

Home work - 2

Coding of the application launcher

Bear in mind that the main program should instantiate the different shared regions of information and the different intervening entities, then start the different entities and wait for their termination. Also remember that the simulation parameters should be coded in an independent distinct non-instantiated data type where each parameter value is public and constant.

All code should compile without errors and warnings.