Object-Oriented Concurrent Programming: Practical Exam Example

Name: _____ Nº Mec.: _____

Problem: The provided program is a sequential sketch of a simulation of a working office. The office if composed of four different areas: hallway, work, eating, and vending machines. Employees may move themselves inside the office fulfilling their agenda.
In the first version, vending machines have an unlimited amount of food, but there is a limited number of workplaces and eating chairs.
To better understand some of the features provided, try to compile (command `./compile` in a terminal opened in `P-example` path) and run (`./run`) the program[1]. Also there is a complete working solution in a (obscured) jar format (`./run-jar`).

a) Take a close look at the code provided to identify all shared mutable objects that might exist in the program. Then, provide a safe implementation of shared classes for such objects[2]. Be aware that the semantics of classes are (also) defined using Design-by-Contract.

b) Implement an employee active entity (i.e. with autonomous execution), with the agenda given as example. Note that chair are shared resources than can only be used by a person at a time.

c) Change employees activity to enable a limited amount of food in vending machines. Note that with this modification it might happen that some employees will be blocked forever (waiting for food).

d) To solve the previous problem, implement a new active entity (`ActiveFoodSupplier`) that refills empty vending machines[3]. This entity should share as much as possible behavior with the existing active entity. Its agenda may require an explicit request (from an employee) for food refilling (carefully analyze the program to ensure liveness). A request should only be made when all vending machines are empty.

In the implementation of this problem, don't forget to use an Object-Oriented structure to enhance modularity and to ease program extension with new types of objects.

[1] Command `view-javadoc` shows the documentation of library classes (e.g. `view-javadoc CThread`)
[2] Consider the use of interfaces to aid in an OO implementation.
[3] Use figure `image/person2.png` to differentiate from employees.