Developers:

David Furtado Monteiro

Description:

The program I developed has multiple classes. Requests, WatcherControl, Person, EntryControl, ExitControl, CarParkControl, CarPark and Main. In this program I am using a combination of two syschronization solutions. I use a similar solution to the "Sleeping barber" for synchronization between Person to Requests, EntryControl to Requests and ExitControl to Requests.

I also use the model "Reader Writer" with the addition of a Watcher to ensure mutual exclusion when EntryControl class and ExitControl class try to access the class CarPark. I made sure to make the program give priority to writers to avoid starvation. Each EntryControl & ExitControl read and write and since there was multiple of them running at the same time this "Reader Writer" was a must.

Requests:

This class is used for communication between Person class and a free EntryControl class.

Person:

This class represents a person. It has three variables that will make it be distinguished from other of the same class. String name, int ID and char type. The type can be 'S' for staff, 'L' for learner(student) and 'V' for visitor. This class implements the class Runnable from the java library and as so it makes use of the method run(). Communication with carparkControl is done through class Requests and the sysnchrnization problem is solved using a model based on the "Sleeping Barber solution".

EntryControl & ExitControl:

This classes access classes Requests and CarPark. When accessing the class CarPark it makes use of the "read writer solution" making use of another class watcherControl to properly insert mutual exclusion. This class will only access CarPark class to read if the person as space/permission to enter or leave the car park. This class will only write in the CarPark to add or remove a vehicle/person info.

This class is a virtual barrier at the entrance and exit of the car park. It waits for a person to try make a request to enter or exit. Once the request is done, it reads the CarPark making sure the request made will be accepted or rejected. It proceeds by placing a response on the Request class.

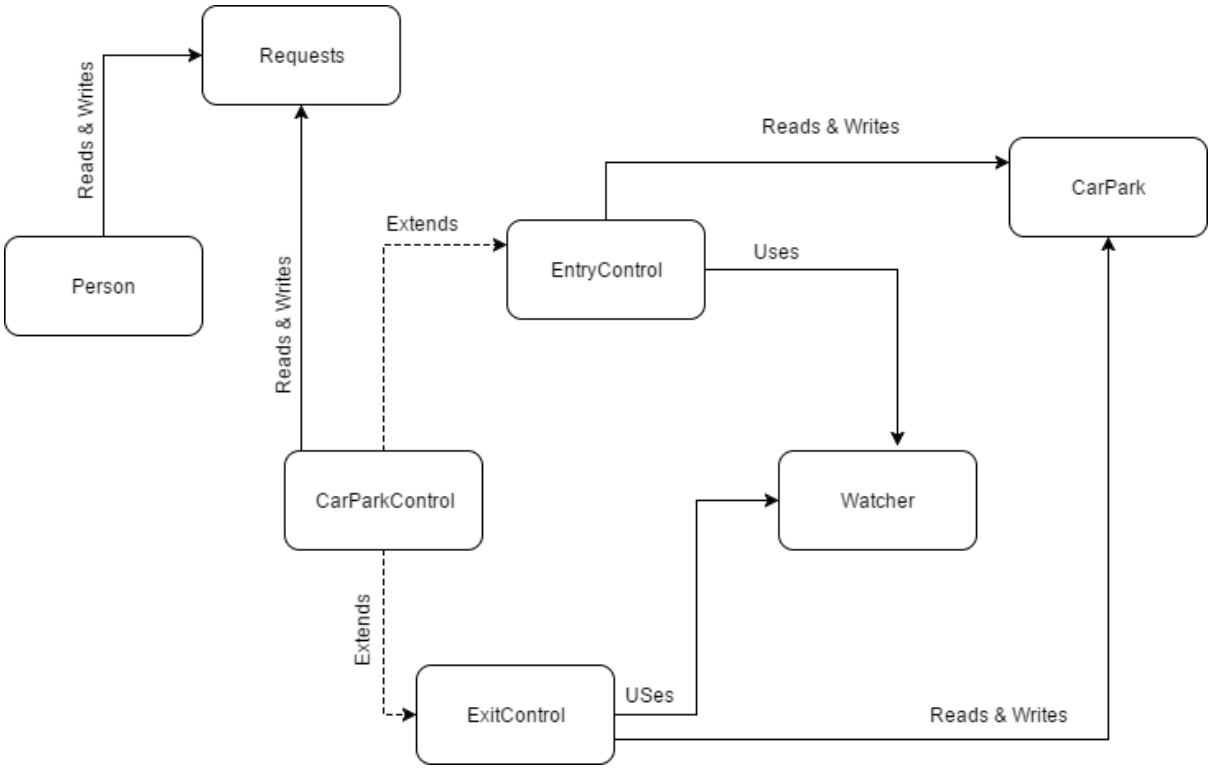
CarPark:

Class uses a combination of elements. This class virtually represents a car park with limited space. If a person gets in, it is guaranteed that there will be one less parking space.

WatcherControl:

Watcher class ensures the writer preference patter is done correctly. Easy to manage and eases debugging

Class Diagram



Run:

1.To Compile

javac CarPark/*.java

3.To run

java CarPark/Main

Expected:

Perfect synchronised system where information swapped between people and carParkControlles with mutual exclusion when accessing CarPark class and Requests class without starvation or a any type of deadlock.

Result:

Multiple java.lang.IllegalMonitorStateException