TURNS CONTROL SYSTEM FOR USER’S ATTENTION 2.0

Sebastián García Acosta

Table of contents:

[**Functional requirements:**](#_pleqwnkl1wqn) **2**

# 

# 

# 

# Functional requirements

The turns control system must be able to:

1. Add new users whose fields are: type of identity document, document number, names, surnames, telephone, and address. These last two fields are not mandatory, while the rest are. Duplicated users are not allowed. Two users are considered duplicated if their document number are equal.
2. Register new turns. These are unique and their nomenclature goes from A00 to Z99. To assign one, is needed to check if the user to whom the turn is going to be assigned exists and has no other turn assigned yet. Moreover, a turn has a type and duration (which depends on the type) that is specified by the user during its creation.  
     
   Eventually, a user that had a turn and was dispatched, can get another turn if and only if was present in the last two turns that requested; otherwise, will be suspended and will not be allowed to register a new turn for the next 2 days.
3. Dispatch all pending turns automatically starting with the oldest one up to the turn that satisfies the condition that the summed datetime durations from the oldest to it, is before the current datetime. A turn is considered "dispatched" in two scenarios: the first, the user who owned that turn has already been attended; or second, the user who owned that turn wasn’t present. App must allows attend a turn assigned to some user regardless of whether there are more turns to be attended ahead of it. The scenario assigned to every turn must be randomly selected.
4. Display on console the current system datetime composed by year, month, day, hour, minute and second.
5. Update the system datetime manually (typed by the user) or using the current datetime of computer.
6. Generate a report with all the turns that a person has ever requested, indicating the turn code, whether it was already attended or not and whether the person was present when they were called to be attended or not.
7. Generate a report with all the people that ever owned a turn specified by user. E.g., user can search all the people that ever owned the turn C91.
8. Randomly generate people registered in the system. The number of people to generate must be indicated by the user.
9. Randomly generate shifts associated with people previously registered in the system. The user must indicate how many shift days will be generated. And the generation always starts from the current day. The user must also indicate how many shifts will be generated for each day.

# Non-functional requirements

This section describes the internal constraints of the proposed program. The number of every item correspond to the number of the functional requirement whose constraints are going to be defined; if the item has not a number as enumerator, then it is a global constraint.

* The model must define datetime class.
* The program must be persistent through the using of serializable interface.
* (8) The random generation of people should take the name values of several base text files (one for names, another for surnames) generated from a web tool.
* (6, 7) Reports must be presented either in console or exported in a text file according to the user selection.
* Every time that the user use a function of the system menu, it must be displayed on console how much time it took.
* It is needed least a sequential and a binary search.
* Throughout the development of the previous requirements, it is needed to have at least 5 orderings using the sort method of the Arrays class or the Collections class, using the Comparable interfaces on some class of the model, Comparator as an external class, Comparator as an anonymous class, a Comparator in reverse of the Collections method and a Comparator of a class already implemented from the Java API such as Integer or String (in the latter you should not implement the Comparator, just order an array or list of objects of a class Java that already comes with Comparator implemented).

# Guidelines to sorting and searching algorithms

BinarySearachForUserId -> TurnsManager-> line 2017.

binarySearchTurnById -> TurnsManager -> line 389

(sequential Search) searchUser -> TurnsManager -> 190

(sequential Search) searchUser -> TurnsManager -> 372

getSortedUsersByNameAndSurname -> TurnsManager -> line 238

getSortedUsersByNumberOfAbsences -> TurnsManager -> line 256

getSortedTurnsByEndingDateTime -> TurnsManager -> line 275

Comparable Interface: DateTime -> line 208

ReverseOrder: -> TurnsManager -> line 392

AnonymousClass -> TurnsManager -> line 209

External Comparator class -> model -> UserNameAndSurnameComparator

String comparator -> TurnsManager -> line 405

Class diagram

