TURNS CONTROL SYSTEM FOR USER’S ATTENTION

Sebastián García Acosta

Table of contents:

[**Context**](#_4p5dd98xhust) **3**

[**Functional requirements:**](#_pleqwnkl1wqn) **6**

[**Test cases design**](#_3g9vr2niv3zo) **6**

[Scenarios configuration](#_zdtkc1lrryep) 6

[Test cases:](#_b8iq3z6ryn6x) 7

[Encapsulation tests.](#_qec8ipjb0v20) 7

[Turn class](#_xedono9smxji) 7

[User class](#_c75u2lpjclwb) 7

[Business logic tests](#_wrwecchu1scl) 7

[Sign up functionality](#_msmkuphcm69r) 8

[Searching methods](#_u90oj54id6wf) 9

[Turn id code generation](#_aiotqddiif58) 9

[Registering functionality](#_51d1qjw2aa1x) 10

[Attend functionality](#_uujuny82vg3a) 10

# Context

A company has hired you to develop a program in his customer service department that solves the problem of control and attention of people who arrive at their service centers to be served.

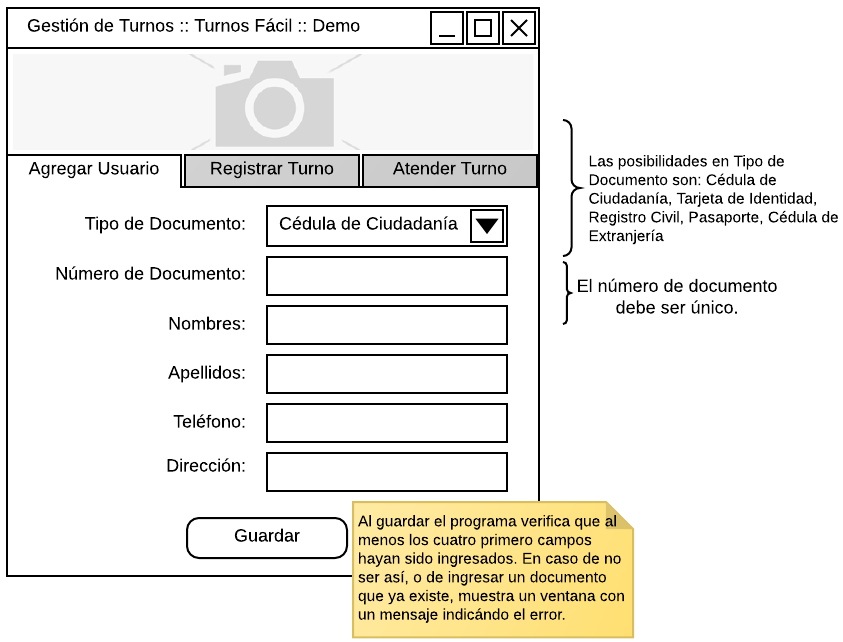
The program will be used by an employee of the company that is at the entrance of the service center and with whom the person who enters to be treated will always communicate in the first instance.

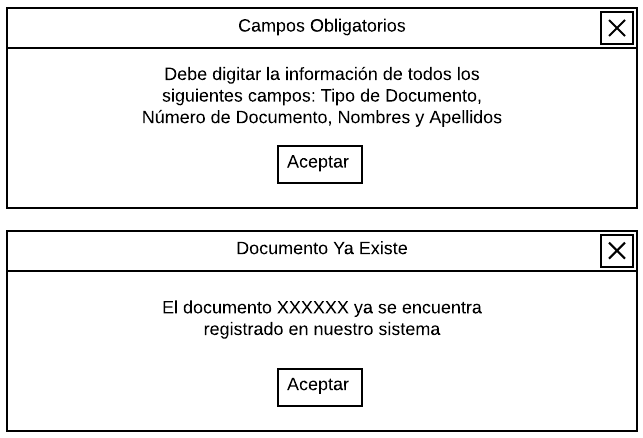
The employee will look for you with your identity document number and then assign a turn to be attended. The turn consists of a letter and a number between 00 and 99. The first turn is the A00, the next the A01 and so on. When the last number of a letter (99) is reached, it will be changed to the first number of the next letter and will continue. For example of the A99 follow the B00.

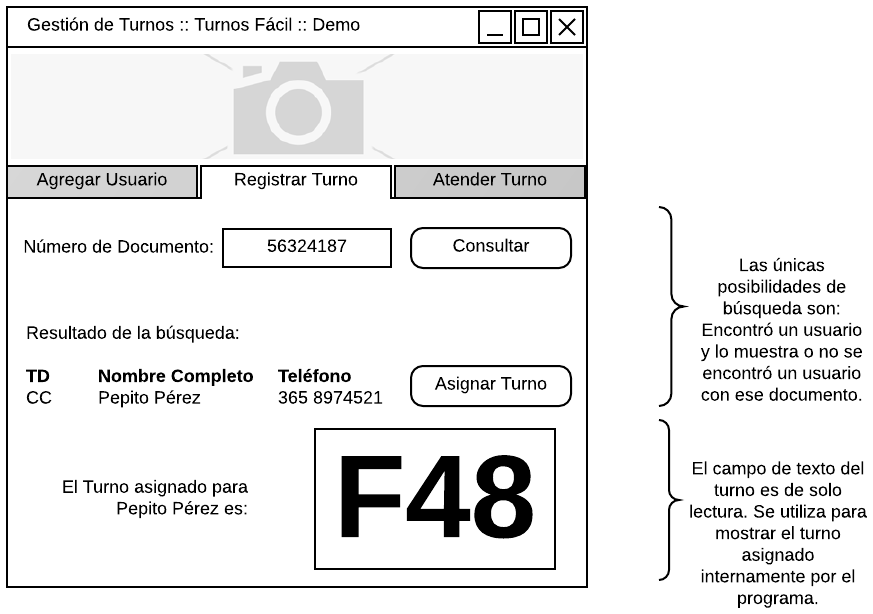
The program will also have the possibility of registering a new user in case it is not already registered in the system. The data of a user are: type of identity document, document number, names, surnames, telephone and address. When adding a new user, it must be validated that at least the type of document, the document number, the names and surnames of the person (ie they are mandatory) must be completed.

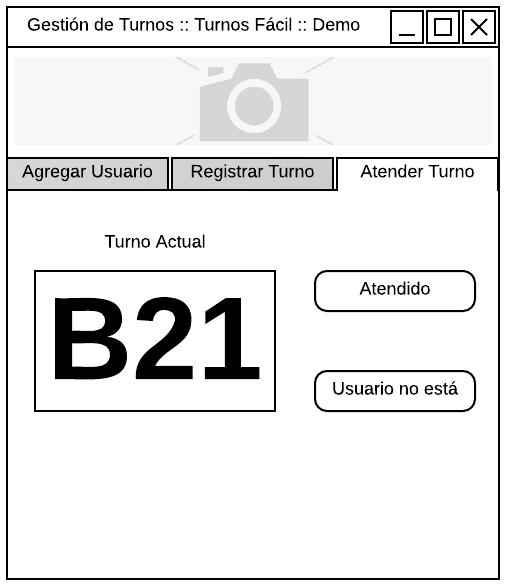
On the other hand, the program must allow progress in the shift to the extent that another employee is serving each client according to the assigned shift. That person who is attending has the possibility to indicate to the program if he really attended the user with that turn or if he did not attend it because he was no longer in the place when he was called to be attended.

The last functionality explained should be in a different program (or at least in a different profile) to that of the other functionalities since it is used by an employee with a different role than the first functionalities explained, however because this is the first version of the program and with demonstration reasons, all the functionalities will be implemented in the same program. Each in a different tab, as shown in the following mockups. The mockups were designed for a second version of the program, you should rely on them to be guided by the options that the program must have but you must develop its interface using a menu of options in a text console.









# Functional requirements:

The turns control system must be able to:

1. Add new users taking into account the following fields: 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.
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. Eventually, a user that had a turn and was dispatched, can get another turn.
3. Dispatch turns starting with the current one. 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 has not appeared. App must allows attend a turn assigned to a user even if there are more turns to be attended ahead of it.

# Test cases design

## Scenarios configuration

|  |  |  |
| --- | --- | --- |
| **Name** | **Class** | **Scenario** |
| emptyObjsSetup | TurnsManagerTest | An object of class TurnsManager with two initialized (and empty) ArrayLists of objects of class Users and Turns. |
| threeUsrsSetup | TurnsManagerTest | An object of class TurnsManager object with the following 3 users added and an empty ArrayList of objects of type Turn:  usr1: object from User class with names=”Melchor Manuel”, surnames=”Reyes García”, id=”123”, typeOfDocument=”CC”, cellphoneNumber=”321”, address=””, turn=null.  usr2: object from User class with names=”Daniel”, surnames=” Gasparín Ordoniez”, id=”124”, typeOfDocument=”IC”, cellphoneNumber=”421”, address=””, turn=null.  usr3: object from User class with names=”Elon”, surnames=”Musk”, id=”125”, typeOfDocument=”FI”, cellphoneNumber=”422”, address=”Cra 107 # 121-20 Desepaz borough”, turn= Turn(“A00”, this). |

## Test cases:

### Encapsulation tests.

#### Turn class

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test objective**: Check that constructor method of the class Turn works properly, assigning values from parameters to its respective attributes. This case is testing Turn(String, User) method, as well as getters and setters methods: getTurn(), getState(), getId() | | | | |
| **Class** | **Method** | **Scenario** | **Inputs** | **Result** |
| Turn | Turn | None | id = “A00”, user = nulll | Object Turn has been created correctly with the values passed in parameters. |

#### User class

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test objective**: Check that constructor method of the class User works properly, assigning values from parameters to its respective attributes. This case is testing User(String, String, String, String, String, String, User) method, as well as getters and setters methods defined: getId(), getTurn() | | | | |
| **Class** | **Method** | **Scenario** | **Inputs** | **Result** |
| User | User | None | in = "Sebas",  s = "García",  id = "123", tod = User.CC,  cpn = "3150550123",  a = "Some Address"  turn = new Turn("A00", null); | Object User has been created correctly with the values passed in parameters. |

### Business logic tests

The following test cases are oriented to verify that every specification declared in the functional requirements is present without problems.

#### Sign up functionality

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test objective**: Verify the correct behavior of addUser(String, String, String, String, String, String, User) method when:   * User to be added have not been registered yet. * User to be added have already been registered. * Required fields are blank (i.e. when name, surnames, id, and type of document parameters are passed as empty Strings). * Alphabetic fields not only contains alphabetic characters * Numeric fields not only contains numeric characters | | | | |
| **Class** | **Method** | **Scenario** | **Inputs** | **Result** |
| Turns Manager | addUser | emptyObjsSetup | n= "Sebastian"; s="Garcia Acosta";  id="123456"; tod=”Identity card” cpn="";  a="";  turn = null | User is added correctly. Can be verified getting the first element in the users list and checking that is equal to the added. |
| Turns Manager | addUser | threeUsrsSetup | n = "Juancho Manuel", s = "Reyes Garcia",  id ="123",  tod=”citizenship card”, cpn="321",  a= "",  turn= null | UserAlreadyRegisteredException is thrown with message: “User with id 123 is already registered.” |
| Turns Manager | addUser | emptyObjsSetup | n = "",  s = "",  id ="",  tod=””,  cpn="123",  a= "Cra 8 #23-12",  turn= null | BlankRequiredFieldException is thrown with message: “Parameters: names, document number, surnames, type of document must be filled.” |
| Turns Manager | addUser | emptyObjsSetup | n="Noobmaster69", s="Salvador123", id="123",  tod= User.CC,  cpn="",  a="",  turn=null | InvalidInputException is thrown with message:  "Invalid format for fields: names, surnames. Fields must contain only alphabetic characters." |
| Turns Manager | addUser | emptyObjsSetup | n="Nando",  s= "Salvador Angulo", id="id3ntity",  tod=”citizenship card”, cpn="abs",  a="",  turn=null | nvalidInputException is thrown with message:  "Invalid format for fields: document number, cellphone number. Fields must contain only numerical characters." |

#### Searching methods

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test objective**: Check that searchUser(String) and searchTurn(String) methods are finding the correct user or turn searched and when there are no users or turns, returns null. | | | | |
| **Class** | **Method** | **Scenario** | **Inputs** | **Result** |
| Turns Manager | searchUser | emptyObjsSetup | id = "36378532" | Returns null |
| Turns Manager | searchTurn | emptyObjsSetup | id=”A00” | Returns null |
| Turns Manager | searchUser | threeUsrsSetup | id=”125” | Returns object from User class with names=”Elon”, surnames=”Musk”, id=”125”, typeOfDocument=”FI”,cellphoneNumber=”422”, address=”Cra 107 # 121-20 Desepaz borough”, turn= Turn(“A00”, this). |
| Turns Manager | searchTurn | threeUsrsSetup | id=”A00” | Returns object of class Turn with id = ”A00” and us r= object from User class with names=”Elon”, surnames=”Musk”, id=”125”, typeOfDocument=”FI”,cellphoneNumbe=”422”, address=”Cra 107 # 121-20 Desepaz borough”, turn= Turn(“A00”, this) |

#### Turn id code generation

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test objective**: Check that for **ALL (26x10²)** **possible turns,** the method generateNextTurnId(Turn) is able to return the next **valid** turn id according to what is defined in functional requirements. E.g., If current turn is an object of class Turn with id=“D99” and usr=null , then generateNextTurnId(Turn) method should return “E0” (usr attribute does not affect), the same logic for: “Z99” and “A00”; “A00” and “A01”; “B99” and “C00”, and so on... | | | | |
| **Class** | **Method** | **Scenario** | **Inputs** | **Result** |
| Turns Manager | generateNextTurnId | emptyObjsSetup | currtTurnId = “<some letter between A-Z><some integer within interval [0,99]>” | Next **valid** turn id according to what is defined in functional requirements. |

#### Registering functionality

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test objective**: Check that registerTurn(String) method is able to assign the next turn available to the user that has the id passed in parameter. In addition, verify that the method throws the correct exceptions when the user owner of the id passed as parameter already have a turn assigned. | | | | |
| **Class** | **Method** | **Scenario** | **Inputs** | **Result** |
| Turns Manager | generateNextTurnId | threeUsrsSetup | id = “123” | User with “123” has a turn assing with id “A01” (Because turn A00 is already assigned to user with id “125”). |
| Turns Manager | generateNextTurnId | threeUsrsSetup | id =”125” | UserAlreadyHasATurnException is thrown with message: "User with id 125 already has turn A00 assigned and its state is: On hold…” |

#### Attend functionality

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test objective**: Check that consultNextTurnToBeAttended() method works properly when there are next turns to be attended or not. In addition, verify that dispatch(Turn, String) method works properly whether there are turns to be attended or not; also, that allows attend a turn assigned to a user even if there are more turns to be attended ahead of it. | | | | |
| **Class** | **Method** | **Scenario** | **Inputs** | **Result** |
| Turns Manager | getCurrentTurn | emptyObjsSetup | None | When n new users and n turns are assigned to each of them. getCurrentTurn returns the first turn that has not be attended. In this case, should return an object of class Turn with id = ”A00” and the user object to whom belongs. |
| Turns Manager | consultNextTurnToBeAttended | emptyObjsSetup | None | An object of clas Turn whose id is “A01” |
| Turns Manager | consultNextTurnToBeAttended | emptyObjsSetup | None | Throws NoSuchElementException with message: "There are no next turn to be attended”. |
| Turns Manager | getCurrentTurn | emptyObjsSetup |  | Throws NoSuchElementException with message "There are no turns in 'On hold' state." when there are no turns waiting for be attended. |