**REQUERIMIENTOS FUNCIONALES**

**Req1. Buscar** usuario a partir de su documento de identidad. Este método recibe el número de identificación del usuario buscado y lo devuelve o, en caso de que no exista, lanzar una excepción que indica que no fue encontrado.

**Req2. Registrar** el nuevo usuario que es ingresado, validando que se diligencien los campos requeridos de este.

**Req3. Avanzar el turno** actual cada vez que se atiende un cliente, indicando además si el usuario fue atendido o se había salido de la fila.

**ESCENARIOS**

|  |  |  |
| --- | --- | --- |
| Nombre | Clase | Escenario |
| setupScenary1 | CompanyTest | Un objeto de clase Company. |
| setupScenary2 | CompanyTest | Un objeto de la clase Company con un objeto de la clase User con los siguientes atributos:   * id = “09876”; * idtype = 4; * firstName = “Taemin”; * lastName = “Lee”; * phone = “58394”; * dir = “random”   Tanto el objeto de la clase Company como el objeto de la clase User tienen asociados un objeto de la clase Turn con los siguientes atributos:   * actualLetter = ‘A’ * actualNumber = 1 |
| setupScenary3 | CompanyTest | Un objeto de la clase Company  con dos objetos asociado de la clase Turn con los siguientes atributos:  Turn t1:   * actualLetter = ‘A’ * actualNumber = 1   Turn t2:   * actualLetter = ‘Z’ * actualNumber = 99 |

**DISEÑO DE PRUEBAS UNITARIAS**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Objetivo de la Prueba:** Comprobar que el constructor y los getters de la clase User funcionen correctamente. | | | | |
| **Clase** | **Método** | **Escenario** | **Valores de Entrada** | **Resultado** |
| User | User() |  | id = “123456”;  idtype = 1;  firstName = “Toshiro”;  lastName = “Kang”;  phone = “333333”;  dir = “random” | Se crea un objeto de User con las entradas correspondientes correctamente. Al usar el get de cada atributo, se devuelve el valor correcto. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Objetivo de la Prueba:** Comprobar que el constructor y los getters de la clase Turn funcionen correctamente. | | | | |
| **Clase** | **Método** | **Escenario** | **Valores de Entrada** | **Resultado** |
| Turn | Turn() |  | actualNumber = 1;  actualLetter = ‘A’; | Se crea un objeto de Turn con las entradas correspondientes correctamente. Al usar el get de cada atributo, se devuelve el valor correcto. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Objetivo de la Prueba:** Comprobar que el arreglo de números de la clase Turn se llena correctamente. | | | | |
| **Clase** | **Método** | **Escenario** | **Valores de Entrada** | **Resultado** |
| Turn | fillNumbers() | setupScenary1() |  | Se llena el arreglo de números del turno de números del 1 al 99. Al recorrer el arreglo, la primera posición debe contener el número 1, y la última el 99. A su vez se comprueba que, desde la primera posición, todas las posiciones siguientes deben ser la suma de la anterior con el número 1. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Objetivo de la Prueba:** Comprobar que un usuario es agregado correctamente. | | | | |
| **Clase** | **Método** | **Escenario** | **Valores de Entrada** | **Resultado** |
| Company | addUser() | setupScenary1()  setupScenary2() | Nuevo usuario a agregar, con los siguientes atributos:   * id = “123456”; * idtype = 1; * firstName = “Toshiro”; * lastName = “Kang”; * phone = “333333”; * dir = “random” | El nuevo usuario debe ser agregado en la última posición del arraylist de usuarios. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Objetivo de la Prueba:** Probar que el método que devuelve el último turno funciona correctamente. | | | | |
| **Clase** | **Método** | **Escenario** | **Valores de Entrada** | **Resultado** |
| Company | getLastTurn() | setupScenary2()  setupScenary3() |  | El objeto Turn devuelto por el método debe ser igual al último objeto Turn del arreglo. |