ESCOLA
POLITÈCNICA SUPERIOR
UNIVERSITAT DE LLEIDA

Universitat de Lleida

Escola Politècnica Superior

Práctica 3 : Pruebas Unitarias

Diseño y Testing de la Plataforma Unificada de Gestión Ciudadana

Autores:
Alexandru Cristian Stoia - Y2386362B
Enric Tobeña Casanovas - 48059124A
Pol Triquell Lombardo - 48054396J

1 de enero de 2023

Índice de Contenidos

1. Introducción **2**

2. Sistema de Control de Versiones **2**

3. Testing de la Plataforma Unificada de Gestión Ciudadana (PUGC) **2**
3.1. Implementación del paquete *Data* 2
3.2. Implementación del paquete *PublicAdministration* 3

4. Decisiones de diseño **3**
4.1. Implementación de la clase *UnifiedPlatformExecutable* 3
4.2. Implementación de la clase *Citizen* 3

5. Análisis de los Resultados **4**

Bibliografía **4**

1. Introducción

En esta práctica estamos aprendiendo a utilizar el sistema de versiones de control Git, como también la implementación de tests unitarios utilizando la librería JUnit5.

Para conseguir lograr este objetivo, utilizamos el proyecto de la UnifiedPlatform que hemos ido desarrollando al largo de la asignatura.

2. Sistema de Control de Versiones

En esta práctica se nos pedia utilizar una herramienta de control de versiones, en nuestro caso, consideramos que la mejor opción para disponer de un historial de todos los cambios realizados es GitHub. [Enr22]

La que hemos considerado, la mejor manera para organizarnos ha sido, tal y como se nos especificó en las clases, el hecho de utilizar una rama para cada una de las modificaciones sustanciales hechas en el código. Para cada implementación, se ha creado una rama con la finalidad de aislar a todos los intengrantes del grupo, haciendo así, un trabajo autonómo para cada posible prueba unitaria. Así conseguimos no depender unos de los otros con las clases y pruebas que cada integrante del grupo implemente por separado.

De esta forma, conseguimos que todo el código que esté en la rama *main* sea la versión final de este, aunque puede ser que haya algun cambio a posteriori.

3. Testing de la Plataforma Unificada de Gestión Ciudadana (PUGC)

Uno de nuestros objetivos en esta práctica era aprender a realizar pruebas unitarias del código, cosa que hemos hecho con la ayuda de la librería de Java JUnit 5.

Hemos implementado todos los test en función de las excepciones que teníamos, es decir, excepciones que, por lo general, se daban a la hora de invocar al constructor de cierta clase con unos parámetros erroneos.

Para ello, hemos implementado diferentes tests para cada paquete definido en el proyecto.

3.1. Implementación del paquete *Data*

En este paquete, hemos probado todos los getters que asignaban los atributos a las clases y las correspondientes excepciones y sus respectivos mensajes de error definidos en las cabeceras de todos los métodos.

Además también hemos testeado el correcto formato de ciertos atributos requeridos, los cuales son:

- Nif : Se ha probado que su medida sea de nueve números más una letra en mayúsculas. También se ha comprobado que la letra sigue la fórmula correcta.
- Password : Se ha probado que la medida mínima sea de cuatro y contenga almenos, un carácter, un carácter especial, un número y una letra mayúscula.
- SmallCode : Se ha probado que su medida sea de tres dígitos y que solo sean válidos carácteres numéricos.

3.2. Implementación del paquete *PublicAdministration*

En este paquete, hemos probado todos los getters que asignaban los atributos a las clases y las correspondientes excepciones y sus respectivos mensajes de error definidos en las cabeceras de todos los métodos. Haciendo más incapié en las fechas en las que se piden los trámites y el control de instancias *null* que lanzaban *NullPointerException*.

Además también hemos testeado el correcto formato de ciertos atributos requeridos, los cuales son:

- CardPayment : Se ha probado que dados un NIF, una fecha y un importe, correspondan con el generado por el sistema. También hemos tenido en cuenta que no se pueda pedir un importe negativo.
- Citizen : Se ha probado que dados un nombre, una dirección y un número de teléfono, cumplan con el correcto formato requerido.
- CreditCard : Se ha probado que la tarjeta sea válida y cumpla con el formato de los atributos, es decir, que tenga dieciséis dígitos, que no haya ningún carácter no númerico entre esos, y que no esté caducada la tarjeta.
- CrimConvictionsColl : Se ha probado que se pueden agregar los delitos correctamente, que no pueden ser *nulls* y que se pueden repetir los delitos en fechas diferentes.
- CrimConvictions : Se ha probado que los atributos de un delito sean correctos y cumplan con el formato estipulado.
- CriminalRecordCertf : Se ha probado que genera un certificado con los atributos apropiados.

4. Decisiones de diseño

4.1. Implementación de la clase *UnifiedPlatformExecutable*

Siguiendo con nuestro metodología de programación, hemos considerado conveniente implementar una clase que simulará el funcionamiento real de nuestra *UnifiedPlatform* y por consiguiente, seguir el DSS que se nos ha otorgado como modelo.

De esta manera, teniendo un seguido de pasos a tener en cuenta del orden de las acciones, nos ayudará durante todo el proceso de implementación del proyecto y ver esos cambios que hemos ido realizando.

4.2. Implementación de la clase *Citizen*

En el caso de la clase *Citizen* tenemos muchas dependencias con todos los datos de un ciudadano, como por ejemplo, el NIF, la dirección, el número de teléfono... Con lo que hemos considero oportuno crear diferentes constructores con la posibilidad de no tener alguno de los atributos disponibles y ayudarnos en la implementación. También hemos comprobado que los datos que nos introduciesen fuesen todos corerctos y con el formato demandado.

Una vez esta clase fue implementada, la necesidad de la misma quedó muy natural y en la estructura del proyecto se puede ver reflejada, tanto en la implementación como en el posterior *testing*.

Para analizar esta casuística nos hemos inspirado en el Principio S de los principios SOLID, el de Responsabilidad Única.

5. Análisis de los Resultados

Una vez terminado todo el proyecto, nos gustaría puntualizar el hecho de haber creído necesario hacer ciertos cambios de estructura en algunas de las funcionalidades que se pedían implementar en el enunciado.

Para precisar un poquito mejor, hemos realizado el cambio en la interfície *CertificationAuthorityInterface*, donde hemos identificado el code smell refused bequest, dentro de la categoría *Object-Oriented Abusers*. Este code smell es producido en nuestra interfície debido a las dos *CertificationAuthorities*, como son, *ClavePINCertificationAuthority* y *ClavePermanenteCertificationAuthority*. Cada clase puede utilizar métodos comunes pero en el caso que se escoja Cl@vePermamente, habrá que implementar ese segundo paso de autenticación reforzado y por lo tanto, en algunos métodos se les dará un uso en concreto a cada uno de ellos de esa clase.

Haciendo eso intentamos promover el hecho que se pierda cierta parte de las funciones que las *CertificationAuthorities* heredan de su interfície padre, violando asó el Principio I de los principios SOLID.

Por último pero no menos importante, hemos considerado oportuno utilizar protección en la branca *main* para no tener problemas con la versión final del proyecto. También hemos decidido utilizar como un doble factor de verificación antes de hacer un *commit* y subir los cambios a nuestro repositorio de *Git*, así, cualquier miembro del grupo que desee subir algun cambio de la implementación tendrá que recibir el visto bueno de los dos otros miembros del equipo.

Bibliografía

[Enr22] Alexandru Cristian Stoia Enric Tobeña Casanovas Pol Triquell Lombardo. *Práctica 3 sobre Pruebas Unitarias.* Diciembre de 2022. URL: `https://github.com/aleex0309/UDLEngProg-Pruebas-Unitarias.git`.