ANDREI GARCÍA CUADRA

100405803

MIGUEL HERNÁNDEZ CASSEL

100405956

PRINCIPIOS DESARROLLO SOFTWARE

PRÁCTICA 3 - SECCIÓN COMPLETA

PRÁCTICA 3

1. ANÁLISIS DE CLASES DE EQUIVALENCIA Y VALORES LÍMITE

Criterio	ID requerimiento	Descripción	Expected output	ID Test
Existencia del fichero	**LM-RF-01-E1** **LM-RF-02-E1**	El fichero no se encuentra en la ruta disponible	*TokenManagementException("No se encuentra el fichero con los datos de entrada.")*	GenericInputFileTest
Implementación de interfaces	**LM-RF-01**	Debe implementar la interfaz TokenRequestGeneratorInterface	*TestAbortedException("Class must implement interface")*	TokenRequestGeneratorTest
	LM-RF-02	Debe implementar la interfaz TokenRequestInterface.	*TestAbortedException("Class must implement interface")*	TokenRequestTest
	LM-RF-03	Debe implementar la interfaz TokenManagerInterface	*TestAbortedException("Class must implement interface")*	TokenManagerTest
Sintaxis, semántica y valores límite del fichero JSON/ CSV	**LM-RF-01-O1**	El campo "Device Name" debe tener de 1 a 20 caracteres.	De	TokenRequestGeneratorTest
		El campo "Type of device" debe tener el valor "Sensor" o "Actuador" respetando mayúsculas.		TokenRequestGeneratorTest
		El campo "Driver Version" debe contener dígitos separados por ., sin poder empezar/ acabar por . y siendo menor o igual a 25 caracteres.		TokenRequestGeneratorTest
		El campo "Support e-mail" debe contener un email válido (RFC-5322).		TokenRequestGeneratorTest
		El campo "Serial Number" debe contener una combinación de al menos una letra, un número y una barra. No puede contener espacios.		TokenRequestGeneratorTest
		El campo "MAC Address" debe contener una MAC válida (RFC-2469).		TokenRequestGeneratorTest
		Los campos deben tener el nombre exacto, validando mayúsculas y minúsculas.		TokenRequestGeneratorTest
	LM-RF-02-E1 **LM-RF-02-O1**	El campo "Token Request" debe tener un tokenReques válido.		TokenRequestTest
		El campo "Notification e-mail" debe ser un email válido (RFC-5322).		TokenRequestTest
		El campo "Request Date" debe ser una fecha válida siguiendo el formato "dd/mm/yyyy HH:MM:SS".		TokenRequestTest
	LM-RF-02-O2	El token será almacenado en memoria y, en caso de no poder, se lanzará una excepción.		TokenRequestTest
	LLM-RF-01-S1 **M-RF-01-S2**	Se deben codificar las salidas y excepciones.		GenericInputFileTest TokenRequestGeneratorTest TokenRequestTest TokenManagertTest
	LM-RF-02-S2			

Criterio	ID requerimiento	Descripción	Expected output	ID Test
	LM-RF-03-O1	El componente deberá verificar que el token es válido		TokenRequestGenera-torTest
	LM-RF-03-02	El componente verifica que el token no está expirado.		TokenRequestTest
	LM-RF-03-02	El componente verifica que el token estaba almacenado.		TokenManagerTest
Análisis de nulos	**LM-RF-01-E1**	La ruta del fichero es un valor nulo.	*TokenManagement-Exception("No se encuentra el fichero con los datos de entrada.")*	GenericInputFileTest
	LM-RF-02-E1			
	LM-RF-03-E1			
	LM-RF-01-E1	El fichero contiene valor nulo en cualquiera de sus atributos.	*TokenManagement-Exception("No se encuentra el fichero con los datos de entrada.")*	TokenRequestGenera-torTest
	LM-RF-02-E1	El fichero contiene valor nulo en cualquiera de sus atributos.		TokenRequestTest
	LM-RF-03-E1	El fichero contiene valor nulo en cualquiera de sus atributos.		TokenManagerTest
Lanzamiento de excepciones	**LM-RF-01-S2**	Excepción de tipo (*TokenManagementException*) con mensaje "No se encuentra el fichero con los datos de entrada".	Excepción definida en JUnit.	TokenRequestGenera-torTest
	LM-RF-01-S2/2	Excepción de tipo (*TokenManagementException*) con mensaje "El fichero de entrada no contiene los datos o el formato esperado".		TokenRequestGenera-torTest
	LM-RF-01-S2/3	Excepción de tipo (*TokenManagementException*) con mensaje "Se ha producido un error interno en la generación del Token Request.".		TokenRequestGenera-torTest
	LM-RF-02-O1/LM-RF-02-S2:	Excepción de tipo (*TokenManagementException*) con mensaje "Se ha producido un error interno en la generación del token.".		TokenRequestTest
	LM-RF-02-O2/LM-RF-02-O2	Excepción de tipo (*TokenManagementException*) con mensaje "Se ha producido un error interno en el almacenamiento del token.".		TokenRequestTest
	LM-RF-02-S1/LM-RF-02-O2	Excepción de tipo (*TokenManagementException*) con mensaje "Se ha producido un error interno en la codificación del token.".		TokenRequestTest
Mostrar mensajes por pantalla	**LM-RF-01-O2 LM-RF-01-S1**	Se deberá mostrar el valor de Token Request por pantalla (MD5 + SALT).	{MSG}	N/A
	LM-RF-03-S1 LM-RF-03-S2	Se deberá mostrar por pantalla si el token es válido o se ha producido un error.		N/A

*1: Se pueden combinar clases de equivalencia válida en un único caso de prueba si es posible. Cada clase de equivalencia inválida y valor límite relevante deberá considerarse en un caso de prueba independiente.

*2: Se pueden dar casos en los que la memoria no contenga todos los casos de prueba, no obstante, todos están documentados en JUnit. En este caso, se han denominado de modo "friendly" las ID de los test en lugar de enlazar con este documento, el cual tiene un comportamiento meramente informativo.

2. FUNCIONALIDAD TÉCNICA INTERNA

- Se ha testeado en base a todos los casos no válidos, de este modo los válidos salen por descarte de éstos. Por ejemplo, no requerimos validar si el archivo existe, ya que validamos que no existe. Si el fichero no cede la excepción de no existir, intrínsecamente existe, por ende nos ahorramos este test.

Funcionalidad LM-RF-01:

- Se debe **crear** una **interfaz** *TokenRequestGeneratorInterface* con los siguientes métodos:
  - *Nombre*: TokenRequestGeneration
    - *Return*:
      (1) String. ID de generación dede solicitud del token.
    - *Argumentos*:
      (1) String - inputFile.
    - *Excepciones* lanzadas:
      (1) *TokenManagementException*.
- Se debe **crear** una **clase** *TokenRequestGenerator* con los siguientes métodos:
  - *Nombre*: TokenRequestGeneration
    - *Return*:
      (1) String.
    - *Argumentos*:
      (1) String - *inputFile*.
    - *Excepciones* lanzadas:
      (1) *TokenManagementException*.
    - Comportamiento:

(1) Debe verificar que existe el fichero de entrada. En caso de no existir, debe devolver una excepción de tipo *TokenManagementException* con el mensaje *"No se encuentra el fichero con los datos de entrada."*.

(2) Debe verificar el fichero de entrada. Debe contener los campos (respetando mayúsculas y minúsculas). *NOTA: Se aceptan campos extra ya que nos será indiferente y no afectará a la ejecución del programa*:

i. "Device Name": Caracteres alfabéticos con una longitud de 1 a 20. En caso de que no se cumpla, lanzará una excepción del tipo *InvalidTokenRequestException,* con el mensaje *"El nombre del dispositivo no es válido para la solicitud de creación de petición del token."*.

ii. "Type Of Device": El valor debe ser Sensor o Actuator (validando mayúsculas y minúsculas). En caso de que no se cumpla, lanzará una excepción del tipo *InvalidTokenRequestException,* con el mensaje *"El tipo de dispositivo no es válido para la solicitud de creación de petición del token."*.

iii. "Driver Version": Dígitos separados por el caracter "." de longitud mayor o igual que 1 y menor o igual que 25. No pueden empezar ni acabar por el caracter ".". En caso de que no se cumpla, lanzará una excepción del tipo *InvalidTokenRequestException,* con el mensaje *"El driver del dispositivo no es válido para la solicitud de creación de petición del token."*.

iv. "Support e-mail": Email con formato válido según RFC-5322. En caso de que no se cumpla, lanzará una excepción del tipo *InvalidTokenRequestException,* con el mensaje *"El email de soporte no es válido para la solicitud de creación de petición del token."*.

v. "Serial Number": Combinación de letras, números y barras sin espacios y de longitud indeterminada. En caso de que no se cumpla, lanzará una excepción del tipo *InvalidTokenRequestException,* con el mensaje *"El serial no es válido para la solicitud de creación de petición del token."*.

vi. "MAC Address": MAC con formado válido según RFC-2469. En caso de que no se cumpla, lanzará una excepción del tipo *InvalidTokenRequestException,* con el mensaje *"La dirección MAC es válida para la solicitud de creación de petición del token."*.

(3) Debe verificar el fichero de entrada. En caso de no cumplir todas las restricciones del apartado (2) debe devolver una excepción de tipo *TokenManagementException* con el mensaje *"El fichero de entrada no contiene los datos o el formato esperado."*.

(4) Deberá codificar el objeto TokenRequest utilizando el algoritmo MD5 y un salt ("contraseña") definido.

(5) En caso de no poder llevar a cabo el hash del apartado (4), debe devolver una excepción de tipo *TokenManagementException* con el mensaje *"Se ha producido un error interno en la generación del Token Request."*.

(6) Deberá mostrar un mensaje en pantalla (*system.out.println*) el valor de *TokenRequest*.

Funcionalidad LM-RF-02:

- Se debe **crear** una **interfaz** *TokenRequestInterface* con los siguientes métodos:

  - *Nombre*: *RequestToken*

    - *Return*:

      (1) String. Devuelve el token consolidado.

    - *Argumentos*:

      (1) String - *inputFile*.

    - *Excepciones* lanzadas:

      (1) *TokenManagementException*.

- Se debe **crear** una **clase** *ManageTokenRequest* con los siguientes métodos:

  - *Nombre*: *RequestToken*

    - *Return*:

      (1) String.

    - *Argumentos*:

      (1) String - *inputFile*.

    - *Excepciones* lanzadas:

      (1) *TokenManagementException*.

    - Comportamiento:

      (1) Debe verificar que existe el fichero de entrada. En caso de no existir, debe devolver una excepción de tipo *TokenManagementException* con el mensaje *"No se encuentra el fichero con los datos de entrada"*.

      (2) Debe verificar el fichero de entrada. Debe contener los campos (respetando mayúsculas y minúsculas). *NOTA: Se aceptan campos extra ya que nos será indiferente y no afectará a la ejecución del programa*:

i. "Token Request": Debe ser un TokenRequest válido. En caso de que no se cumpla, lanzará una excepción del tipo *InvalidTokenException,* con el mensaje "La solicitud de token no es válida.".

ii. "Notification e-mail": Debe ser un email válido según RFC-5322. En caso de que no se cumpla, lanzará una excepción del tipo *InvalidTokenException,* con el mensaje "El email de notificación del token no es válido.".

iii. "Request date": Fecha válida con el siguiente formato dd/mm/yyyy HH:MM:SS. No puede contener días inferiores a 01, ni puede tener días superiores a 31. No puede contener meses inferiores a 1 ni superiores a 12. Se debe validar el año 0000 y el año 9999. En caso de que no se cumpla, lanzará una excepción del tipo *InvalidTokenException,* con el mensaje "La fecha de expiración del token no es válida.".

(3) Debe verificar el fichero de entrada. En caso de no cumplir todas las restricciones del apartado (2) debe devolver una excepción de tipo *TokenManagementException* con el mensaje *"El fichero de entrada no contiene los datos o el formato esperado."*.

(4) Deberá generar el token final, que será un objeto de tipo Token (generado a partir de las bases del punto (10).

(5) Deberá almacenar el token en memoria.

(6) Deberá codificar el token en base64 siguiendo el algoritmo de codificación *base64urlencoding* con los tres elementos del token generado.

(7) En caso de no cumplir el apartado (5), debe devolver una excepción de tipo *TokenManagementException* con el mensaje *"Se ha producido un error interno en el almacenamiento del token."*.

(8) En caso de no cumplir el apartado (6), debe devolver una excepción de tipo *TokenManagementException* con el mensaje *"Se ha producido un error interno en la codificación del token."*.

(9) En caso de no poder llevar a cabo cualquier operación no contemplada en estos apartados que lanzara una excepción, debe devolver una excepción de tipo *TokenManagementException* con el mensaje *"Se ha producido un error interno en la generación del Token Request."*.

(10) Crear objeto de tipo Token, que contendrá la siguiente estructura:

➡ Header:

- *alg*: Algoritmo a utilizar. Debe ser el mismo que el campo “Signature”. En este caso, HS256 (hardcodeado, aunque este valor puede cambiar, se utilizará un enum).

- *typ*: Tipo de token a generar. Debe admitir el valor “PDS”, pero se utilizará un enum ya que en el futuro esta lista se puede expandir.

➡ Payload:

- *device*: (String) Cadena de caracteres de Token Request.

- *issued at*: (timestamp) Fecha de emisión del token.

- *expiration date*: (timestamp) Fecha de expiración del token.

➡ Signature: Firma del algoritmo. Debe contener el valor “HS256” hardcodeado (mismo valor que campo Header -> Payload).

Funcionalidad LM-RF-03:

- Se debe **crear** una **interfaz** *TokenManagerInterface* con los siguientes métodos:

  - *Nombre*: *VerifyToken*

    - *Return*:

      (1) boolean. Verifica si el token es válido.

    - *Argumentos*:

      (1) String - *token*.

    - *Excepciones* lanzadas:

      (1) *LMException*.

- Se debe **crear** una **clase** *TokenVerifier* con los siguientes métodos (implementando la interfaz *TokenManagerInterface)*:

  - *Nombre*: TokenManager

    - *Return*:

      (1) boolean. Verifica si el token es válido.

    - *Argumentos*:

      (1) String - *token*. Codificado en *base64urlencoding*.

    - *Excepciones* lanzadas:

      (1) *LMException*.

    - Comportamiento:

      (1) Debe decodificar el token.

      (2) En caso de no poder llevar a cabo el apartado (1) debe devolver una excepción de tipo *TokenManagementException* con el mensaje *"La cadena de caracteres de la entrada no se corresponde con un token que se pueda procesar."*.

(3) Debe verificar que la fecha de expiración del token no ha expirado. En dicho caso, debe devolver una excepción de tipo *TokenManagementException* con el mensaje *"Token expirado."*.

(4) Debe verificar que el token está registrado y en vigor, registrado en memoria.

(5) En caso de no poder llevar a cabo el apartado (4) debe devolver una excepción de tipo *TokenManagementException* con el mensaje *"No se encuentra registrado el token para el cual se solicita verificación."*.

3. EXTRAS

1. Se ha aplicado la herramienta de testing sonarcloud. Se adjuntan datos a continuación.

2. Se ha agregado integración continua para compilar este fichero de documentación, el cual utiliza la tecnología Pages, para compilarlo a PDF en cada subida de commit y guardarlo bajo el nombre de fichero "Documentación.pdf" en la ruta principal del repositorio.

3. Se ha ejecutado un análisis de código de "*coverage*" para verificar la calidad del código y que no exista código muerto. Se adjunta la información de los resultados del análisis debajo.

4. Definición de rutas para ayuda a la corrección: La ruta de los tests (JUnit) se localiza bajo la carpeta src/test/java/transport4future/tokenManagement. El código funcional se ubica en la ruta src/main/java/transport4future/tokenManagement.

5. Se ha generado un informe generado en XML por JUnit que incluye los resultados de la ejecución de los casos incluidos para cada una de las clases de prueba que contienen los casos considerados para cada una de las funcionalidades y técnicas consideradas, en la ruta principal del repositorio bajo el nombre "INFORME_PRUEBAS.xml".

6. Se ha validado que todos los casos de prueba (clases equivalencia/valores límite) contienen la siguiente cabecera:

```
/**
* Caso de Prueba: ID/NOMBRE
* Clase de Equivalencia o Valor Límite Asociado: VALOR
*  Técnica de prueba: CLASE_EQUIVALENCIA/VALOR_LIMITE
*  Resultado Esperado: DESCRIPCION
* */
```

7. Se ha revalidado todo el código para verificar que es bullet-proof, y no contiene errores detectables. Se han validado las fugas de datos de código.

8. Se ha subido un commit con cada caso de prueba.

9. Se ha realizado un commit inicial con todos los cambios relativos a esta memoria.

10. Se ha dejado un registro XML con la evidencia de la correcta ejecución de los casos de prueba dados.

11. Se han agregado ficheros de configuración en base al entorno.

12. Se ha validado que todos los casos de prueba (código de prueba/funcional) contienen la siguiente cabecera:

```
/**
* Caso de Prueba: ID/NOMBRE
* Nodo/s del Árbol de Derivación: VALORES_ESPECIFICOS
* Tipo de Prueba: Valor Normal | Omisión | Repetición | Omisión
*  Técnica de prueba: Análisis Sintáctico
* Resultado Esperado: DESCRIPCION
* */
```

13. Se han realizado tests sobre los controladores (paquete controllers) del proyecto debido a que así se pide. No obstante, el equipo lo habría hecho sobre los modelos, los servicios, etc, ya que de esta forma testaríamos sobre la base y no sobre la capa (ya que, si hiciéramos otros controladores que usen los mismos modelos, podrían seguir teniendo los fallos, ya que no son validados a nivel de modelo sino de controlador).

14. Los test agregan “hints”, de modo que si uno falla, compartirá al desarrollador un mensaje amigable para solucionar el código.

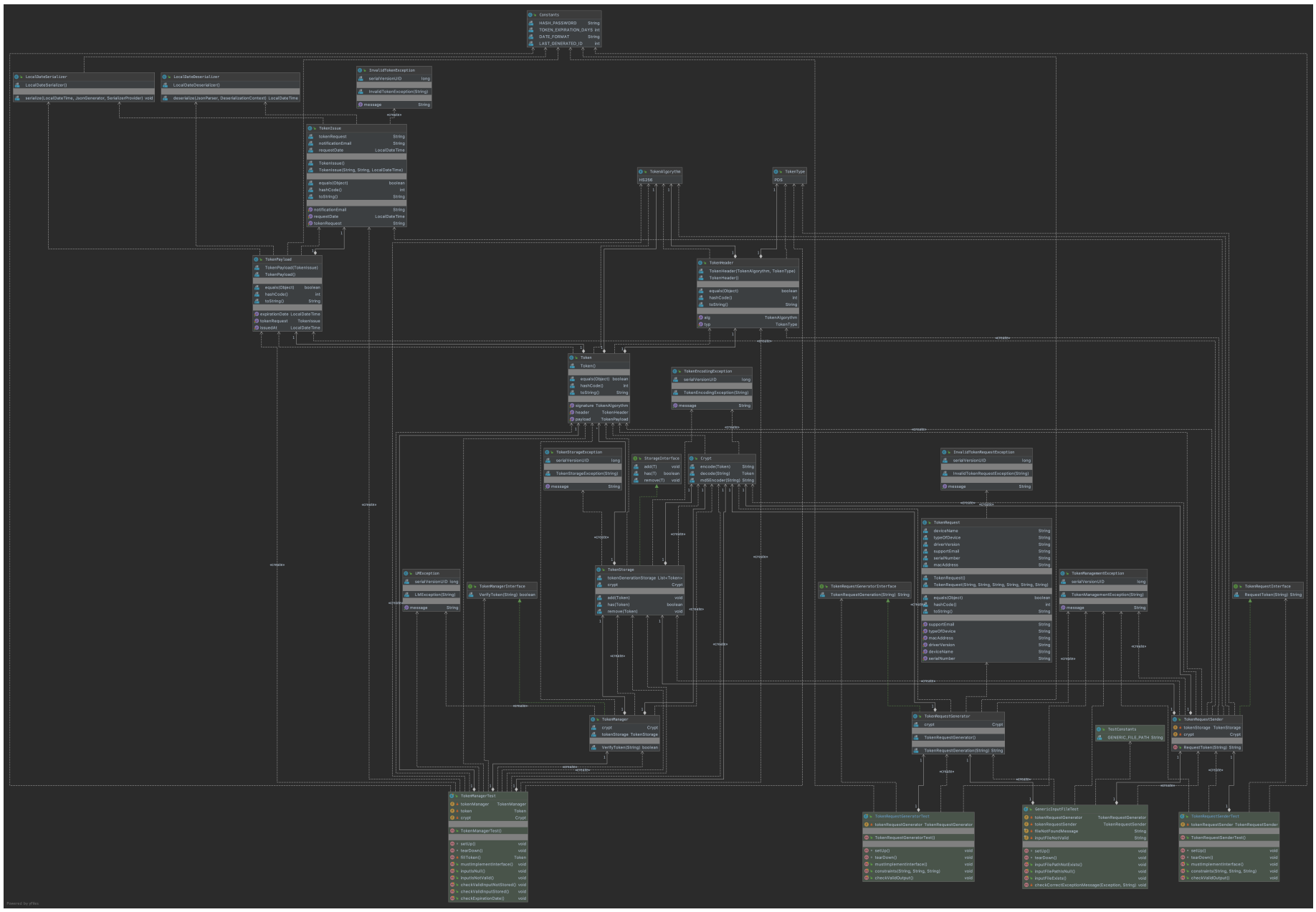
15. Se ha generado una carpeta “javadocs” que contiene toda la documentación relativa al proyecto. Se puede acceder a él mediante la apertura de “javadocs/index.html”.

16. Los XML resultado de los test se han guardado en la carpeta ./src/test/java/results.

17. Se ha implementado un patrón de diseño arquitectónico MVC*, son servicios integrados.

18. Se ha integrado un sistema de constantes para la validación y consistencia de datos.

4. DIAGRAMA UML



Para ver el diagrama más amplio, se puede ver desde la carpeta docs/UML_DIAGRAM.png

5. GRAMÁTICA

5.1. Gramática

5.1.1. Fichero valid-token-request.json

Este fichero se usa en la clase TokenRequestSender.

```
STARTFILE := INICIO_FICHERO_OBJ
STARTLINE := CORCHETE_APERTURA
STARTLINE := INICIO_ATRIBUTO
CHAR := COMILLAS_DOBLES
STR := Token Request
CHAR := COMILLAS_DOBLES
CHAR := DOS_PUNTOS
CHAR := COMILLAS_DOBLES
INPUT := [VALOR_ENTRADA]
CHAR := COMILLAS_DOBLES
CHAR := COMA
DEBUG := FIN ATRIBUTO
STARTLINE := INICIO_ATRIBUTO
CHAR := COMILLAS_DOBLES
STR := Notification e-mail
CHAR := COMILLAS_DOBLES
CHAR := DOS_PUNTOS
CHAR := COMILLAS_DOBLES
INPUT := [VALOR_ENTRADA]
CHAR := COMILLAS_DOBLES
CHAR := COMA
DEBUG := FIN ATRIBUTO
STARTLINE := INICIO_ATRIBUTO
CHAR := COMILLAS_DOBLES
STR := Request Date
CHAR := COMILLAS_DOBLES
CHAR := DOS_PUNTOS
CHAR := COMILLAS_DOBLES
INPUT := [VALOR_ENTRADA]
CHAR := COMILLAS_DOBLES
CHAR := COMA
STARTLINE := CORCHETE_CIERRE
ENDFILE := FIN_FICHERO_OBJ

CONSTRAINT[Token Request] := CHARLENGTH(UNDEFINED),CHARS(a-Z,1-9),MUST_BE_VALID_TOKEN
CONSTRAINT[Notification e-mail] := RFC-5322.
CONSTRAINT[Request Date] := STRING_ENCODED_DATE_FORMAT[dd/mm/yyyy HH:MM:SS]
```

5.1.2. Fichero valid-token.json

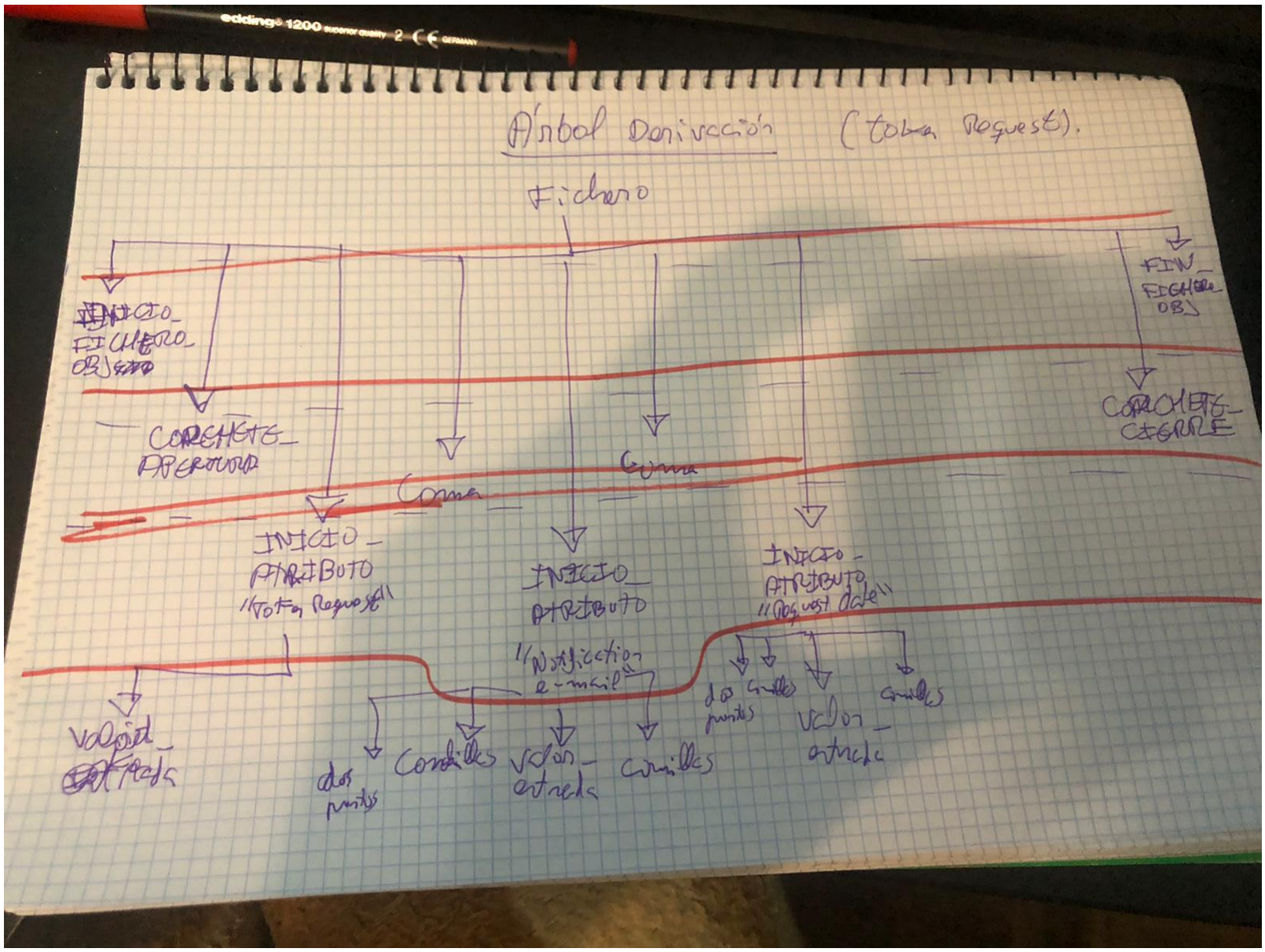
Este fichero se usa en la clase TokenRequestGenerator.

```
STARTFILE := INICIO_FICHERO_OBJ
STARTLINE := CORCHETE_APERTURA
STARTLINE := INICIO_ATRIBUTO
CHAR := COMILLAS_DOBLES
STR := Type of device
CHAR := COMILLAS_DOBLES
CHAR := DOS_PUNTOS
CHAR := COMILLAS_DOBLES
INPUT := [VALOR_ENTRADA]
CHAR := COMILLAS_DOBLES
CHAR := COMA
DEBUG := FIN ATRIBUTO
STARTLINE := INICIO_ATRIBUTO
CHAR := COMILLAS_DOBLES
STR := Driver Version
CHAR := COMILLAS_DOBLES
CHAR := DOS_PUNTOS
CHAR := COMILLAS_DOBLES
INPUT := [VALOR_ENTRADA]
CHAR := COMILLAS_DOBLES
CHAR := COMA
DEBUG := FIN ATRIBUTO
STARTLINE := INICIO_ATRIBUTO
CHAR := COMILLAS_DOBLES
STR := Support e-mail
CHAR := COMILLAS_DOBLES
CHAR := DOS_PUNTOS
CHAR := COMILLAS_DOBLES
INPUT := [VALOR_ENTRADA]
CHAR := COMILLAS_DOBLES
CHAR := COMA
DEBUG := FIN ATRIBUTO
STARTLINE := INICIO_ATRIBUTO
CHAR := COMILLAS_DOBLES
STR := Serial number
CHAR := COMILLAS_DOBLES
CHAR := DOS_PUNTOS
CHAR := COMILLAS_DOBLES
INPUT := [VALOR_ENTRADA]
CHAR := COMILLAS_DOBLES
CHAR := COMA
DEBUG := FIN ATRIBUTO
STARTLINE := INICIO_ATRIBUTO
CHAR := COMILLAS_DOBLES
STR := Mac Address
CHAR := COMILLAS_DOBLES
CHAR := DOS_PUNTOS
CHAR := COMILLAS_DOBLES
INPUT := [VALOR_ENTRADA]
CHAR := COMILLAS_DOBLES
CHAR := COMA
DEBUG := FIN ATRIBUTO
STARTLINE := CORCHETE_CIERRE
ENDFILE := FIN_FICHERO_OBJ

CONSTRAINT[Type of device] := CHARLENGTH(1,20),CHARS(a-Z,1-9)
CONSTRAINT[Driver version] := ENUM[Sensor,Actuator]
CONSTRAINT[Support e-mail] := RFC-5322.
CONSTRAINT[Serial number] := CHARLENGTH(1,20),CHARS(a-Z,1-9,-)
CONSTRAINT[Mad Address] := RFC-2469.
```

5.2. Árbol de derivación

5.2.1 Se ha realizado para la casuística del fichero valid-token-request.json



Nota: No se ha utilizado draw.io debido a periodos de latencia álgidos en otras tareas.

5.3. Identificación e implementación de casos de prueba

En este caso, se han implementado todos los casos de prueba en la primera tanda de tests: tanto, es por ello que aquí los reidentificamos, pero no podemos reimplementarlos ya que duplicaríamos el código. Sentimos mucho las molestias causadas por ello, a veces vamos algo anticipados y pasa factura. En este caso, hemos considerado las siguientes casuísticas:

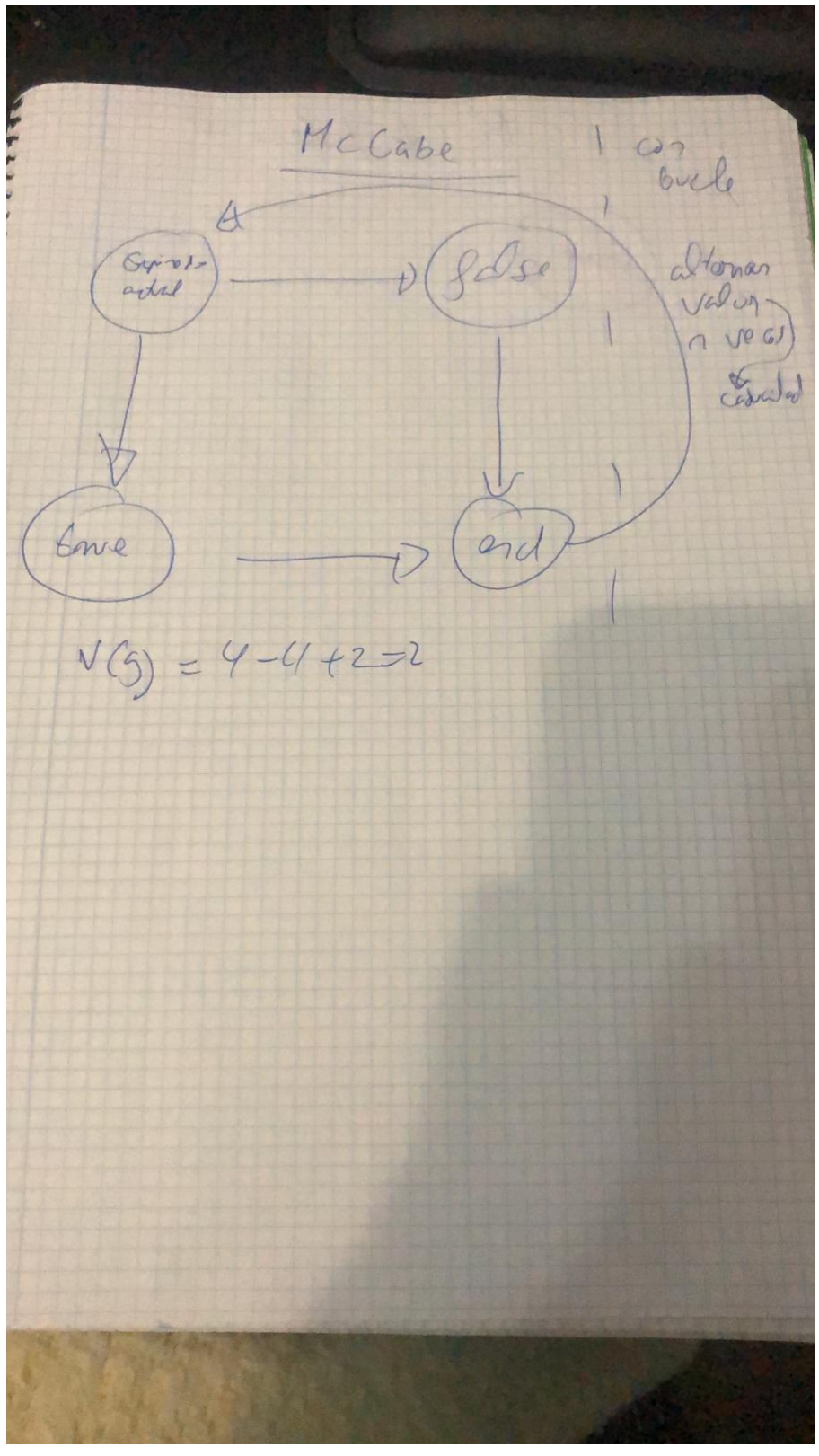
- **Caso verífico**: el fichero es correcto.
- **Omisión léxica**: El fichero contiene un JSON mal formado (falta de brackets, dos puntos, comillas, comas….).
- **Omisión de valor**: Se han omitido valores e incluso puesto a null.
- **Repetición**: Se han repetido valores, se ha comprobado que se coge el último valor.
- **Adhesión**: Se han añadido campos extra que no vienen a cuento, y se ha comprobado que estos se ignoran ya que no afectan a nuestra casuística y tratarlos generaría latencias innecesarias.

No obstante, las pruebas citadas están en los tests parametrizados y se ha documentado su cabecera. El fichero de dichos test se denomina *TokenRequestTest*.

Denotar, del mismo modo, que nuestro TokenStore es dinámico y funciona en memoria. De este modo, podemos tratar de manera más rápida y ágil los elementos, además de interponer una capa de servicio, la cual en un futuro próximo podría tratarse como un objeto dinámico de storing en bases de datos de cualquier tipo (capa abstracta).

6. TÉCNICAS DE PRUEBA ESTRUCTURALES

En primer lugar, queremos comentar que en este apartado, nuestro objeto Token contiene objetos nesteados los cuales, de modo más eficiente y containerizados, nos permite de/serializarlos de una manera óptima y tratarlos de manera más simple, además de ser más mantenible y sofisticado.



6.3. McCabe: Implementación

Volvemos a la misma quimera anterior… Esto ya se implementó al comienzo, junto al resto de la pila de tests… Por ende, no nos fue necesario realizar mucha tarea en este aspecto.

Se puede encontrar en la clase *TokenManagerTest, en los métodos checkValidInputStored y checkValidInputStoredLoop*.