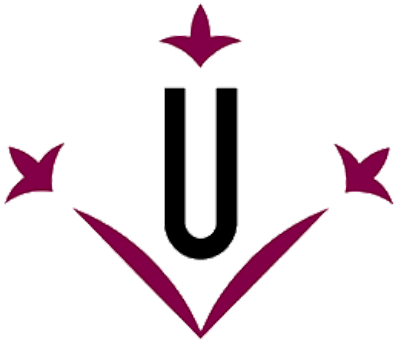

Universitat de Lleida
Escola Politècnica Superior

Enginyeria del programari
Pràctica de proves unitàries

Grup: PraLab1

Aarón Arenas Tomás – 78098697N

Marc Cervera Rosell – 47980320C

Pau Escolà Barragán – 48253559L

Data d'entrega: 4 de gener del 2022

Curs 2021 – 20

Introducció

En aquesta última activitat, es demana realitzar la implementació i les proves pertinents del cas d'ús *Obtener certificación.* En altres paraules, la creació de test i la implementació del codi que passa aquests tests.

Es començarà per formalitzar algunes classes considerades bàsiques, atès que la seva única responsabilitat és la d'emmagatzemar certs valors. Totes elles aniran en un paquet anomenat data.

Están realizadas todas las partes Opcionales

Github

https://github.com/aaron-at97/EP-PlataformaUnificada

Paquet data

Excepcions

En aquest paquet és crea una classe que conté una excepció anomenada BatPathException que serà llençada quan hi hagi un DocPath erroni o null. Aquesta excepció serà usada en la classe UnifiedPlatform en els mètodes que impliquin qualsevol tipus de gestió d'algun document.

Nif

Aquesta classe representa el NIF d'una persona. El NIF és el sistema d'identificació tributària que s'utilitza a l'estat espanyol per tota persona física o jurídica.

La classe codificada, és quasi la mateixa que es proporciona amb l'enunciat però s'han realitzat algunes modificacions. Una d'aquestes modificacions, ha estat realitzada en el mètode *getter* que retorna el NIF. Originalment (en l'enunciat),

aquest mètode és un *getter* clàssic que solament retorna el NIF, però després de la modificació, és possible que es llanci una excepció informant d'un NIF invàlid.

L'última modificació d'aquesta classe, és l'afegit d'un nou mètode *boolean.* Aquest mètode, retornarà cert si el NIF és vàlid i fals en cas contrari. Per comprovar que és vàlid, s'empra la següent estratègia:

1. Es comprova que la instància *nif* no sigui nul·la. En cas de ser-ho és retornarà fals.
2. Atès que tenim un valor per a la instància *nif* (sigui o no correcte), el següent pas és crear un vector de caràcters per introduir el susdit valor de la instància per poder recórrer amb més facilitat el *String*.
3. Es comprova que el valor de *nif* tingui una longitud de 9. En cas de no tenir-la és retornarà fals.
4. Seguidament, com ja es segur que el valor de *nif* té la longitud correcta, s'ha de comprovar el format. Per fer-ho, es comprova que tot el vector de caràcters fins a la penúltima posició (inclosa) són xifres numèriques. En cas de no ser així, es retornarà fals.
5. Finalment, com totes els 8 primers caràcters tenen el format correcte (són xifres numèriques) cal comprovar que l'ultima posició del vector de caràcters sigui una lletra. Si el mètode ha arribat fins aquí significa que el valor *nif* no és null, té una longitud de 9 caràcters i que els 8 primers caràcters son nombres, per tant, si l'últim caràcter és una lletra, el mètode retornarà cert. En cas contrari, fals.

Aquest mètode s'usa per comprovar si cal o no llançar una excepció en el *getter* que retorna el NIF.

AccredNumb

Aquesta classe representa l'identificador de la SS.

En primer lloc, esmentar que aquesta és una classe totalment implementada per l'alumnat, és a dir, en l'enunciat no s'ha proporcionat res del codi d'aquesta classe.

En segon lloc, cal remarcar que és una classe bastant semblant a la classe *NIF*.

Aquesta classe *AccredNumb*, igual que la classe *NIF*, és una classe final atès que solament interessa el valor de la seva instància.

Després de definir el constructor de la classe (el qual rep el nombre de la SS d'una persona com a paràmetre), és defineix un mètode *getter* que retorna el nombre de la SS en cas de ser vàlid. En cas contrari, llança una excepció indicant que el nombre de la SS introduït no és vàlid.

Per poder saber si s'ha de llançar la excepció o no, és defineix un mètode per comprovar la correctesa del format del valor del nombre de la SS. Per fer-ho, es segueix la estratègia a continuació detallada:

1. Es comprova que la instància *ssNum* no sigui nul·la. En cas de ser-ho és retornarà fals.
2. Un cop es segur que no és nul·la, es crea un vector de caràcters que contindrà el valor de *ssNum*. D'aquesta manera serà més senzill poder recórrer el *String.*
3. Seguidament, si el *String* te una longitud de 12, es comprova tot recorrent el vector de caràcters, que tots i cadascun d'aquests caràcters siguin valors numèrics atès que en la vida real, el nombre de la SS d'una persona només conté xifres numèriques. En cas de trobar alguna xifra no numèrica, es retornarà fals.
4. Finalment, es retornarà cert si i solament si la longitud del *String* és de 12.

Aquest mètode s'usa per comprovar si cal o no llançar una excepció en el *getter* que retorna el *ssNum*.

Darrerament, es creen els mètodes *equals()* (cosa que comporta la creació del mètode *hashCode()*), i el mètode *toString()*.

DocPath

Aquesta classe, com les dues anteriors és una classe final atès que solament interessa el valor de la seva instància.

Com les dues classes anteriors, aquesta conté els mètodes *equals(), hashCode()* i *toString()* que són mètodes *@Override.*

També com en les classes anteriors, es disposa d'un mètode *getter* que retorna el *path* del document en cas sigui correcte. En cas contrari, es llança una excepció informant de la invalidesa del *path.*

Per comprovar si el *path* és vàlid o no, solament cal comprovar que aquest sigui no nul. Aquesta tasca la duu a terme el mètode booleà *compPathCode()*.

EncryptedData (Certificat digital)

De la mateixa manera que en les classes anteriors, aquesta és una classe final que conté el seu constructor, el *equals(),* el *hashCode(),* el *toString(),* el *getter* que retorna les dades encriptades en funció de si són vàlides (cosa que es

comprova mitjançant una excepció en el *getter*) i el mètode que determina si les dades encriptades són vàlides o no.

Per comprovar si les dades encriptades són vàlides, solament cal comprovar si la instància *data* és *null* o no. En cas de ser *null,* les dades encriptades no seran vàlides.

EncryptingKey (Certificat digital)

Aquesta classe segueix la mateixa estructura que les anteriors però adaptada a variable d'instància d'aquesta classe.

En aquesta ocasió per comprovar la validesa de la clau, solament és necessari que no sigui nul·la.

PINcode

Aquesta classe segueix la mateixa estructura que les anteriors però adaptada a variable d'instància d'aquesta classe.

En aquesta classe, el mètode que comprova la validesa del PIN, segueix l'estratègia a continuació detallada:

1. Si el valor de *pin* és *null*, és retornarà fals.
2. En cas contrari, es crearà un vector de caràcters que contindrà el *pin*.
3. Seguidament, si la longitud del *String pin* (variable de la classe), té una longitud de 3, és comprovarà, mitjançant el vector de caràcters, que tots els caràcters siguin dígits. En cas contrari es retornarà fals.
4. Finalment, el mètode retornarà cert, és a dir, el PIN és vàlid si i solament si el *String pin* te una longitud de 3.

Password

Aquesta classe segueix la mateixa estructura que les anteriors però adaptada a variable d'instància d'aquesta classe.

El que cal remarcar d'aquesta classe, és el mètode que dona per vàlida o no la contrasenya.

Tal i com està implementat el mètode hi ha 2 coses fonamentals perquè la contrasenya sigui acceptada. Ha de tenir una longitud mínima de 7 caràcters i ha

de contenir dígits alfanumèrics. Per tant, perquè la contrasenya sigui acceptada, com a mínim haurà de contenir un dígit numèric, un dígit alfabètic i una longitud mínima de 7 caràcters.

Test

NifTest

En la realització d'aquest test en concret es comprova si es realitzen correctament les excepcions de la implementació de la classe NIF. Per a poder realitzar aquestes comprovacions es necessari un `@BeforeAll`, on s'inicialitzaran 6 objectes diferents. Un amb un NIF correcte, i la resta incorrectes. Amb més números del compte, sense cap caràcter al final, amb caràcters de sobra, amb l'ordre alterat de lletres i números, i per últim un NIF buit.

Una vegada inicialitzats els objectes anteriors, realitzem un `@Test` per a comprovar que la funció equals que hem implementat funcioni correctament.

També hem realitzat un `@Test` per a la comprovació de les excepcions. Per a la realització d'aquesta tasca el que fem es cridar a la funció de comparació assertTrue/False que ens comprova si el NIF es correcte (retorna Vertader o Fals depenent del cas).

AccredNumbTest

En la realització d'aquest test en concret es comprova si es realitzen correctament les excepcions de la implementació de la classe AccredNumb. Per a poder realitzar aquestes comprovacions es necessari un `@BeforeAll`, on s'inicialitzaran 5 objectes diferents. Un amb un Número d'Acreditació correcte, i la resta incorrectes. No havent número d'acreditació, amb més i amb menys números del compte, i sent els caràcters en comptes de números.

Una vegada inicialitzats els objectes anteriors, realitzem un @Test per a comprovar que la funció equals que hem implementat funcioni correctament.

També hem realitzat un `@Test` per a la comprovació de les excepcions. Per a la realització d'aquesta tasca el que fem es cridar a la funció de comparació assertTrue/False que ens comprova si el Número d'Acreditació es correcte (retorna Vertader o Fals depenent del cas).

DocPathTest

En la realització d'aquest test en concret es comprova si es realitzen correctament les excepcions de la implementació de la classe DocPath. Per a poder realitzar aquestes comprovacions es necessari un `@BeforeAl`l, on s'inicialitzaran 2 objectes diferents. Un amb una ruta d'un document, i una altre inicialitzat a null.

Una vegada inicialitzats els objectes anteriors, realitzem un @Test per a comprovar que la funció equals que hem implementat funcioni correctament.

També hem realitzat un `@Test` per a la comprovació de les excepcions. Per a la realització d'aquesta tasca el que fem es cridar a la funció de comparació assertTrue/False que ens comprova si el Número d'Acreditació es correcte (retorna Vertader o Fals depenent del cas).

EncryptedDataTest

En la realització d'aquest test en concret es comprova si es realitzen correctament les excepcions de la implementació de la classe DocPath. Per a poder realitzar aquestes comprovacions és necessari un `@BeforeAll`, on s'inicialitzaran 3 objectes diferents. Dos d'aquests són correctes i l'altre es null.

Una vegada inicialitzats els objectes anteriors, realitzem un @Test per a comprovar que la funció equals que hem implementat funcioni correctament.

Un altre `@Test` comprova les excepcions. Per a la realització d'aquesta tasca el que fem es cridar a la funció de comparació assertTrue/False que ens comprova si el Número d'Acreditació es correcte (retorna Vertader o Fals depenent del cas).

I per últim tenim un @Test que el que fa es comprovar que la funció toString que hem implementat funcioni correctament.

EncryptingKeyTest

En la realització d'aquest test en concret es comprova si es realitzen correctament les excepcions de la implementació de la classe EncryptingKey. Per a poder realitzar aquestes comprovacions es necessari un @BeforeAll, on s'inicialitzaran 2 objectes diferents. Un de correcte, sent aquest una clau d'encriptació, i l'altre es incorrecte, sent aquest null.

Una vegada inicialitzats els objectes anteriors, realitzem un @Test per a comprovar que la funció equals que hem implementat funcioni correctament.

Un altre `@Test` comprova les excepcions. Per a la realització d'aquesta tasca el que fem es cridar a la funció de comparació assertTrue/False que ens comprova si el Número d'Acreditació es correcte (retorna Vertader o Fals depenent del cas).

PINcodeTest

En la realització d'aquest test en concret es comprova si es realitzen correctament les excepcions de la implementació de la classe PINcode. Per a poder realitzar aquestes comprovacions es necessari un `@BeforeAll`, on s'inicialitzaran 5 objectes diferents. Un amb un codi PIN correcte, i la resta incorrectes. Sent un d'ells null, també amb més i amb menys dígits del compte, i sent els caràcters en comptes de números.

Una vegada inicialitzats els objectes anteriors, realitzem un @Test per a comprovar que la funció equals que hem implementat funcioni correctament.

També hem realitzat un `@Test` per a la comprovació de les excepcions. Per a la realització d'aquesta tasca el que fem es cridar a la funció de comparació assertTrue/False que ens comprova si el Número d'Acreditació es correcte (retorna Vertader o Fals depenent del cas).

PasswordTest

En la realització d'aquest test en concret es comprova si es realitzen correctament les excepcions de la implementació de la classe Password. Per a poder realitzar aquestes comprovacions es necessari un `@BeforeAl`l, on s'inicialitzaran 5 objectes diferents. Un amb un password correcte, i la resta incorrectes. Sent un d'ells null, amb menys dígits del compte, essent format solament de números i estant format solament de caràcters.

Una vegada inicialitzats els objectes anteriors, realitzem un @Test per a comprovar que la funció equals que hem implementat funcioni correctament.

També hem realitzat un `@Test` per a la comprovació de les excepcions. Per a la realització d'aquesta tasca el que fem es cridar a la funció de comparació assertTrue/False que ens comprova si el Número d'Acreditació es correcte (retorna Vertader o Fals depenent del cas).

Paquet Services

Excepcions

En aquest paquet, hi ha un total de 6 excepcions diferents que es detallen a continuació:

1. *IncorrectValDateException*
2. *NifNotRegisteredException*
3. *NotAfiliatedException*
4. *NotValidCertifiacteException*
5. *NotValidCredException*
6. *NotValidPinException*

Interfaces CertificationAuthority y SS

Les interfícies s'implementen posteriorment per a la construcció de les classes següents i per a la comprovació de la classe **UnifiedPlatform.**

Decryptor

En el cas de **Certificacio Digital** hem volgut profundir mes y hem implementat un algoritme RSA de encryptacio i desencriptacio. El cual está implementat en la Clase **Decryptor.**

CertificationAuthorityImpl

En aquesta classe, s'implementa la CertificationAuthority. Es fa una petita introducció de diferents valors d'autenticació. En la qual s'inicialitzen tots els valors de les funcions amb el funcionament correcte.

SSImpl

En aquesta classe s'implementa la **SS**.

Es fa una introducció de diferents valors dels **Nif**. Aquests nif es recorren amb un HashMap, on es guarda una crida `getLaboralLife(nif)`.

El qual tindrà també el document de vida laboral o si fem la crida `getMembAccred(nif)` obtindrem número d'acreditació vinculat.

Test

No cal passar tests a les classes substitutes (CertificationAuthorityImplTest i SSImplTest), ni tampoc fer aquestes classes massa elaborades. Encara així com a extra les hem realitzat amb una petita elaboració per facilitar la seva comprovació i entendre els seus funcionaments.

CertificationAuthorityImplTest

Realitzem un @BeforeEach on inicialitzem els objectes. Hem decidit que era convenient utilitzar un mètode de SetUp `(@BeforeEach)` per tal de preparar les classes a ser utilitzades i la resta de passos necessaris per iniciar els tests, evitant així repetir línies de codi.

A continuació tenim 5 @Test on hem implementat la comprovació de les funcions sendPIN, checkPin, checkCredent, sendCertfAuth i sendCertfAuthErrores de la classe CertificationAuthorityImpl, amb els seus respectius assertEquals i assertThrows, per a comprovar les excepcions.

SSImplTest

Realitzem un `@BeforeEach` on inicialitzem els objectes. Hem decidit que era convenient utilitzar un mètode de SetUp `(@BeforeEach)` per tal de preparar les classes a ser utilitzades i la resta de passos necessaris per iniciar els tests, evitant així repetir línies de codi.

A continuació tenim 2 `@Test` on hem implementat la comprovació de les funcions getLaboralLife i getMembAccreed de la classe SSImpl, amb els seus respectius assertEquals i assertThrows, per a comprovar les excepcions.

Paquet public administration

Excepcions

Aquest paquet contindrà una totalitat de 6 excepcions diferents;

1. *AnyKeyWordProcedureException*
2. *AnyMobileRegisteredException*
3. *DecryptationException*
4. *DuplicatedQuotedPeriodOrNullException*
5. *NotValidPasswordException*
6. *PrintingException*

QuotePeriodsColl

Aquesta classe representa els períodes de cotització d'un ciutadà, els quals estan ordenats per data, de la més antiga a la més recent.

Per guardar tots els períodes de cotització d'un ciutadà s'usarà una llista que acceptarà elements de tipus *QuotePeriod* (variable d'instància de la classe).

De la mateixa manera que les classes anteriors, aquesta també contindrà els mètodes *@Override equals(), hashCode()* i *toString()*, que es generen automàticament.

AddQuotePeriod

Finalment, el mètode que conté l'autèntic interès és el mètode *addQuotePeriod(),* que afegeix un període de cotització, sempre respectant l'ordenació per data, de la més antiga a la més recent. Per tenir èxit en la codificació d'aquest mètode, l'estratègia seguida ha estat:

DuplicatedQuotedPeriodOrNullException:

En cas de ser *null*, es llança una excepció del tipus juntament amb un missatge informant de l'error.

Es considera duplicat un període que té la mateixa data d'inici i la mateixa data de finalització d'un ja existent a la llista.

QuotePeriod

Aquesta classe, és semblant a l'anterior però amb la diferència que l'anterior representava tots els període de cotització d'una persona mentre que aquesta solament representa un sol període de cotització.

Les variables d'instància de la classe, són les proporcionades en l'enunciat, és a dir, no se n'han afegit de noves.

Òbviament, s'ha afegit un constructor per poder crear una instància d'aquesta classe.

Com ja és habitual, la classe conté els mètodes *@Override equals(), hashCode()* i *toString()* que es generen automàticament.

En aquesta classe també s'ha afegit un mètode *getter* per cada variable d'instància.

PDFDocument

Després de definir les variables d'instància de la classe, en aquesta ocasió es creen un total de 3 constructors diferents.

El primer dels quals, no rep cap paràmetre i realitza tres accions:

1. Crea una nova *Date*.
2. Crea un nou *DocPath* i li dona per *deffault value* "src/Docs/".

3. Inicialitza el fitxer al fitxer amb el fitxer PDF que hi ha dintre del directori "src/Docs/". El nom d'aquest arxiu és "default.pdf".

El segon dels contructors, rep per paràmetre el *path*, per tant, l'única operació que canviarà respecte al primer constructor, és la inicialització del *path*, que en comptes d'inicialitzar-lo amb un *new DocPath()* s'inicialitzarà al que es rep per paràmetre.

El tercer i últim constructor, en comptes de rebre el *path* del arxiu, rep directament l'arxiu, per tant, l'operació que canvia respecte al primer constructor és la inicialització del fitxer, que s'inicialitza al que es rep per paràmetre.

Cal remarcar que els dos últims constructors (els que reben paràmetres) solament s'utilitzen en el moment del testing.

Seguidament, hi ha un *getter* per cada variable d'instància de la classe.

Després dels *getters*, hi ha un *setter* per poder canviar el document.

Un cop generats aquests mètodes, hi ha els *@Override methods equals(), hashCode()* i *toString()*.

A continuació, hi ha el mètode *moveDoc()* (d'implementació opcional), que mou el document al path que s'indica. Aquest mètode es basa en la utilització de l'estructura *try{} catch() {}* on s'intentarà moure el document al *path* indicat mitjançant la sentencia `File.move()`. En cas de no poder moure's es llançarà una excepció *IOException* informant de l'error.

L'últim mètode de la classe, és el mètode *openDoc(),* que obre el document PDF en el *path* indicat. En cas de no poder obrir el document, es llança una nova excepció.

LaboralLifeDoc

Aquesta és una subclasse de *PDFDocument,* per tant en el constructor d'aquesta classe hi haurà una crida al constructor de la superclasse mitjançant la sentència *super()*.

La resta de la classe són mètodes que l'IDE amb el que s'està treballant genera automàticament. Aquests mètodes són els diferents *getters*, els *getters* de la superclasse, el *setter* de la superclasse per poder canviar el fitxer i els *@Override methods equals(), hashCode()* i *toString().*

MemberAccreditationDoc

Aquesta classe és subclasse de *PDFDocument.*

Aquesta classe, igual que l'anterior, conté el constructor, els *getters* i els *getters* de la superclasse, el *setter* de la superclasse per poder canviar el fitxer i els *@Override methods equals(), hashCode()* i *toString().*

UnifiedPlatform

En aquesta classe, tot i tenir moltes variable d'instància, el constructor solament n'inicialitza tres. Aquestes tres són: *cert* de tipus *CertificationAuthority* que representa el servei, *ss* de tipus *SS* (SS == Seguretat Social) i *doc* de tipus *PDFDocument* que es el document PDF que es genera al final del tràmit.

El primer mètode de rellevància de la classe és el mètode *processSearcher()* que estableix la varible *searcher* a cert per poder buscar el tràmit des del cercador.

El següent mètode de rellevància en aquesta classe és *enterKeyWords()* quer permetrà buscar el tràmit en el cercador introduint el nom del tràmit.

Seguidament, s'han implementat dos mètodes bastant pareguts; *selects()* i *selectCitizens()*. Mètodes que estableixen a cert les seves respectives variables d'instància (el nom dels mètodes és el mateix que el de les variables d'instància). El primer mètode simula el fet de entrar en l'apartat de la SS i el segon representa entrar a l'apartat de "Ciutadans" un cop dins de l'apartat de la SS.

El quart mètode implementat, *selectReports()*, mostra per pantalla els diferents tràmits que són possibles i representa el fet d'entrar a l'apartat d'informes i certificats i simula la tria del tràmit a realitzar.

A continuació, el mètode *selectCertificationReport()*, mostra les diferents opcions d'autenticació (certificat digital, *Cl@ve PIN* i *Cl@ve permanente*) i simula el triatge de una de les opcions.

Un cop mostrats els diferents mecanismes d'identificació, arriba el moment de seleccionar-ne un. Tasca de la qual s'encarrega el mètode *selectAuthMehod()* mitjançant el *byte* representatiu del tràmit.

Després, es troben els mètodes que implementen els mecanismes d'autenticació. El primer dels quals és *enterNIF_PINobt()* (*Cl@ve PIN*). Per tant, si s'ha seleccionat aquesta opció d'identificació, el NIF de la persona no és *null*, la data de validació tampoc ho és i s'ha enviat el PIN, es corroborarà el NIF i el mètode *checkPIN()* retornarà cert. En cas de no complir alguna de les condicions anteriors, es llançarà la excepció corresponent en cada cas amb un missatge informant de l'error. És a dir, representa el pas en que l'usuari utilitza el formulari on introdueix les seves dades del sistema d'identificació.

Seguidament, es troba el mètode *enterPIN()* que si i solament si el PIN és vàlid, es completarà el "*log in"* i es generarà l'informe corresponent. En cas contrari es llançarà una excepció.

Per implementar el mètode d'autenticació de la *Cl@ve Permanente*, s'ha destinat el mètode *enterCred()* que actuarà, s'hi s'ha escollit aquesta opció, i comprovarà; primer de tot que el NIF i la *password* no siguin *null* (En cas de ser-ho es llançarà una excepció del tipus *NotValidCredException* amb un missatge informant de l'error). En segon lloc

és comprova l'estat de les credencials del ciutadà. En cas de no tenir el NIF registrat, és llança una excepció informant de l'error, en cas de estar registrat i no tenir el mètode reforçat activat, es treu per pantalla la situació de les credencials i es generen els informes i finalment si el NIF està registrat i el mètode reforçat activat s'estableix com a correcte el PIN (this.*checkPIN = true*). Dit en altres paraules; és una simulació del formulari on el ciutadà introdueix les seves dades del sistema d'identificació *Cl@ve Permanente.*

En cas de no poder realitzar cap d'aquestes accions es llança una *ConnectException* informant de l'error.

Per implementar el tercer i últim mètode d'autenticació s'han destinat no un sinó tres mètodes.

El primer de tots *selectCertificate()* comprova que efectivament s'ha escollit aquesta opció d'identificació i estableix la *password* a la que marca el mètode *checkCertificate* amb l'opció desitjada (paràmetre del mètode *selectCertificate()*).

El segon dels mètodes destinat al certificat digital és *enterPasswd()* que en primer lloc comprova que la contrasenya sigui correcta. En cas de no ser-ho llança una excepció. Un cop verificada la contrasenya, s'utilitzen els mètodes de la classe *Decryptor* per generar les classes públiques i procedir a l'encriptació del NIF. En cas en que en els susdits mètodes d'encriptat hi hagi algun error en el moment d'obtenir la clau pública o la encriptació es mostraran errors.

El tercer i últim mètode destinat al certificat digital, és el mètode *decryptData()*. En primer lloc si les dades encriptades rebudes per aquest mètode tenen un valor de *null*, es llançarà una excepció. Un cop comprovat que les dades encriptades no són nul·les, es tractarà de desencriptar el NIF mitjançant el mètode mètode *decryptData* de la classe *Decryptor.* Si després d'intentar desencriptar el NIF aquest és *null* és llançarà una excepció informant que hi ha havigut un error en el moment de desencriptar el NIF. Si tot va bé, finalment, es retornarà el NIF.

Seguidament, hi ha una operació que estableix les paraules clau que permeten usar el cercador de tràmits. Aquest mètode retorna un *String* amb el nom de l'administració encarregada del tràmit.

L'operació *openDocument()* és s'encarrega d'obrir el document generat. En cas de tenir un *path null* es llança una excepció i en cas de no poder obrir el document també s'informa que no s'ha pogut obrir el document.

Finalment s'han afegit els *getters* i els *setters* de les variables d'instància.

Test

QuotePeriodCollTest

Realitzem un @BeforeEach on inicialitzem els objectes. A continuació tenim 3 @Test on hem implementat la comprovació de les funcions addPeriod, addPeriodThrows i getInit de la classe QuotePeriodColl. Al @Test de addPeriod fem les comprovacions necessàries amb assertEquals, al @Test addPeriodThrows comprovem les excepcions mitjançant assertThrows, i per últim tenim el @Test del getter getInit, que comprova que funcioni correctament.

QuotePeriodTest

Realitzem un @BeforeEach on inicialitzem els objectes. A continuació tenim 4 @Test on hem implementat la comprovació de les funcions equals, notEquals, getNumDays i getInitDay de la classe QuotePeriod. Als `@Test` equals i notEquals fem les comprovacions necessàries amb assertEquals, i als `@Test` getNumDays i getInitDay es realitzen els getters que el que fan es comprovar que les funcions implementades funcionin correctament i no es realitzin test falsos, mitjançant les funcions assertEquals/assertNotEquals, respectivament.

PDFDocumentTest

Realitzem un @BeforeEach on inicialitzem l'objecte. A continuació tenim 5 @Test on hem implementat la comprovació de les funcions open, move, getCreateDate, getPath i getFile de la classe PDFDocument. Als `@Test` open i move fem les comprovacions de les funcions implementades i en el cas de que no es pogués fer qualsevol de les 2 funcions, llançarà una Exception. Als `@Test` getCreateDate, getPath i getFile es realitzen els getters que el que fan es comprovar que les funcions implementades funcionin correctament i no es realitzin test falsos, mitjançant les funcions assertEquals/assertNotEquals, respectivament.

LaboralLifeDocTest / MemberAccreditationDocTest

A les següents classes s'utilitza una estructura semblant.

A cada classe s'inicialitzen al `@BeforeEach` els objectes pertinents i als `@Test` s'executen els seus respectius equals i getters, que s'estan utilitzant des de les

diferents classes per comprovar que aquestes funcionen correctament.

La principal diferència entre les classes és que en MemberAccreditationDocTest validem el `@Test` mitjançant el nombre d'acreditació, i en canvi a LaboralLifeDocTest es valida mitjançant quote periods.

UnifiedPlatform

Sempre és convenient definir diferents classes substitutes, per diferenciar els escenaris d'èxit i fracàs, o bé parametrizar-les, sempre i quan la classe no quedi massa llarga o complexa.

Per aquest cas hem realitzat el següents tipus cas de test:

UnifiedPlatformClavePINOpenDocSearchTest / UnifiedPlatformPermanenteTest/ UnifiedPlatformCertificadoDigitalTest

En **UnifiedPlatform** se hace la comprobación de todos los métodos funcionales. Para realizar dicha comprobación se utilizan tests dobles. Se realiza la comprobación con las clases ya implementadas retornando los valores correctos para realizar sus dichos equals, para comprobar si se realizan las modificaciones y getters correctamente.

Para facilitar la comprensión de nuestros Test hemos dividido los test por los distintos métodos de autenticación Certificado digital / Cl@ve Permanente y Cl@ve PIN

Empren un mètode de test per a cada esdeveniment (aquest contindrà la crida a l'esdeveniment en qüestió, però sempre seguint l'ordre establert al cas d'ús), i així anar provant el correcte funcionament del cas d'ús de manera progressiva, esdeveniment a esdeveniment

UnifiedPlatformClavePINNullTest /UnifiedPlatformPermanenteNullTest / UnifiedPlatformCertificadoDigitalNullTest

En **UnifiedPlatform** se hace la comprobación de todos los métodos funcionales. Para realizar dicha comprobación se utilizan tests dobles. En una clase se hacen

las comprobaciones de errores implementando dos clases todo null las cuales implementaran las interfaces.

Para facilitar la comprensión de nuestros Test hemos dividido los test por los distintos métodos de autenticación Certificado digital / Cl@ve Permanente y Cl@ve PIN.

Cabe tener en cuenta que en los test PIN null también comprobamos la funcionalidad de los select distintos capturando el System.out.printl() y comparando el resultado esperado.

Dichos forzamos que se disparen las excepciones pasando parámetros esperados.

Principis SOLID reflecteixen a la pràctica

- **Principi de Responsabilitat Única:** En la implementació de la tasca es pot apreciar com totes les classes implementades tenen un sol eix, és a dir, les classes tenen una funció clara, única i definida. En cap moment no s'ha vist necessari duplicar una classe perquè tingui una única responsabilitat, ja que es percep que ja ho fa.

- **Principi Obert-Tancat** En els tests, interfícies per fer un codi susceptible a canvis, però que no comportin, al seu torn, modificacions en alguns fragments de codi.

- **Principi de substitució de Liskov:** El principi de Liskov ens dóna una sèrie de pautes per guiar l'herència entre classes. La principal que ha de complir si estem realitzant l'herència d'una manera correcta és que cada classe que hereta d'una altra es pot fer servir com a pare sense necessitat de conèixer les diferències entre elles.

Aquest cas és completat amb les herències **MemberAccreditationDoc y LaboralLifeDoc**

- **Principi de Segregació d'Interfícies:** Se pot veure que al codi presentat no hi ha existència d'interfícies grosses, totes les interfícies implementades

consten de pocs mètodes i tenen un sentit, per la qual cosa es pot afirmar que les interfícies estan correctament construïdes i no cal segregar més del que es ha fet.

- **Principi d'inversió de dependències:**

Els mòduls dalt nivell no han de dependre dels de baix nivell. Tots dos han de dependre d'abstraccions

Les abstraccions no han de dependre de detalls concrets. Els detalls concrets han de dependre de les abstraccions

Patrons Grasp reflectits a la pràctica

Durant la realització de la tasca hem pogut veure que es reflecteixin alguns dels patrons explicats a la docència de l'assignatura com per exemple:

- **El patró Creador:** En aquest cas, classifiquem el patró creador com la classe UnifiedPlatform, ja que mitjançant aquesta classe es pot crear qualsevol instància de què es pugui disposar.

- **Patró expert en informació (expert):** Es tracta de la classe Unified Platform , ja que conté tota la informació necessària per al desenvolupament de les accions plantejades mitjançant getters o setters.

- **Patró de alta cohesión:** Per exemple, QuoteperiodColl pot crear instàncies de QuotePeriod entre d'altres que podeu utilitzar la vostra classe més gran (UnifiedPlatform).

- **Patró de baix acoplament:** Aquest patró va de la mà amb l'anterior i és que si la classe és d'alta cohesió, aleshores complirà que és de baix acoblament.

- **Patró controlador:** UnifiedPlatform juga el paper de controlador de façana, i que per tant s'aplica el patró GRASP Controller.

Observaciones

1. El test sobre la classe que fa de controlador del cas d'ús (en el nostre cas la classe **UnifiedPlatform**) sempre ha de fer-se tenint en compte l'ordre dels esdeveniments del cas d'ús, el qual està reflectit al DSS de referència. Això és així perquè en un context real, aquests mètodes s'invoquen seqüencialment.
2. Utilitzar un mètode de SetUp **(@BeforeEach)** per tal de preparar les classes a ser utilitzades i la resta de passos necessaris per iniciar els tests, evitant així repetir línies de codi
3. Als mètodes de test no utilitzem la instrucció try/catch, ni tampoc el mètode fail(). Com ja a estat esmentat a clase fem servir Assertion assertThrows(), i la possibilitat de declarar els mètodes amb la clàusula throws, quedant el codi molt més clar i net.
4. No utilitzem atributs públic per tal de poder ser utilitzats a les classes de test. No hem de descuidar l'aplicació dels fonaments de la Programació Orientada a Objectes. Per facilitar els codes Smell em emprat les següents funcions per reduir el codi repetitiu del nostre projecte.

5. Mètode emprat per generar el informe i estalviarnos línies de codi:

```
private void informes() throws NotAffiliatedException, ConnectException {
    if (opcion == 0) {
        if (ss.getLaboralLife(nif) == null) {
            throw new NotAffiliatedException("");
        }
        setLaboralLifeDoc(ss.getLaboralLife(nif));
    } else if (opcion == 1) {
        if (ss.getMembAccred(nif) == null) {
            throw new NotAffiliatedException("");
        }
        setAccreditationDoc(ss.getMembAccred(nif));
    }
}
```

6. Creació d'un hasMap per el cas de que un futur vulguem aprofundir en la implementació d'una altra AAPP:

```java
// Other operations
private String searchKeyWords(String keyWord) throws AnyKeyWordProcedureException {

    Map<String, String> listkeyWord = new HashMap<>();
    listkeyWord.put("vida laboral", "SS");
    listkeyWord.put("numero de afiliacion", "SS");
    listkeyWord.put("afiliacion", "SS");
    listkeyWord.put("laboral", "SS");
    listkeyWord.put("datos fiscales", "AEAT");
    listkeyWord.put("declaración de la renta", "AEAT");
    listkeyWord.put("puntos carnet", "DGT");
    listkeyWord.put("certificado de nacimiento", "MJ");

    if (listkeyWord.containsKey(keyWord)) {
        return listkeyWord.get(keyWord);
    }
```

7. En els test: Sempre és com aquí: assertEquals(expected, actual)