

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 – 22**

Índex

[Introducció: 1](#_Toc92128887)

[Implementació de mètodes: 1](#_Toc92128888)

[Paquet data 1](#_Toc92128889)

[Excepcions 1](#_Toc92128890)

[Classe NIF 1](#_Toc92128891)

[Classe AccredNumb 2](#_Toc92128892)

[Classe DocPath 2](#_Toc92128893)

[Classe EncryptedData 3](#_Toc92128894)

[Classe EncryptingKey 3](#_Toc92128895)

[Classe PINcode 3](#_Toc92128896)

[Classe Password 3](#_Toc92128897)

[Paquet Services 4](#_Toc92128898)

[Excepcions 4](#_Toc92128899)

[Interfícies 4](#_Toc92128900)

[Classes 5](#_Toc92128901)

[Paquet publicadministration 8](#_Toc92128902)

[Classe QuotePeriodColl 8](#_Toc92128903)

[Classe QuotePeriod 8](#_Toc92128904)

[Classe PDFDocument 8](#_Toc92128905)

[Classe LaboralLifeDoc 8](#_Toc92128906)

[Classe MemberAccreditationDoc 8](#_Toc92128907)

[Classe UnifiedPlatform 8](#_Toc92128908)

[Tests: 8](#_Toc92128909)

[Paquet data 8](#_Toc92128910)

[Classe NIF 8](#_Toc92128911)

[Classe AccredNumbTest 8](#_Toc92128912)

[Classe DocPathTest 8](#_Toc92128913)

[Classe EncryptedDataTest 8](#_Toc92128914)

[Classe EncryptingKeyTest 8](#_Toc92128915)

[Classe PINcodeTest 8](#_Toc92128916)

[Classe PasswordTest 8](#_Toc92128917)

[Paquet Services 8](#_Toc92128918)

[Classe CertificationAuthorityImplTest 8](#_Toc92128919)

[Classe SSImplTest 8](#_Toc92128920)

[Paquet publicadministration 8](#_Toc92128921)

[Classe QuotePeriodCollTest 8](#_Toc92128922)

[Classe QuotePeriodTest 8](#_Toc92128923)

[Classe PDFDocumentTest 8](#_Toc92128924)

[Classe LaboralLifeDocTest 8](#_Toc92128925)

[Classe MemberAccreditationDocTest 8](#_Toc92128926)

[Classe UnifiedPlatformNormalTest 8](#_Toc92128927)

[Classe UnifiedPlatformNullTest 8](#_Toc92128928)

# 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.

# Implementació de mètodes:

## 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.

### Classe 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.

### Classe 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()*.

### Classe 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()*.

### Classe EncryptedData

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.

### Classe EncryptingKey

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.

### Classe 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.

### Classe 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.

## 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*

### Interfícies

Les interfícies s’expliquen molt per sobre en aquest informe atès que han estat proporcionades, juntament amb l’explicació del comportament dels mètodes que contenen, en l’enunciat de la pràctica

#### Interfície CertificationAuthority

És un servei extern que representa les diferents entitats de certificació.

Conté les operacions que ha d’implementar com a mínim aquella classe que s’encarregui de codificar el servei de l’autoritat de certificació.

En el cas d’aquesta interfície, la classe que la implementi haurà de contenir, mínim, aquestes 4 operacions:

1. *sendPin()*
2. *checkPIN()*
3. *checkCredent()*
4. *sendCertfAuth()*

L’única modificació realitzada per l’alumnat ha estat l’afegir la quarta operació *sendCertfAuth()*.

#### Interfície SS

Conté les operacions que ha de implementar com a mínim, 2 en aquest cas, aquella classe que s’encarregui de implementar el servei de la SS.

Les operacions que com a mínim contindrà la classe encarregada d’implementar aquesta interfície seran:

1. *getLaboralLife()*
2. *getMemAccred()*

En el cas d’aquesta interfície l’alumnat no ha afegit cap mètode més.

### Classes

#### Classe CertificationAuthorityImpl

Com el propi nom de la classe indica, aquesta és la classe encarregada d’implementar la interfície que defineix el servei *CertificationAuthority*.

En primer lloc, el mètode *sendPin()* que proporciona les credencials del ciutadà en el sistema Cl@ve PIN, a la mateixa vegada que sol·licita l’emissió del PIN per completar la identificació. L’estratègia d’implementació seguida és detalla seguidament:

1. Primerament, es comprova que el NIF d’aquell ciutadà estigui registrat. En cas de no estar-ho, es llança una excepció del tipus *NotRegisteredException.*
2. En segon lloc, es comprova si hi ha un telèfon de contacte del ciutadà. En cas de no haver-hi cap telèfon es llança una excepció del tipus *AnyMobileRegisteredException*.
3. Seguidament, mitjançant una variable booleana, es comprova que existeixi la parella (clau, valor) on la clau és el NIF del ciutadà i el valor la data de validació de quan aquest es va donar d’alta. Aquesta comprovació és realitza en *Map*<Nif, Date> *listClave.*
4. Un cop aquesta variable booleana obté el valor que li pertoca, es comprova si es falsa o vertadera. En cas de ser falsa, cosa que indicarà que o bé no existeix el NIF d’aquella persona, o bé que la data de validació no és la correcta, es llançarà una excepció del tipus *IncorrectValException*.
5. Abans de finalitzar, un cop superats tots els controls d’errors, s’intentarà generar el PIN per tal de poder completar la autenticació de l’usuari. En cas que tot vagi correctament, s’injectarà en el mapa de pins una parella (clau, valor) amb el NIF del ciutadà com a clau i un codi PIN com a valor associat. En cas de no poder dur a terme aquesta tasca, es llançarà una excepció del tipus *ConnectException.*
6. Finalment, es retornarà el valor de la variable booleana del pas 3 confirmant que tot el procediment s’ha dut a terme de manera correcta. Retornar aquest valor implica que tot el procediment s’ha dut executat correctament.

El segon dels mètodes de la interfície a implementar és *checkPIN(),* que retorna un booleà indicant si el PIN es correspon amb el PIN generat per el sistema Cl@ve PIN a aquell ciutadà a més també és necessari que el PIN continuï vigent. L’estratègia per implementar aquest mètode es detalla a continuació:

1. En primer lloc, es comprova que el NIF del ciutadà i el PIN generat anteriorment existeixin dintre del *Map* que associa PINs i NIFs. Òbviament s’ha de trobar una parella (clau, valor) que tingui un NIF concret (el del ciutadà que realitza el tràmit) i que aquest concordi amb el PIN que el sistema ha generat. En cas de no trobar aquesta tupla, ja sigui perquè no existeix el NIF en el *Map* o bé perquè el PIN no es correspon, es llança una excepció del tipus *NotValidPinException.*
2. Seguida i finalment, es comprova que efectivament el PIN és correspon amb el generat per el sistema per al ciutadà en concret. En cas de no ser així, es llança una excepció del tipus *ConnectException*.

El tercer mètode *@Override@,* és *checkCredent()*, que retorna un *byte* que indica diferents situacions: 0 si el ciutadà no està registrat en el sistema Cl@ve permanente amb aquelles credencials, 1 si ho està però no té activat el mètode reforçat; i 2 si ho està i utilitza el mètode reforçat.

Per implementar aquest mètode s’han seguit els següents passos:

1. Primerament, es comprova que el NIF de l’usuari existeix en el *Map* que relaciona els NIFs del ciutadà amb el *byte* que correspongui (0, 1 o 2) . En cas de no existir el NIF, es llança una excepció del tipus *NifNotRegisteredException.*
2. Com en l’ocasió anterior, es comprova que hi hagi un telèfon de contacte associat a un NIF. En cas de no existir es llança una excepció del tipus *AnyMobileRegisteredException.*
3. Seguidament, es comprova si el *byte* associat al NIF és un 0. En cas de ser-ho es retorna el 0.
4. Si el *byte* associat no és el 0, es comprova que les credencials siguin les correctes mitjançant una comparació de la contrasenya associada al NIF i la contrasenya introduïda. En cas de no ser així, es llança una excepció del tipus *NotValidCredException.*
5. Finalment, mitjançant una estructura *try {} catch() {},* es comprova si el byte associat al NIF és un 2. En cas de ser-ho es genera un nou PIN i s’associa al NIF pertinent. En cas que el 2 no sigui el *byte* associat a aquest NIF, és retorna el *byte* associat al NIF (serà l’1). Per acabar, si no es pot realitzar cap d’aquestes accions, es llançarà una excepció del tipus *ConnectException* (s’entrarà en el entrarà en el *catch() {}*).

L’últim mètode *@Override,* és el mètode afegit per l’equip de desenvolupament. *sendCertAuth()* en primer lloc comprova que existeixi la *pubKey* que es passa per paràmetre i que tingui un valor associat. Per tant, si no existeix o té un valor associat de *null* es llançarà una excepció del tipus *NotValidCertificateException* informant de l’error.

Seguidament s’intentarà retornar noves dades encriptades (les corresponents al certificat). En cas de no poder, es llançarà una nova excepció del tipus *ConnectException* informant de l’error.

#### Classe SSImpl

El primer dels mètodes de la interfícies és *getLaboralLife().* Aquest mètode, a partir del NIF, verifica l’existència d’activitats laborals amb alta a la SS, i en aquest cas genera un pdf i transmet l’informe de vida laboral.Perimplementar amb èxit el mètode l’estratègia seguida és la que es detalla a continuació:

1. Comprovar que el NIF no es *null*. En cas de ser-ho es llança una excepció del tipus *NotAfiliatedException* informant de l’error.
2. En segon lloc, es prova de buscar l’informe de vida laboral gràcies al mètode auxiliar *buscarLaboralLife()* el qual rep el NIF del ciutadà com a paràmetre. En cas que no sigui possible generar l’informe, es llançarà una excepció del tipus *ConnectException*.
3. Seguidament, si la variable que emmagatzema l’informe és nul·la, cosa que significa que no s’ha pogut generar l’informe, es llança una excepció del tipus *NotAfiliatedException.*
4. Finalment, si tot els controls d’errors s’han superat satisfactòriament, es retornarà l’informe de vida laboral del ciutadà en qüestió.

Per implementar el mètode *buscarLaboralLife(),* senzillament és busca en el *Map* que el NIF introduït sogui el mateix que el NIF que consta a la taula. En cas de no trobar cap NIF, es retornarà *null.*

El segon i últim mètode pertinent a la interfície és *getMembAccred().* Aquest mètode, a partir del NIF, verifica si el ciutadà es troba afiliat a la SS, i en aquest cas genera un document PDF i transmet l’acreditació del nombre d’afiliació de la SS. Per implementar amb èxit aquest mètode s’ha seguit la següent estratègia:

1. Si el NIF es *null*, es llança una excepció del tipus *NotAfiliatedException.*
2. En segon lloc, es cerca l’informe de vida laboral del ciutadà en la llista que conté tots els informes de vida laboral de tots els ciutadans donats d’alta a la SS. (El mètode que s’encarrega d’aixo és *buscarMembAccredNif()* el qual rep el NIF del ciutadà com a paràmetre). En cas de no poder realitzar aquesta acció, es llança una excepció del tipus *ConnectException.*
3. Si el valor de la variable que emmagatzema el PDF és *null*, significa que no s’ha trobat ningun informe associat a aquell NIF i per tant es llança una excepció del tipus *NotAfiliatedException* informat de l’error*.*
4. Finalment, si tot ha anat bé, es retorna el document. En cas contrari *null.*

## Paquet publicadministration

### Excepcions

Aquest paquet contindrà una totalitat de 6 excepcions diferents;

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

### Classe 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.

Com a mètode imprescindible de tota classe, aquesta tindrà un constructor que inicialitzarà la llista anomenada anteriorment a una nova *ArrayList<>().*

També s’ha afegit un *getter* per poder obtenir la susdita llista.

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:

1. En primer lloc, guardar en una variable la longitud de la llista que emmagatzema els períodes de cotització.
2. Segon, es comprova que el nou període de cotització (que es passa per paràmetre al mètode) no es *null.* En cas de ser *null*, es llança una excepció del tipus *DuplicatedQuotedPeriodOrNullException* juntament amb un missatge informant de l’error.
3. En tercer lloc, es comprova que el període de cotització que es vol introduir a la llista, no sigui un període que ja existeix, és a dir, un duplicat. En aquest cas es llança una excepció del tipus *DuplicatedQuotedPeriodOrNullException* juntament amb un missatge informant de l’error. Si el període que es vol afegir no és un duplicat s’afegeix a la llista d’emmagatzemament dels períodes de cotització d’aquella persona.

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.

1. Finalment, s’afegeix un condicionale que comprova el cas en que no hi hagi períodes de cotització en la llista, és a dir, listQuote.size() == 0. Per tant, si la longitud de la llista es 0 o la *flag* que controla si s’ha afegit algun període de cotització recentment és falsa, s’afegirà el nou període de cotització a la llista.

### Classe 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.

### Classe 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.

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.

### Classe 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 superclasseper poder canviar el fitxer i els *@Override methods equals(), hashCode()* i *toString().*

### Classe 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().*

### Classe UnifiedPlatform

# Tests:

## Paquet data

### Classe NIF

### Classe AccredNumbTest

### Classe DocPathTest

### Classe EncryptedDataTest

### Classe EncryptingKeyTest

### Classe PINcodeTest

### Classe PasswordTest

## Paquet Services

### Classe CertificationAuthorityImplTest

### Classe SSImplTest

## Paquet publicadministration

### Classe QuotePeriodCollTest

### Classe QuotePeriodTest

### Classe PDFDocumentTest

### Classe LaboralLifeDocTest

### Classe MemberAccreditationDocTest

### Classe UnifiedPlatformNormalTest

### Classe UnifiedPlatformNullTest