# Enunciado de Práctica
# Diseño y Pruebas Unitarias

## 1. Temática

El dominio de esta práctica es el mismo que habéis estado trabajando en la parte de análisis, y que conocéis en detalle: la *Plataforma unificada de gestión ciudadana*.

**Se pide** implementar y probar una versión simplificada del caso de uso *Obtener certificación*. En otras palabras, la creación de test y la implementación del código que pasa esos tests (*o a la inversa, el orden lo decidís vosotros*). En cualquier caso, se recomienda escribir el código en orden creciente de complejidad, tal y como se propone en este documento.

Comenzaremos formalizando algunas clases consideradas básicas (igual que lo son String, BigDecimal, etc.), dado que su única responsabilidad es la de guardar ciertos valores. Todas ellas irán en un paquete denominado data.

## 2. El paquete data

El paquete data contendrá algunas clases, la única responsabilidad de las cuales es la de guardar un valor de tipo primitivo o clase de java (concretamente String y Date). Pensad los diversos motivos que hacen que sea conveniente hacerlo así (aparte de que, como ya sabéis, hay algún que otro *code smell* que hace alusión a este aspecto).

Se trata de las clases Nif (el nif del ciudadano), a guardar como un String, PINcode (código PIN de tres dígitos generado por el sistema Cl@ve[1]), también un String, DocPath (una ruta), a representar con un String y AccredNumb (el número de afiliación a la SS), nuevamente un String. Además, para trabajar tanto con el certificado digital como con Cl@ve permanente utilizaremos también la clase Password (un String), tal y como se presenta más adelante.

A continuación se presenta la clase Nif.

[1] Por simplicidad, prescindiremos de la consideración del sistema Cl@ve PIN 24 h.

## Clase Nif

Representa el Número de Identificación Fiscal (NIF) del ciudadano. El Nif es el sistema de identificación tributaria utilizada en España para las personas físicas y jurídicas.

En nuestro caso se utiliza en el momento de autenticarse mediante el sistema Cl@ve, así como para obtener cualquier tipo de certificación por parte de la Seguridad Social.

Esta es su implementación:

```
package data;

/**
* Essential data classes
*/

final public class Nif {

// The tax identification number in the Spanish state.

    private final String nif;

    public Nif (String code) { this.nif = code; }

    public String getNif () { return nif; }

    @Override
    public boolean equals (Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Nif niff = (Nif) o;
        return nif.equals(niff.nif);
    }

    @Override
    public int hashCode () { return nif.hashCode(); }

    @Override
    public String toString () {
        return "Nif{" + "nif ciudadano='" + nif + '\'' + '}';
    }
}
```

Definid vosotros las clases PINcode, DocPath, AccredNumb y Password (Cl@ve permanente).

Todas estas clases serán **inmutables** (por eso el **final** y la no existencia de setters), y tienen definido un **equals**, que comprueba si dos instancias con el mismo valor son iguales. Estas clases se denominan también **clases valor**, ya que de sus instancias nos interesa tan sólo el valor.

Es conveniente añadir las excepciones que consideréis oportunas. Para el caso de la clase Nif, podemos definir las dos situaciones siguientes: que al constructor le llegue null (objeto sin instanciar), y también un nif mal formado.

¿Cuál es el caso de las otras clases básicas? ¿Qué excepciones convendrá contemplar?

**Implementar y realizar test para estas clases (es suficiente con comprobar las excepciones consideradas).**

# 3. El caso de uso *Obtener certificación*

En lo que queda de documento se presenta el caso de uso a desarrollar, junto a sus casos de uso de segundo orden relacionados. Se trata de **Obtener certificación**[2].

Este caso de uso resuelve la obtención por parte del ciudadano del informe de vida laboral y de la acreditación del número de afiliación a la SS, pudiendo optar por descargar o imprimir ambos informes, una vez presentados en pantalla. Por supuesto, la realización de estos trámites requiere la autenticación del ciudadano mediante una de las herramientas de identificación consideradas. Además, se ofrece la opción de utilizar el buscador de trámites. Todo ello se desarrollará en concordancia con los diagramas utilizados como referencia durante el análisis.
Los casos de uso considerados para resolver la identificación son: **Autenticar con Cl@ve PIN** y, *opcionalmente*, **Autenticar con Cl@ve Permanente**. Además, se propone otra parte opcional en la que resolver el caso de uso **Autenticar con certificado digital**.

Definiremos una clase llamada UnifiedPlatform[3] para implementar el caso de uso. Será la clase responsable de manejar los eventos de entrada (*controlador de fachada*).

Además, se implementarán los servicios involucrados. Estos se presentan a continuación.

## Servicios involucrados

Agruparemos los servicios involucrados en este caso de uso en un paquete denominado services.

La parte central del caso de uso **Obtener certificación** requiere de dos servicios externos. Estamos hablando de la SS (la Seguridad Social) y de la CertificationAuthority (la autoridad de certificación), cuya interacción con la plataforma unificada se relaciona a continuación:

- SS. Interviene en el momento de obtener la certificación requerida por el ciudadano –la acreditación del número de afiliación o la vida laboral-, tras haberse identificado con éxito (consultar el *DSS-ObtCertific-SolRef.jpg*):

[2] Como estamos realizando pruebas unitarias, no se presenta ninguna interfaz de usuario, sino que tendremos métodos que son los que usará la interfaz de usuario (los eventos de entrada). Tal y como se presenta en el tema de *Patrones GRASP*, implementaremos y probaremos un *controlador de fachada* para el caso de uso (es decir, un objeto del dominio escogido específicamente como controlador).

[3] Corresponde a la clase del dominio UnifiedPlatform (modelo del dominio pasado como referencia).

- getLaboralLife(Nif nif): a partir del nif (un objeto de la clase NIF), verifica la existencia de actividades laborales con alta en la SS, y en ese caso genera en pdf y transmite el informe de vida laboral.
- getMembAccred(Nif nif): a partir del nif, verifica si el ciudadano se encuentra afiliado a la SS, y en ese caso genera en pdf y transmite la acreditación del número de afiliación a la SS.

La definición del servicio SS queda tal y como se muestra a continuación:

```
package services;

/**
 * External services involved in procedures from population
*/

public interface SS {   // External service for Social Security Govern administration

    LaboralLifeDoc[4] getLaboralLife (Nif nif) throws NotAffiliatedException,
                ConnectException[5];

    MemberAccreditationDoc[6] getMembAccred (Nif nif) throws NotAffiliatedException,
        ConnectException;
}
```

A parte de la excepción proporcionada por la API ConnectException, definiréis la excepción NotAffiliatedException, que representa la situación en que el ciudadano no se encuentra afiliado a la SS. Todas ellas las podéis consultar en los contratos de referencia (documento *ModelCasosUsPlatfUnif-PartContrates.pdf*).

Este servicio se inyectará a la clase pertinente, por ejemplo, mediante un setter.

Por otro lado, para simular el proceso de autenticación del ciudadano, a través de una autoridad de certificación, utilizaremos la siguiente componente:

- CertificationAuthority. Interviene en dos ocasiones en el proceso de autenticación mediante Cl@ve PIN (*DSS-ObtCertific-SolRef.jpg*):
  - sendPIN(Nif nif, Date valID): proporciona las credenciales del ciudadano en el sistema Cl@ve PIN, a la vez que solicita la emisión del PIN para completar su identificación. Retorna un booleano indicando si todo es correcto.

---

[4] El informe de la vida laboral en formato PDF, representado como un objeto LaboralLifeDoc.

[5] Excepción proporcionada por la API. Señala un error producido al intentar conectar un socket a una dirección y puerto remotos. Por lo general, es debido a que la conexión fue rechazada remotamente (e.g. ningún proceso fue escuchado en la dirección/puerto remoto).

[6] La acreditación del número de afiliación a la SS en formato PDF, representado como un objeto MemberAccreditationDoc.

- checkPIN(Nif nif, PINcode pin): retorna un booleano indicando si el pin corresponde al generado por parte del sistema Cl@ve PIN a ese ciudadano (nif), y éste sigue vigente.

En el caso de utilizar la Cl@ve permanente, la interacción con este servicio cambia ligeramente, y se definen los siguientes métodos:

- ckeckCredent(Nif nif, Password passw): retorna un byte que indicará diferentes situaciones: 0 si el ciudadano no está registrado en el sistema Cl@ve permanente con esas credenciales; 1 sí lo está y no tiene activado el método reforzado; y 2 si lo está y utiliza el método reforzado. *Su implementación es opcional*.
- checkPIN(Nif nif, PINcode pin): *la misma que para el Sistema de Cl@ve PIN presentada arriba*, a aplicar si el ciudadano ha activado el método reforzado para el uso de Cl@ve permanente.

La definición del servicio CertificationAuthority queda tal y como se muestra a continuación:

```
public interface CertificationAuthority {// External service that represents the
                                         different trusted certification entities

      boolean sendPIN (Nif nif, Date date) throws NifNotRegisteredException,
          IncorrectValDateException, AnyMobileRegisteredException, ConnectException;

      boolean checkPIN (Nif nif, PINcode pin) throws NotValidPINException,
          ConnectException;

      byte ckeckCredent (Nif nif, Password passw) throws NifNotRegisteredException,
          NotValidCredException, AnyMobileRegisteredException, ConnectException;
}
```

Las situaciones excepcionales a tratar son las siguientes:

- NifNotRegisteredException indica que el nif proporcionado no está registrado en el sistema Cl@ve PIN.
- IncorrectValDateException: el nif y la fecha de validez del ciudadano no corresponden.
- AnyMobileRegisteredException: el ciudadano no tiene un móvil registrado.
- NotValidPINException: excepción asociada al segundo paso de autenticación, que indica que el PIN proporcionado no se corresponde con el PIN emitido por el sistema Cl@ve, o bien ya ha expirado.
- NotValidCredException: indica que la contraseña proporcionada es incorrecta. *Su implementación es opcional*.

A continuación, se presentan el resto de clases relacionadas con la gestión de los dos trámites relacionados con la SS considerados en el caso de uso. Agruparemos todas ellas en el paquete publicadministration.

# 4. El paquete publicadministration

Comenzaremos con las clases que representan la información que se incluye en los documentos generados por la SS. En concreto, en el documento de la vida laboral.

Sus operaciones serán, básicamente, los constructores y getters (dado que como mucho podremos consultarlos).

## Las clases QuotePeriodsColl y QuotePeriods

La información que recoge el documento de la vida laboral es relativa a los periodos de cotización en la SS por parte del ciudadano. Los representaremos mediante las clases QuotePeriod (un periodo de cotización) y QuotePeriodsColl (todos los periodos de cotización del ciudadano).

Para representar la información relativa a un periodo de cotización utilizaremos la siguiente estructura:

```
public class QuotePeriod {  // Represents a quote period as a registered worker

    private Date initDay;
    private int numDays;

  public QuotePeriod (Date date, int ndays){ . . . }  //  Initializes attributes

  ???   // the getters

  public String toString () { . . . }  // converts to String

}
```

La clase QuotePeriodsColl representa los periodos de cotización de un ciudadano, los cuales deberán estar ordenados por fecha, de la más antigua a la más actual.
La estructura, aunque incompleta, de la clase QuotePeriodsColl es la siguiente:

```
public class QuotePeriodsColl {  // Represents the total quote periods known as a
                                                registered worker

    ??? // Its components, that is, the set of quote periods

  public QuotePeriodsColl (){ . . . }  // Initializes the object

  ???   // the getters

  public addQuotePeriod (QuotePeriod qPd){ . . . }  //  Adds a quote period,
            always respecting the sorting by date, from oldest to later in time

  public String toString () { . . . }  //  Converts to String
}
```

**Implementar y realizar test para estas clases.**

## Las clases PDFDocument, LaboralLifeDoc y MemberAccreditationDoc

La clase PDFDocument representa un documento pdf. Encapsula la ruta en la que se encuentra el documento (un DocPath –paquete data), la fecha de generación (un Date), y un objeto de la clase File.

Esta es su estructura:

```
public class PDFDocument {  // Represents a PDF document

    private Date creatDate;
    private DocPath path;
    private File file;

  public PDFDocument () { . . . }  //  Initializes attributes and
                                   emulates the document download at a default path
    ???   // the getters

  public String toString () { . . . } // Converts to String members Date and DocPath

    // To implement only optionally

  public void moveDoc (DocPath destPath) throws IOException; // Moves the document
                                   to the destination path indicated

  public void openDoc (DocPath path) throws IOException;  // Opens the document
                                   at the path indicated
}
```

El constructor de la clase emulará la descarga del documento en una determinada ruta establecida por defecto. El método toString retorna un String con el valor de los campos creatDate y path.
Con el propósito de que la emulación sea más completa, podéis implementar, *opcionalmente*, los siguientes métodos adicionales: moveDoc y openDoc.
El método moveDoc permitirá mover el documento a la ruta especificada como argumento. Por su parte, el método openDoc cargará el documento pdf en pantalla mediante un visor de pdf. Para ello se puede utilizar la clase de la API Desktop, que permite abrir cualquier tipo de fichero con la aplicación que haya sido definida por defecto en el sistema operativo para ese tipo de archivo.
Un ejemplo típico de uso de la clase Desktop es el siguiente, para abrir un archivo en una determinada ruta:

```
1 try {
2     File path = new File ("temp\laboralLife.pdf");
3     Desktop.getDesktop().open(path);
4    } catch (IOException ex) {
5          ex.printStackTrace();
6     }
```

PDFDocument tiene dos subclases: LaboralLifeDoc y MemberAccreditationDoc.

La clase LaboralLifeDoc representa el documento pdf y la información asociada a la vida laboral de un ciudadano. Esta es su estructura:

```
public class LaboralLifeDoc extends PDFDocument {  // Represents the laboral life
    private Nif nif;
    private QuotePeriodsColl quotePds;

  public LaboralLifeDoc (Nif nif, QuotePeriodsColl qtP) { . . . }
                                    //  Initializes attributes

  ???   // the getters
}
```

Por su parte, la clase MemberAccreditationDoc representa el documento pdf con la información correspondiente al número de afiliación a la SS de un ciudadano. Esta es su estructura:

```
public class MemberAccreditationDoc extends PDFDocument {
                      // Represents the member accreditation document
    private Nif nif;
    private AccredNumb numAffil;

  public MemberAccreditationDoc (Nif nif, AccredNumb nAff){ . . . }
                                    //  Initializes attributes
  ???   // the getters
}
```

Estas clases corresponden a las clases con ese mismo nombre del modelo del dominio de referencia (documento *DCl-PlatfUnif-SolRef.jpg*).

**Implementar y realizar test para estas clases (se recomienda combinar ambas tareas en paralelo, con la finalidad de ir probando poco a poco el código, conforme se va desarrollando).**

## La clase UnifiedPlatform

A continuación, se presenta la estructura, aunque incompleta, de la clase UnifiedPlatform. Esta clase define, entre otros, los eventos de entrada para resolver el caso de uso que nos ocupa.

La estructura, aunque incompleta, de la clase UnifiedPlatform es la siguiente:

```
public class UnifiedPlatform {

    ???   // The class members
```

```
    // Input events
  public void processSearcher () { . . . };

  public void enterKeyWords (String keyWord) { . . . } throws
                AnyKeyWordProcedureException;

  public void selectSS () { . . . };

  public void selectCitizens () { . . . };

  public void selectReports () { . . . };

  public void selectCertificationReport (byte opc) { . . . };

  public void selectAuthMethod (byte opc) { . . . };

  public void enterNIF-PINobt (Nif nif, Date valDate) { . . . } throws
                NifNotRegisteredException, IncorrectValDateException,
                AnyMobileRegisteredException, ConnectException;

  public void enterPIN (PINcode pin) { . . . } throws NotValidPINException,
                NotAffiliatedException, ConnectException;

  public void enterCred (Nif nif, Password passw) { . . . } throws
                NifNotRegisteredException, NotValidCredException,
                AnyMobileRegisteredException, ConnectException;

  private  void printDocument () { . . . } throws BadPathException,
                PrintingException;

  private  void downloadDocument () { . . . };

  private  void selectPath (DocPath path) { . . . } throws BadPathException;

    // Other operations

  private String searchKeyWords (String keyWord) { . . . } throws
                AnyKeyWordProcedureException;

  private  void OpenDocument (DocPath path) { . . . } throws BadPathException;

  private  void printDocument (DocPath path) { . . . } throws BadPathException,
                PrintingException;

  private  void downloadDocument (DocPath path) { . . . } throws BadPathException;

    ???    // Possibly more operations
}
```

A continuación se presentan a grandes rasgos dichos métodos (*algunos de sobras conocidos*). Los contratos de referencia detallados se pueden consultar en el documento *ModeloCasosUsoPlatUnif-ParteContratos.pdf*.

*Eventos de entrada:*

- processSearcher(): se procede a utilizar el buscador de trámites. Este evento emula la acción de clicar en el buscador para desplegar el campo de texto en el que introducir la o las palabras clave.
- enterKeyWords(String keyWord): se introduce/n la/s palabra/s clave en el buscador de trámites. Obtiene la AAPP responsable de ese trámite y la muestra por pantalla.
  *Excepciones*: AnyKeyWordProcedureException que se lanzará en el caso que la palabra introducida no se identifique con ningún trámite registrado en el portal.
- selectSS(): evento que emula la acción de clicar la sección SS en el mosaico inicial.
- selectCitizens(): evento que emula la acción de clicar el enlace 'Ciudadanos', en la sección de la SS.
- selectReports(): evento que emula la acción de clicar el enlace 'Informes y certificados', en el apartado 'Ciudadanos' de la SS.
- selectCertificationReport(byte opc): evento que emula la acción de seleccionar el informe o certificado concreto que se desea obtener, tras presentar un menú con las dos opciones disponibles. Utilizaremos un byte para indicar de qué informe se trata.
- selectAuthMethod(byte opc): evento que representa la acción de seleccionar la herramienta de identificación a utilizar, de entre las disponibles. Igual que antes, se presentará un menú con la/s opción/es disponible/s. La opción escogida se salva en un byte.
- enterNIF-PINobt(Nif nif, Date valDate): emula el uso del formulario donde el usuario introduce los datos que lo acreditan en el sistema Cl@ve PIN, a la vez que solicita el PIN para poder completar su identificación.
  *Se puede desglosar en dos eventos de entrada*: uno para el nif y otro para la fecha de validez.
  *Excepciones*: todas ellas ya mencionadas anteriormente. NifNotRegisteredException (el usuario no está registrado en el sistema Cl@ve PIN); IncorrectValDateException (el nif y la fecha de validez no se corresponden); AnyMobileRegisteredException (el ciudadano no tiene un móvil registrado); y ConnectException.
- enterPIN(PINcode pin): el usuario introduce el PIN recibido vía SMS, con objeto de completar su identificación. Esta operación se aplica siempre en el proceso de identificación con Cl@ve PIN, pero también en los casos en los que se escoge Cl@ve permanente y el usuario tiene activado el método reforzado.
  *Excepciones*: NotValidPINException (el PIN no se corresponde con el PIN emitido por el sistema Cl@ve, o bien ya ha expirado); NotAffiliatedException (ese usuario no está afiliado a la SS); y ConnectException.
- printDocument(): El usuario lanza la orden de imprimir el documento. *No se pide su implementación*.
- downloadDocument(): El usuario lanza la orden de descargar el documento. *No se pide su implementación*.
- selectPath(DocPath path): El usuario escoge la ruta en la que guardar el documento. *No se pide su implementación*.

Además, *de manera opcional*, definir también el siguiente evento de entrada para la autenticación con Cl@ve permanente:

- enterCred(Nif nif, Password passw): emula el uso del formulario donde el usuario introduce las credenciales que lo acreditan en el sistema Cl@ve permanente.
  *Se puede desglosar en dos eventos de entrada*: uno para el nif y otro para la contraseña.

Si el usuario ha activado el método reforzado, automáticamente se generará un PIN, para completar el segundo paso de identificación. Ello implica haber de introducir ese PIN mediante el método enterPIN(pin) anterior.
*Su implementación es opcional.*
*Excepciones*: las mismas mencionadas anteriormente (NifNotRegisteredException, AnyMobileRegisteredException, ConnectException) y, adicionalmente, NotValidCredException, para indicar que las credenciales proporcionadas no son correctas.

*Operaciones internas:*

- searchKeyWords(String keyWord): operación interna del sistema para emular la búsqueda de la administración pública responsable del trámite sugerido mediante keyWord. Retorna un String con el nombre de la administración pública
  *Excepciones*: AnyKeyWordProcedureException, mencionada anteriormente.
- openDocument(DocPath path): operación interna del sistema para cargar y mostrar el documento mediante un visor de pdf. *Su implementación es opcional.*

- printDocument(DocPath path): El sistema ejecuta la orden de imprimir el documento. *No se pide su implementación*.
- downloadDocument(DocPath path): El sistema ejecuta la orden de descargar el documento en la ruta indicada. *No se pide su implementación*.

*Consideraciones:*

- No hace falta contemplar la parte correspondiente a los servicios de impresión ni de descarga de los documentos. Es por ello que no se pide la implementación de los métodos ni de las excepciones relacionadas.
- Definiréis las excepciones como clases propias (subclases de Exception –excepciones *checables*) y, adicionalmente, la excepción proporcionada por la API: ConnectException, tal y como aparece en las cabeceras de los métodos.
- Por lo que respecta a las excepciones correspondientes a las clases del paquete data se dejan para vosotros.

**Implementar y realizar test para el caso de uso descrito aquí, utilizando dobles para los servicios colaboradores.**

# 5. Partes OPCionales

## Los métodos adicionales de la clase PDFDocument *(0,25 puntos)*

Se trata de los métodos de la clase PDFDocument: moveDoc y openDoc, presentados anteriormente, que emularán operaciones básicas de la clase (mover el documento a una ruta destino y abrir el documento para su visualización).

## El Caso de Uso *Autenticar con Cl@ve permanente (0,5 puntos)*

Se trata de implementar los eventos de entrada, servicios de la autoridad de certificación y excepciones relacionadas con este caso de uso, las cuales aparecen detalladas a lo largo del documento.

Se emularán ambos métodos de autenticación: el método simple (tan sólo comprobación de las credenciales) y el método reforzado (con segundo paso de comprobación del PIN).

## El Caso de Uso *Autenticar con certificado digital* (0,75 puntos)

Esta parte, que es **opcional**, trata de emular el proceso de identificación utilizando el certificado digital. Corresponde al caso de uso **Autenticar con certificado digital** cuyos aspectos implicados se describen a continuación.

En primer lugar, se requieren dos nuevos eventos de entrada, a definir en la clase UnifiedPlatform, los cuales se describen a continuación:

```
public void selectCertificate(byte opc);

public void enterPassw(Password pas) throws NotValidPasswordException;
```

- selectCertificate(byte opc): mediante la presentación de un menú, se emula el paso de escoger un certificado digital de entre los que se encuentran instalados (emulando así el uso de una ventana emergente con los distintos certificados).
- enterPassw(Password pas): introducción de la contraseña.
  *Excepciones*: NotValidPasswordException, que contempla la situación de contraseña no válida.

*Procedimiento de autenticación con certificado digital*:

1. Una vez introducida la contraseña, ya se tiene acceso a la clave pública. Ésta debe ser enviada a la autoridad de certificación para chequear su legitimidad. Es así como se prueba la identidad del propietario de la clave pública.
2. Si todo es correcto, la autoridad de certificación utiliza la clave pública para encriptar los datos vinculados con ella, los cuales identifican a su propietario. A continuación, se procede a la entrega del mensaje encriptado.
3. Por su parte, el sistema procede a descifrar el mensaje encriptado (los datos identificativos entregados por la autoridad de certificación) utilizando la clave privada (ésta se encuentra custodiada en el portal unificado). Es así como se completa el proceso de cifrado/descifrado de los datos identificativos. En nuestro caso se trata del nif del usuario para resolver el trámite con la SS.

Este proceso queda plasmado en el DSS proporcionado como anexo, donde se resaltan todas las operaciones implicadas (también publicado en carpeta *Práctica-Proyecto de Software/Soluciones/AnDom*, archivo: *DSS-ObtCertific-CertfDgtal.jpg*).

Siguiendo con los cambios que afectan a la clase UnifiedPlatform, se añadirá también la siguiente operación interna:

```
NIF decryptIDdata(EncryptedData encrypData) throws DecryptationException;
```

- decryptIDdata(EncryptedData encrypData): se encarga de llevar a cabo el paso de descifrar los datos identificativos proporcionados por la autoridad de certificación, contenidos en encrypData. Es el método encargado del paso 3 anterior.
  *Excepciones*: DecryptationException, que saltará ante cualquier incidencia relacionada con el paso de descifrado.

Dado que el paso de descifrado conlleva la aplicación de un algoritmo matemático de cierta complejidad, este paso se delega en un colaborador. Se trata de la componente Decryptor, cuyo servicio es proporcionado mediante un método con el mismo nombre: decryptIDdata siguiente:

```
NIF decryptIDdata(EncryptedData encrypData, EncryptingKey privKey)
        throws DecryptationException;
```

- decryptIDdata(DigitalSignature encrypData, EncryptingKey privKey): aplica un algoritmo matemático para descifrar los datos identificativos proporcionados mediante encrypData. Para ello es indispensable utilizar la clave privada (segundo argumento). De este modo se obtiene el nif del usuario.
  La excepción DecryptationException ya se ha mencionado arriba.

Decryptor formará parte también del paquete services.

Por lo que respecta al servicio CertificationAuthority, éste incorpora ahora un nuevo método para emular la certificación de la identidad utilizando certificado digital. Es el siguiente:

```
EncryptedData sendCertfAuth(EncryptingKey pubKey)
        throws NotValidCertificateException, ConnectException;
```

- sendCertfAuth(EncryptingKey pKey): se envía la clave pública pubKey, a fin de probar la identidad de su propietario y obtener sus datos identificativos. Retorna dichos datos encriptados (en el paso de cifrado la autoridad de certificación hace uso de nuevo de la clave pública). Es el método encargado de los pasos 1 y 2 anteriores.
  La excepción NotValidCertificateException representa cualquier incidencia relacionada con el certificado.

En cuanto a las clases implicadas en este caso de uso, se incorporarán las siguientes: EncryptingKey para representar las claves pública y privada (un valor numérico largo: un BigInteger) y EncryptedData (un mensaje cifrado, que se representará mediante un byte[]). Se trata de clases inmutables, por lo que se incorporarán también en el paquete data.

**Implementar este escenario y realizar test utilizando dobles para los sistemas colaboradores.**
**Se pide añadir lo necesario para completar las pruebas de todos los escenarios del caso de uso implementados (identificación mediante Cl@ve PIN, Cl@ve permanente y/o mediante certificado digital).**

# 6. Consideraciones generales

- La práctica se puede realizar **en grupos de dos o tres personas** (preferiblemente los mismos grupos que hasta ahora).
- Esta práctica tiene un valor de un **20%** sobre la nota final y **no es recuperable**.
- Utilizaréis el sistema de **control de versiones** git y un **repositorio remoto**, a fin de coordinaros con vuestros compañeros (e. g. GitHub, Bitbucket o GitLab –podrán ser repositorios privados). Algunas recomendaciones:
  - Cada vez que hagáis un test y el sistema lo pase, haced un commit. Nunca incorporar a la rama remota código que no ha pasado un test.
  - Cada vez que apliquéis un paso de refactoring en el código, haced un commit indicando el motivo que os ha llevado a hacerlo (¿quizás algún *code smell* o principio de diseño?), así como del refactoring aplicado.
  - Con el fin de facilitar los test, podéis definiros otros constructores además de los sugeridos aquí, simplificando la inicialización de las clases.
  - Es recomendable ir trabajando cada miembro del grupo en ramas distintas para así lograr una mejor colaboración y sincronización de vuestro trabajo (e.g. desarrollo de distintos requisitos/funcionalidades y/o unidades funcionales por separado).
- Como entregaréis un ZIP con el directorio del proyecto, entregaréis también el repositorio git (subdirectorio .git), por lo que podré comprobar los commits (*no os lo dejéis para el último día !*).
- Por lo que respecta al SUT (System Under Test), los eventos de entrada **deben satisfacer los contratos** de referencia facilitados en detalle (documento *ModeloCasosUsoPlatUnif-ParteContratos.pdf*), a excepción de las simplificaciones introducidas en este documento.
- No hagáis test dobles complicados para poder aprovecharlos más de una vez. Se trata de definir test dobles lo más simples posible, con objeto de pasar los test.
- Los test dobles los podéis definir internamente en las clases de test, o bien en un paquete separado.

# 7. ¿Qué se ha de entregar?

Un **ZIP** que contenga:

- El **proyecto desarrollado** (podéis utilizar IntelliJ IDEA o cualquier otro entorno).
- Un pequeño **informe** en el que expliquéis con vuestras palabras las situaciones que habéis probado en cada uno de los test realizados, así como el/los criterio/s empleado/s para tomar vuestras decisiones de diseño (principios SOLID, patrones GRASP, etc.), y justificación de los mismos, si es el caso (*la implementación se os da ya masticada*).
  Podéis añadir tambe cualquier otro detalle que pueda ayudar a valorar mejor vuestro trabajo.

Como siempre, haced la entrega a través del CV **tan sólo uno de los miembros del grupo**, indicando el nombre de vuestros compañeros.

# 8. *Anexo*. DSS para escenario opcional ***Autenticar con certificado digital***