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* (2ª rama).

Se pide implementar y probar una versión simplificada del caso de uso *Solicitar certificado antecedentes penales*. 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). 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, SmallCode (código de tres dígitos utilizado para el Safe Verification Code[1], y también por el sistema Cl@ve[2]), también un String, DocPath (una ruta en el sistema de archivos), a representar como un String y DigitalSignature (para representar la firma electrónica[3]), que representaremos mediante un byte[].

[1] Es el Código de Verificación Seguro de la tarjeta de crédito (el conocido CVS).

[2] Se tratará el sistema Cl@ve PIN y Cl@ve permanente. Por simplicidad, prescindiremos del sistema Cl@ve PIN 24 h.

[3] Un mensaje cifrado que contiene los datos del Ministerio de Justicia.

Además, se utilizará la clase Goal para representar las distintas opciones tipificadas que definen la finalidad del certificado de antecedentes penales (dato requerido para su solicitud).

Consideraremos la existencia del siguiente enumerado (las constantes serán definidas por vosotros):

```
enum goalTypes {WORKWITHMINORS, GAMESECTOR, PUBLICWORKERS, PUBLICADMINCONSORTIUM};
```

A continuación se presenta la clase Nif.

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 también al cumplimentar los datos relativos al ciudadano al iniciar el trámite.

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 SmallCode, DocPath, Goal, DigitalSignature y, *opcionalmente*, 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 *Solicitar certificado antecedentes penales (9,75 puntos)*

En lo que queda de documento se presenta el caso de uso a desarrollar. Se trata de **Solicitar certificado antecedentes penales**[4].

Este caso de uso resuelve la obtención por parte del ciudadano del certificado de antecedentes penales. Por supuesto, la realización de este trámite requiere la autenticación del ciudadano mediante una de las herramientas de identificación consideradas. Todo ello se desarrollará en concordancia con los diagramas utilizados como referencia durante el análisis y, en particular, con el DSS que se incluye como anexo al final de este documento.
Los casos de uso considerados para resolver la identificación son: **Autenticar con Cl@ve PIN** y, *opcionalmente*, **Autenticar con Cl@ve Permanente**.

Definiremos una clase llamada UnifiedPlatform[5] 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.

[4] 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).

[5] Corresponde a la clase del dominio UnifiedPlatform.

Servicios involucrados

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

La parte central del caso de uso **Solicitar certificado antecedentes penales** requiere de cuatro servicios externos. Estamos hablando de JusticeMinistry (el Ministerio de Justicia), CertificationAuthority (la autoridad de certificación), GPD (la Dirección General de la Policía –General Police Direction) y CAS (el Servicio de Autorización de Crédito –Credit Authorization Service).

La interacción de estos servicios con la plataforma unificada se relaciona a continuación.

JusticeMinistry. Este servicio interviene en el momento de obtener el certificado de antecedentes penales requerido por el ciudadano, tras haber completado con éxito todos los pasos previos (consultar *DSS-SolCertAntPen-NoApos.jpg*, incorporado en el anexo):

- getCriminalRecordCertf(Citizen persD, Goal g): a partir del nif (un atributo de la clase Citizen) verifica la existencia de condenas penales de ese ciudadano. Genera el certificado en pdf, estampa la firma electrónica y lo transmite al sistema. En el caso de no constar ninguna condena penal, así se indica. En cualquier caso, se constata la finalidad del certificado.

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

```
package services;

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

public interface JusticeMinistry {  // External service for the Justice Ministry

    CriminalRecordCertf[6] getCriminalRecordCertf (Citizen persD, Goal g)
                throws DigitalSignatureException,ConnectException[7];
}
```

A parte de la excepción proporcionada por la API ConnectException, se tratará la siguiente:

- DigitalSignatureException: indica cualquier tipo de incidencia con la firma electrónica. Todas ellas las podéis consultar en los contratos de referencia (documento *PlatUnifRama2-Contratos.pdf*).

[6] El certificado de antecedentes penales en formato PDF, representado como un objeto CriminalRecordCertf.

[7] 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).

Por otro lado, para simular el paso de autenticación del ciudadano utilizaremos la siguiente componente, la cual, por razones de simplificación (de igual forma que habéis realizado en el análisis), representa de forma genérica las distintas autoridades de certificación:

CertificationAuthority. Interviene en dos ocasiones en el proceso de autenticación mediante Cl@ve PIN (consultar DSS anexo):

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

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, SmallCode pin) throws NotValidPINException,
          ConnectException;
}
```

A parte de 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.

GPD (General Police Direction). Simula el proceso de verificación de los datos personales del ciudadano por parte de la Dirección General de la Policía. Interviene en una ocasión (consultar DSS anexo):

- verifyData(Citizen persData, Goal goal): proporciona los datos personales del ciudadano, juntamente con la finalidad del certificado de antecedentes penales. Retorna un booleano indicando si la verificación se ha realizado con éxito.

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

```
public interface GPD {// External service that represents the
                                        General Police Direction
      boolean verifyData(Citizen persData, Goal goal)
                throws IncorrectVerificationException, ConnectException;
}
```

A parte de ConnectException, las situaciones excepcionales a tratar son las siguientes:

- IncorrectVerificationException: indica que se ha producido una incidencia con la verificación de los datos del ciudadano.

CAS (Credit Authorization Service). Simula el Servicio de Autorización de Crédito. Se encarga de verificar la validez y el estado de la cuenta bancaria facilitada, a fin de cargar el importe del trámite (3.86 €). Interviene en una ocasión para llevar a cabo la transferencia bancaria, caso de ser aprobada:

  - askForApproval(String nTrans, CreditCard cardData, Date date, BigDecimal imp): proporciona el código de la transferencia y los datos de pago del ciudadano, juntamente con la fecha y el importe asociado al trámite. Retorna un booleano indicando si la transferencia se ha realizado con éxito.

La definición de este servicio CAS queda tal y como se muestra a continuación:

```
public interface CAS {// External service that represents the
                                        Credit Authorization Service

      boolean askForApproval(String nTrans, CreditCard cardData,
                Date date, BigDecimal imp) throws NotValidPaymentDataException,
                                InsufficientBalanceException, ConnectException;
}
```

A parte de ConnectException, las situaciones excepcionales a tratar son las siguientes:

- NotValidPaymentDataException: indica que los datos de la tarjeta proporcionados no son correctos.
- InsufficientBalanceException: indica que el importe del pago supera el saldo disponible en la cuenta asociada a la tarjeta.

Todos estos servicios se inyectarán a la clase pertinente, por ejemplo, mediante un *setter*.

A continuación, se presentan el resto de clases relacionadas con la gestión del trámite correspondiente al caso de uso. Agruparemos todas ellas en el paquete publicadministration.

4. El paquete publicadministration

Comenzaremos con las clases que representan las distintas entidades conceptuales involucradas en el dominio del problema: Citizen, CreditCard y CardPayment.
Sus operaciones serán, básicamente, los constructores y getters (dado que como mucho podremos consultarlas).

Las clases Citizen, CreditCard y CardPayment

En primer lugar, la clase Citizen, que representa al ciudadano que hace uso del sistema, y que recoge toda la información necesaria.

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

```
public class Citizen {

// Represents all the information required for a citizen

    private Nif nif;
    private String name;
    private String address;
    private String mobileNumb;
    (. . .)   // Other additional information (not required)

  public Citizen (String name, String add, String mobile){ . . . }
                              //  Initializes attributes
  (. . .)   // the getters

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

La clase CreditCard representa los datos asociados a una tarjeta de crédito, los cuales son indispensables para efectuar, a través del sistema, el pago de aquellos trámites que conllevan un coste.

Esta es su implementación:

```
public class CreditCard {

// The payment information for using the card via internet

    private final Nif nif;   // The nif of the user

    private final String cardNumb;   // The number of the credit card

    private final Date expirDate;    // The expiration date for the credit card

    private final SmallCode svc;        // The Safe Verification Code

    public CreditCard (Nif nif, String cNum, Date d, SmallCode c) {  . . .  }
                                                    //  Initializes attributes

    (. . .)   // the getters

    @Override
    public String toString () {  . . .  }  // converts to String
}
```

Por último, la clase CardPayment, que representa un pago realizado, a ser registrado por el sistema.

Esta es su implementación:

```
public class CardPayment {

// The information associated to the payment realized via internet

    private final String reference;   // The code of the operation

    private final Nif nif;   // The nif of the user

    private final Date date;    // The date of the operation

    private final BigDecimal import;   // The import of the payment

    public CardPayment (Nif nif, BigDecimel imp) {  . . .  }
                                        //  Initializes attributes

    (. . .)   // the getters

    @Override
    public String toString () {  . . .  }  // converts to String
}
```

Implementar y realizar test para estas clases.

Las clases CrimConviction y CrimConvictionsColl

A continuación, se presentan las clases que representan la información que se incluye en el certificado de antecedentes penales. Se trata de información relativa a las condenas penales que tiene registrada una persona mayor de edad.
Utilizaremos las clases CrimConviction (una condena penal) y CrimConvictionsColl (todas las condenas penales del ciudadano). Sus operaciones serán los constructores, getters y una operación para añadir una nueva condena penal sobre la componente CrimConvictionsColl.

La estructura de la clase CrimConviction es la siguiente::

```
public class CrimConviction {  // Represents a criminal conviction registered

    private Date commitDate;
    private String offense;
    private String sentence;

  public CrimConviction (Date commit, String off, String sentc){ . . . }
                              //  Initializes attributes
  (. . .)      // the getters

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

La clase CrimConvictionsColl representa las distintas condenas penales registradas de un ciudadano. Para su implementación deberá escogerse una estructura adecuada para facilitar la consulta de las condenas por fecha.

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

```
public class CrimConvictionsColl {  // Represents the total criminal
                                       convictions registered for a citizen

    ??? // Its components, that is, the set of criminal convictions

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

    (. . .)      // the getters

  public addCriminalConviction (CrimConvictionsColl crmC){ . . . }
              // Adds a criminal conviction

  public CrimConviction getCriminalConviction (Date date){ . . . }
              // Gets a specific criminal conviction by date

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

Implementar y realizar test para estas clases.

Las clases PDFDocument y CriminalRecordCertf

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 & 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 esté definida por defecto en el sistema operativo para este tipo de archivos.
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      }
```

La clase CriminalRecordCertif es una subclase de PDFDocument.

Esta clase representa el documento pdf con la información asociada al certificado de antecedentes penales de un ciudadano. Esta es su estructura:

```
public class CriminalRecordCertf extends PDFDocument {  // Represents
                                          the Criminal Record Certificate
    private Nif nif;
    private String name;
    private Goal goal;
    private DigitalSignature digSign;
    private CrimConvictionsColl crimConvs;

  public CriminalRecordCertf (Nif nif, String name, Goal g,
                              DigitalSignature ds, CrimConvictionsColl crmC)
       { . . . }   //  Initializes attributes

  (. . .)      // the getters
}
```

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).

5. El paquete citizenmanagementplatform

La clase UnifiedPlatform

A continuación, se presenta 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
    ???   // The constructor/s
    // Input events
  public void selectJusMin () { . . . };
  public void selectProcedures () { . . . };
  public void selectCriminalReportCertf () { . . . };
  public void selectAuthMethod (byte opc) { . . . };
  public void enterNIFandPINobt (Nif nif, Date valDate) { . . . } throws
                NifNotRegisteredException, IncorrectValDateException,
                AnyMobileRegisteredException, ConnectException;
  public void enterPIN (SmallCode pin) { . . . } throws NotValidPINException,
                ConnectException;
  private  void enterForm (Citizen citz, Goal goal) { . . . }
                throws IncompleteFormException, IncorrectVerificationException,
                ConnectException;
  private  void realizePayment () { . . . };
  private  void enterCardData (CreditCard cardD) { . . . }
                throws IncompleteFormException, NotValidPaymentDataException,
                InsufficientBalanceException, ConnectException;
  private  void obtainCertificate () { . . . } throws BadPathException,
                DigitalSignatureException, ConnectException;
  private  void printDocument () { . . . } throws BadPathException,
                PrintingException;
    (. . .) // The setter methods for injecting the dependences
    (. . .) // Other input events (not required)
      // Other internal operations (not required)
  private  void registerPayment () { . . . };
  private  void openDocument (DocPath path) { . . . } throws BadPathException;
  private  void printDocument (DocPath path) { . . . }
                throws BadPathException, PrintingException;
}
```

A continuación se presentan a grandes rasgos dichos métodos. Los contratos de referencia se pueden consultar en el documento *PlatUnifRama2-Contratos.pdf*[8].

Eventos de entrada:

- selectJusMin(): evento que emula la acción de clicar la sección Ministerio de Justicia en el mosaico inicial.
- selectProcedures(): evento que emula la acción de clicar el enlace 'Trámites', en la sección de la SS.
- selectCriminalReportCertf(): evento que emula la acción de seleccionar el trámite para obtener el certificado de antecedentes penales.
- selectAuthMethod(byte opc): evento que representa la acción de seleccionar la herramienta de identificación a utilizar, de entre las disponibles. Se presentará un menú con la/s opción/es disponible/s. La opción escogida se guarda en un byte.
- enterNIFandPINobt(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.
  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(SmallCode 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 (parte opcional).
  Excepciones: NotValidPINException (el PIN proporcionado no se corresponde con el PIN emitido por el sistema Cl@ve, o bien ya ha expirado) y ConnectException.
- enterForm(Citizen citz, Goal goal): emula la introducción de los datos personales por parte del usuario, a través de un formulario. Entre los datos requeridos se encuentra la finalidad a la que se destinará el certificado de antecedentes penales. Todos esos datos serán verificados por la DGP.
  Excepciones: IncompleteFormException (campos de información vacíos en el formulario); IncorrectVerificationException (la DGP indica que no puede verificar dichos datos) y ConnectException.
- realizePayment(): el usuario indica que está listo para realizar el pago del importe correspondiente al trámite (emula la acción de un click).
- enterCardData(CreditCard cardD): emula la introducción de los datos requeridos para realizar el pago por parte del usuario. Estos serán transmitidos (junto con el resto de información) al Servicio de Autorización de Crédito para su aprobación y, en consecuencia, efectuar la transferencia bancaria.
  Excepciones: IncompleteFormException (campos de información vacíos en el formulario); NotValidPaymentDataException (los datos de pago son incorrectos); InsufficientBalanceException (saldo insuficiente) y ConnectException.
- obtainCertificate(): evento que emula la acción de indicar al sistema que proceda a generar el certificado de antecedentes penales estándar (sin el registro de la

[8] Este documento no incluye los contratos de los eventos que únicamente emulan la acción de un click (no tienen ningún efecto en los objetos del dominio).

apostilla), mediante conexión con el Ministerio de Justicia, una vez completados todos los pasos requeridos.
Excepciones: BadPathException (problemas en las operaciones de acceso al directorio de archivos); DigitalSignatureException (problemas con la firma electrónica) y ConnectException.

- printDocument(): el usuario lanza la orden de imprimir el documento. *No se pide su implementación*.

Operaciones internas:

- registerPayment(): operación interna del sistema para registrar el pago del trámite. *No se pide su implementación*.
- 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*.

Consideraciones:

- **Deben tratarse también las situaciones descritas en las precondiciones** (documento *PlatUnifRama2-Contratos.pdf*), derivadas del hecho de no haber completado con éxito los pasos previos del caso de uso (una única excepción, ProceduralException, que unifica todas las situaciones relacionadas con este aspecto).
- 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** (su definición y su tratamiento).

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

6. 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 servirán para emular 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 las clases inmutables, los servicios de la autoridad de certificación y excepciones específicas para este caso de uso, así como los eventos de entrada involucrados, los cuales se detallan a continuación.

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

En primer lugar, para trabajar con Cl@ve permanente se utilizará la clase Password (un String), a incorporar en el paquete data.

Por lo que respecta a los servicios externos, CertificationAuthority debe incorporar un método específico para Cl@ve permanente, puesto que la interacción con este servicio cambia ligeramente. Se trata del siguiente:

- ckeckCredent(Nif nif, Password passw): retorna un byte que indicará las siguientes 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*.

Nos referimos al siguiente método:

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

Tal y como se observa, la situación excepcional específica a tratar es la siguiente:

- NotValidCredException: indica que las credenciales proporcionadas no son correctas.

Suponiendo que el ciudadano haya activado el método reforzado, se llevará a cabo el segundo paso invocando a:

- checkPIN(Nif nif, SmallCode pin): *el mismo que para el Sistema Cl@ve PIN* (presentado anteriormente).

Por otro lado, la clase UnifiedPlatform ahora tendrá un evento de entrada adicional. Se trata del siguiente:

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

- 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.
  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

dicho PIN mediante el método enterPIN(pin), presentado anteriormente para Cl@ve PIN.
Excepciones: las ya mencionadas y anteriormente descritas (NifNotRegisteredException, NotValidCredException, AnyMobileRegisteredException y ConnectException).

Implementar este escenario y añadir lo necesario para completar las pruebas y realizar test utilizando dobles para los sistemas colaboradores.

7. 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 *PlatUnifRama2-Contratos.pdf*), a excepción de las simplificaciones introducidas en este documento.
- **Nivel de exhaustividad en los test**:
  - Para las clases del paquete data es suficiente con probar las excepciones.
  - Deben probarse mínimamente las clases del paquete publicadministration.
  - **Donde las pruebas deben ser exhaustivas** es en el paquete citizenmanagementplatform. **Deberán tratarse todos los escenarios posibles** del caso de uso, con todas sus situaciones excepcionales. Para ello **es imprescindible planear una cierta estructura para el código de test**, tanto para las clases de test, como para los test dobles (e.g., podrían definirse por separado los distintos casos de test: los de éxito y los de fracaso).

    - Además, podéis recurrir a la definición de **interfaces de test**, así como la definición de **métodos default**, tal y como se ha sugerido en clase para la resolución de los problemas de la colección de problemas de testing.
- Os indico, además, **cómo testear la clase controladora del caso de uso**:
    - Los eventos de entrada del caso de uso no se testean individualmente. Cada evento es testeado precediéndolo de los eventos que van por delante en el caso de uso. De esta forma se va probando el progreso del caso de uso y, de no ser el esperado, deberá comprobarse el lanzamiento de la excepción ProceduralException, asociada a las **precondiciones**.
- 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 probar los distintos escenarios.
- Los test dobles pueden definirse internamente en las clases de test, o bien como clases separadas en paquetes separados.

8. ¿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 también 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.

9. *Anexo*. DSS adaptado: Solicitar certificado antecedentes penales sin apostilla y sin usar buscador