

DESARROLLO DE APLICACIONES CON DNIE

**CASO PRÁCTICO DE DESARROLLO DE APLICACIÓN DE
FIRMA**

Copyright © 2011 Instituto Nacional de Tecnologías de la comunicación (INTECO)



El presente documento está bajo la licencia Creative Commons Reconocimiento-No comercial-Compartir Igual versión 2.5 España.

Usted es libre de:

- copiar, distribuir y comunicar públicamente la obra
- hacer obras derivadas

Bajo las condiciones siguientes:

- **Reconocimiento.** Debe reconocer los créditos de la obra de la manera especificada por el autor o el licenciador (pero no de una manera que sugiera que tiene su apoyo o apoyan el uso que hace de su obra).
- **No comercial.** No puede utilizar esta obra para fines comerciales.
- **Compartir bajo la misma licencia.** Si altera o transforma esta obra, o genera una obra derivada, sólo puede distribuir la obra generada bajo una licencia idéntica a ésta.

Al reutilizar o distribuir la obra, tiene que dejar bien claro los términos de la licencia de esta obra.

Alguna de estas condiciones puede no aplicarse si se obtiene el permiso del titular de los derechos de autor

Nada en esta licencia menoscaba o restringe los derechos morales del autor.

Esto es un resumen legible por humanos del texto legal (la licencia completa) disponible en:

<http://creativecommons.org/licenses/by-nc-sa/2.5/es/>

ÍNDICE

1. ÍNDICE	3
2. MOTIVACIÓN	4
3. ANÁLISIS	5
3.1. Toma de Requisitos	5
3.1.1. Requisitos Generales	5
3.1.2. Requisitos operacionales	6
3.1.3. Requisitos de Seguridad	7
3.2. Casos de Uso	9
3.2.1. Caso de Uso Firmar una Factura Electrónica	9
3.2.2. Caso de Uso Validar una Factura Electrónica	10
3.2.3. Caso de Uso Verificar Aplicación	11
4. DISEÑO	13
4.1. Diagramas de Flujo	13
4.1.1. Diagrama de Flujo Firmar Factura Electrónica	14
4.1.2. Diagrama de Flujo Verificar Factura	15
4.1.3. Diagrama de Flujo Verificar Aplicación	16
4.2. Diagrama de Clases	16
4.3. Definición de Funciones	18
5. DESARROLLO	21
5.1. Herramientas	21
5.2. Configuración del entorno de Desarrollo	22
5.2.1. Configuración de las librerías del MITyC	22
5.3. Codificación	25
5.3.1. Operación Verificar Aplicación	27
5.3.2. Operación Firma de facturas	30
5.3.3. Operación Verificar Factura	39
6. ABREVIATURAS	48

Motivación

Para ejemplificar el desarrollo de aplicaciones basadas en el DNle se ha desarrollado un caso práctico basado en la creación de una aplicación Java cliente que permita, desde el menú contextual del explorador de ficheros de Microsoft Windows, realizar las operaciones de firma y validación de facturas electrónicas usando como dispositivo seguro de creación de firma el DNI electrónico.

Actualmente en el mercado existen aplicaciones, algunas gratuitas, que ya permiten la firma y verificación de facturas electrónicas.

Durante este ejemplo práctico de desarrollo de una Aplicación de creación y verificación de firma electrónica, o SCVA, se realiza un análisis de lo que queremos desarrollar, se realiza un diseño en base al análisis y por último se dan las pautas para la codificación del SCVA.

Análisis

Toma de requisitos

Como en cualquier desarrollo de software lo primero que hay que hacer es una toma de requisitos. Hay que tener en cuenta no sólo los requisitos generales sino los requisitos específicos en cuanto a términos de seguridad.

Requisitos Generales

Como ya se ha comentado la aplicación del caso práctico es muy sencilla en cuanto a funcionalidad si bien se pueden establecer los siguientes requisitos generales:

Requisito	Descripción
RG-001	Se deben poder firmar facturas electrónicas
RG-002	El formato de las facturas para ser firmadas será Facturae 3.1
RG-003	La política de firma de las facturas será la política de firma de Facturae 3.1 del MITyC (Ministerio de industria turismo y comercio)
RG-004	El formato de firma será XADES-EPES
RG-005	Se deben poder verificar facturas electrónicas
RG-006	El formato de las facturas firmadas será Facturae 3.1
RG-007	El tipo de firma que se aceptará para la validación será XADES-EPES
RG-008	Para la validación se seguirán las normas de la política de Firma 3.1 del MITyC
RG-009	La aplicación se ejecutará desde el menú contextual del explorador de archivos de Windows XP y Windows 7
RG-010	No se soportaran firmas o validaciones en lote

RG-011	Las facturas firmadas tendrán el mismo nombre que las facturas originales más la extensión .xsig
RG-012	La aplicación será desarrollada en Java 1.6
RG-013	La aplicación será desarrollada con las APIs ofrecidas por el MITyC.
RG-014	Funcionará sólo con el DNle

Para este caso práctico la lista de requisitos generales está muy acotada aunque para una aplicación más compleja se deberán contemplar muchos más requisitos generales.

Requisitos operacionales

En los requisitos operacionales se debe definir qué debe cumplir el entorno operacional para ser considerado un entorno seguro sobre el que la aplicación se pueda ejecutar.

A la hora de definir estos requisitos más allá de los requisitos que sean necesarios para la aplicación también hay que tener en cuenta las restricciones de seguridad que hay que cumplir. Estas restricciones son las siguientes:

- **O.SSCD, dispositivo seguro de creación de firma**

El dispositivo seguro de creación de firma que usa la **Aplicación de firma y verificación de facturas electrónicas en formato Facturae** será el DNle.

- **O.ITENV, protecciones y mecanismos de seguridad**

La plataforma de propósito general que la **Aplicación de firma y verificación de facturas electrónicas en formato Facturae** necesita para operar y para facilitar los interfaces de firmante y con el DNle, facilita las protecciones y mecanismos de seguridad adecuados para proteger los activos de la SCVA, mediante una combinación eficaz de medidas de índole técnico, de procedimientos y de aseguración de su entorno.

Para este caso práctico se definen los siguientes requisitos operacionales:

Requisito	Descripción
RO-001	Se requiere Windows 7 o Windows XP

RO-002	No se soporta ningún otro sistema operativo excepto los definidos en RO-001
RO-003	Se requiere Java 1.5.X o Java 1.6.X
RO-004	No se soporta ninguna versión de Java excepto las definidas en RO-002
RO-005	Se requiere tener instalado el middleware del DNle
RO-006	Para la validación de certificados se requiere conexión a Internet

Estos requisitos operacionales se deberán especificar claramente en la guía de instalación de la aplicación.

Requisitos de seguridad

A la hora de diseñar la aplicación no hay que tener en cuenta únicamente qué funcionalidades se quieren cubrir de cara al usuario final, sino también, qué funcionalidades de seguridad tiene que cumplir.

Para este caso práctico se tienen en cuenta los requisitos de seguridad que se muestran en la siguiente tabla.

Requisito	Descripción
RS-001	Antes de realizar la firma se verificará que el formato de la misma se corresponde con lo definido en RG-002
RS-002	Se mostrará el documento a Firmar
RS-003	Se mostrará la política de Firma definida en RG-006
RS-004	El usuario deberá dar su conformidad antes de realizar la firma de una factura

RS-005	Se mostrará por pantalla un aviso sobre la LOPD
RS-006	Se comprobará la conexión de la aplicación con el DNI electrónico en cada operación
RS-007	Se enviarán los datos a firmar al DNle por un canal seguro
RS-008	Los datos resultantes de una firma se guardaran en un fichero
RS-009	En el proceso de validación de una factura se comprobará que el esquema es correcto
RS-010	En el proceso de validación de una factura se comprobará que la contabilidad es correcta
RS-011	Al validar una factura se validará la firma
RS-012	Al validar una factura se validará el certificado
RS-013	Se mostrará el resultado de las validaciones por pantalla
RS-014	Se podrá validar que la aplicación funciona correctamente
RS-015	Se podrá validar que el entorno operacional es el correcto
RS-016	El usuario podrá ver un informe de si la aplicación funciona

	correctamente
--	---------------

Casos de uso

Una vez están definidos los requisitos de la aplicación el siguiente paso recomendable es especificar los casos de uso. Los casos de uso resultan de utilidad a la hora de abordar un desarrollo, ya que especifican la comunicación y comportamiento de un sistema mediante su interacción con los usuarios y otros sistemas.

Ya que la aplicación es muy sencilla a priori se pueden identificar dos casos de uso a partir de los requisitos generales:

- **Firmar una factura electrónica**
- **Verificar una factura electrónica**

Sin embargo si analizamos los requisitos de seguridad RS-014 a RS-016 observaremos que también hay que considerar otro caso de uso:

- **Verificar que la aplicación funciona correctamente**

Como se puede comprobar tener bien catalogados los requerimientos ayuda a identificar los casos de uso de la SCVA.

Para definir los casos de uso se pueden utilizar diferentes plantillas. Como en este ejemplo práctico se utilizan casos de uso sencillos se definirán en una simple tabla, aunque se podría utilizar alguna herramienta específica.

Caso de uso: firmar una factura electrónica

Actores	Usuario de la aplicación
Desencadenante	El usuario activa la opción de firmar factura desde el menú contextual
Descripción	Se verifica que el fichero corresponda a una factura y se realiza su firma usando el DNI electrónico
Precondiciones	La aplicación está correctamente instalada y el entorno operativo es el

	adecuado
Postcondiciones	Se obtiene la factura firmada
Flujo normal	Se muestran los datos que se van a firmar
	Se muestra la política de Firma
	Se muestra el Aviso del mensaje de la LOPD
	El usuario selecciona el certificado de Firma
	Se accede al dispositivo de firma
	El usuario introduce el código pin (CHV, Card Holder Verification) del dispositivo de Firma
	Se muestra un aviso de seguridad
	Se crea un fichero con la factura firmada
	Se muestra un informe del resultado de la operación
Excepciones	El usuario cancela la operación en algún punto
	El usuario no acepta los avisos de seguridad
	El dispositivo de firma no está conectado
	El usuario introduce un pin erróneo

Caso de uso: validar una factura electrónica

Actores	Usuario de la aplicación
Desencadenante	El usuario activa la opción de validar factura desde el menú contextual
Descripción	Se verifica que el fichero corresponda a una factura y se realiza su

	validación
Precondiciones	La aplicación está correctamente instalada y el entorno operativo es el adecuado
Postcondiciones	Se obtienen los datos de verificación de la firma
Flujo normal	Se muestra la factura a validar
	Se muestra la política de validación
	Se valida el esquema de la factura
	Se realiza la validación contable
	Se valida la firma
	Se valida el certificado
	Se muestra un informe del resultado de la operación
Excepciones	El usuario cancela la operación en algún punto

Caso de uso: verificar aplicación

Actores	Usuario de la aplicación
Desencadenante	El usuario activa la opción de Verificar la aplicación
Descripción	Se verifica que la aplicación funciona correctamente
Precondiciones	La aplicación está correctamente instalada y el entorno operativo es el adecuado
Postcondiciones	
Flujo normal	Se comprueba la versión del sistema operativo

	Se comprueba la versión de Java instalada
	Se comprueba la existencia de las librerías necesarias
	Se comprueba si las rutas están bien configuradas
	Se muestra un informe del resultado de la operación
Excepciones	

Diseño

Una vez realizada la fase de análisis con la toma de requisitos y la definición de casos de uso se debe proceder al diseño de la solución.

Lo primero que se debe decidir es la tecnología que se va a utilizar para desarrollar la solución. Para tomar esta decisión se debe tener en cuenta criterios de seguridad, experiencia de los desarrolladores, funcionalidades requeridas, etc.

En el caso de este ejemplo práctico se ha decidido utilizar Java, por los siguientes motivos:

- Es un lenguaje orientado objetos lo que ofrece mayor control sobre el código y sus excepciones que los lenguajes procedurales.
- Existen utilidades para Java que facilitan la creación y verificación de firmas electrónicas.
- Experiencia del equipo de desarrollo en esta tecnología.

Diagramas de flujo

Una herramienta recomendable de cara a diseñar un desarrollo es realizar diagramas de flujo, a continuación se representan los anteriores casos de uso como diagramas de flujo para una mejor asimilación de los procesos de firma, validación y verificación del sistema.

Los diagramas de flujo permiten ver cuál es la secuencia de las operaciones que componen los casos de uso así como las posibles excepciones o variantes que pueden ocurrir. Considerar las excepciones y tratarlas correctamente evita posibles vulnerabilidades en el resultado final.

Existen en el mercado distintas herramientas para la confección de diagramas de flujo que pueden resultar de utilidad cuando se manejan casos de uso complejos. Aunque los casos de uso son sencillos se puede comprobar que los diagramas de flujo pueden ser enrevesados. Es por ello recomendable definir los casos de uso como operaciones simples, y si un caso de uso es muy complejo intentar dividirlo en varios casos de uso más sencillos.

Diagrama de flujo: firmar factura electrónica

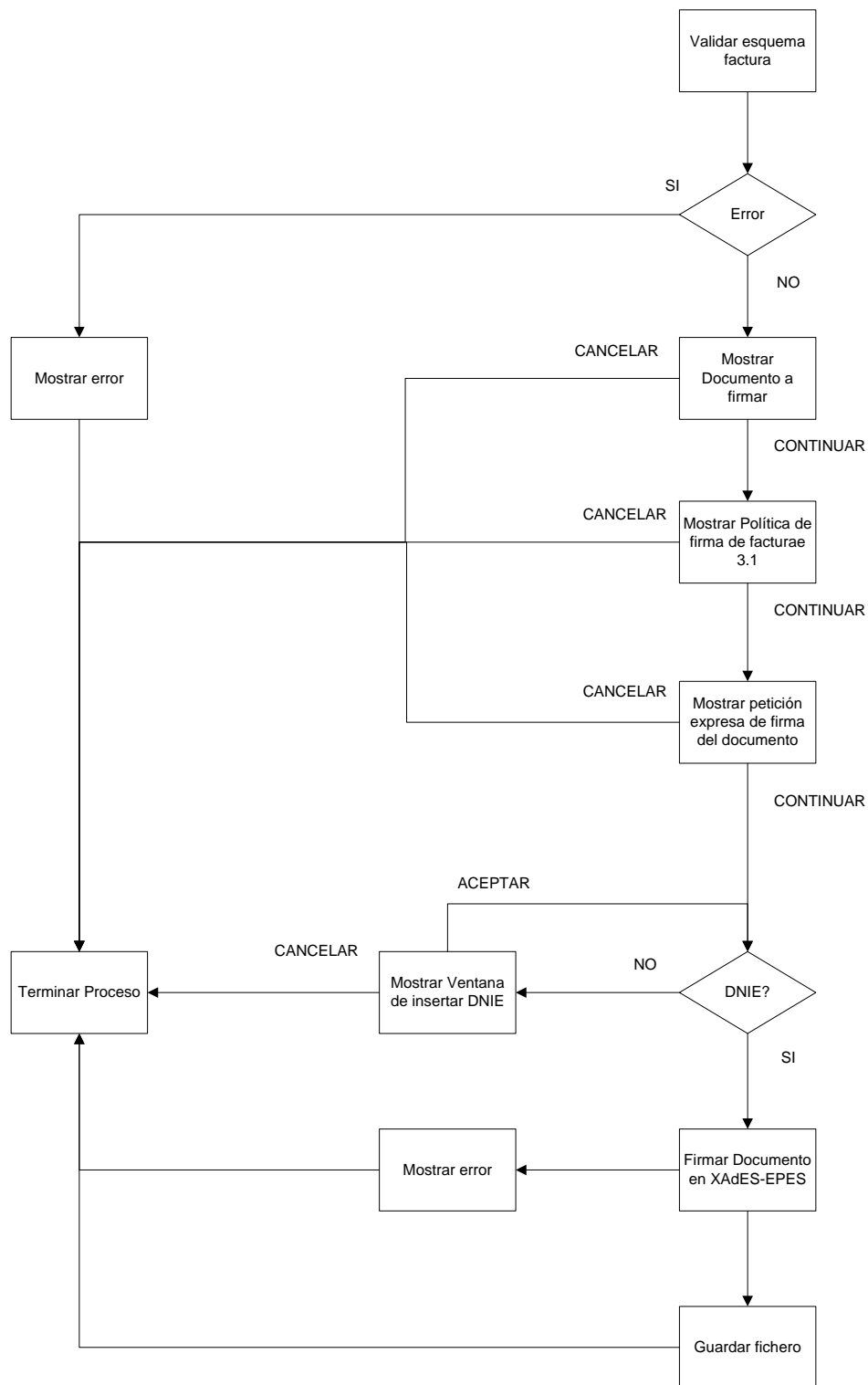


Diagrama Flujo Operación Firma

Diagrama de flujo: verificar factura

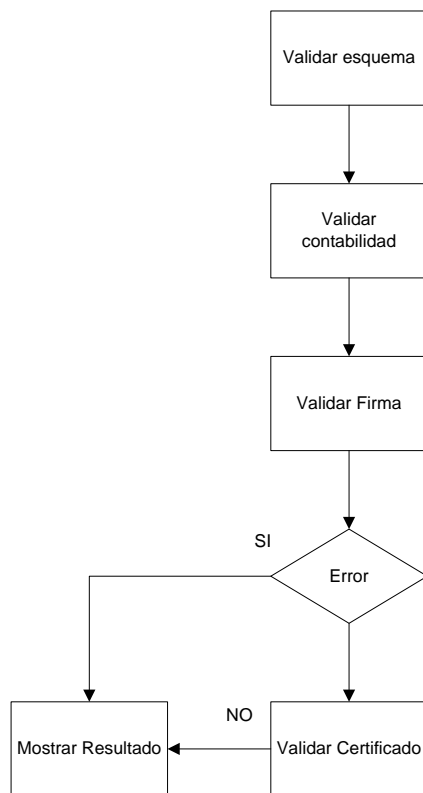


Diagrama Flujo Operación Verificación Factura

Diagrama de flujo: verificar aplicación

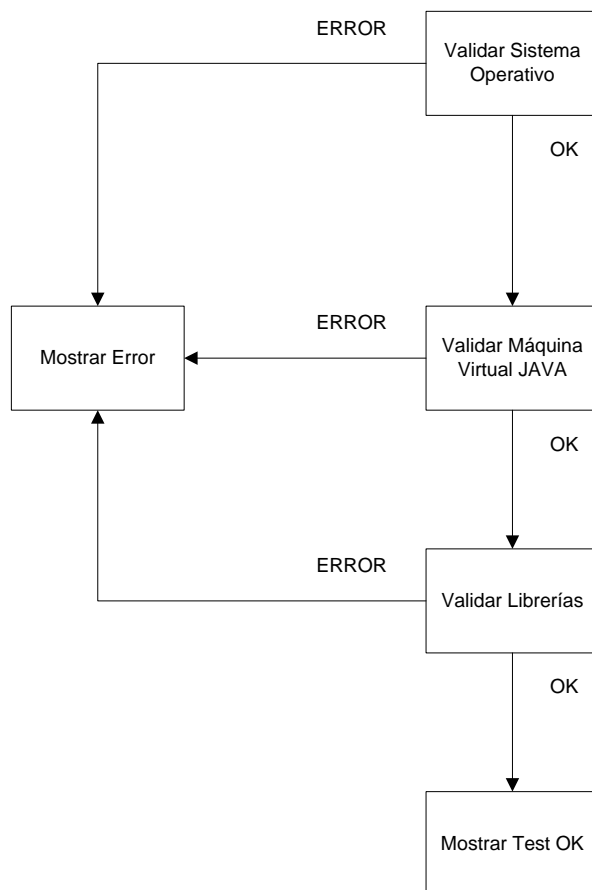


Diagrama Flujo Operación Verificación Aplicación

Diagrama de clases

Un **diagrama de clases** es un tipo de diagrama estático que describe la estructura de un sistema mostrando sus clases, atributos y las relaciones entre ellos. Esta herramienta sólo tiene sentido en el caso de utilizar una tecnología orientada a objetos, como es el caso de Java.

Es recomendable el uso de diagramas de clases para definir el diseño conceptual de la estructura y organización del sistema, y los componentes que se encargarán del funcionamiento y la relación entre uno y otro.

Existe mucha bibliografía sobre cómo distribuir las clases en base a patrones UML (*Unified Modelling Language*) que es un conjunto de diagramas con los que describir sistemas

informáticos. En el caso de aplicaciones complejas es recomendable considerar el uso de alguno de estos patrones predefinidos.

Para el caso de esta aplicación de ejemplo se ha realizado una estructura de clases sencilla como se puede ver en el siguiente diagrama:

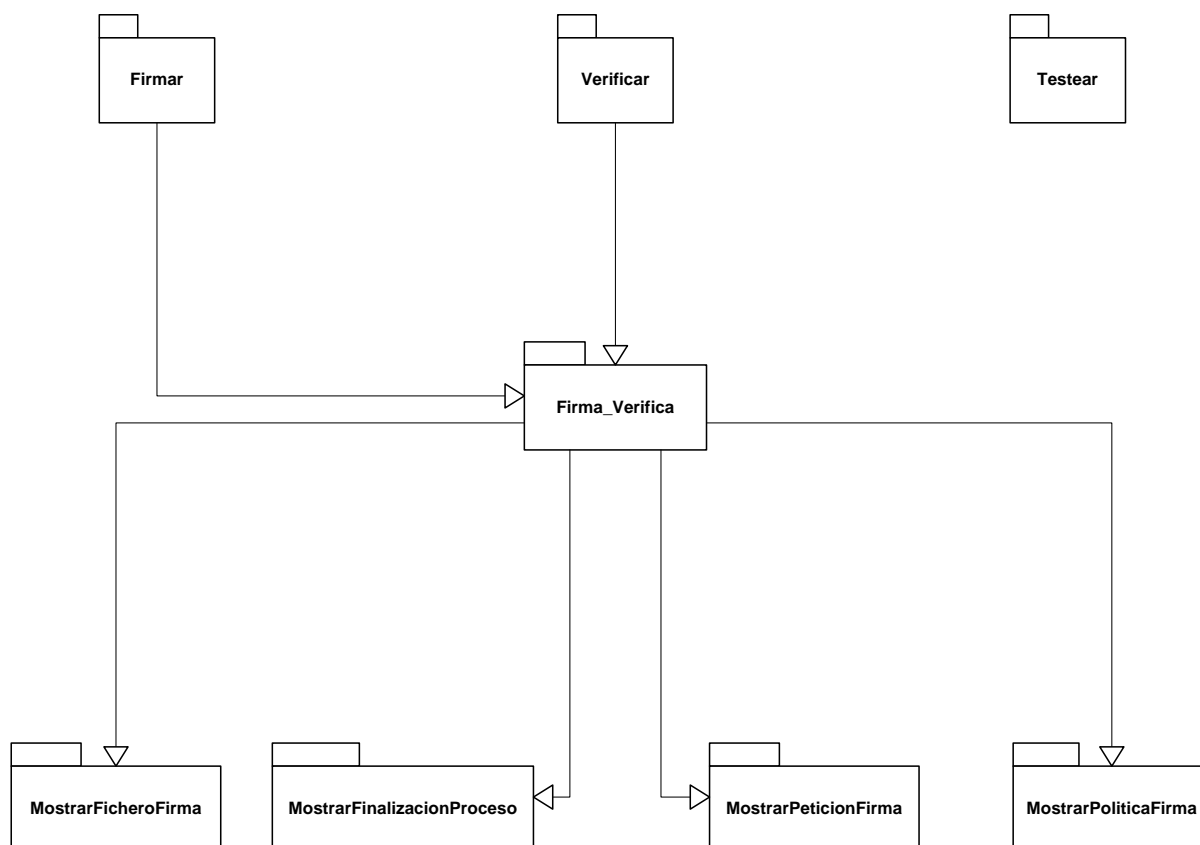


Diagrama de clases

Tal como se puede observar se crea una clase para cada uno de los casos de uso identificados:

- **Clase Firmar:** Caso de Uso Firmar una Factura Electrónica
- **Clase Verificar:** Caso de Uso Verificar una Factura Electrónica
- **Clase Testear:** Caso de Uso Verificar Aplicación

Para minimizar los puntos por donde la aplicación puede ser atacada se ha definido una clase que funciona como *dispatcher* para todas las operaciones de firma y verificación:

- **Clase FirmaVerifica**

Esta clase contiene la lógica de la aplicación y aglutina la llamada de las clases *Firmar* y *Verificar*.

Por último se ha definido una serie de clases necesarias para la interfaz que se va a presentar al usuario:

- **Clase MostrarFicheroFirma**
- **Clase MostrarFinalizaciónProceso**
- **Clase MostrarPeticiónFirma**
- **Clase MostrarPolíticaFirma**

Tal como se puede apreciar en el diagrama todas las llamadas de firma y verificación pasan a través de la clase *dispatcher* FirmaVerifica lo que supone una mayor mantenibilidad, legibilidad, flexibilidad y seguridad ya que la lógica está centralizada en un único punto.

Definición de funciones

Como parte del diseño de las clases se deberá especificar qué métodos contiene cada clase. Es muy recomendable realizar este proceso antes de la codificación ya que se consigue que esta sea mucho más estructurada y que simplemente consista en implementar la lógica de cada método.

Idealmente se podría encomendar el diseño y la codificación a perfiles diferentes, siendo un analista la persona encargada de la definición de las clases y el código que éstas contienen y un desarrollador el que codifica estas funciones.

En la siguiente tabla se muestran las funciones que componen las clases del ejemplo y se describe qué función realiza cada una:

Clase	Método	Descripción
Firmar	main	Este método se invoca desde el menú contextual y redirige la llamada a la clase FirmaVerifica
Verificar	main	Este método se invoca desde el menú

		contextual y redirige la llamada a la clase FirmaVerifica
Test	validar	Realiza el proceso de comprobación de que la aplicación funciona correctamente
	mostrarResultado	Muestra un informe con el resultado de la validación
FirmaVerifica	firma	Ejecuta el proceso de firma llamando a todas las operaciones que son necesarias para completar el proceso
	verificacion	Ejecuta el proceso de verificación de una factura llamando a todas las operaciones que son necesarias para completar el proceso
	mostrarFichero	Muestra el fichero que se va a firmar
		Retorna si ha aceptado o rechazado la operación
	mostrarFicheroValidar	Muestra el fichero que se va a validar
		Retorna si ha aceptado o rechazado la operación
	mostrarPeticionFirma	Muestra una ventana pidiendo la petición expresa de firmado del documento
		Retorna si ha aceptado o rechazado la operación
	mostrarPolitica	Muestra el fichero de política de Facturae 3.1

		Retorna si ha aceptado o rechazado la operación
	mostrarResultado	Muestra el resultado de la operación realizada
MostrarFicheroFirma	n/a	Clase con el diálogo gráfico que se muestra al usuario para visualizar el fichero que se va a Firmar
MostrarFinalizacionProceso	n/a	Clase con el diálogo gráfico que se muestra al usuario para visualizar el resultado del Proceso
MostrarPeticionFirma	n/a	Clase con el diálogo gráfico que se muestra al usuario para pedir la petición expresa
MostrarPoliticaFirma	n/a	Clase con el diálogo gráfico que se muestra al usuario para visualizar la política de Firma

Desarrollo

Herramientas

Antes de empezar la codificación hay que decidir que herramientas se van a utilizar en el desarrollo. Lo primero que hay que decidir es qué IDE (*Integrated Development Environment*) se va a utilizar. La utilización de un IDE no es un requisito aunque sí que es recomendable, ya que permite tener un mayor control sobre el código e incorpora utilidades como la depuración del código que son muy necesarias.

Para este ejemplo se ha usado [NetBeans](#) que es un IDE libre y gratuito, si bien existen otras alternativas como [Eclipse](#), también de gran difusión entre la comunidad de desarrolladores Java.

Además, para realizar las operaciones de firma, la aplicación se ha basado en la utilización de unas APIs proporcionadas por el MITyC (Ministerio de Industria Turismo y Comercio) que permiten la firma, verificación, y gestión de facturas en formato Facturae.

La utilización de estas APIs simplifica de una forma significativa el desarrollo, ejemplificando de esta forma la sencillez en la programación de aplicaciones que hacen uso del DNIE empleando componentes de programación con un alto nivel de abstracción.

Se pueden descargar las [APIs empleadas](#) en este caso práctico y el [formato del documento factura](#).

Aunque no se ha considerado para este ejemplo debido a la sencillez del mismo, es muy recomendable el uso de un repositorio de código para controlar la evolución del desarrollo. Esta herramienta permite llevar un control de las versiones que se van generando y es especialmente necesario si va a haber varios desarrolladores implicados en el proyecto.

En el mercado existen varias alternativas de herramientas para repositorio de código fuente (SVN, CVS, GIT,...), algunas de ellas gratuitas.

Configuración del entorno de desarrollo

Las herramientas de desarrollo que se han seleccionado se deben configurar adecuadamente antes de empezar a escribir la primera línea de código. Aunque no es un requisito de cara a la certificación es recomendable tener una guía de la “Gestión de la Configuración”.

Los pasos y convenios que se han empleado deben escribirse para permitir a un hipotético nuevo desarrollador que se incorpore al proyecto configurar su sistema de forma sencilla. En la guía se debe recoger el nombre interno del proyecto, directorios del proyecto, variables de entorno, acceso al repositorio, etc.

En el caso de la aplicación que nos ocupa la configuración de la misma prácticamente se limita a la configuración de las librerías del MITyC mencionadas anteriormente.

Configuración de las librerías del MITyC

El MITyC provee distintas librerías a través de su página web:

- **API para Internet Explorer:** sirve para acceder al almacén de certificados de Windows desde una aplicación Java.
- **API para Mozilla bajo Microsoft Windows:** sirve para el acceder a las clases del almacén de certificados del un navegador Mozilla que se ejecute bajo Microsoft Windows.
- **API para Mozilla bajo Linux:** sirve para el acceder a las clases del almacén de certificados del un navegador Mozilla que se ejecute bajo Linux.

Tal como se especifica en los requisitos generales la aplicación va a ejecutarse en sistemas operativos Windows por lo que el API utilizado para esta aplicación ha sido el API para IExplorer.

Las librerías además de contener las clases necesarias vienen acompañadas por un tutorial que muestra:

- proceso de instalación del API
- componentes que conforman el API
- descripción de las funcionalidades del API

- ejemplos de uso del API
- guías de configuración del API
- extensiones del API

En el API existen tres librerías que hay que utilizar y que deben indicarse en el classpath **en el orden** que se indica a continuación:

- Facturae-API.jar
- Facturae-API_CompPack.jar
- FacturaEBridge.jar

Si no se hace uso de ningún IDE la configuración del Classpath se deberá hacer por línea de comandos, o bien a través de un fichero .bat, cuando se proceda a compilar la aplicación. El siguiente ejemplo muestra la sentencia que se debería ejecutar para la compilación de la aplicación desde línea de comando:

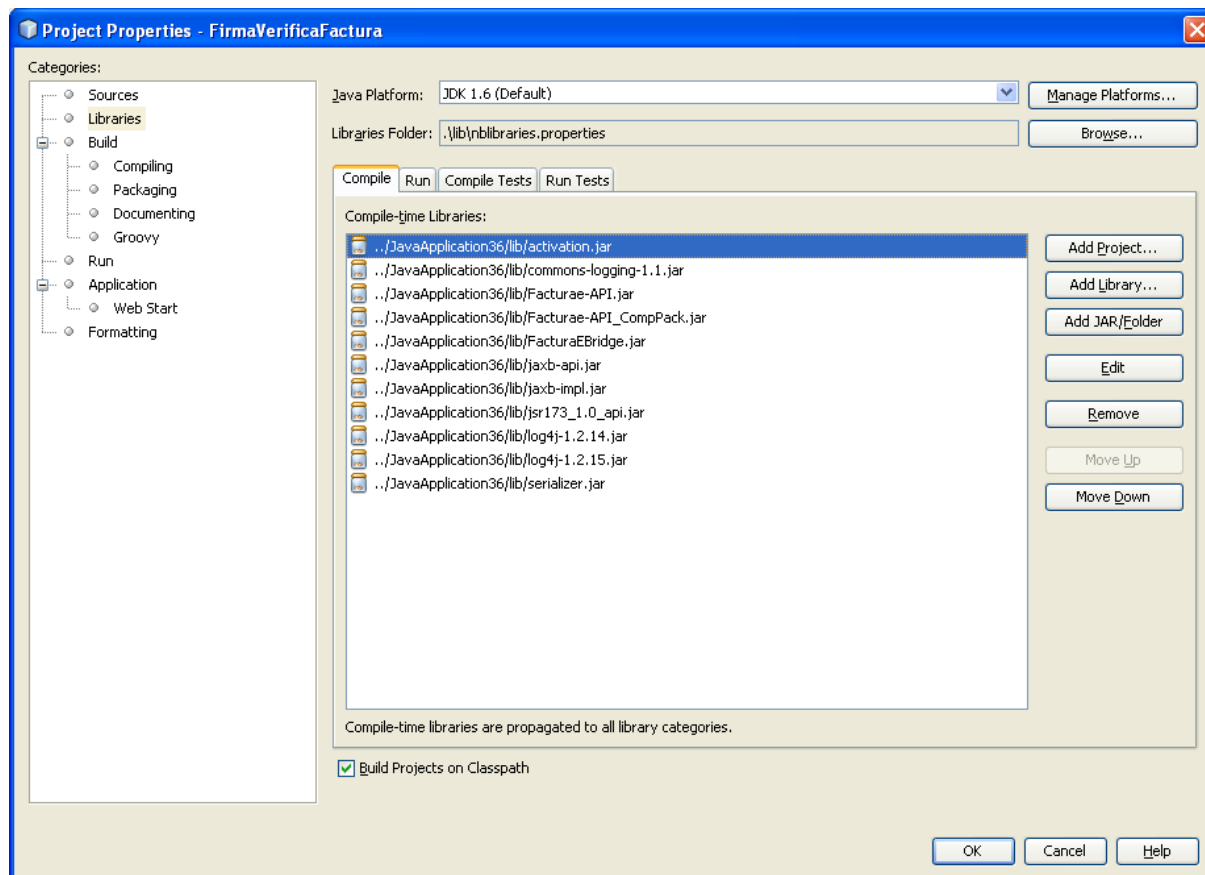
```
javac -classpath lib/activation.jar;lib/commons-logging-  
1.1.jar;lib/Facturae-API.jar;lib/Facturae-  
API_CompPack.jar;lib/FacturaEBridge.jar;lib/jaxb-api.jar;lib/jaxb-  
impl.jar;lib/jsr173_1.0_api.jar;lib/LibXADESJNI_IE_W.jar;lib/log4j-  
1.2.15.jar;lib/swing-layout-1.0.3.jar;. ClaseCompilar.java
```

Asimismo para la ejecución de la aplicación también deberían incluirse las librerías mencionadas en el Classpath tal como se muestra en el siguiente ejemplo:

```
java -classpath lib/activation.jar;lib/commons-logging-  
1.1.jar;lib/Facturae-API.jar;lib/Facturae-  
API_CompPack.jar;lib/FacturaEBridge.jar;lib/jaxb-api.jar;lib/jaxb-  
impl.jar;lib/jsr173_1.0_api.jar;lib/LibXADESJNI_IE_W.jar;lib/log4j-  
1.2.15.jar;lib/swing-layout-1.0.3.jar;. ClaseEjecutar
```

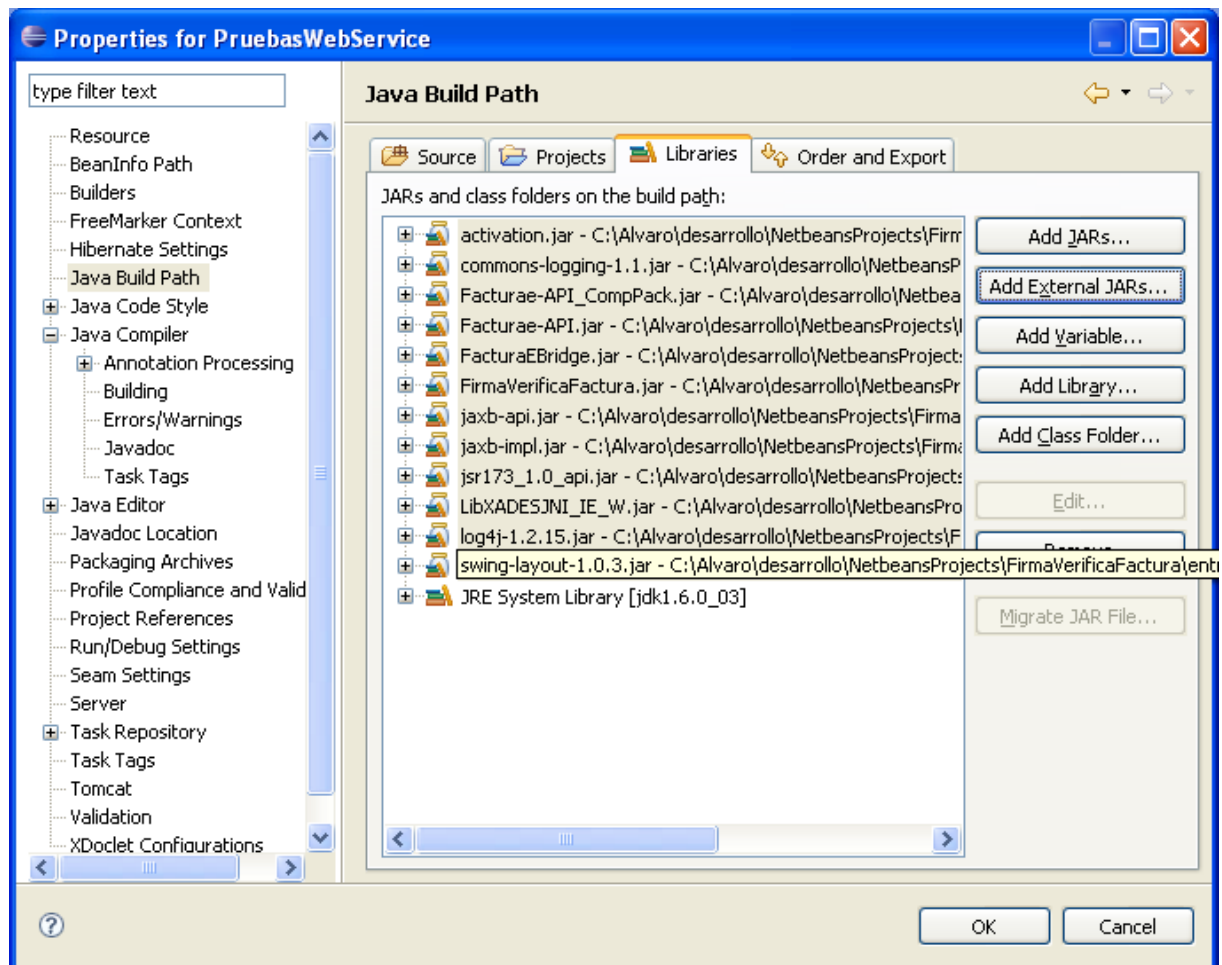
Tal como se señalaba al inicio de la sección para el desarrollo de la aplicación se va a utilizar NetBeans como IDE. Una de las ventajas de usar un IDE es que la configuración del entorno de desarrollo se puede realizar a través de una interfaz gráfica.

Para configurar las librerías del MITyC se hará desde la ventana de propiedades del proyecto tal como se muestra en la siguiente imagen:



NetBeans Classpath

En el caso de que el IDE que se utilizara no fuera NetBeans el proceso resulta muy parecido. En la siguiente imagen se muestra como realizar la configuración de las librerías desde Eclipse:



Eclipse Classpath

Codificación

La realización del proceso de diseño facilita mucho el trabajo a la hora de realizar la codificación. Una vez están definidas las clases y las operaciones que estas deben realizar el proceso de codificación se limita a añadir la lógica en las funciones.

De cara a que la codificación sea lo más clara posible, y por tanto hayan menos posibles errores se recomienda lo siguiente:

- Realizar una correcta gestión de las excepciones
 - Tratar las excepciones en lugar de ignorarlas dentro del manejador sin más
 - Transformar las excepciones genéricas a excepciones propias de la aplicación

- Usar constantes
 - Evitar incluir datos insertados directamente en el código
 - Agrupar las constantes en una sola clase en lugar de tenerlas desperdigadas por todas las clases
- Las funciones deben ser fáciles de entender
 - En lugar de hacer funciones muy grandes que realicen muchas operaciones es mejor tener varias funciones que hagan operaciones atómicas
 - Comentar el código
- Seguir una convención en la nomenclatura
 - Usar nombres de clases, métodos y variables que sean explicativos de para qué sirve cada elemento
 - Se puede usar cualquier convenio mientras se siga sistemáticamente. La siguiente tabla muestra una de las convenciones más habituales:

Regla	Convención	Ejemplo
Regla general para nombres con varias Palabras	Usar la convención “ <i>CamelCase</i> ”	VariableDeTresPalabras
Nombres de paquetes	Usar solo minúsculas	package lprg.p7;
Nombres de Clases	Se sigue la regla general	ColaLlamadas
	Usar sustantivos	
Nombre de Variables	Se sigue la regla general	int numeroDeFirmas;
	Primera letra en minúscula	
Constantes	En mayúsculas	static final String MENSAJE_ALERTA=“Hola Mundo”;
	Separadas por un guión bajo	

Nombres de métodos	Se sigue la regla general	void firmaFactura(String factura);
	La primera letra en minúscula	

Todo el código de la aplicación de ejemplo puede encontrarse en los materiales del caso práctico. Aunque el código es bastante sencillo a continuación se describe parte del mismo siguiendo el orden de ejecución de las operaciones que se describe en los casos de uso.

Operación Verificar aplicación

Esta operación se lanza desde el menú contextual de Windows a través del método *main* de la clase Test que invoca al método *validar* de la misma clase.

Validar la versión del Sistema Operativo

Se comprueba que el sistema operativo se corresponde con alguno de los definidos en el requisito RO-001 con el código que se muestra a continuación:

```
if (!(System.getProperty("os.name") != null) &&
    ((System.getProperty("os.name").equals("Windows XP")) ||
     (System.getProperty("os.name").equals("")) ||
     (System.getProperty("os.name").equals("Windows 7"))))
{
    mostrarResultado(<ERROR_MESSAGE>);
}
```

Validar la versión de la máquina virtual de Java

Se comprueba que la máquina virtual de Java se corresponde con alguna versión de las definidas en el requisito RO-003 con el código que se muestra a continuación:

```
if (!(System.getProperty("java.runtime.version") != null) &&
    ((System.getProperty("java.runtime.version").startsWith("1.6")) ||
     (System.getProperty("java.runtime.version").startsWith("1.5"))))
{
    mostrarResultado(<ERROR_MESSAGE>);
}
```

Validar las dependencias

Se comprueba que las librerías de Java que utiliza la aplicación son las correctas. Para ello se realiza un *hash* (resumen) de las librerías y se verifica que coincide con el *hash* que se tiene almacenado. Esto se realiza mediante el código que se muestra a continuación:

```
MessageDigest md = MessageDigest.getInstance("MD5");
FileInputStream libreria = null;
byte[] fichero = null;
byte[] hash = null;
int i = 0;
String sHash = "";
boolean correcto = true;
BASE64Encoder base64 = new BASE64Encoder();
//Recorrer todas las librerías
while (correcto && (i < librerias.length))
{
    //leer la librería
    libreria = new FileInputStream(librerias[i]);
    fichero = new byte[libreria.available()];
    libreria.read(fichero);
    libreria.close();
    //calcular el Hash
    md.update(fichero);
    hash = md.digest();
    sHash = base64.encode(hash);
    //Comparar el hash con el que se tiene guardado
    if (!sHash.equals(hashesLibrerias[i]))
    {
        correcto = false;
    }
    i++;
}
```

Informe de la validación

Se mostrará un informe sobre el funcionamiento de la aplicación. Se indicará si el sistema operativo en el que se está ejecutando la aplicación es el correcto, al igual que la máquina

virtual de Java con la que se ejecuta. Por último se indica si las librerías que está utilizando también son las correctas.

Esto se hace simplemente invocando el método *mostrarResultado* de la clase *FirmaVerifica* mediante el código que se puede ver a continuación:

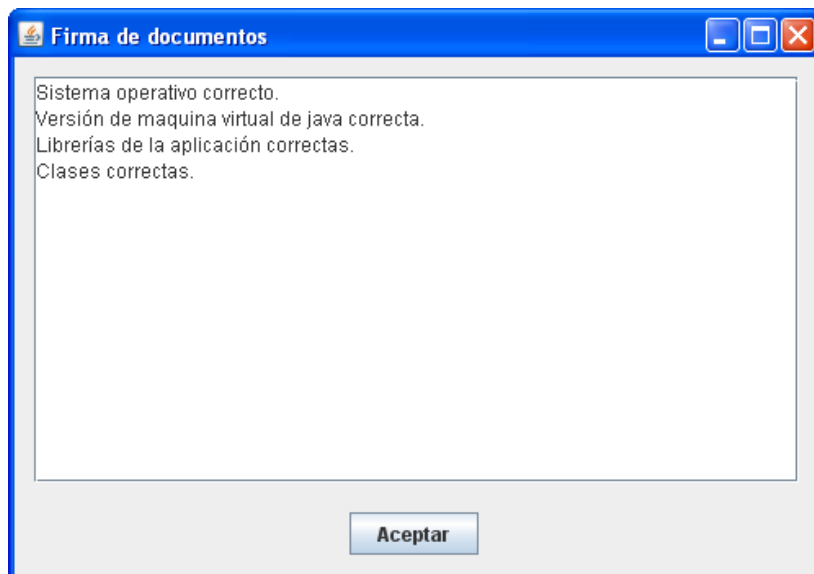
```
mostrarResultado(<RESULT_DETAILS>);
```

Donde <RESULT_DETAILS> recoge el resultado de las comprobaciones realizadas según se indica en los apartados anteriores.

El método muestra el resultado por pantalla mediante el siguiente código:

```
private void mostrarResultado(String textoMostrar)
{
    //Generamos la pantalla a mostrar
    MostrarFinalizacionProceso mostrar = new
    MostrarFinalizacionProceso();
    mostrar.setTitle("Firma de documentos");
    mostrar.setSize(500, 350);
    mostrar.setLocationRelativeTo(null);
    mostrar.setTexto(textoMostrar);
    mostrar.setVisible(true);
}
```

Si la validación ha sido correcta el resultado es que se muestra la siguiente ventana al usuario:



Pantalla Informe Funcionamiento Aplicación

Operación Firma de facturas

Esta operación se lanza desde el menú contextual de Windows a través del método *main* de la clase *Firmar* que invoca al método *firma* de la clase *FirmaVerifica* que realiza las operaciones que se describen a continuación.

Validar el esquema de la Factura

Lo primero que realiza la aplicación es comprobar que la factura cumple con el esquema definido por Facturae 3.1.

Para realizar esta operación se hace uso de las librerías del MITyC tal como se puede ver en el código adjunto:

```
java.io.File factura31 = new java.io.File(facture_a_firmar);
ValidatorUtil vu = ValidatorUtil.getInstance();
try {
    vu.validate(factura31, "3.1");
}
catch (Exception e) {
    mostrarResultado(<ERROR_MESSAGE>);
}
```

En caso de que la operación falle se recogerá la excepción y se mostrará el error en el informe mediante el método *mostrarResultado*.

Mostrar la Factura por pantalla

Se mostrarán por pantalla los datos de la factura a firmar. Se mostrarán todos los datos de forma que el usuario de la aplicación pueda ver exactamente toda la información que va a resultar firmada en la operación. **El usuario deberá expresar su consentimiento.**

Se lee la factura que se va a mostrar por pantalla y se muestra por pantalla tal como se muestra en el siguiente código:

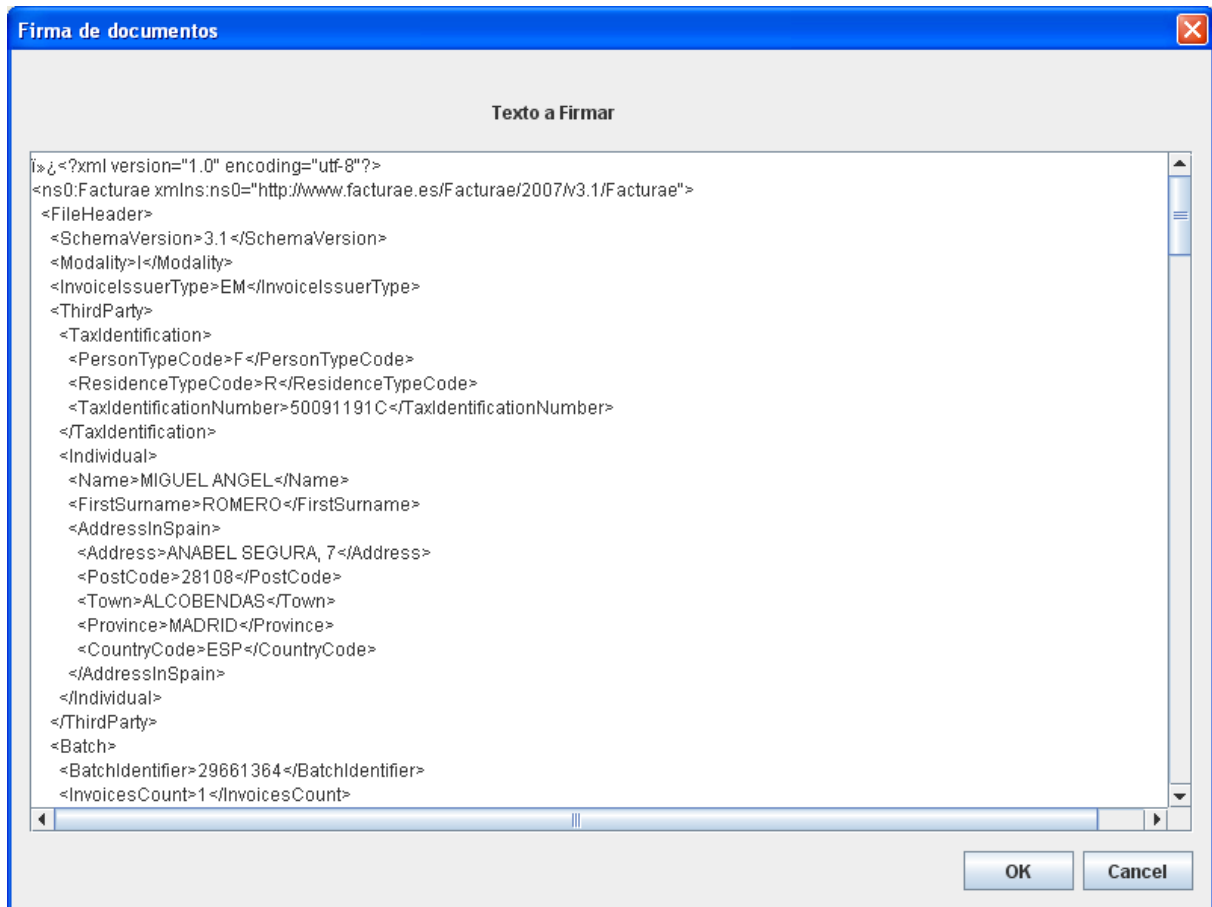
```
byte[] fichFirma = null;
// Leer fichero
try {
    FileInputStream f = new FileInputStream(factura31);
    fichFirma = new byte[f.available()];
    f.read(fichFirma);
    f.close();
}
catch (Exception e){
    mostrarResultado(<ERROR_MESSAGE>);
}
// Mostrar Fichero
mostrarFichero(new String(fichFirma))
```

El método *mostrarFichero* finaliza en mostrar un dialogo con los datos que el usuario va a firmar, haciendo uso de la clase *MostrarFicheroFirma* tal como se puede ver en el código adjunto:

```
//Generamos la pantalla a mostrar
    MostrarFicheroFirma mostrar = new MostrarFicheroFirma(null, true);
    mostrar.setTitle("Firma de facturas");
    mostrar.setSize(800, 600);
    mostrar.setLocationRelativeTo(null);
    mostrar.setTexto(textoMostrar);
    mostrar.setTitulo("Factura a firmar:");
    mostrar.setVisible(true);

    return mostrar.getReturnStatus() == mostrar.RET_OK;
```

Este código hace que aparezca la pantalla que se muestra en la siguiente imagen:



Documento a firmar

En el caso de que el usuario pulse el botón *Cancel* se deberá abortar el proceso y mostrar el resultado de la operación. Esto se controla mediante el código de retorno del método *mostrarFichero* mencionado anteriormente.

Mostrar Política de firma

Se mostrará al usuario de la aplicación la política de firma conforme a la que se va a realizar la operación. **El usuario deberá expresar su consentimiento pulsando en Ok.**

Esto se hace mediante el método *mostrarPolitica* de la clase *FirmaVerifica* mediante el código que se puede ver a continuación:

```
private boolean mostrarPolitica()
{
    byte[] politica = null;
    //Se lee la política a mostrar
```

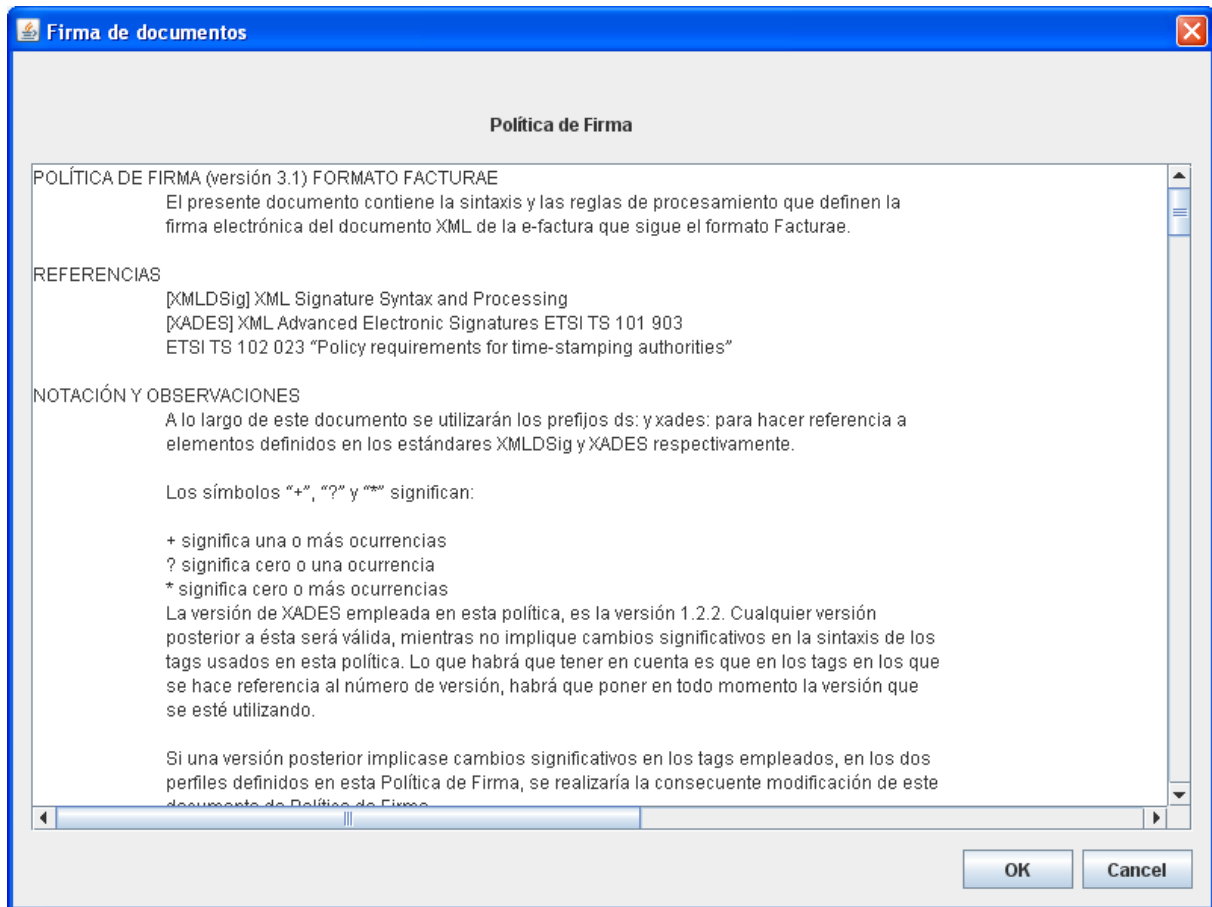


```
try {
    // La política está en el directorio de recursos
    FileInputStream fichPolitica = new
FileInputStream(<FICH_POLITICA>);
    politica = new byte[fichPolitica.available()];
    fichPolitica.read(politica);
    fichPolitica.close();
}
catch (Exception e) {
    mostrarResultado(<ERROR_MESSAGE>);
}
//Se genera la pantalla a mostrar
MostrarPoliticaFirma mostrar = new MostrarPoliticaFirma(null, true);
mostrar.setTitle("Firma de documentos");
mostrar.setSize(800, 600);
mostrar.setLocationRelativeTo(null);
mostrar.setTexto(new String(politica));
mostrar.setVisible(true);

return mostrar.getReturnStatus() == mostrar.RET_OK;
}
```

Como se puede apreciar en el código del método consiste en dos operaciones, una leer la política y otra mostrarla por pantalla haciendo uso de la clase `MostrarPoliticaFirma`.

El resultado es que se muestra la siguiente ventana al usuario:



Política de firma

En el caso de que el usuario pulse el botón *Cancel* se deberá abortar el proceso y mostrar el resultado de la operación. Esto se controla mediante el código de retorno del método *mostrarPolíticaFirma* mencionada anteriormente.

Aviso de LOPD

Se alertará al usuario del tratamiento al que serán sometidos sus datos. **El usuario deberá expresar su consentimiento pulsando en OK.**

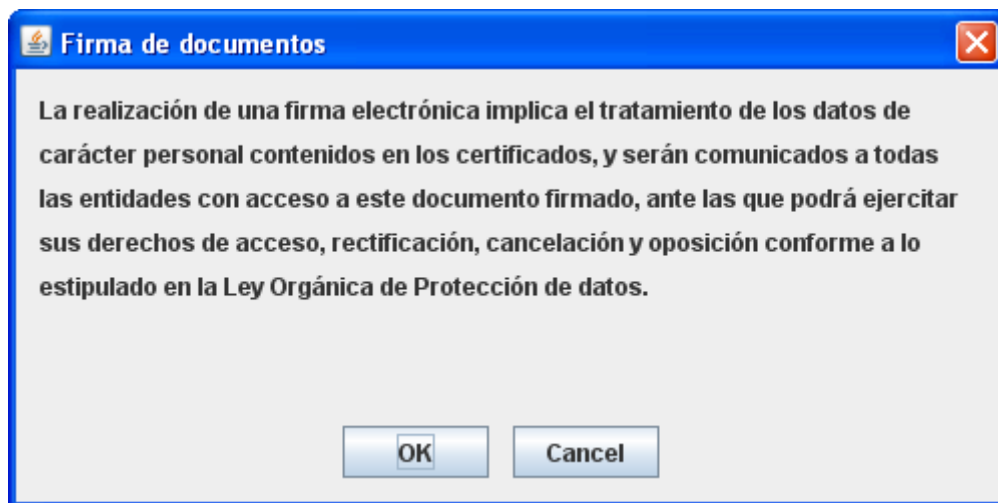
Esto se hace mediante el método *mostrarPeticiónFirma* de la clase *FirmaVerifica* mediante el código que se puede ver a continuación:

```
private boolean mostrarPeticiónFirma()
{
    //Generamos la pantalla a mostrar
    MostrarPeticiónFirma mostrar = new MostrarPeticiónFirma(null, true);
```

```
mostrar.setTitle("Firma de documentos");  
mostrar.setSize(500, 250);  
mostrar.setLocationRelativeTo(null);  
mostrar.setVisible(true);  
  
return mostrar.getReturnStatus() == mostrar.RET_OK;  
}
```

Como se puede apreciar en el código del método, consiste en mostrar el aviso por pantalla haciendo uso de la clase `mostrarPeticiónFirma`.

El resultado es que se muestra la siguiente ventana al usuario:



Petición de firma

En el caso de que el usuario pulse el botón *Cancel* se deberá abortar el proceso y mostrar el resultado de la operación. Esto se controla mediante el código de retorno del método `mostrarPeticiónFirma` mencionado anteriormente.

Selección de certificado

Se mostrarán los certificados disponibles para realizar la operación. Si el usuario tiene el DNle conectado se le mostrará la pantalla de petición de contraseña para acceder a los certificados del DNle para poderlo mostrar.

Este proceso hace uso de las librerías del MITyC dejando que sean estas junto con los drivers del DNI electrónico las que se encarguen de las llamadas a bajo nivel. Desde el punto de vista de la aplicación únicamente hay que realizar la siguiente llamada:


```
File fichSalida = es.mityc.facturae.utils.SignatureUtil.sign(factura31,  
pathSalida);
```

Esta llamada finaliza en una serie de operaciones a bajo nivel:





- El *middleware* del DNI electrónico pide al usuario que introduzca el pin del dispositivo.
- Se muestran los certificados del almacén de Microsoft Windows.
- El usuario selecciona el certificado que va a utilizar la firma.
 - En este punto el usuario tiene la posibilidad de comprobar la validez del certificado mediante la opción “Validación OCSP” que comprueba que el certificado no está revocado.
- El usuario debe seleccionar el certificado del DNI electrónico adecuado para firmar (contiene la palabra FIRMA en el nombre del certificado).
- El *middleware* del DNI electrónico pide al usuario que introduzca de nuevo el pin del dispositivo.
- El *middleware* del DNI muestra una alerta de seguridad indicando que se va a realizar una firma que el usuario debe aceptar.
- Se guarda la factura firmada en la ubicación *<pathsalida>* que debe ser el nombre original de la factura más la extensión *.xsig* tal como se recoge en RG-011.

Para realizar estas operaciones al usuario le aparecen las pantallas que se muestran a continuación:









Por favor, seleccione un certificado

 *Gestión de Facturación Electrónica 1.0*
Formatos factura-e soportados: 3.0 y 3.1



Certificados Disponibles



Emitido para	Emisor	Fecha de expir...	Tipo de certificado
 LevelBCAOK	LevelBCAOK	sáb, 13 feb 17:12...	X.509
 SigningUser...	LevelBCAOK	sáb, 5 mar 11:24:...	X.509
		mié, 7 oct 10:43:4...	X.509
		jue, 16 sep 11:51...	X.509

Datos del certificado seleccionado

Campo	Valor
 Versión	
 Número de serie	
 Algoritmo de firma	
 Emisor	
 Válido desde	
 Válido hasta	
 Sujeto	
 Clave Pública	

Detalles

 Continuar  Validacion OCSP

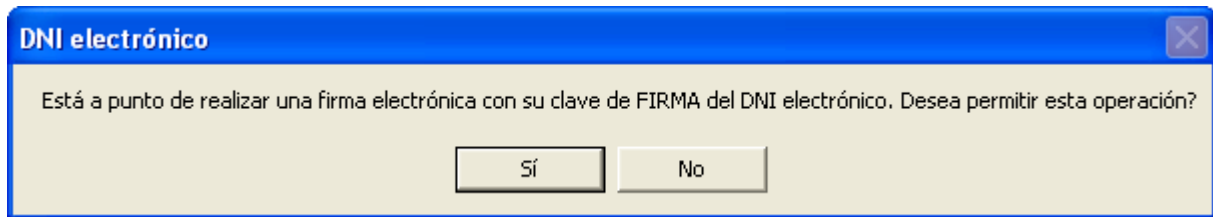
Pantalla de certificados

DNI electrónico

 **Introduzca el PIN**

XXXXXXXXXX

Pantalla petición pin



Pantalla de acceso a clave privada

En el caso de que se produzca algún error o que el usuario cancele en algún momento se deberá abortar el proceso y mostrar el resultado de la operación. Esto se controla mediante el código de retorno del método *sign* mencionado anteriormente.

Informe del resultado de la operación

Se presenta un informe con el resultado de la operación indicando si se ha firmado correctamente o se ha producido algún tipo de error.

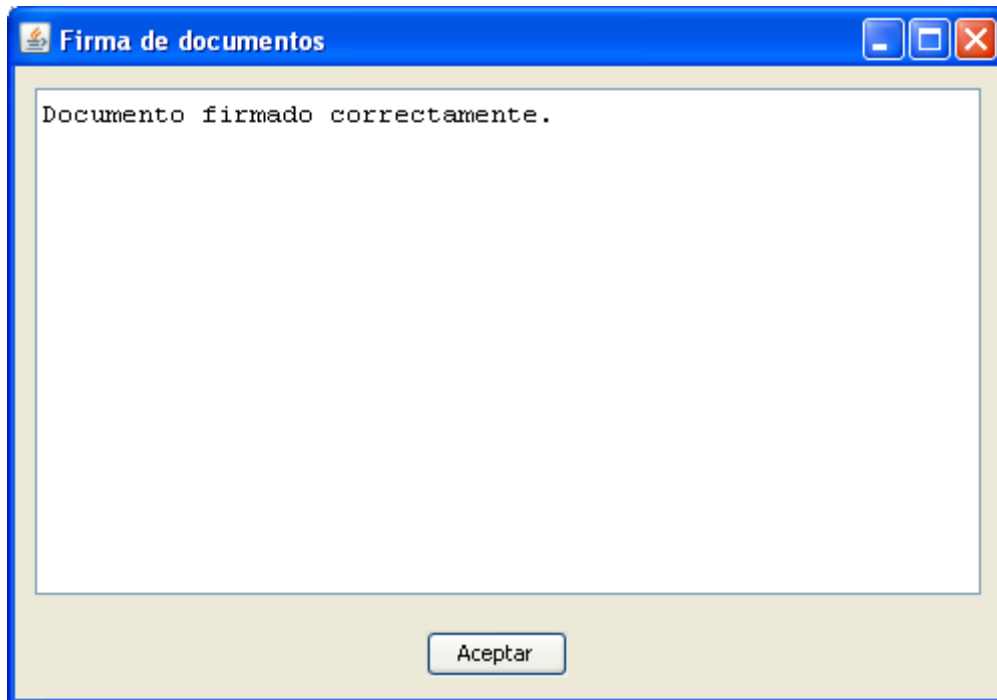
Esto se hace simplemente invocando el método *mostrarResultado* de la clase *FirmaVerifica* mediante el código que se puede ver a continuación:

```
mostrarResultado(<SUCCESS_MESSAGE>);
```

El método muestra el resultado por pantalla mediante el siguiente código:

```
private void mostrarResultado(String textoMostrar)
{
    //Generamos la pantalla a mostrar
    MostrarFinalizacionProceso mostrar = new
    MostrarFinalizacionProceso();
    mostrar.setTitle("Firma de documentos");
    mostrar.setSize(500, 350);
    mostrar.setLocationRelativeTo(null);
    mostrar.setTexto(textoMostrar);
    mostrar.setVisible(true);
}
```

El resultado es que se muestra la siguiente ventana al usuario:



Pantalla de resultado Firma

Operación Verificar factura

Esta operación se lanza desde el menú contextual de Windows a través del método main de la clase Verificar que invoca al método *verificacion* de la clase FirmaVerifica que realiza las operaciones que se describen a continuación.

Comprobar que el fichero existe

El primer paso consiste en verificar que el fichero que se quiere verificar existe. Esto se hace mediante el siguiente código:

```
//Comprobar que el fichero existe.  
java.io.File factura31 = new java.io.File(pathEntrada);  
boolean error = false;  
String resultado = "";  
if (!factura31.exists()) {  
    mostrarResultado(<ERROR_MESSAGE>);  
}
```

En caso de que el fichero no exista se aborta el proceso y se muestra la pantalla de resultado de la operación indicando el error.

Cargar el fichero

Esta operación se encarga de cargar el fichero en un objeto *Document* para documentos DOM XML y comprobar que la estructura del mismo es correcta.

Se puede ver el código a continuación:

```
//Preparar la factura para validar el formato
    DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
    dbf.setNamespaceAware(true);
    org.w3c.dom.Document document31 = null;
    try {
        javax.xml.parsers.DocumentBuilder parser =
dbf.newDocumentBuilder();
        document31 = parser.parse(factura31);
    }
    catch (Exception e) {
        mostrarResultado(<ERROR_MESSAGE>);
    }
```

En caso de que el fichero no se pueda cargar en un objeto *Document* se aborta el proceso y se muestra la pantalla de resultado de la operación indicando el error.

Mostrar Factura a validar

Se mostrarán por pantalla los datos de la factura a verificar. Se mostrarán todos los datos de forma que el Usuario de la Aplicación pueda ver exactamente toda la información que va a verificarse. **El usuario deberá expresar su consentimiento.**

Se lee la factura que se va a mostrar por pantalla y se muestra por pantalla tal como se muestra en el siguiente código:

```
byte[] fichFirma = null;
// Leer fichero
try {
    FileInputStream f = new FileInputStream(factura31);
    fichFirma = new byte[f.available()];
    f.read(fichFirma);
    f.close();
}
catch (Exception e){
```

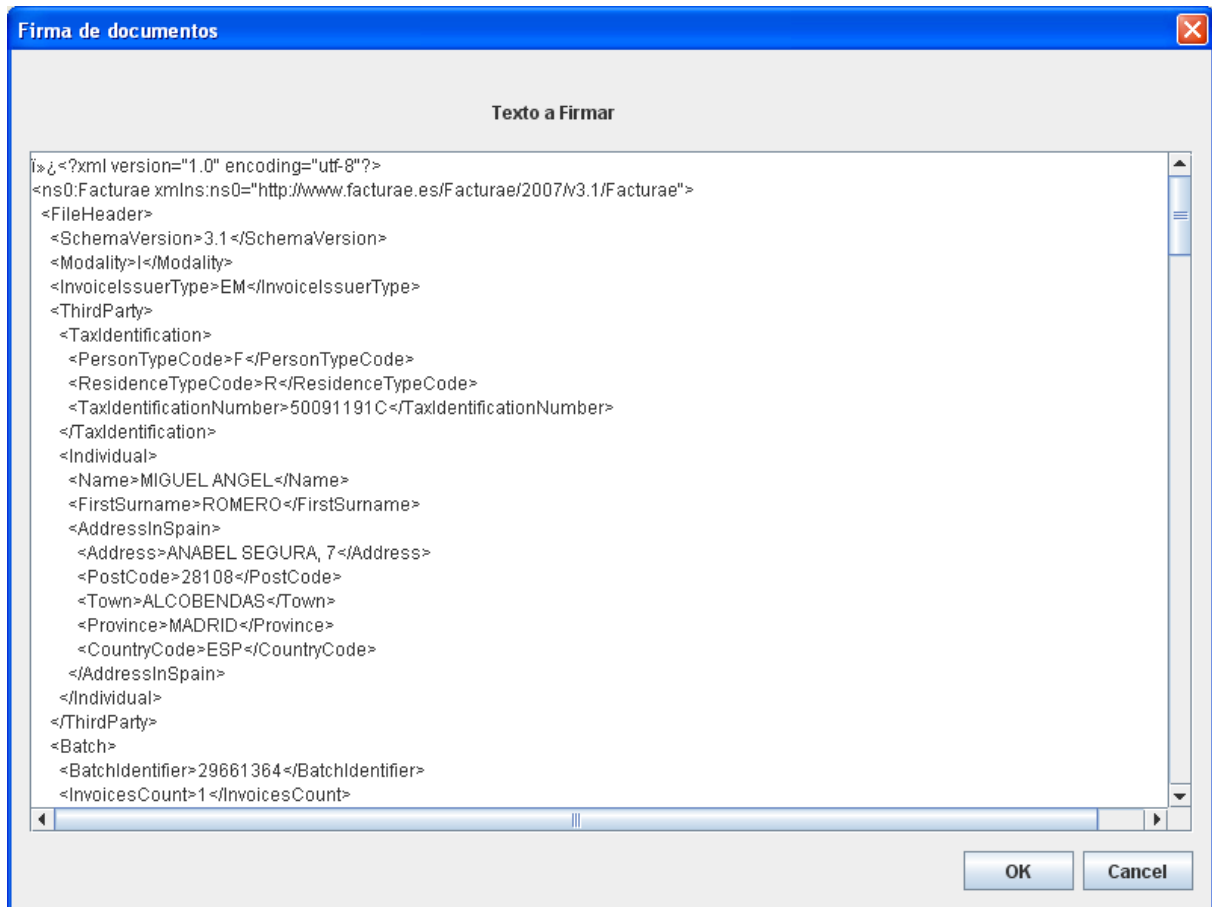


```
mostrarResultado(<ERROR_MESSAGE>);}  
  
// Mostrar Fichero  
mostrarFicheroValidar (new String(fichFirma))
```

El método *mostrarFicheroValidar* finaliza en mostrar un dialogo con los datos que el usuario va a firmar, haciendo uso de la clase *MostrarFicheroFirma* tal como se puede ver en el código adjunto:

```
private boolean mostrarFicheroValidar(String textoMostrar)  
{  
    //Generamos la pantalla a mostrar  
    MostrarFicheroFirma mostrar = new MostrarFicheroFirma(null, true);  
    mostrar.setTitle("Validación de Factura");  
    mostrar.setSize(800, 600);  
    mostrar.setLocationRelativeTo(null);  
    mostrar.setTexto(textoMostrar);  
    mostrar.setTitulo("Factura a verificar:");  
    mostrar.setVisible(true);  
  
    return mostrar.getReturnStatus() == mostrar.RET_OK;  
}
```

Este código hace que aparezca la pantalla que se muestra en la siguiente imagen:



Documento validado

En el caso de que el usuario pulse *Cancel* se deberá abortar el proceso y mostrar el resultado de la operación. Esto se controla mediante el código de retorno del método *mostrarFicheroValidar* mencionado anteriormente.

Mostrar Política de verificación

Se mostrará al usuario de la Aplicación la política de verificación conforme a la que se va a realizar la operación. **El usuario deberá expresar su consentimiento pulsando en Ok.**

Esto se hace mediante el método *mostrarPolitica* de la clase *FirmaVerifica* mediante el código que se puede ver a continuación:

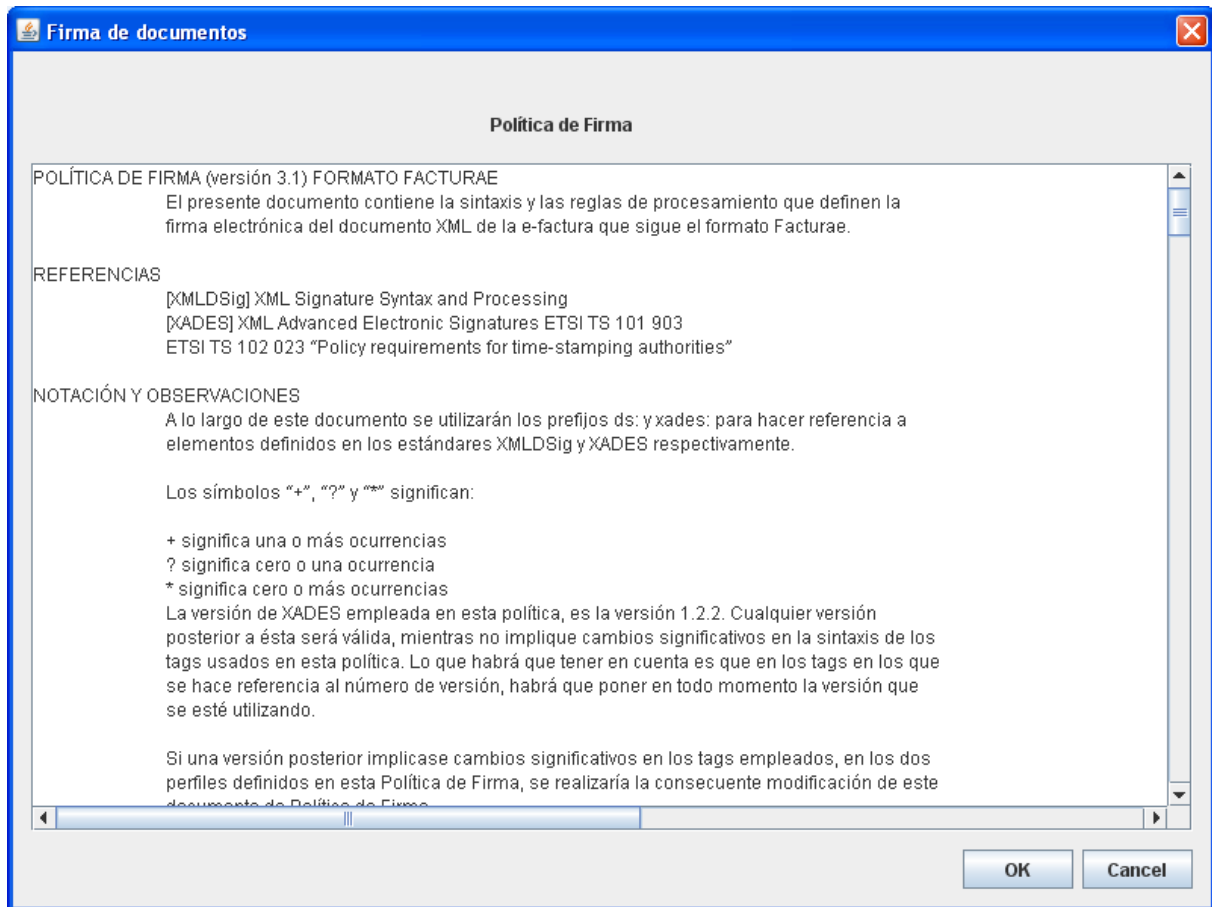
```
private boolean mostrarPolitica()
{
    byte[] politica = null;
    //Se lee la política a mostrar
```

```
try {
    // La política está en el directorio de recursos
    FileInputStream fichPolitica = new
FileInputStream(<FICH_POLITICA>);
    politica = new byte[fichPolitica.available()];
    fichPolitica.read(politica);
    fichPolitica.close();
}
catch (Exception e) {
    mostrarResultado(<ERROR_MESSAGE>);
}
//Se genera la pantalla a mostrar
MostrarPoliticaFirma mostrar = new MostrarPoliticaFirma(null, true);
mostrar.setTitle("Firma de documentos");
mostrar.setSize(800, 600);
mostrar.setLocationRelativeTo(null);
mostrar.setTexto(new String(politica));
mostrar.setVisible(true);

return mostrar.getReturnStatus() == mostrar.RET_OK;
}
```

Como se puede apreciar en el código, el método consiste en dos operaciones, una leer la política y otra mostrar la por pantalla haciendo uso de la clase `MostrarPoliticaFirma`.

El resultado es que se muestra la siguiente ventana al usuario:



Política de firma

En el caso de que el usuario pulse *Cancel* se deberá abortar el proceso y mostrar el resultado de la operación. Esto se controla mediante el código de retorno del método *mostrarPolíticaFirma* mencionado anteriormente.

Validación del esquema de la factura

Se debe comprobar que el esquema de la factura se corresponde con lo especificado en Facturae 3.1. Esto se realiza mediante el siguiente código del método *verificación* de la clase *FirmaVerifica*:

```
ValidatorUtil vu = ValidatorUtil.getInstance();  
try {  
    vu.validate(factura31, "3.1");  
}  
catch (Exception e) {  
    mostrarResultado(<ERROR_MESSAGE>);  
}
```

```
}
```

Como se puede ver en el código se usa la clase `ValidatorUtil` de las librerías del MITyC que está dentro del paquete `es.mityc.facturae.utils`.

En caso de que la validación no sea correcta se mostrará el error en la pantalla con el informe del resultado.

Validación contable de la factura

Se debe comprobar que la contabilidad de la factura es correcta. Esto se realiza mediante el siguiente código de método *verificación* de la clase `FirmaVerifica`:

```
ValidadorEfactura31 validator31 = new ValidadorEfactura31();  
try {  
    validator31.valid(document31);  
} catch (Exception e) {  
    mostrarResultado(<ERROR_MESSAGE>);  
}
```

Como se puede ver en el código se usa la clase `ValidadorEfactura31` de las librerías del MITyC que está dentro del paquete `es.mityc.facturae.utils`

En caso de que la validación no sea correcta se mostrará el error en la pantalla con el informe del resultado.

Verificar la firma de la factura

Se debe comprobar que la contabilidad de la factura es correcta. Esto se realiza mediante el siguiente código del método *verificación* de la clase `FirmaVerifica`:

```
Map m =  
es.mityc.facturae.utils.SignatureUtil.validateSignature(document31);
```

Como se puede ver en el código se usa la clase `SignatureUtil` de las librerías del MITyC que está dentro del paquete `es.mityc.facturae.utils`.

Si el resultado `m` es distinto de `null` significa que la firma es correcta.

Validación del certificado

Se debe comprobar que el certificado con el que se ha firmado la factura es válido. Esto se realiza mediante el código de la función *verificación* de la clase `FirmaVerifica`.

Esta operación se divide en varios pasos. El primero es extraer el certificado de la factura y comprobar que es criptográficamente correcto, es decir, que no está corrupto. Esto se hace con el código que se muestra a continuación:

```
//Recuperamos el certificado y lo validamos
X509Certificate cert = (X509Certificate) m.get("sign.certificate");
```

El siguiente paso es comprobar que el certificado no está caducado:

```
//Validamos la caducidad del certificado
//Comprobamos que ha sido emitido antes de la fecha actual
if (new Date().after(cert.getNotBefore())) {
    //Comprobamos que la fecha de caducidad es posterior a la actual
    if (cert.getNotAfter().after(new Date()));
}
else {
    mostrarResultado(<ERROR_MESSAGE>);
}
```

La última comprobación que se debe efectuar es que el certificado no está revocado. Para ello se utilizará el servicio de OCSP ofrecido por el DNI electrónico:

```
//Comprobamos que no está revocado
OCSPCliente ocs = new OCSPCliente("http://ocsp.dnie.es");
try {
    RespuestaOCSP res = ocs.validateCert(cert);
    if (res.getNroRespuesta() != 0) {
        mostrarResultado(<ERROR_MESSAGE>);
    }
}
```

Informe de la validación

Se mostrará un informe con el resultado de la operación realizada. El informe contendrá información acerca de:

- Validación de esquema
- Validación contable
- Validación de firma
- Validación de certificado

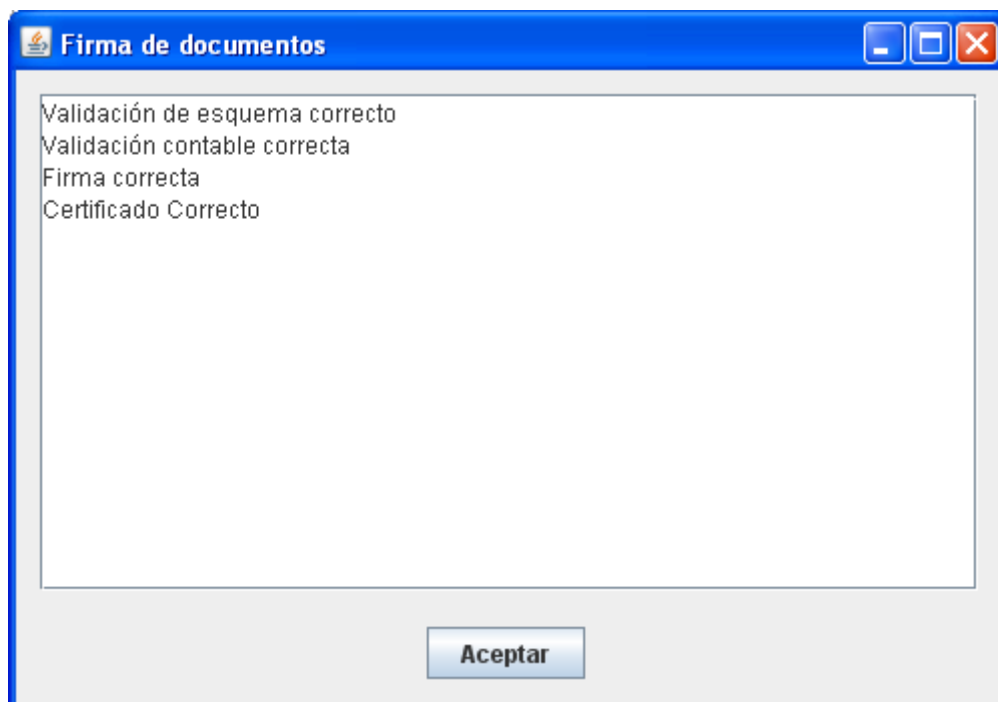
Esto se hace simplemente invocando el método *mostrarResultado* de la clase *FirmaVerifica* mediante el código que se puede ver a continuación:

```
mostrarResultado(<RESULT_DETAILS>);
```

El método muestra el resultado por pantalla mediante el siguiente código:

```
private void mostrarResultado(String textoMostrar)
{
    //Generamos la pantalla a mostrar
    MostrarFinalizacionProceso mostrar = new
MostrarFinalizacionProceso();
    mostrar.setTitle("Firma de documentos");
    mostrar.setSize(500, 350);
    mostrar.setLocationRelativeTo(null);
    mostrar.setTexto(textoMostrar);
    mostrar.setVisible(true);
}
```

Si la validación ha sido correcta el resultado es que se muestra la siguiente ventana al usuario:



Pantalla de resultado Verificación

Abreviaturas

- **Aplicación de creación y verificación de firma electrónica (SCVA)** – los medios utilizados para la creación y verificación de firma electrónica, sin incluir el SSCD.
- **Certificado** – es una garantía electrónica que une los datos de verificación de firma (SVD) a una persona y confirma la identidad de esa persona.
- **Datos a ser firmados (DTBS)** – son los datos electrónicos completos que hay que firmar (incluyendo tanto los atributos del mensaje del usuario como los de la firma).
- **Datos de verificación de autenticación (VAD)** – son datos de entrada de autenticación proporcionados por el usuario para la autenticación de su identidad bien sea demostrando el conocimiento o bien derivados de las características biométricas del usuario.
- **Datos de verificación de firma (SVD)** – son los datos, como códigos o claves criptográficas públicas, que se utilizan para de verificar una firma electrónica.
- **Dispositivo seguro de creación de firma (SSCD)** – es el software o hardware configurado para aplicar los datos de creación de firma (SCD).
- **Firmante** – es la persona que está en posesión de un dispositivo de creación de firma y que actúa en su propio nombre o en el de la entidad o persona física o jurídica a la que representa.