

DDT Cobro Modelo N28

Documento de Diseño Técnico

**Archivo: ERU Cobro Modelo N28.docx**

**Versión: 1**

**Equipo: Pasarelas de Pago**

**Fecha: 16/08/2018**

# Índice

[Índice 2](#_Toc522702213)

[1. Descripción del documento 3](#_Toc522702214)

[1.1. **Referencias** 3](#_Toc522702215)

[1.2. **Definiciones y acrónimos** 3](#_Toc522702216)

[1.3. **Control de modificaciones sobre el documento** 3](#_Toc522702217)

[2. Introducción 4](#_Toc522702218)

[2.1. Objetivo 4](#_Toc522702219)

[3. Arquitectura de la Aplicación 5](#_Toc522702220)

[3.1. Arquitectura general 5](#_Toc522702221)

[3.2. Arquitectura interna 5](#_Toc522702222)

[3.3. Versiones de las Librerías 7](#_Toc522702223)

[4. Organización Interna de los Módulos 8](#_Toc522702224)

[5. Módulo de Cobro N28 9](#_Toc522702225)

[5.1. Funcionalidad 9](#_Toc522702226)

[5.2. Diseño 9](#_Toc522702227)

[6. Módulo de Cifrado de Datos 10](#_Toc522702228)

[6.1. Funcionalidad 10](#_Toc522702229)

[6.2. Diseño 10](#_Toc522702230)

[7. Módulo de Integración 13](#_Toc522702231)

[7.1. Funcionalidad 13](#_Toc522702232)

[7.2. Diseño 13](#_Toc522702233)

[8. Ficheros de Configuración de la Aplicación 14](#_Toc522702234)

[9. Convenciones del Desarrollo 15](#_Toc522702235)

[9.1. Estructura del Proyecto 15](#_Toc522702236)

[9.2. Definición de los Paquetes de la Aplicación 15](#_Toc522702237)

[9.3. Nombres de Clases, Métodos y Variables 15](#_Toc522702238)

[9.4. Gestión de Excepciones 16](#_Toc522702239)

# Descripción del documento

Este documento refleja las necesidades de negocio relativas a un sistema de información (en adelante, sistema).

El documento trata de describir de forma rigurosa las necesidades del negocio y para ello trata evitar el manejo de conceptos ajenos a su ámbito en la definición de estas necesidades (requisitos de usuario).

El objetivo del documento consiste en formalizar los requisitos funcionales y de aquellas capacidades adicionales (características no funcionales) que ha de suministrar el sistema.

Este documento será utilizado por los Clientes de la Unidad de Organización y Sistemas para validar si el contenido del mismo coincide con sus necesidades manifestadas. También será utilizado por las personas de la Gerencia de Sistemas responsables de efectuar el desarrollo correspondiente.

* 1. **Referencias**

| **Título** | **Autor** | **Fecha** | **Editor** | **Fuentes** |
| --- | --- | --- | --- | --- |
| ERU Cobro Modelo N28.docx | Esteban Martín-Tembleque Poves | Junio 2018 |  |  |

Lista completa de los documentos referenciados en el ERU (Especificación de Requisitos de Usuario)

Identificación de cada documento por medio de un título, un número de informe, fecha y el organismo que lo ha publicado

Fuentes para conseguir los documentos referenciados

* 1. **Definiciones y acrónimos**

Definiciones de todos los términos, abreviaturas y siglas precisas para interpretar correctamente el documento.

Esta información puede proporcionarse haciendo referencia a uno o más apéndices o a otros documentos

| **Término** | **Definición** |
| --- | --- |
| CARM | Comunidad Autónoma de la Región de Murcia |
| CCT | Código de Control Tributario |
| MAC | Código de Control de verificación de campos |

* 1. **Control de modificaciones sobre el documento**

Identificación de las modificaciones hechas sobre el documento, indicando la fecha, la versión sobre la que se realizó el cambio, los capítulos afectados, una breve descripción del cambio y el autor o responsable de la modificación.

| **Fecha** | **Versión** | **Capítulo Afectado** | **Observaciones** | **Autor Modificación** |
| --- | --- | --- | --- | --- |
| 16/08/2018 | 1.0 | Todos | Versión inicial | Esteban Martín-Tembleque Poves |
| 20/08/2018 | 1.1 | Todos | Ajuste de Formato y modificación del Módulo de Cifrado | Esteban Martín-Tembleque Poves |
| 22/08/2018 | 1.2 | Todos | Inserción de Esquemas | Esteban Martín-Tembleque Poves |

# Introducción

## Objetivo

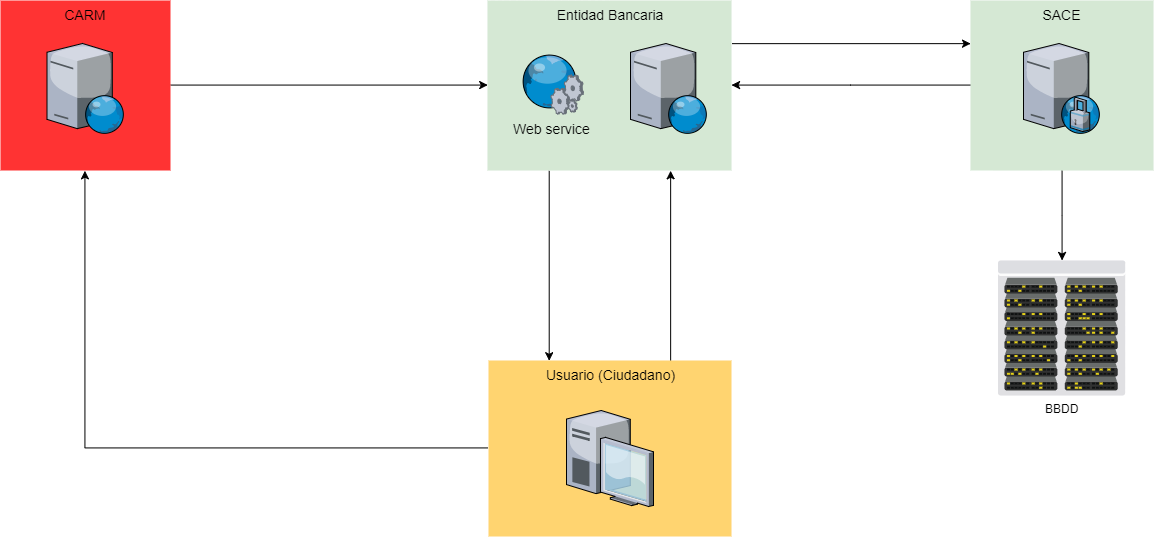
El objetivo fundamental del documento que se presenta a continuación es la realización de un diseño técnico pormenorizado de los procesos necesarios para la implementación de la aplicación del Cobro del Modelo N28 para la CARM (Comunidad Autónoma de la Región de Murcia).

Así, este documento tiene por objeto la recopilación del diseño técnico de la aplicación para guiar al equipo de desarrollo durante la fase de implementación.

# Arquitectura de la Aplicación

## Arquitectura general

El siguiente diagrama muestra la arquitectura general del sistema. De un lado están los usuarios que se conectan a la página de tributos de la CARM y desde allí son redirigidos a la página de introducción de datos de la pasarela de pago de la entidad bancaria. La Entidad Bancaria se comunica con el SACE para la gestión del pago y su inclusión en la BBDD. La interacción entre los actores y módulos de CARM y los módulos de la entidad bancaria se realizará a través de una aplicación web e invocación de servicios de negocio.



Los siguientes apartados abordan la arquitectura interna de los módulos de la entidad bancaria (aplicaciones y servicios) y su implementación.

## Arquitectura interna

Teniendo en cuenta lo anterior, a continuación, se muestra el diseño detallado de la aplicación. Los siguientes apartados, explican la función de cada una de las capas y los componentes que se usan para su implementación.

La siguiente tabla resume las características de cada una de las capas. Los apartados que vienen a continuación las describen con más detalle.

|  |  |  |
| --- | --- | --- |
| Capa | Descripción | Componentes |
| **Presentación** | Implementa las páginas de la interfaz gráfica de los usuarios. Es la cara visible de la aplicación. | Páginas JSP, plantillas HTML, CSS e iText. |
| **Negocio** | Implementa la lógica del funcionamiento de la aplicación, gestión de las transacciones, conversiones de tipos de datos. Es el corazón de la aplicación. | Objetos de negocio y servicios implementados con clases Java y que se integran con Spring. |
| **Integración** | Gestiona el acceso a todos los módulos externos: SACE | SNG de acceso a componentes de negocio Bankia. |

###### Capa de Presentación

Esta capa es la que se encarga de implementar la interacción con los usuarios finales a la hora de realizar el cobro. Se encarga de presentar tanto la pantalla de introducción de cuenta, como la pantalla de resultado de la operación y el justificante de pago en caso de ser solicitado. Se implementa con páginas JSP y código HTML y para la generación del PDF de justificante de pago utilizamos iText.



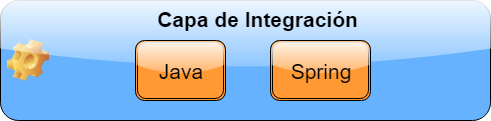
###### Capa de Negocio

La capa de negocio está compuesta objetos de negocio (controladores y servicios de Java). Los objetos de negocio son las entidades identificadas en la aplicación, como por ejemplo: las operaciones, los indicadores, los beneficiarios. Los servicios se encargan de gestionar las transacciones y coordinan el flujo de las operaciones que se inician con las acciones de los usuarios desde la capa de presentación.



###### Capa de Integración

Esta capa gestiona las invocaciones a SACE mediante SNG.



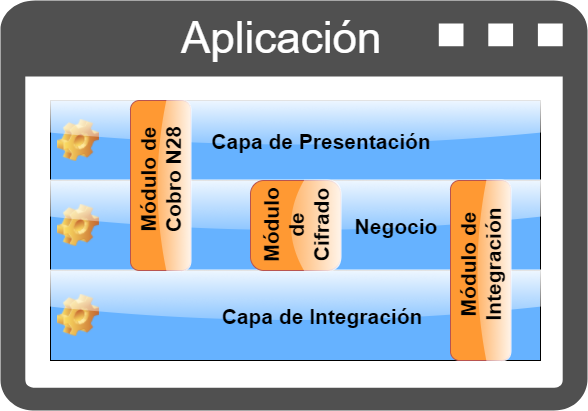
## Versiones de las Librerías

La siguiente tabla resume las versiones de las principales librerías utilizadas en la implementación de los módulos.

|  |  |
| --- | --- |
| Librería | Versión |
| <Nombre de la Librería> | <Versión> |

# Organización Interna de los Módulos

En el apartado anterior se presentaba la siguiente jerarquía de capas “lógicas”: Capa de Presentación, Capa de Negocio y Capa de Integración. Los módulos definidos son transversales y se organizan de la siguiente forma:



# Módulo de Cobro N28

## Funcionalidad

El módulo de cobro N28 implementa toda la gestión de cobro:

(INSERTAR TAREAS DE MÓDULO N28)

## Diseño

El siguiente diagrama de clases muestra las entidades de la capa de negocio que participan en este módulo:

(INSERTAR DIAGRAMA DE CLASES DE MÓDULO N28)

# Módulo de Cifrado de Datos

## Funcionalidad

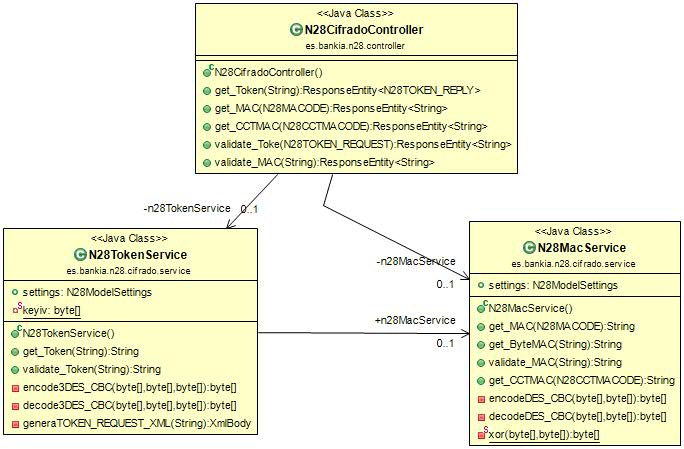
El módulo de cifrado implementa todas las operaciones de cifrado/descifrado de datos en la aplicación:

* Descifrado de TOKEN\_REQUEST enviado en la petición de la CARM.
* Validación de documento XML bien formado.
* Validación del MAC enviado con los datos del pago.
* Generación del MACODE necesario para la obtención del CCT.
* Cifrado de TOKEN\_REPLY para el envío de respuesta a la CARM.

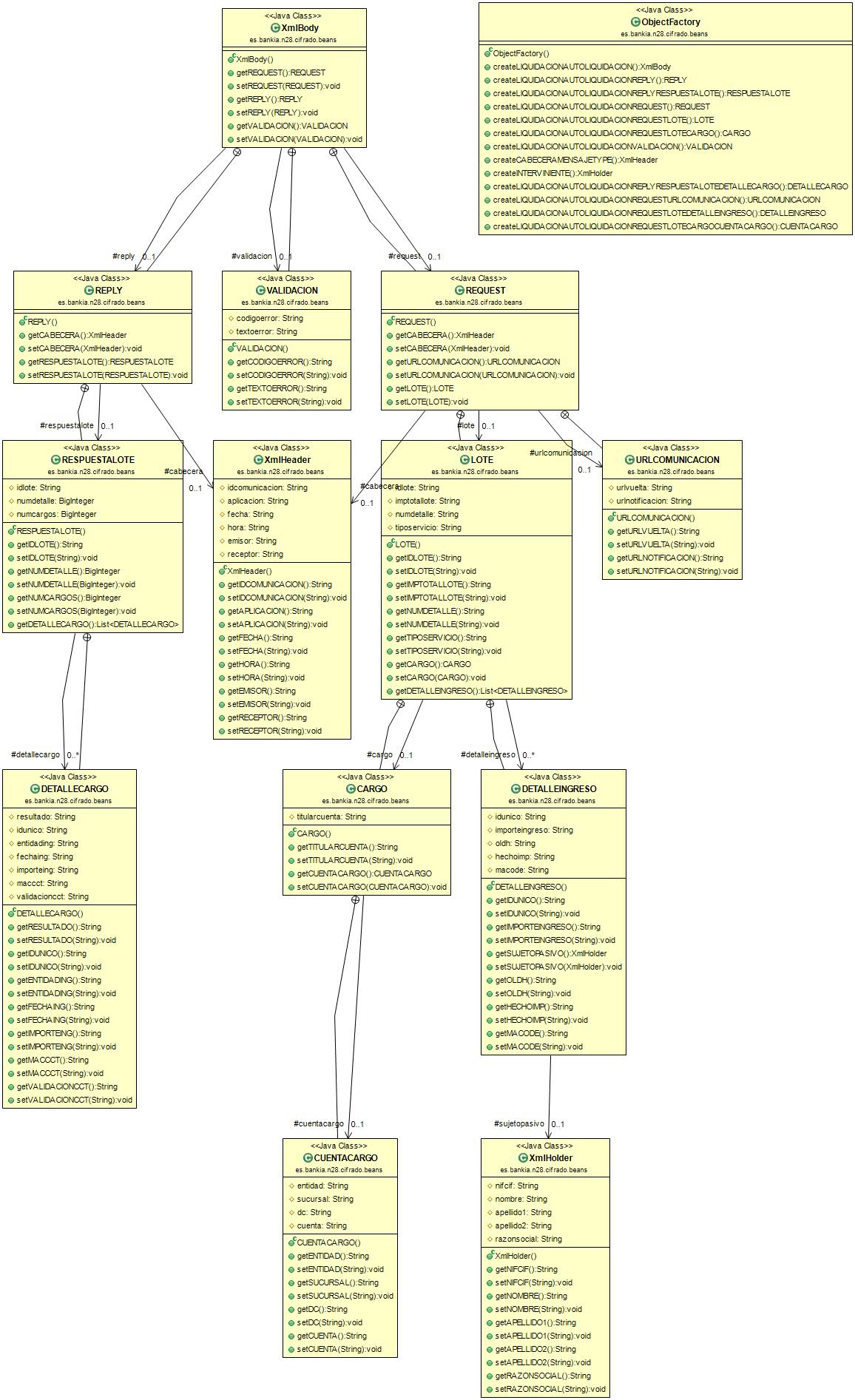
## Diseño

Los siguientes diagramas de clases muestra las entidades de la capa de negocio que participan en este módulo:

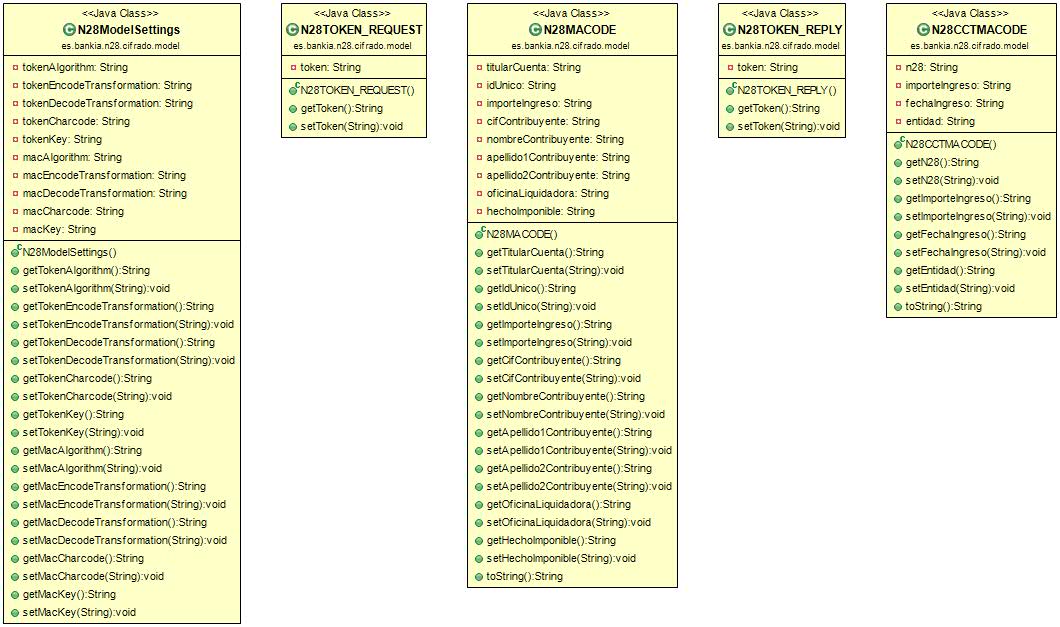
* Paquete **Controller**:



* Paquete **Beans**:



* Paquete **Model**:



# Módulo de Integración

## Funcionalidad

El módulo de integración implementa toda la gestión de llamadas a los servicios de SACE para la ejecución del cobre del N28:

(INSERTAR TAREAS DE MÓDULO INTEGRACIÓN)

## Diseño

El siguiente diagrama de clases muestra las entidades de la capa de negocio que participan en este módulo:

(INSERTAR DIAGRAMA DE CLASES DE MÓDULO INTEGRACIÓN)

# Ficheros de Configuración de la Aplicación

La siguiente tabla muestra todos los ficheros de configuración de la aplicación. Todos ellos van empaquetados en el fichero de despliegue.

|  |  |
| --- | --- |
| Nombre del fichero | Descripción |
| pom.xml | Este fichero es utilizado por Maven. Contiene la declaración de todas las dependencias (librerías). |
| application.properties | Este fichero contiene todas las definiciones de cifrado, claves de cifrado, juego de caracteres, etc.. |

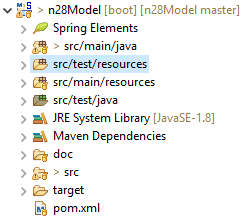
Parámetros de **application.properties**

|  |  |
| --- | --- |
| Nombre del fichero | Descripción |
| n28.token.algorithm | Es el algoritmo utilizado en la encriptación de los datos enviados por la CARM (TOKEN\_REQUEST) como en los datos de respuesta de la Aplicación de cobro del N28 (TOKEN\_REPLY). |
| n28.token.encode.transformation | Es el tipo de codificación utilizado en los tokens de comunicación entre aplicaciones. |
| n28.token.decode.transformation | Es el tipo de decodificación utilizado en los tokens de comunicación entre aplicaciones. |
| n28.token.charcode | Es la codificación de caracteres utilizado en el cifrado de tokens. |
| n28.token.key | Es la clave utilizada en el cifrado de tokens. |
| n28.mac.algorithm | Es el algoritmo utilizado en el cálculo del MACODE enviado por la CARM y en el cálculo del MACODE del CCT en la respuesta de la aplicación. |
| n28.mac.encode.transformation | Es el tipo de codificación utilizado en los MACODE. |
| n28.mac.decode.transformation | Es el tipo de decodificación utilizado en los MACODE. |
| n28.mac.charcode | Es la codificación de caracteres utilizado en el cifrado de MACODE. |
| n28.mac.key | Es la clave utilizada en el cifrado de MACODE. |

# Convenciones del Desarrollo

## Estructura del Proyecto

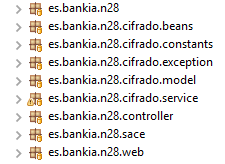
La aplicación se diseña siguiendo la definición del estándar J2EE. La estructura del proyecto en Eclipse está creada con el arquetipo de aplicaciones web de Maven, que se muestra a continuación.



## Definición de los Paquetes de la Aplicación

El proyecto tiene por un lado, las clases y recursos que corresponden a la implementación de la aplicación **(src/main)** y por otro las clases y recursos de las pruebas unitarias **(src/test)**. El fichero **pom.xml** contiene la definición de las dependencias, targets y demás parámetros utilizados por Maven.

Siguiendo las convenciones ya definidas en BANKIA, los paquetes del proyecto están organizados de la siguiente manera:



## Nombres de Clases, Métodos y Variables

En general, se adoptan todos los estándares y convenciones definidos para el lenguaje Java. Los nombres de las clases, métodos y variables deben ser siempre castellanos o castellanizados, evitando caracteres especiales como ñ, tildes, etc. Se admite el uso de prefijos o sufijos en inglés a efectos de homogeneizar la nomenclatura de clases, propiedades y métodos para adecuarlos a la tecnología específica que se ha decidido utilizar. Por ejemplo: usar **“get”**, **“set”** o **“is”** para los beans.

Los nombres de las clases corresponden a sustantivos y son descriptivos de la función dentro del modelo de objetos de la aplicación. En algunos casos también se añade un sufijo que especifica su papel dentro de la jerarquía de capas. Por ejemplo: **OperacionController**, **OperacionService**, etc.

Los nombres de los métodos deben indicar siempre el tipo de acción. Si se trata de nombres compuestos por varias palabras, la primera palabra indica la acción y el resto, el objeto(s) sobre lo(s) que se ejecuta la acción. La primera letra de cada palabra es una mayúscula excepto en la primera palabra. Por ejemplo: **setNombre(…)**, **procesaOperacionesPendientes()**, etc.

Los nombres de las variables de instancia y variables locales no pueden comenzar con **“\_”** o **“$”**. Deben ser breves y comunicar el significado de forma evidente. Se evitarán las variables de un carácter, con la sola excepción de las variables de las cláusulas **catch()** y los contadores temporales de los bucles (e, i, j, k, m). Los nombres de las variables no tendrán ningún prefijo que indique el tipo del objeto u otro clasificador. Los nombres de constantes van en mayúsculas y con un “\_” para separar las palabras (por ejemplo: **CANTIDAD\_MAXIMA**).

## Gestión de Excepciones

Las excepciones son errores que ocurren durante la ejecución de la aplicación y afectan el flujo normal de funcionamiento. Java ofrece una infraestructura robusta y basada en objetos para gestionar los diferentes escenarios de excepciones.

En general, cuando se produce una excepción, la aplicación debe tratarla, propagarla o generar una traza en el fichero de log. Si el bloque de código donde ocurre la excepción puede tratarla, entonces deberá hacerlo. Caso contrario, por normal general, las excepciones serán propagadas y tratadas en los niveles superiores.

Nunca deben usarse cláusulas **try/catch** vacías y siempre debe dejarse una traza en el fichero de log con toda la pila de ejecución correspondiente. Las trazas serán generadas con la librería Log4J y el fichero de configuración irá embarcado dentro de la misma aplicación (**.war**). En ningún caso se deben generar trazas con **System.out** o **System.err**.

Las clases de excepciones específicas de la aplicación seguirán las convenciones definidas previamente para el nombrado de clases y además deben terminar con el sufijo **“Exception”**. Por ejemplo: **“ValidacionUsuarioException”**.

Finalmente, debe tenerse en cuenta que las excepciones son caras de gestionar y que deben usarse razonablemente. Esto significa que debe analizarse si en algunos casos es más simple retornar un flag booleano para indicar que una operación se realizó con éxito o no en lugar de lanzar y propagar una excepción.