Librería Java

Guías de Integración

Versión: 1.0
Fecha: 24/03/2026
Referencia: RD.DCE.GUI.0002

Redsys, Servicios de Procesamiento, S.L. – c/ Francisco Sancha, 12 – 28034 Madrid (España)
www.redsys.es

Autorizaciones y control de versión

EMITIDO POR: Francisco Sáenz Ruiz	VALIDADO POR: Comercio Electrónico	APROBADO POR: Comercio Electrónico
Empresa: Redsys	Empresa: Redsys	Empresa: Redsys
Firma:	Firma:	Firma:
Fecha: 24/03/2026	Fecha: 24/03/2026	Fecha: 24/03/2026
Comentarios: La gestión de la documentación impresa es responsabilidad de la persona que la imprime. Las versiones impresas de las normas de seguridad no garantizan ser la última versión aprobada. Para consultar la última versión acceder a la base de datos de Alejandría. **Para el tratamiento de la información contenida en este documento se deberá seguir las pautas establecidas en la Normativa Redsys, y en particular en la norma *RS.RI.SEG.NOR.0003 NORMA DE CLASIFICACIÓN Y TRATAMIENTO DE LA INFORMACIÓN*** 		

Versión	Fecha	Afecta	Breve descripción del cambio
1.0	24/03/2026	TODO	Versión Inicial

ÍNDICE

1. Introducción a la librería

Estas librerías están diseñadas para simplificar al máximo la integración con el TPV Virtual de Redsys, eliminando la necesidad de implementar manualmente los procesos más técnicos: definición de parámetros, construcción de la petición, codificación, firma, diversificación de claves y generación del formato requerido por el TPV Virtual.

Además, la librería evita que el integrador tenga que conocer los endpoints o el entorno correspondiente a cada tipo de operación. El propio **API Key** ya incluye toda la información del comercio y del entorno, permitiendo que la librería seleccione automáticamente la URL adecuada (TEST o PROD) y aplique la configuración correspondiente.
Si, excepcionalmente, una integración requiere enviar la petición a un endpoint distinto del flujo estándar, la API también permite **sobrescribir la URL** directamente desde los parámetros antes de realizar la operación.

Gracias a este enfoque, el comercio solo necesita comprender el **flujo funcional de la integración**, sin encargarse de procesos criptográficos, del empaquetado de la solicitud o de la lógica para determinar endpoints.

Esta documentación no sustituye la documentación oficial de Redsys (manuales de integración por Redirección, integración REST u otros documentos específicos). Su objetivo es **complementarla**, ofreciendo una forma más sencilla y directa de aplicar los requisitos que Redsys establece utilizando esta librería.

1.1 Antes de empezar

1.1.1 Qué documentación necesitas

La integración oficial sigue existiendo y sigue siendo necesaria:

- **Manual de redirección y rest** o la documentación reflejada en la **web de desarrolladores** (https://pagosonline.redsys.es/desarrolladores-inicio/)
  Ej. envío de parámetros, retorno OK/KO, notificación online
- Glosario de códigos de error y respuesta (DS_RESPONSE)
  Tu librería no sustituye esta tabla oficial.
- Requisitos del entorno de pruebas y producción
  (URLs, aspectos de seguridad, PCI, etc.).

Esta librería te ayuda a cumplir esos requisitos, no a reemplazar la lectura.

1.1.2 Cómo obtengo mi API Key

El API Key de la librería puede obtenerse directamente desde el Portal, en el mismo apartado donde tradicionalmente se han gestionado las claves de firma del comercio.
Este API Key es **igual de sensible y crítico** que dichas claves: contiene la información necesaria para identificar al comercio, seleccionar el entorno correspondiente y realizar las firmas de las operaciones.

Por ese motivo, debe almacenarse siempre en un entorno seguro.
Compartirlo equivaldría a compartir la clave de firma de tu comercio, comprometiendo la integridad de las operaciones y la seguridad del TPV Virtual.

2. Guía de uso

2.1 Instalación de la librería

La librería está preparada para integrarse como un paquete **Maven**, aunque por el momento solo es posible obtenerla mediante **descarga directa desde la web de Redsys**.

En cualquier caso, para aquellos proyectos que no utilicen Maven, la instalación puede realizarse fácilmente instalando el .jar distribuido en el repositorio de tu proyecto e importándola en el código de la siguiente manera:

```
import es.redsys.lib.*;
```

2.2 Clases de la librería

La librería está compuesta por varias clases principales que el integrador deberá conocer para trabajar con ella de forma efectiva, junto con un conjunto de clases auxiliares que, si bien pueden utilizarse directamente, **no son necesarias para un flujo estándar ni forman parte del uso habitual recomendado**.

El integrador es libre de decidir qué partes de la librería emplea y cuáles prefiere implementar por su cuenta. No obstante, **se recomienda utilizarla de forma completa**, ya que los flujos que no estén íntegramente gestionados por la librería **no podrán contar con soporte**, al quedar fuera del comportamiento previsto y validado.

2.3 Clases principales

2.3.1 Merchant

En esta clase se inicializan todas las variables necesarias para identificar correctamente a qué comercio corresponde la operación.

Almacena el fuc, el terminal, la clave de firma y el entorno al que pertenece. Esta clase se inicializa con el API Key obtenido en el portal:

```
final Merchant merchant = Merchant.initWithApiKey(Secrets.REDSYS_API_KEY);
```

Aunque también se pueden asignar a posteriori cada uno de los parámetros de forma independiente inicializando la clase con su constructor vacío.

```
final Merchant merchant = new Merchant();
merchant.setFuc("999XXXXXX");
merchant.setTerminal("1");
merchant.setKc(clave);
merchant.setEnv(Environment.TEST.getValue());
```

2.3.2 Parameters

En esta clase se concentran la mayoría de los parámetros necesarios para prácticamente cualquier integración con el TPV Virtual.

Los valores se asignan con los setters correspondientes de cada uno de los parámetros, utilizando propiedades legibles y equivalentes a los campos estándar del TPV Virtual. Por ejemplo:

```
final Parameters params = new Parameters();
params.setAmount("999"); // En céntimos
params.setOrder("ABCD1234");
```

Si en algún caso la librería no contemplara un parámetro específico que necesite tu integración, la clase permite indicarlo utilizando su nombre literal tal como lo exige el TPV Virtual llamando a la función setParam. Por ejemplo, si order no estuviera definido como propiedad (equivalente a DS_MERCHANT_ORDER) podrías declararlo del siguiente modo:

```
params.setParam("DS_MERCHANT_ORDER", "ABCD1234");
```

Aunque DS_MERCHANT_ORDER no figure explícitamente como propiedad de la clase, **se incluirá correctamente en la serialización y en el empaquetado final enviado al TPV Virtual**, garantizando su funcionamiento dentro del flujo normal de la integración.

Si la librería es utilizada de forma completa, existe la posibilidad de **especificar un endpoint customizado** si existiera el caso de que las peticiones genéricamente recogidas en esta librería no contemplaran correctamente el destino de la petición que se desea realizar. Este parámetro es customUrl y se puede invocar de la siguiente forma:

```
params.setCustomUrl("https://url-de-destino.com/endpoint");
```

Las propiedades y sus equivalencias son las siguientes:

Propiedad (librería)	Equivalencia IN (DS_MERCHANT_*)	Equivalencia OUT (DS_*)
amount	DS_MERCHANT_AMOUNT	DS_AMOUNT

authorisationCode	DS_MERCHANT_AUTHORISATIONCODE	DS_AUTHORISATIONCODE
cofTxnId	DS_MERCHANT_COF_TXNID	DS_MERCHANT_COF_TXNID
consumerLanguage	DS_MERCHANT_CONSUMERLANGUAGE	DS_CONSUMERLANGUAGE
currency	DS_MERCHANT_CURRENCY	DS_CURRENCY
emv3ds	DS_MERCHANT_EMV3DS	DS_EMV3DS
tokenizedCard	DS_MERCHANT_IDENTIFIER	DS_MERCHANT_IDENTIFIER
fuc	DS_MERCHANT_MERCHANTCODE	DS_MERCHANTCODE
merchantData	DS_MERCHANTDATA	DS_MERCHANTDATA
order	DS_MERCHANT_ORDER	DS_ORDER
terminal	DS_MERCHANT_TERMINAL	DS_TERMINAL
transactionType	DS_MERCHANT_TRANSACTIONTYPE	DS_TRANSACTIONTYPE
dcc	DS_MERCHANT_DCC	DS_DCC
dccFlag	DS_MERCHANT_DCC * (Cuando se iguala a un String)	—
dccDetails	DS_MERCHANT_DCC * (Cuando se iguala a un Map)	—
excepSca	DS_MERCHANT_EXCEP_SCA	DS_EXCEP_SCA
rts	DS_MERCHANT_RTS	DS_RTS / rts
cardExpiryDate	DS_MERCHANT_EXPIRYDATE	DS_EXPIRYDATE
cardNumber	DS_MERCHANT_PAN	DS_CARD_NUMBER
bizumData	DS_MERCHANT_BIZUM_DATA	DS_BIZUM_DATA
cofIni	DS_MERCHANT_COF_INI	—
cofType	DS_MERCHANT_COF_TYPE	—
cvv2	DS_MERCHANT_CVV2	—
directPayment	DS_MERCHANT_DIRECTPAYMENT	—

group	DS_MERCHANT_GROUP	—
idOper	DS_MERCHANT_IDOPER	—
merchantName	DS_MERCHANT_MERCHANTNAME	—
merchantUrl	DS_MERCHANT_MERCHANTURL	—
payMethods	DS_MERCHANT_PAYMETHODS	—
productDescription	DS_MERCHANT_PRODUCTDESCRIPTION	—
taxReference	DS_MERCHANT_TAX_REFERENCE	—
cardholderName	DS_MERCHANT_TITULAR	—
transactionDate	DS_MERCHANT_TRANSACTIONDATE	—
urlKo	DS_MERCHANT_URLKO	—
urlOk	DS_MERCHANT_URLOK	—
tokenData	DS_MERCHANT_TOKENDATA	—
shippingAddressPyp	DS_MERCHANT_SHIPPINGADDRESSPYP	—
merchantDescriptor	DS_MERCHANT_MERCHANTDESCRIPTOR	—
persoCode	DS_MERCHANT_PERSOCODE	—
matchingData	DS_MERCHANT_MATCHINGDATA	—
externalMpi	DS_MERCHANT_MPIEXTERNAL	—
customerMobile	DS_MERCHANT_CUSTOMER_MOBILE	—
customerMail	DS_MERCHANT_CUSTOMER_MAIL	—
p2fExpiryDate	DS_MERCHANT_P2F_EXPIRYDATE	—
customerSmsText	DS_MERCHANT_CUSTOMER_SMS_TEXT	—
p2fXmlData	DS_MERCHANT_P2F_XMLDATA	—
financing	DS_MERCHANT_FINANCING	—
ota	DS_MERCHANT_OTA	—

clientIp	DS_MERCHANT_CLIENTIP	—
xPayType	DS_XPAYTYPE	—
xPayOrigin	DS_XPAYORIGEN	—
xPayDecodedData	DS_XPAYDECODEDDATA	—
xPayData	DS_XPAYDATA	—
bizumMobileNumber	DS_MERCHANT_BIZUM_MOBILENUMBER	—
dccAmount	—	DS_AMOUNT_DCC
amountEuro	—	DS_AMOUNT_EURO
cardBrand	—	DS_CARD_BRAND
cardCountry	—	DS_CARD_COUNTRY
cardPsd2	—	DS_CARD_PSD2
cardType	—	DS_CARD_TYPE
dccCurrency	—	DS_CURRENCY_DCC
date	—	DS_DATE
errorCode	—	DS_ERRORCODE
hour	—	DS_HOUR
processedPayMethod	—	DS_PROCESSEDPAYMETHOD
codResponse	—	DS_RESPONSE
securePayment	—	DS_SECUREPAYMENT
signature	—	DS_SIGNATURE
p2fUrl	—	DS_URLPAGO2FASES
code	—	CODIGO
responseDescription	—	DS_RESPONSE_DESCRIPTION
cardTypology	—	DS_CARD_TYPOLOGY
cardIssuer	—	DS_CARD_ISSUER

cardTokenIndicator	—	DS_CARD_TOKEN_INDICATOR
cardTreatment	—	DS_CARD_TREATMENT
cardProductType	—	DS_CARD_PRODUCTTYPE
bizumIdStatus	—	DS_BIZUMIDSTATUS
bizumIdResponse	—	DS_BIZUMIDRESPONSE
bizumIdDescription	—	BD_BIZUMIDDESCRIPTION

2.3.3 PaymentMethod

Clase sencilla que contiene una lista de métodos de pago admitidos por el TPV Virtual. El nombre de las variables son el nombre común atribuido al método de pago.

Por ejemplo, bizum está representado de la siguiente forma:

```
BIZUM("z");
```

Y podrá invocarse de la siguiente forma:

```
params.setPayMethods(PaymentMethod.BIZUM);
```

De nuevo, tal cual ocurre con las monedas, si hay algún método de pago soportado que no se encuentre en esta lista y se quiere operar con él, es tan fácil como introducir en el parámetro payMethods el código identificativo del método de pago deseado para poder enviarlo al TPV Virtual. Por ejemplo:

```
params.setPayMethods("z");
```

2.3.4 GenerateRedirectForm

Clase que te ayuda a generar un formulario para realizar la integración por redirección. Devuelve un html listo para poder renderizar.

Sus funciones reciben la variable inicializada de la clase Merchant, y la variable completada con los datos del pago de la clase Parameters.

Cada una de estas funciones llaman a un tipo de operación distinto, como por ejemplo, authentication(), que es la utilizada para realizar una operación de autenticación (Comprobar que el cliente tiene el dinero disponible).

Las funciones disponibles son las siguientes:

Función	Descripción	Equivalencia DS_TRANSACTION_TYPE
authorisation()	Realiza una operación de autorización, también llamado Pago.	0
preauthorisation()	Realiza una preautorización, reteniendo el dinero al cliente hasta el momento de confirmar el pago.	1
preauthorisationConfirmation()	Confirma la preautorización y termina el pago, haciendo definitivo el cobro de la cantidad retenida en la preautorización.	2
refund()	Devolución de una operación.	3
authentication()	Realiza una verificación de la tarjeta del cliente que va a realizar un pago en el comercio.	7
authenticationConfirmation()	Confirma la operación previamente enviada al verificar la tarjeta y termina el pago, haciendo el cobro de la cantidad solicitada.	8
preauthorisationPartialConfirmation()	Confirma de forma parcial preautorización y termina el pago, haciendo definitivo el cobro de una parte de la cantidad retenida en la preautorización.	48
challenge()	Redirige al cliente a la autenticación necesaria para realizar el pago	No aplica
generic()	Redirección genérica al TPV Virtual, por defecto realiza una Autorización. Si no se quiere indicar ninguna operación, es necesario llamarla con null en el tercer parámetro transactionType.	No aplica

Si por algún caso fuera necesario realizar una operación que tenga un DS_TRANSACTION_TYPE diferente a los recogidos en la tabla, se podrá utilizar la función generic() indicando el DS_TRANSACTION_TYPE en el tercer parámetro, o en su defecto inicializando el parámetro transactionType de la clase Parameters al tipo de transacción deseando antes de realizar la llamada de redirección.

Un ejemplo podría ser el siguiente:

```
GenerateRedirectForm.generic(merchant, params, TransactionType.AUTHENTICATION);
```

O también:

```
params.setTransactionType(TransactionType.AUTHENTICATION);
GenerateRedirectForm.generic(merchant, params);
```

O incluso de la siguiente forma si el transactionType no está definido en la clase TransactionType (ver más adelante en el punto *¿Qué ocurre si el tipo de operación que quiero realizar no está contemplada en la librería*?):

```
params.setTransactionType("7");
GenerateRedirectForm.generic(merchant, params);
```

2.3.5 Rest

Esta clase se utiliza para realizar operaciones a través de peticiones Rest. Sus funciones reciben la variable inicializada de la clase Merchant, y la variable completada con los datos del pago de la clase Parameters.

También todas estas funciones se llaman de forma estática.

Las funciones disponibles son las siguientes:

Función	Descripción	Equivalencia DS_TRANSACTION_TYPE
authorisationInit()	Consulta las opciones de verificación de la tarjeta para una operación de autorización o Pago.	0
authorisation()	Realiza una operación de autorización, también llamado Pago.	0
preauthorisationInit()	Consulta las opciones de verificación de la tarjeta para una operación de preautorización.	1
preauthorisation()	Realiza una preautorización, reteniendo el dinero al cliente hasta el momento de confirmar el pago.	1
preauthorisationConfirmation()	Confirma la preautorización y termina el pago, haciendo definitivo el cobro de la cantidad retenida en la preautorización.	2
refund()	Devolución de una operación.	3

authenticationInit()	Consulta las opciones de verificación de la tarjeta para una operación de autenticación.	7
authentication()	Realiza una autenticación del cliente que va a realizar un pago en el comercio.	7
authenticationConfirmation()	Confirma la operación previamente autenticada y termina el pago, haciendo definitivo el cobro de la cantidad solicitada en la autenticación.	8
preauthorisationPartialConfirmation()	Confirma de forma parcial preautorización y termina el pago, haciendo definitivo el cobro de una parte de la cantidad retenida en la preautorización.	48
challengeResponse()	Comunica al TPV Virtual el resultado de la verificación realizada a la tarjeta del cliente.	No aplica
genericInit()	Consulta las opciones de verificación de la tarjeta para una operación genérica.	No aplica
generic()	Petición Rest genérica al TPV Virtual, al igual que en redirección, por defecto realiza una Autorización. Si no se quiere indicar ninguna operación, es necesario llamarla con null en el tercer parámetro transactionType.	No aplica

Al igual que en el apartado de generar formulario de redirección, si por algún caso fuera necesario realizar una operación que tenga un DS_TRANSACTION_TYPE diferente a los recogidos en la tabla, se podrá utilizar la función genericInit() o generic() indicando el DS_TRANSACTION_TYPE en el tercer parámetro, o en su defecto inicializando el parámetro transactionType de la clase Parameters al tipo de transacción deseando antes de realizar la llamada.

Un ejemplo podría ser el siguiente:

```
initResponse = Rest.genericInit(merchant, params, "7");
```

2.3.6 Response

Wrapper utilizado por la clase Rest para devolver las respuestas del TPV Virtual.

Los parámetros que contiene esta clase son los siguientes:

Parámetro	Descripción	Tipo/s
httpCode	Código HTTP devuelto por el TPV Virtual	Integer
headers	Headers devueltos por la petición al TPV Virtual	Map
data	Contenido de la respuesta recibido desde el TPV Virtual, ya interpretado y normalizado por la clase Parameters	Parameters
raw	Cuerpo de la petición sin procesar recibido desde el TPV Virtual	String
errorCode	Código de error devuelto por el TPV Virtual	String
description	Descripción del estado de la operación (si lo hubiera)	String

2.4 Clases secundarias

2.4.1 Utils

Clase con diferentes funciones de ayuda que podrían ser de utilidad para el integrador. Por norma general no serían necesarias.

Función	Descripción
randomString()	Genera un string aleatorio, por defecto de 12 caracteres de longitud.
getSystemNumLang()	Devuelve el idioma del sistema en el que se está ejecutando la librería en formato numérico, el que recibe el TPV Virtual para mostrar el idioma del cliente correctamente. Por defecto: Inglés (en, 2)
getNumLangFromLang()	Recibe el idioma en string de 2 caracteres (pe: 'es') y devuelve el equivalente numérico para el TPV Virtual. Solo idiomas soportados. Por defecto: Inglés (en, 2)
encodeB64UrlString()	Encode Base64 URL
encodeB64UrlSafeString()	Encode Base64 URL Safe
encodeB64UrlSafeString()	Decode Base64 URL Safe

2.4.2 TransactionType

Clase que contiene una lista de tipos de transacción admitidos por el TPV Virtual.

Si hubiera algún tipo de transacción soportada que no se encuentre en esta lista y se quiere operar con ella, se puede, como ya se explicó en el punto de Redirección y Rest, incluir un literal en los parámetros de la clase Parameters de la siguiente forma con el tipo de operación que se desea realizar:

```
params.setTransactionType("7");
```

2.4.3 Currency

Clase sencilla que contiene la lista de monedas admitidas por el TPV Virtual. El nombre de las variables son el código de la moneda en formato ISO 4217, y el valor de la variable es el código numérico de esta misma moneda, que es lo que necesita el SIS para poder identificar la moneda correctamente.

Por ejemplo, el euro y el dólar están representados de la siguiente forma:

```
EUR("EUR", "978", "Euro"),
USD("USD", "840", "US Dollar"),
```

Y podrán utilizarse de la siguiente forma:

```
params.setCurrency(Currency.EUR);
```

Si hay alguna moneda soportada que no se encuentre en esta lista y se quiere operar con ella, es tan fácil como introducir en el parámetro currency el número de la moneda deseada para poder enviarla al TPV Virtual. Por ejemplo:

```
params.setCurrency("978");
```

Si la librería es utilizada de forma completa y **la operación se desea realizar en euros**, **no es necesario especificarlo** en los parámetros **ya que se tomará como moneda por defecto.**

2.4.4 Endpoints

Contiene los endpoints principales para poder operar con el TPV Virtual. Es una clase auxiliar que el integrador no tiene por qué utilizar en ninguna parte del proceso.

2.4.5 RequestData

Clase auxiliar que contiene utilidades para ayudarte a generar la petición al TPV Virtual en el formato correspondiente. No es necesario utilizarla si se va a usar para la integración todas las clases principales.

2.4.6 Signature

Ayuda a generar todos los tipos de firma utilizados en el TPV Virtual. Por defecto la firma se genera con la versión HMAC_SHA512_V2.

2.4.7 Version

Clase que contiene la versión actual de la librería. No debería ser modificada bajo ningún concepto.

2.4.8 Environment

Contiene el conjunto de entornos posibles a los que se puede apuntar en el TPV Virtual. Para la correcta integración del TPV Virtual solamente son necesarios TEST y PROD.

2.5 Integración

Todos estos ejemplos deben ser debidamente adaptados al framework o librerías en uso por el integrador, ya que no son directamente traducibles a cualquier proyecto.

2.5.1 Integración por Redirección Simple

Para poder realizar, por ejemplo, un pago lo más simple posible de una Autorización (tipo 0) por redirección, el código podría ser el siguiente:

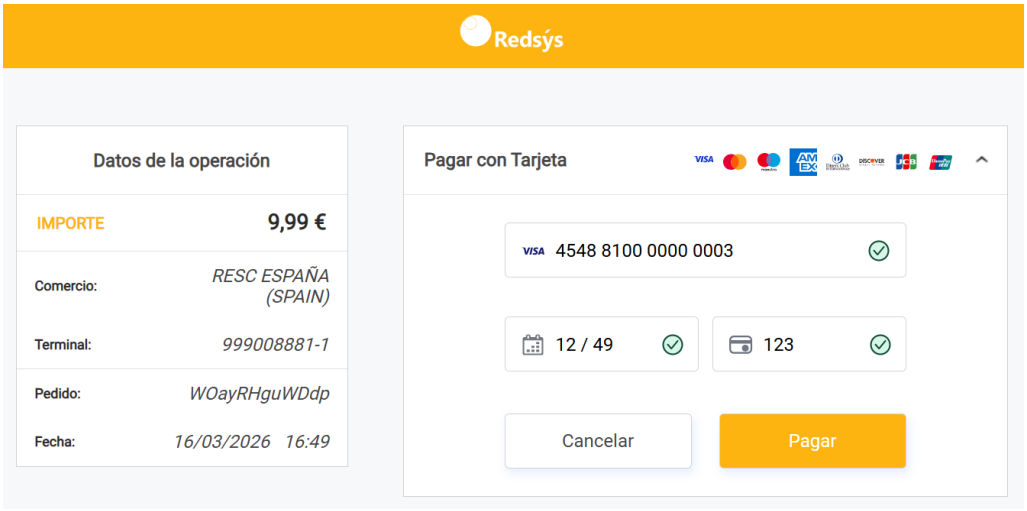
```java
package demo.examples.redirect;

import es.redsys.lib.*;

public class AuthorisationRedirectExample {
    public static void main() {
        final String token = "TEST_OTk5MDA4ODgxXzAwMV9zcTdIanJVT0JmS21DNTc2SUxnc2tENXNyVTg3MGdKNw";
        final Merchant merchant = Merchant.initWithApiKey(token);
        final Parameters params = new Parameters();
        params.setAmount("999");
        params.setOrder(Utils.randomString(12));

        final String redirectForm = GenerateRedirectForm.authorisation(merchant, params);
        // Render redirectForm
    }
}
```

En este ejemplo podemos ver como utilizando el API Key de nuestro comercio para nuestro entorno, en este caso el comercio 999008881-001 de TEST, se puede realizar la integración de redirección muy fácilmente con un archivo de 16 líneas, considerando que donde está el comentario "// Render redirectForm" colocamos una llamada por parte de nuestro framework/aplicación capaz de renderizar un string con contenido HTML.

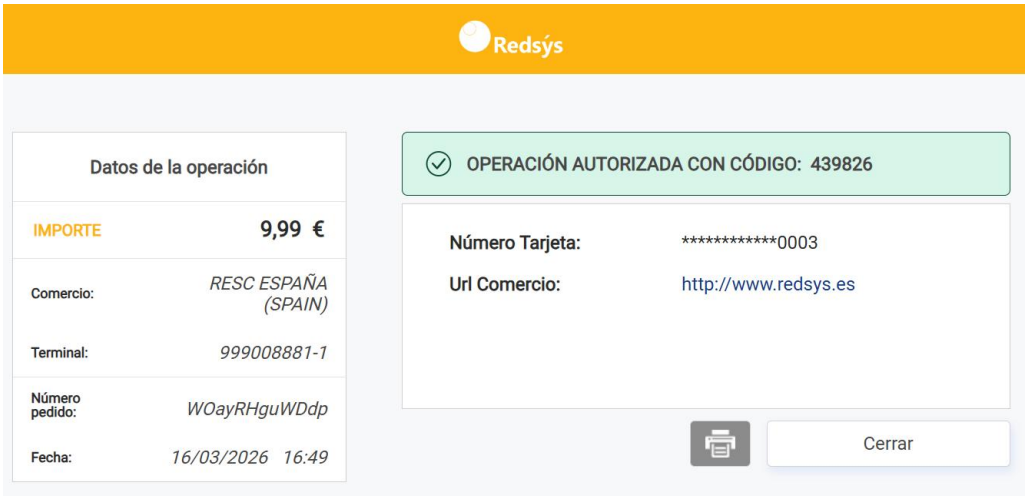
Como se puede comprobar, los únicos parámetros necesarios son la cantidad a pagar (en céntimos) y el número de pedido, ya **que los valores identificativos del comercio vienen**

integrados en el API Key. Si no se indica ninguna moneda, **por defecto se configura el Euro**. El tipo de **transacción también se infiere de la llamada a la función** authorisation().

Al ejecutar el código, nos redirige a la página de pago:



Y al realizar el pago - por ejemplo, con cualquiera de las tarjetas de pruebas de nuestra web https://pagosonline.redsys.es/desarrolladores-inicio/integrate-con-nosotros/tarjetas-y-entornos-de-prueba/- podremos realizar la transacción sin problema y nos mostrará la página de éxito:



Evidentemente si el comercio quiere mantener el control total de la transacción, no puede quedarse aquí, ya que necesitará, al menos, especificar cuáles son las URLs en las que quiera recuperar el control, y la URL en la que recibirá la información de la operación una vez autorizada.

2.5.2 Integración por Redirección Estándar

Para recuperar el control una vez el cliente realiza el pago es necesario decirle al TPV Virtual dónde queremos redirigir al usuario. Esto se realiza configurando las variables urlOk (para redirigir en operaciones correctas), urlKo (para redirigir cuando sucede algún error) y el merchantUrl (para que el sistema del comercio reciba la notificación desde el TPV Virtual).

Un ejemplo podía ser el siguiente:

```java
package demo.examples.redirect;

import es.redsys.lib.*;

public class AuthorisationRedirectExample {
    public static void main() {
        final String token = "TEST_OTk5MDA4ODgxXzAwMV9zcTdIanJVT0JmS21DNTc2SUxnc2tENXNyVTg3MGdKNw";
        final Merchant merchant = Merchant.initWithApiKey(token);
        final Parameters params = new Parameters();
        params.setAmount("999");
        params.setOrder(Utils.randomString(12));

        params.setMerchantUrl("https://url-del-comercio.com/notificaciones");
        params.setUrlOk("https://url-del-comercio.com/pagoCorrecto.html");
        params.setUrlKo("https://url-del-comercio.com/error.html");

        final String redirectForm = GenerateRedirectForm.authorisation(merchant, params);
        // Render redirectForm
    }
}
```

Al recuperar el control en el urlOk o en el urlKo no es necesario que esta sea una página estática html, puede contener lógica.

Es en la URL de notificaciones (la indicada en **merchantUrl**) donde sí es necesario aplicar lógica sobre los datos recibidos, ya que es obligatorio procesar el resultado de la operación a partir de la información enviada en dicha notificación.

Aunque la redirección a **urlOk** indica que la operación aparentemente se ha completado con éxito, el estado definitivo solo se confirma cuando se recibe la notificación correspondiente.

Para mapear correctamente la notificación a la clase Parameters de la librería, podemos usar Parameters.digest(), que recibe el Merchant inicializado y la notificación recibida, comprobando en el proceso la integridad de la respuesta. Si la firma es errónea, digest() lanza una excepción.

Un ejemplo de endpoint donde se reciba una notificación podría ser el siguiente:

```java
package demo.examples.rest;

import java.util.Map;

public class ReceiveNotificationExample {

    public static void handle(HttpExchange exchange) throws Exception {
        final String token = "TEST_OTk5MDA4ODgxXzAwMV9zcTdIanJVT0JmS21DNTc2SUxnc2tENXNyVTg3MGdKNw";
        final Merchant merchant = Merchant.initWithApiKey(token);

        final Map<String, Object> receivedParams = Server.getFormParams(exchange);
        final Parameters params = Parameters.digest(merchant, receivedParams);

        if(Integer.parseInt(params.getCodResponse()) == 0) {
            // Operación correcta, guardar resultado en BBDD
        } else {
            // Operación KO, guardar resultado en BBDD
        }
    }
}
```

Para comprobar que la operación ha terminado correctamente, se comprueba que el codResponse (DS_RESPONSE en la notificación original) sea 0. Para otro tipo de operaciones el DS_RESPONSE de finalización correcta puede ser otro.

Qué significa cada posible contenido del DS_RESPONSE se puede ver en el siguiente apartado de nuestra web: https://pagosonline.redsys.es/desarrolladores-inicio/integrate-con-nosotros/parametros-de-entrada-y-salida/

Una vez el cliente termine de realizar el pago y todo salga bien, será redirigido al la pantalla de Operación Autorizada y la notificación del estado de la operación será enviada desde el TPV Virtual.

O en su defecto al urlOk propuesta por el comercio, (o si el usuario pulsa Continuar) dependiendo de la configuración.

Este flujo vale para todas las operaciones de un solo paso, como pagos, devoluciones y anulaciones.

2.5.3 Integración por Redirección para operaciones con Confirmación

En el TPV Virtual, las **operaciones con Confirmación** incluyen tanto las **preautorizaciones** como las **autenticaciones**. Aunque a nivel operativo ambas siguen un flujo prácticamente idéntico, su efecto en la **liquidación** es distinto:

- La **preautorización** bloquea el importe en la tarjeta del usuario hasta que se confirma.
- La **autenticación** no realiza ese bloqueo previo.

El código del **primer paso** de cualquiera de estas operaciones es muy similar al de una autorización estándar. Sin embargo, en este caso es fundamental **conservar el número de pedido** sin modificarlo, ya que para la confirmación posterior deberá enviarse exactamente el mismo número utilizado en la solicitud inicial.

```java
package demo.examples.redirect;

import es.redsys.lib.*;

public class AuthenticationRedirectExample {
    public static void main() {
        final String token = "TEST_OTk5MDA4ODgxXzAwMV9zcTdIanJVT0JmS21DNTc2SUxnc2tENXNyVTg3MGdKNw";
        final Merchant merchant = Merchant.initWithApiKey(token);
        final Parameters params = new Parameters();
        params.setAmount("999");
        params.setOrder("ORDER123456");

        params.setMerchantUrl("https://url-del-comercio.com/notificaciones");
        params.setUrlOk("https://url-del-comercio.com/pagoCorrecto.html");
        params.setUrlKo("https://url-del-comercio.com/error.html");

        final String redirectForm = GenerateRedirectForm.authentication(merchant, params);
        // Render redirectForm
    }
}
```

En este primer paso, el pago **aún no está finalizado**. Cuando el comercio decida que procede completar la operación, podrá enviar la **confirmación**, ya sea mediante la **integración REST** o a través de otra **Redirección**. Más adelante veremos en detalle cómo realizar este proceso utilizando la integración REST.

Un ejemplo para este segundo paso con Redirección podría ser el siguiente:

```java
package demo.examples.redirect;

import es.redsys.lib.*;

public class AuthenticationConfirmationRedirectExample {
    public static void main() {
        final String token = "TEST_OTk5MDA4ODgxXzAwMV9zcTdIanJVT0JmS21DNTc2SUxnc2tENXNyVTg3MGdKNw";
        final Merchant merchant = Merchant.initWithApiKey(token);
        final Parameters params = new Parameters();
        params.setAmount("999");
        params.setOrder("ORDER123456");

        params.setMerchantUrl("https://url-del-comercio.com/notificaciones");
        params.setUrlOk("https://url-del-comercio.com/pagoCorrecto.html");
        params.setUrlKo("https://url-del-comercio.com/error.html");

        final String redirectForm = GenerateRedirectForm.authenticationConfirmation(merchant, params);
        // Render redirectForm
    }
}
```

Como podemos ver, en el primer paso se llama a la función authentication(), que equivale a una operación con transactionType 7, mientras que en el segundo paso invocamos a authenticationConfirmation(), que equivale a una operación con transactionType 8.

En cada uno de los pasos, tal como vimos anteriormente, las redirecciones finalizan en el **urlOk** configurado por el comercio. Además, ambos pasos generan su **notificación correspondiente**, lo que permite al comercio controlar correctamente el resultado de cada fase de la operación.

2.5.4 Integración por Rest

Esta modalidad es más compleja que la redirección tradicional, ya que **exige controlar** aspectos adicionales, como la **verificación del cliente durante el proceso de pago**. Si fuera necesaria una explicación más en profundidad de este flujo, consultar la guía de integración rest o nuestra web de desarrolladores.

```java
package demo.examples.rest;

import java.io.IOException;

public class AuthorisationRestExample {
    @SuppressWarnings("unchecked")
    public static void authorisationRestExample(HttpExchange exchange) throws Exception {
        final Merchant merchant = Merchant.initWithApiKey(Secrets.REDSYS_API_KEY);
        final Parameters params = new Parameters();

        final Map<String, Object> receivedParams = Server.getJsonParams(exchange);
        params.setAmount("999"); // En céntimos
        params.setOrder("ORDER123456");

        // Card
        params.setCardNumber("4548810000000003");
        params.setCardExpiryDate("4912");
        params.setCvv2("123");

        // INIT
        Response initResponse = Rest.authorisationInit(merchant, params);
        Parameters initParameters = initResponse.getData();

        //Notificaciones
        params.setMerchantUrl("https://url-del-comercio.com/notificaciones");

        params.setEmv3ds(initParameters.getEmv3ds());
        params.getEmv3ds().put("threeDSInfo", "AuthenticationData");
        params.getEmv3ds().put("notificationURL", "https://url-del-comercio.com/challenge-response");
        params.getEmv3ds().put("browserAcceptHeader", exchange.getRequestHeaders().getFirst("Accept"));

        // Obteniendolo de una petición desde un navegador
        final Map<String, Object> browserDetailsEmv3ds = (HashMap<String, Object>) receivedParams.get("emv3ds");
        params.getEmv3ds().putAll(browserDetailsEmv3ds);

        Response response2 = Rest.authorisation(merchant, params);

        if((response2.getData().getCodResponse() != null
                || !"0000".equals(response2.getData().getCodResponse()))
                && response2.getData().getEmv3ds() != null) {
            // Challenge
            response2.setChallenge(GenerateRedirectForm.challenge(merchant, response2.getData()));
        }
    }
```

En este tipo de integración, necesitamos información del cliente que, en la integración por Redirección, el TPV Virtual obtiene de forma automática.
Además, antes de ejecutar el pago, es necesario realizar una **consulta previa sobre la tarjeta introducida** para conocer el tipo de verificación que requiere. En el ejemplo, esto se realiza mediante la función authorisationInit().
Las funciones de tipo *Init* añaden automáticamente el valor **"CardData"** dentro de emv3ds["threeDSInfo"], indicando que estamos solicitando información de la tarjeta. En los pasos posteriores, debemos ser nosotros mismos quienes especifiquemos el punto exacto del flujo 3DS en el que nos encontramos.

También es necesario disponer de un **endpoint** donde el comercio recibirá la respuesta del verificador una vez que se confirme que la persona que está realizando la operación es, efectivamente, la titular de la tarjeta. Esta URL se incluye dentro de los parámetros emv3ds, concretamente en notificationURL.

Para poder realizar la verificación del cliente, debemos obtener ciertos **datos del navegador** desde el que se efectúa el pago. Estos datos pueden recopilarse mediante una función de JavaScript y enviarse posteriormente al endpoint encargado de procesar la operación por POST.

```
<!DOCTYPE html>
<body>
    <form id="form" method="POST" action="example">
        <input type="hidden" id="emv3ds" name="emv3ds"/>
    </form>
    <script>
        let emv3dsObj = {};
        emv3dsObj['browserUserAgent'] = navigator.userAgent;
        emv3dsObj['browserJavaEnabled'] = false;
        emv3dsObj['browserJavascriptEnabled'] = true;
        emv3dsObj['browserLanguage'] = navigator.language;
        var colorDepth = window.screen.colorDepth;
        if (colorDepth === undefined || colorDepth === null
            || !(colorDepth == '1' || colorDepth == '4' || colorDepth == '8'  || colorDepth == '15'
                || colorDepth == '16' || colorDepth == '24' || colorDepth == '32' || colorDepth == '48')) {
            colorDepth = '24';
        }
        emv3dsObj['browserColorDepth'] = colorDepth;
        emv3dsObj['browserScreenHeight'] = window.screen.height;
        emv3dsObj['browserScreenWidth'] = window.screen.width;
        emv3dsObj['browserTZ'] = (new Date()).getTimezoneOffset();

        document.getElementById("emv3ds").value = JSON.stringify(emv3dsObj);
        document.getElementById("form").submit();
    </script>
</body>
</html>
```

Cuando ejecutamos la llamada a authentication() en el ejemplo, debemos indicar que nos encontramos en el paso **"AuthenticationData"**, asignándolo dentro de emv3ds["threeDSInfo"].

En el endpoint definido en notificationURL recibiremos por POST una serie de parámetros que tendremos que reenviar al TPV Virtual para dejar constancia de que el cliente ha superado correctamente la verificación. Una vez incluidos en los parámetros de la solicitud, podremos enviarlos mediante la función challengeResponse. En este caso no es necesario asignanar **"ChallengeResponse"** dentro de emv3ds["threeDSInfo"] ya que se hace automáticamente. Esta función devolverá el resultado final de la operación.

```java
package demo.examples.rest;

import java.util.HashMap;

public class ChallengeResponseExample {
    public static void challengeResponseExample(HttpExchange exchange) throws Exception {

        final Merchant merchant = Merchant.initWithApiKey(Secrets.REDSYS_API_KEY);
        final Parameters params = new Parameters();
        params.setOrder("ORDER123456");

        // Parametros recibidos
        final Map<String, Object> receivedParams = Server.getFormParams(exchange);

        final Map<String, Object> emv3ds = new HashMap<String, Object>();
        // Usamos el protocolVersion recibido en el emv3ds de authorisationInit
        // Para este ejemplo imaginemos que es 2.2.0
        emv3ds.put("protocolVersion", "2.2.0");
        emv3ds.put("cres", receivedParams.get("cres"));
        params.setEmv3ds(emv3ds);

        Response challengeResponse = Rest.challengeResponse(merchant, params);
        if(challengeResponse != null && challengeResponse.getData() != null
                && "0000".equals(challengeResponse.getData().getCodResponse())) {
            // Pago correcto
        } else {
            // Pago KO
        }
    }
}
```

Finalmente, y como en los ejemplos anteriores, es necesario **procesar la notificación recibida** para confirmar el resultado definitivo y no guiarse en exclusiva por la respuesta síncrona. Como en la integración por redirección, la URL donde recibir las notificaciones se comunica en merchantUrl.

Si se quisiera hacer una operación con confirmación a través de Rest, a lo anteriormente explicado en este punto sólo habría que añadirle un par de pasos extra a continuación de los ya vistos. Los pasos anteriores a los que vamos a exponer serían exactamente los mismos, simplemente habría que cambiar el nombre de la función a la que invoca la clase Rest por una de las transacciones que sean de operaciones con confirmación (preauthorisation() o authentication().

Al hacer esto y haber realizado todo el flujo, llegaríamos a la confirmación. Los dos pasos extra serían confirmar:

```java
package demo.examples.rest;

import com.sun.net.httpserver.HttpExchange;

public class AuthenticationConfirmationRestExample {
    public static void authenticationConfirmationRestExample(HttpExchange exchange) throws Exception {
        final Merchant merchant = Merchant.initWithApiKey(Secrets.REDSYS_API_KEY);
        final Parameters params = new Parameters();

        params.setAmount("999"); // En céntimos
        params.setOrder("ORDER123456");
        params.setMerchantUrl("https://url-del-comercio.com/notificaciones");

        Response response = Rest.authenticationConfirmation(merchant, params);
        if(response != null && response.getData() != null
                && "0000".equals(response.getData().getCodResponse())) {
            // Pago correcto
        } else {
            // Pago KO
        }
    }
}
```

Y posteriormente, procesar la notificación para comprobar el resultado definitivo.

2.6 Adaptación y flexibilidad en la integración

2.6.1 ¿Puedo especificar el Merchant con los parámetros que ya tengo sin utilizar el API Key?

Sí, esto es posible. El constructor por defecto de la clase Merchant recibe, por este orden, fuc, terminal, clave de firma, y entorno al que pertenece el comercio, además se puede inicializar un constructor vacío y asignarlos individualmente. El entorno se puede indicar con la clase Environment. Un ejemplo podría ser el siguiente:

```
final Merchant merchant = new Merchant();
merchant.setFuc("999XXXXXX");
merchant.setTerminal("1");
merchant.setKc(clave);
merchant.setEnv(Environment.TEST.getValue());
```

Inicializar al comercio de esta forma es más conveniente siempre y cuando el comercio que quiere realizar la integración tenga más de un terminal. No obstante siempre va a ser más sencillo inicializar con API Key.

2.6.2 ¿Puedo llamar a los parámetros como DS_MERCHANT_* o DS_*?

Como respuesta corta: **Sí**, este API permite llamar a todos los parámetros por su nombre original en caso de que el integrador ya conozca el nombre de estos con anterioridad y no quiera aprender la equivalencia a los nuevos.

También cabe destacar que parámetros como fuc y terminal ya se aportan en la clase Merchant y no es necesario concretarlos en los parámetros.

¿Cómo se podría hacer esto? Puedes llamando al parámetro que quieras invocar con getParam() (o getParamString() si sabes que el parámetro es de tipo String) o asignarlo con setParam(), como si de un Map se tratara. Por ejemplo, para amount, que se asignaría con params.setAmount("999") puedes asignarlo como params.setParam("DS_MERCHANT_AMOUNT", "999") y tendría el mismo efecto. Un ejemplo sobre código usando exclusivamente la nomenclatura clásica DS_* podría ser el siguiente:

```java
final Merchant merchant = Merchant.initWithApiKey(Secrets.REDSYS_API_KEY);
final Parameters params = new Parameters();

final Map<String, Object> receivedParams = Server.getJsonParams(exchange);
params.setParam("DS_MERCHANT_AMOUNT", "999"); // En céntimos
params.setParam("DS_MERCHANT_ORDER", "ORDER123456");

// Card
params.setParam("DS_MERCHANT_PAN", "4548810000000003");
params.setParam("DS_MERCHANT_EXPIRYDATE", "4912");
params.setParam("DS_MERCHANT_CVV2", "123");

// INIT
Response initResponse = Rest.authorisationInit(merchant, params);
Parameters initParameters = initResponse.getData();

//Notificaciones
params.setParam("DS_MERCHANT_CVV2", "https://url-del-comercio.com/notificaciones");

final Map<String, Object> emv3ds = initParameters.getEmv3ds();
emv3ds.put("threeDSInfo", "AuthenticationData");
emv3ds.put("notificationURL", "https://url-del-comercio.com/challenge-response");
emv3ds.put("browserAcceptHeader", exchange.getRequestHeaders().getFirst("Accept"));

// Obteniendolo de una petición desde un navegador
final Map<String, Object> browserDetailsEmv3ds = (HashMap<String, Object>) receivedParams.get("emv3ds");
emv3ds.putAll(browserDetailsEmv3ds);
params.setParam("DS_MERCHANT_EMV3DS", emv3ds);

Response response2 = Rest.authorisation(merchant, params);
```

2.6.3 En la respuesta de una llamada recibo un parámetro pero no está contemplado en Parameters, ¿cómo lo obtengo?

Al igual que en el punto anterior, la clase Parameters tiene flexibilidad para recibir cualquier parámetro. Si este no está contemplado, en la respuesta del TPV Virtual recibida en la clase Response, podemos traernos la respuesta traducida a Parameters, que se encuentra en response.getData(), y a continuación buscar en data el parámetro que necesites.

```java
response.getData().getParam("DS_EMV3DS")
```

2.6.4 ¿Qué ocurre si el tipo de operación que quiero realizar no está contemplada en la librería?

La solución a este problema también está prevista.

Tanto en la integración por Redirección como en la integración REST, las clases correspondientes usadas para realizar estas llamadas contienen una función generic() o genericInit() en la que, o bien dentro del parámetro, o bien en la misma función, puedes indicar el número de la transactionType que quieras realizar.

Además, si tu operación NO requiere de transactionType (DS_TRANSACTION_TYPE), puedes llamar a la función generic() con el tercer parámetro de esta a null. Los dos primeros parámetros de esta función generic son los mismos que reciben las funciones que invocan las operaciones vistas anteriormente, que son Merchant y Parameters.

Si adaptamos el ejemplo del punto anterior para operar con lo que aquí hemos explicado, quedaría de la siguiente forma:

```java
final Merchant merchant = Merchant.initWithApiKey(Secrets.REDSYS_API_KEY);
final Parameters params = new Parameters();

final Map<String, Object> receivedParams = Server.getJsonParams(exchange);
params.setParam("DS_MERCHANT_AMOUNT", "999"); // En céntimos
params.setParam("DS_MERCHANT_ORDER", "ORDER123456");

// Card
params.setParam("DS_MERCHANT_PAN", "4548810000000003");
params.setParam("DS_MERCHANT_EXPIRYDATE", "4912");
params.setParam("DS_MERCHANT_CVV2", "123");

// INIT
Response initResponse = Rest.genericInit(merchant, params, "0");
Parameters initParameters = initResponse.getData();

//Notificaciones
params.setParam("DS_MERCHANT_CVV2", "https://url-del-comercio.com/notificaciones");

final Map<String, Object> emv3ds = initParameters.getEmv3ds();
emv3ds.put("threeDSInfo", "AuthenticationData");
emv3ds.put("notificationURL", "https://url-del-comercio.com/challenge-response");
emv3ds.put("browserAcceptHeader", exchange.getRequestHeaders().getFirst("Accept"));

// Obteniendolo de una petición desde un navegador
final Map<String, Object> browserDetailsEmv3ds = (HashMap<String, Object>) receivedParams.get("emv3ds");
emv3ds.putAll(browserDetailsEmv3ds);
params.setParam("DS_MERCHANT_EMV3DS", emv3ds);

Response response = Rest.generic(merchant, params, "0");
```

Cambiando authorisationInit(merchants, params) y authorisation(merchants, params) por genericInit(merchants, params, “0”) y generic(merchants, params, “0”), pasando, como dicho anteriormente, el tercer parámetro de la función el transactionType correspondiente.

2.6.5 Necesito llamar a una URL específica del TPV Virtual que no está contemplada en las funciones de la librería, ¿Qué hago?

Utilizar esta librería en esta ocasiones sigue siendo útil ya que, por norma general, la disposición de los datos y la forma de realizar la firma es la misma para todas las llamadas independientemente del donde se llame.

Antes de saber si esto es posible hay que tener en cuenta que actualmente las llamadas de consulta de operaciones no están soportadas aún por esta librería, aunque llegará.

Si esta URL específica a la que necesitas llamar es del SIS (https://sis.redsys.es/* o https://sis-t.redsys.es:<puerto>/*) SÍ va a ser posible realizarla.

A continuación veremos cómo se puede hacer con un ejemplo. Siguiendo la misma línea de antes, aunque lo que se ve en el ejemplo es una llamada al TPV Virtual que sí está contemplada, la llamada a una que no lo está sería igual, sólo habría que asignar a params.setCustomUrl() la URL necesaria.

```java
final Merchant merchant = Merchant.initWithApiKey(Secrets.REDSYS_API_KEY);
final Parameters params = new Parameters();

final Map<String, Object> receivedParams = Server.getJsonParams(exchange);
params.setParam("DS_MERCHANT_AMOUNT", "999"); // En céntimos
params.setParam("DS_MERCHANT_ORDER", "ORDER123456");

// Card
params.setParam("DS_MERCHANT_PAN", "4548810000000003");
params.setParam("DS_MERCHANT_EXPIRYDATE", "4912");
params.setParam("DS_MERCHANT_CVV2", "123");

params.setCustomUrl("https://sis-t.redsys.es:25443/sis/rest/iniciaPeticionREST");

// INIT
Response initResponse = Rest.genericInit(merchant, params, "0");
Parameters initParameters = initResponse.getData();

//Notificaciones
params.setParam("DS_MERCHANT_CVV2", "https://url-del-comercio.com/notificaciones");

final Map<String, Object> emv3ds = initParameters.getEmv3ds();
emv3ds.put("threeDSInfo", "AuthenticationData");
emv3ds.put("notificationURL", "https://url-del-comercio.com/challenge-response");
emv3ds.put("browserAcceptHeader", exchange.getRequestHeaders().getFirst("Accept"));

// Obteniendolo de una petición desde un navegador
final Map<String, Object> browserDetailsEmv3ds = (HashMap<String, Object>) receivedParams.get("emv3ds");
emv3ds.putAll(browserDetailsEmv3ds);
params.setParam("DS_MERCHANT_EMV3DS", emv3ds);

params.setCustomUrl("https://sis-t.redsys.es:25443/sis/rest/trataPeticionREST");

Response response = Rest.generic(merchant, params, "0");
```

2.6.6 ¿Cómo podría serializar o sacar como array la clase Parameters a un JSON con los nombres de los parámetros clásicos (DS_MERCHANT_*)

Si por alguna razón se quisiera serializar la clase Parameters a un JSON con nombre de los valores con los nombres clásicos, esto se podría hacer con params.toJsonIn() para DS_MERCHANT, recibiendo de esto un JSON en String