Affipay Pos

**Affipay Pos**

Operaciones:

* Autenticación OAuth 2.0
* Inicialización de dispositivo
* Retail:

- Venta  
- Cancelación  
- Devolución

- Reverso

- Venta Offline

* Inicio de sesión del usuario
* Recuperación de contraseña
* Consulta de transacciones
* Envió de ticket
* Detalle de una transacción

**Contenido**

[Metodos interactivos del dspread (Librería dspread) 4](#_Toc33041397)

[Metodos de operativos (Librería Blumonpay\_library) 4](#_Toc33041398)

[Integración de librerías 7](#_Toc33041399)

[Implementación de librería Dspread 7](#_Toc33041400)

[Implementación de la librería Blumonpay\_library 11](#_Toc33041401)

[Aplicación de operativas 12](#_Toc33041402)

[Controlador de servicios tokener 12](#_Toc33041403)

[Inicio de sesión del usuario 12](#_Toc33041404)

[Restaurar contraseña 12](#_Toc33041405)

[Inicialización de llaves del dispositivo 13](#_Toc33041406)

[Controlador de servicios BPC 14](#_Toc33041407)

[Retail Venta 14](#_Toc33041408)

[Actualizar transacción (solo en venta) 16](#_Toc33041409)

[Validar cancelación 16](#_Toc33041410)

[Retail Cancelación 17](#_Toc33041411)

[Reverso 18](#_Toc33041412)

[Almacenar firma 18](#_Toc33041413)

[Consultar transacciones por rango de fecha vía correo electrónico 19](#_Toc33041414)

[Enviar ticket por correo electrónico 19](#_Toc33041415)

[Lista de las ultimas 10 transacciones realizadas 19](#_Toc33041416)

[Detalle de una transacción en específico 20](#_Toc33041417)

[Realizar todas las ventas offline 20](#_Toc33041418)

[Reintentar una venta offline en específico 20](#_Toc33041419)

[Controlador de servicios Element 21](#_Toc33041420)

[Código de verificación 21](#_Toc33041421)

[Registro completo 21](#_Toc33041422)

[Consulta de perfil del usuario 22](#_Toc33041423)

[Carga de catálogos 22](#_Toc33041424)

**Metodos interactivos del dspread (Librería dspread)**

Los métodos principales que se implementan por medio de su interface *QPOSService.QPOSServiceListener* para la interacción con el dispositivo, son:

* **onRequestQposConnected()**: Indica la conexión exitosa del dispositivo con la aplicación.
* **onRequestNoQposDetected()**: Indica la conexión fallida del dispositivo con la aplicación.
* **onRequestQposDisconnected()**: Indica la desconexión del dispositivo con la aplicación.
* **onQposIdResult()**: Extrae el número serial del dispositivo.
* **onRequestSetAmount()**: Se establece el monto a solicitar.
* **onRequestWaitingUser()**: Muestra el monto en el dispositivo.
* **onDoTradeResult()**: Detecta el modo de lectura de la tarjeta.
* **onRequestOnlineProcess()**: Devuelve la información de la tarjeta chip detectada.
* **onRequestTransactionResult()**: Devuelve el status de la transacción.
* **onError()**: Detecta el tipo de error con respecto al dispositivo.
* **onRequestDisplay()**: El mensaje a mostrar en el dispositivo.

Metodos de operativos (Librería Blumonpay\_library)

La instancia de cualquier controlador de servicios requiere los parámetros *OperativesListener* y el contexto de la actividad.

La clase **ServicesTokener** contiene los métodos:

|  |  |  |
| --- | --- | --- |
| METODO | INPUT | DESCRIPCIÓN |
| LoginUser | * String email * String password | Inicio de sesión del usuario, el logueo para poder acceder a la aplicación. |
| RecoverPassword | * String email | Restablece la contraseña del usuario. |
| LoginDevice | * String serialDevice * String addressDevice | Realiza la inicialización de llaves del dispositivo. |

La clase **ServicesBPC** tiene los siguientes métodos:

|  |  |  |
| --- | --- | --- |
| METODO | INPUT | DESCRIPCIÓN |
| Sale | * Hastable decodeData * String amount * EntryMode * Authentication | Realiza una venta y valida si hay internet, en caso de que no, genera una venta offline. |
| ValidateCancel | * Integer operationNumber | Valida la existencia de una transacción en específico para realizar una cancelación. |

|  |  |  |
| --- | --- | --- |
| Cancel | * Hastable decodeData * String amount * EntryMode * Authentication | Realiza la cancelación de una venta. |
| UpdateTransaction | * Integer operationNumber * String signatura * Double latitude * Double longitude * String entryMode * String tc * Double tipAmount * Integer waiter | Actualiza una transacción en específico. |
| SaveSignature | * Bitmap imageFirm * Integer insertOffline | Genera el base64 de un bitmap y actualiza la transacción offline en específico, en caso de que no haya internet. |
| GenerateReverse | * Reverse type | Realiza un reverso de la transacción en curso. |
| ConsultTransactionForDate | * String startDate * String endDate   Formato aaaammdd | Envía por correo electrónico las transacciones entre un rango de fechas. |
| SendTicket | * String emailClient * Integer operationNumber | Envía por correo electrónico el ticket del cliente especificado por el número de esa transacción. |
| LastTenTransactions | ------ | Muestra las ultimas diez transacciones realizadas con los siguientes datos: número de operación, fecha, monto y código de autorización. |
| OperationDetail | * Integer operationNumber | Muestra el detalle completo de una transacción en específico. |
| LaunchAllOffline | * Context | Realiza todas las transacciones hechas offline. |
| LaunchSpecificOffline | * Integer operationNumber | Realiza una transacción offline en específico. |

La clase **ServicesElement** contiene los siguientes métodos:

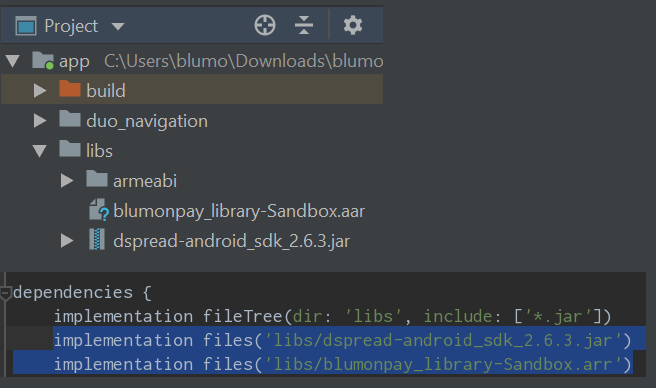
|  |  |  |
| --- | --- | --- |
| METODO | INPUT | DESCRIPCIÓN |
| VerificationCode | * String name * String email * String nameP * String nameM * String phone | Envía un código de verificación por correo electrónico para el registro de un comercio. |
| SignUp | * SignUpData | Genera un registro con los datos del objeto SignUpData. |
| ConsultUserProfile | -------- | Muestra los datos del usuario que se loguea. |
| Catalogs | -------- | Carga el catálogo de régimen, giro y banco en la base de datos del teléfono. |

Para manipular el resultado de cada operativa o continuar con el flujo se debe implementar la interface *OperativesListener* donde se encuentran los siguientes métodos ya predefinidos:

|  |  |  |
| --- | --- | --- |
| METODO | OUTPUT | DESCRIPCIÓN |
| StartTransaction | ---- | Inicializa la solicitud de cualquier servicio. |
| ProcessingTransaction | ---- | Proceso de respuesta del servicio. |
| EndTransaction | * Boolean TransactionStatus * String description * ObjectResponse | Resultado de la solicitud del servicio devolviendo el estatus de respuesta, un mensaje descriptivo al servicio solicitado y el objeto de respuesta. |

Integración de librerías

Agregar la librería del dspread y blumonpay\_library en el proyecto, en la carpeta libs e implementarla los archivos en las dependencias del gradle.

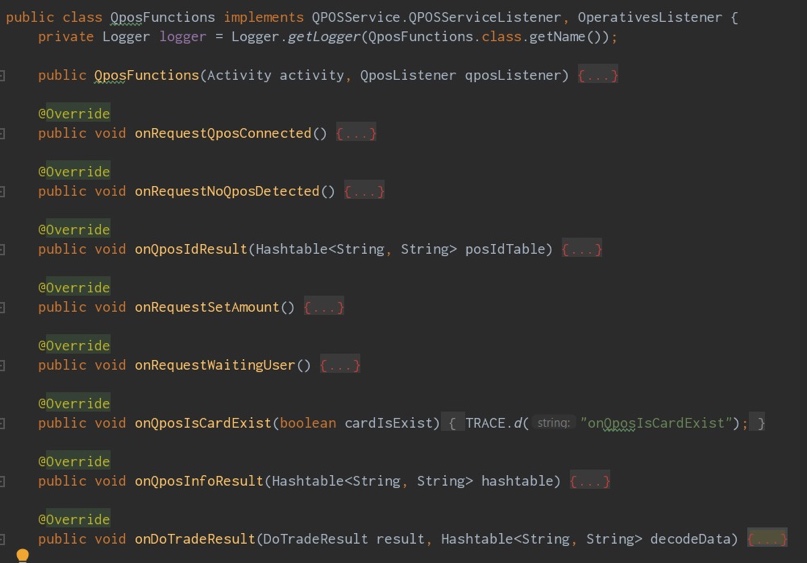


**Figura 1.** Integración e implementación de las librerías.

Implementación de librería Dspread

Implementar los métodos de la interface *QPOSServiceListener* para realizar la interacción con el dspread.

***Nota:*** *La clase QposFunctions que se muestra en la figura 2 es una clase de ejemplo donde BlumonPay la crea como una clase administradora entre los listeners del dspread y las respuestas para los activity, la cual* ***no viene incluida******en la librería*** *ya que* ***depende de la implementación del usuario*** *al igual que la clase QposListener, esa se creo para enviar la respuesta de la clase QposFunctions a una vista.*

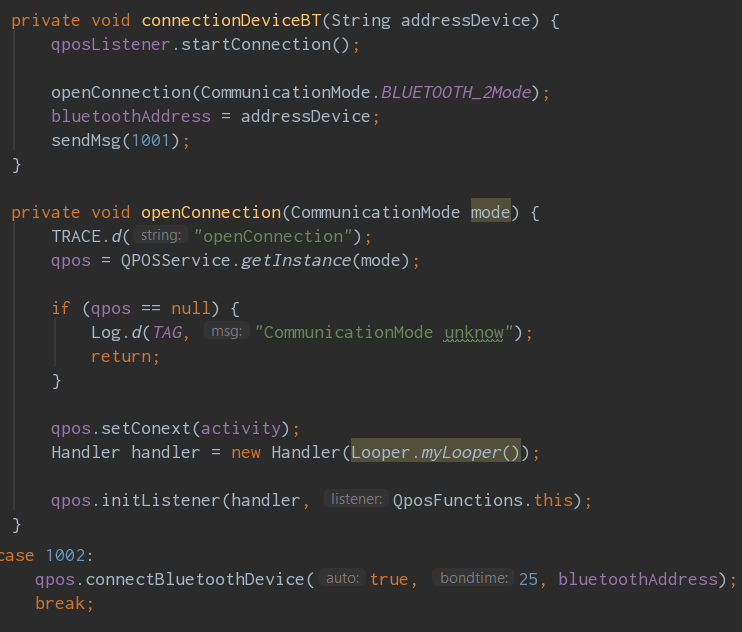


**Figura 2.** Implementación del interface QPOSServiceListener.

Para iniciar el proceso de conexión del dispositivo con la aplicación se requiere implementar las siguientes funciones.

* + - 1. Crear variable **qpos** de tipo **QPOSService**.
      2. Instanciar la variable **qpos** con la siguiente linea **QPOSService.getInstance**(CommunicationMode.BLUETOOTH\_2Mode);
      3. Enviar el contexto por medio de la variable **qpos** y el método **setConext()**
      4. Ejecutar la linea Handler handler = new Handler(Looper.myLooper());
      5. Especificar la clase de implementación de la interface QPOSServiceListener por medio de la siguiente línea **qpos.initListener**(handler, QposFunctions.this);
      6. Por último iniciar la conexión bluetooth del dispositivo con la apliación con la línea **connectBluetoothDevice**(true, 25, bluetoothAddress);

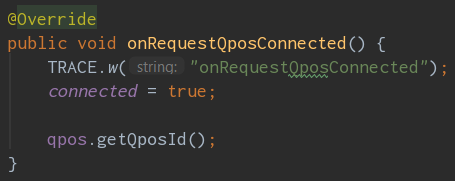
***Nota:*** *Los metodos connectionDeviceBT y openConnection son creados por BlumonPay para ejecutar las líneas de comando mencionadas anteriormente y dar inicio a la utilización de los metodos implementados por la interface QPOSServiceListener.*



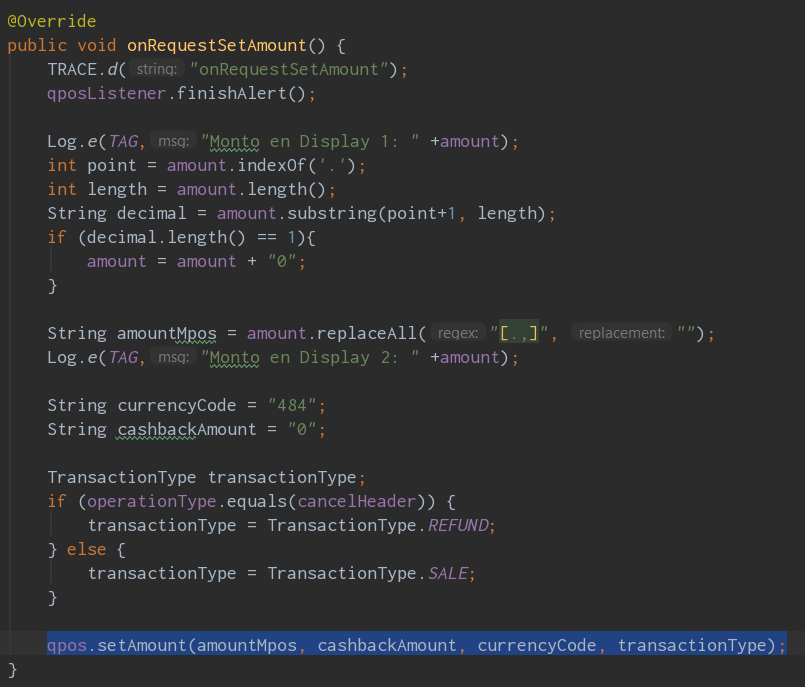
**Figura 3.** Ejecución de las líneas de comando mencionadas anteriormente para establecer la conexión del dispositivo con la aplicación.

Una vez conectado el dispositivo, llamar las siguientes funciones:

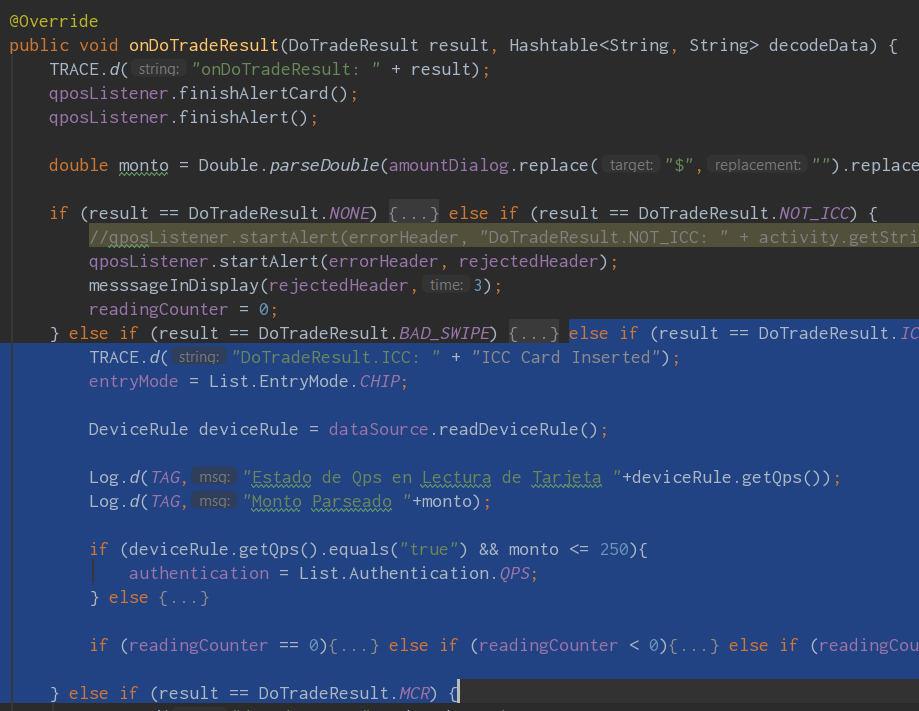
1. En el método ***onRequestQposConnected*** aplicar ***qpos.getQposId()*** para obtener el serial del dspread.



1. En el método ***onRequestSetAmount*** llamar ***qpos.setAmount()*** el cual requiere el monto a cobrar, el siguiente parámetro va en 0, luego 484, REFUND en caso de cancelación o SALE en venta.

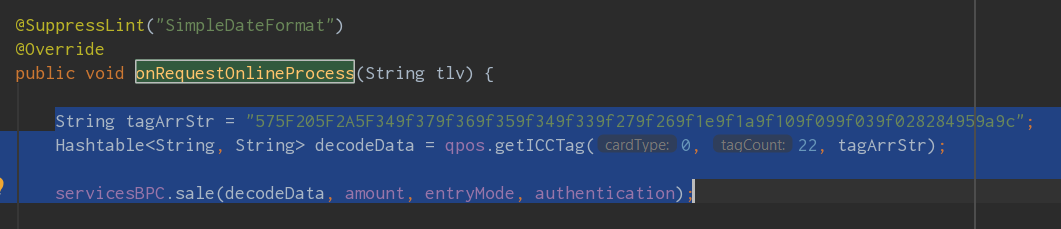


1. En el método ***onDoTradeResult()*** si el valor de la ***variable result*** es ***ICC*** detecto una tarjeta chip, si es ***MCR*** detecto una tarjeta por banda.

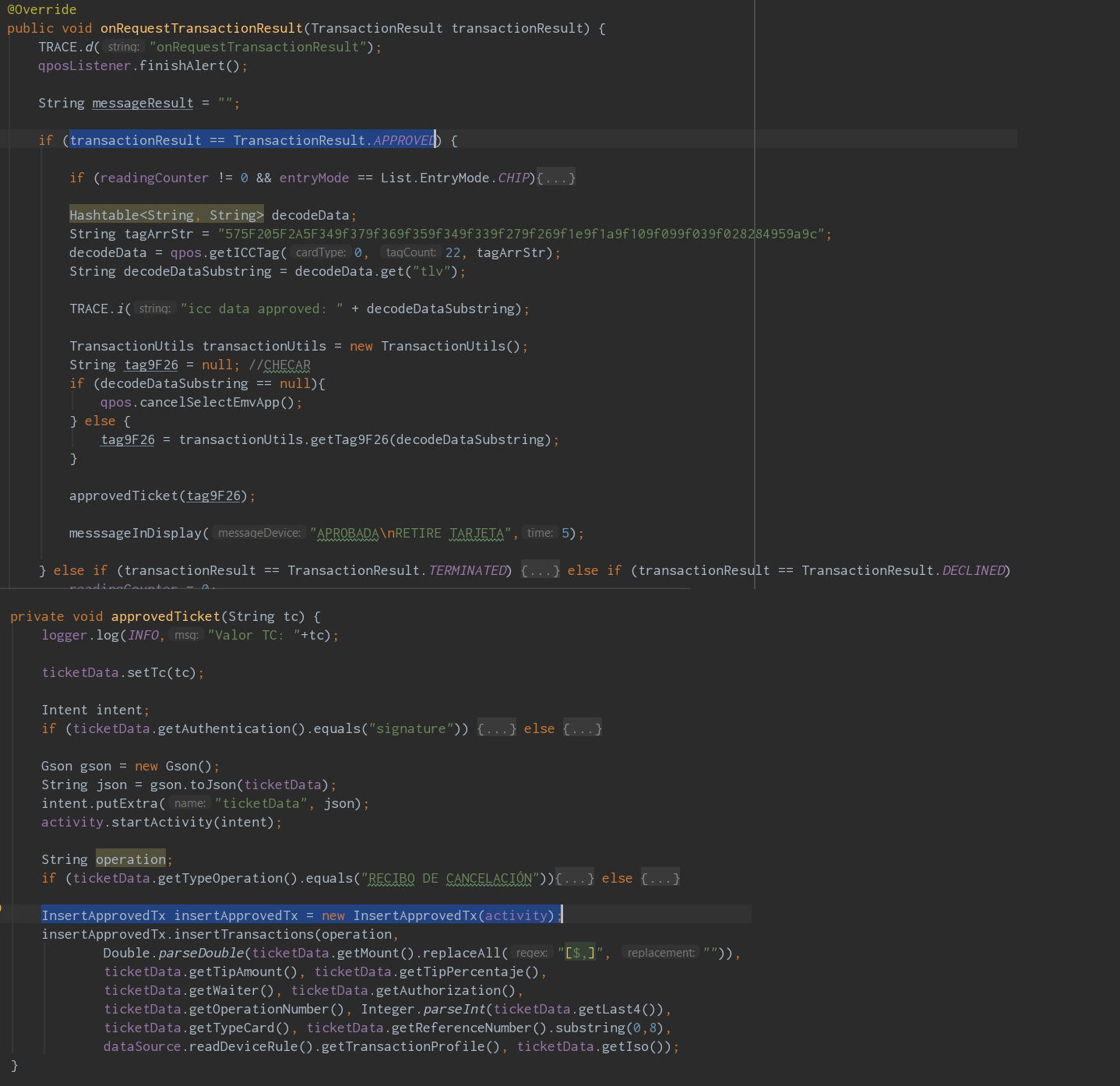


1. En el método **onRequestOnlineProcess** se debe realizar la función ***qpos.getICCTag()*** con los parámetros 0, 22, 575F205F2A5F349f379f369f359f349f339f279f269f1e9f1a9f109f099f039f028284959a9c. En seguida se extrae la variable tlv del HashMap generado con la función anterior.

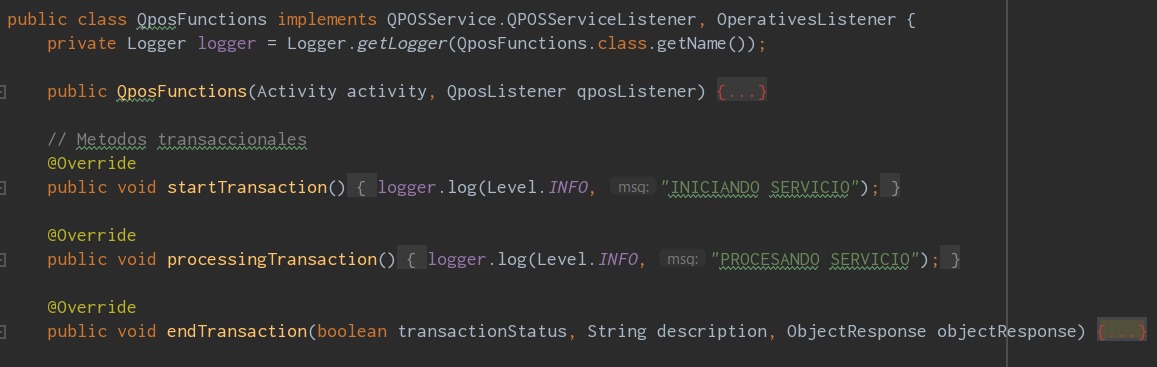
\*Se utilizo servicesBPC.validateTransaction(



1. El método ***onRequestTransactionResult*** devuelve el estado de la transacción ya sea APPROVED, TERMINATED, DECLINED, etc. En caso aprobado se requiere aplicar la misma función getICCTag() con los mismos parámetros. Al ser aprobadas se guardar en la base de datos del teléfono por medio de la clase ***insertApprovedTx***.



Implementación de la librería Affipay\_library

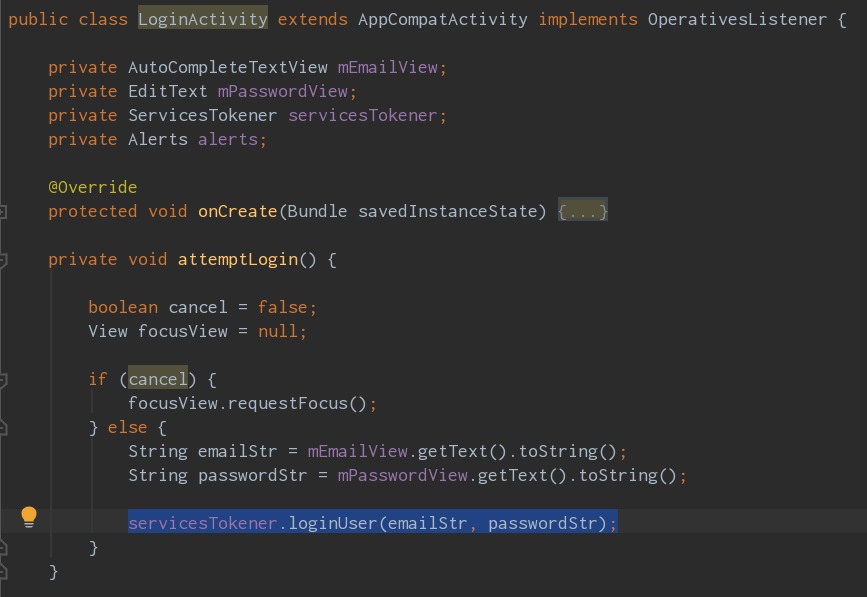


**Figura 4.** Implementación de la interface OperativesListener y sus métodos predefinidos.

Aplicación de operativas

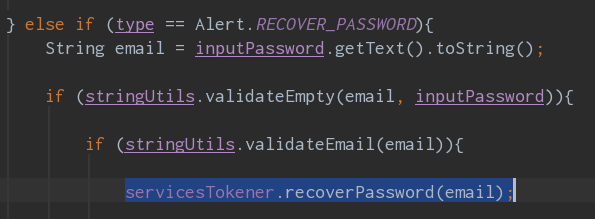
Controlador de servicios tokener

Inicio de sesión del usuario

****

**Figura 5.** Ejecución del método loginUser del controlador de servicios.

Restaurar contraseña

****

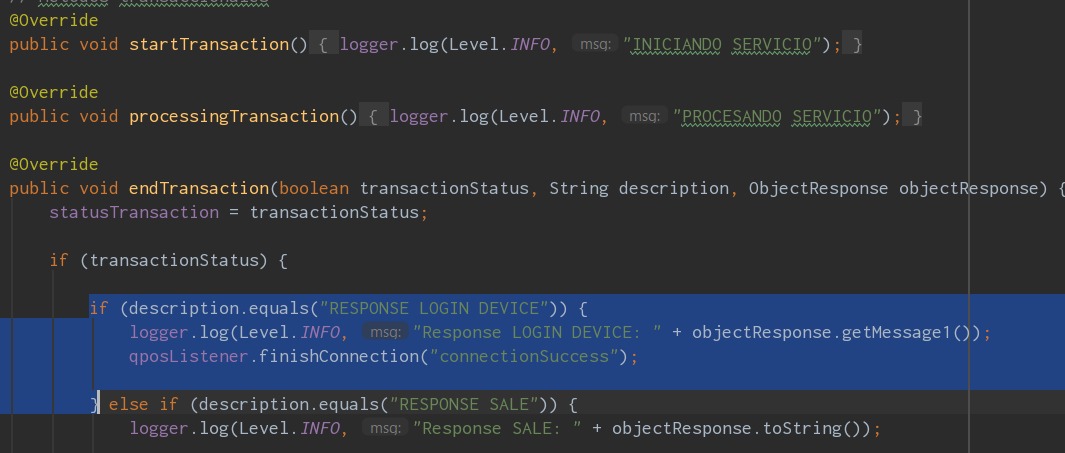
**Figura 6.** Utilización del método recoverPassword.

Inicialización de llaves del dispositivo

En el método *onQposIdResult* implementar la inicialización de llaves ya que es aquí donde se obtiene el serial del dispositivo una vez vinculado.

****

**Figura 7.** Instancia del método loginDevice dispara la inicialización de llaves del dispositivo.



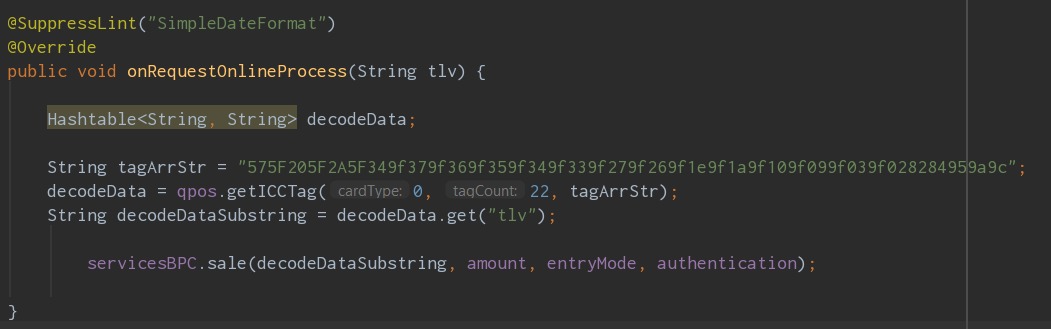
**Figura 8.** Ejemplo de implementación para la respuesta del servicio Login Device.

\*No existe listener.finishConnection()

\* Como implementar Authentication

Controlador de servicios BPC

Retail Venta

****

**Figura 9.** Implementación del método sale para el inicio de una venta, con los siguientes parámetros: decodeData, monto a cobrar, modo de entrada de la tarjeta e identificación de la tarjeta.

Al concluir la venta el objeto de respuesta que devuelve el método *EndTransaction()* contiene los siguientes parámetros en diferentes casos:

**Transacciones online aprobadas:**

Message1: La descripción de la venta ya sea “Venta Online” o “Venta Offline”.

Aid: Código de respuesta de las transacciones solo con tarjetas CHIP.

Arqc: Código de respuesta de las transacciones solo con tarjetas CHIP.

RealCounter: Contador de transacciones realizadas, se resetea cada vez que se hace una inicialización de llaves.

OperationNumber: Identificador de cada transacción, conocido también como “Número de Operación”.

Date: Fecha en la que se autorizó la transacción.

Time: Hora en la que se autorizó la transacción.

Authorization: Código de autorización de una transacción, también se considera como otro tipo de identificador.

Description: Mensaje descriptivo con la transacción.

Bin: Los primeros seis números de la tarjeta.

Bank: Banco de la tarjeta con la que se realizó la transacción.

Type: Tipo de tarjeta “Débito” o “Crédito”.

Brand: Marca de la tarjeta “Visa” o “MasterCard”.

Membership: Número de la afiliación.

EntryMode: Modo de entra de la tarjeta “Chip” o “Banda”.

TypeOperation: Tipo de operación “Venta”, “Cancelación” o “Reverso”.

NameCommerce: Nombre del comercio.

AddressCommerce: Dirección del comercio.

TerminalNumber: Identificador de la terminal con la que se está realizando la transacción.

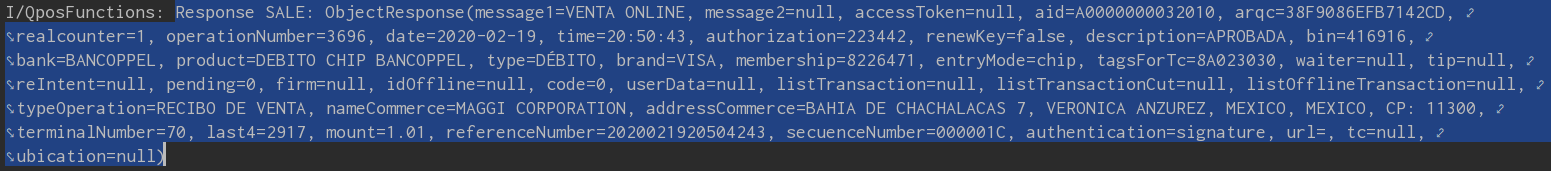
Last4: Los últimos cuatro dígitos de la terminal.

Mount: Monto de la transacción.

ReferenceNumber: Número de referencia de la transacción.

SecuenceNumber: Número de secuencia de la transacción.

Authentication: Mode autenticación de la tarjeta “Firma”, “Pin” o “Sin Firma”.



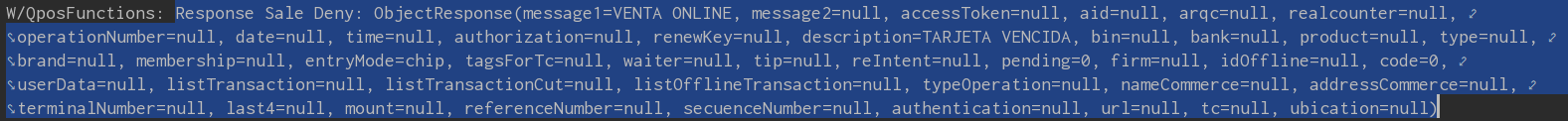
**Figura 10.** Ejemplo del ObjectResponse del método EndTransacción de una transacción online aprobada.

**Transacciones online rechazadas:**

Message1: La descripción de la venta ya sea “Venta Online” o “Venta Offline”.

Description: Motivo del rechazo de la transacción.

EntryMode: Modo de entra de la tarjeta “Chip” o “Banda”.



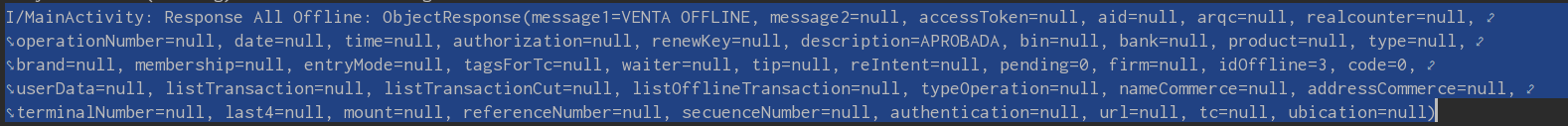
**Figura 11.** Ejemplo del ObjectResponse del método EndTransacción de una transacción online rechazada.

**Transacciones offline aprobadas:**

Message1: La descripción de la venta ya sea “Venta Online” o “Venta Offline”.

Description: Mensaje descriptivo con la transacción.

IdOffline: Identificador de la transacción offline realizada.



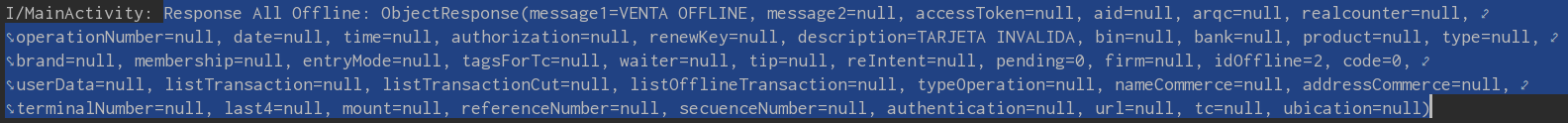
***Figura 12.*** *Ejemplo del* ObjectResponse *del método* EndTransacción *de una transacción offline aprobada.*

**Transacciones offline rechazadas:**

Message1: La descripción de la venta ya sea “Venta Online” o “Venta Offline”.

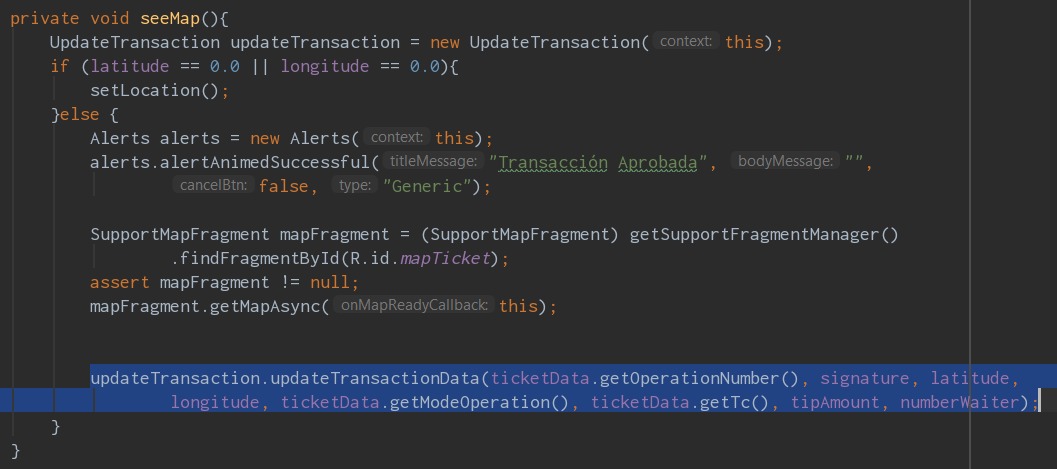
Description: Motivo del rechazo de la transacción.

IdOffline: Identificador de la transacción offline realizada.



**Figura 13.** Ejemplo del ObjectResponse del método EndTransacción de una transacción offline rechazada.

Actualizar transacción (solo en venta)



**Figura 14.** Ejecución del método updateTransaction para la actualización de la transacción.

**Cancelación**

El flujo de una cancelación es similar al flujo de una venta en cuestión de la interacción con el dispositivo dspread, tanto en transacciones con tarjeta CHIP y tarjeta BANDA.

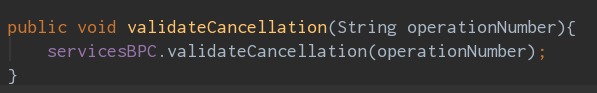
Validar cancelación

Como primer paso se debe de validar si la transacción a cancelar existe, eso se valida con el método *ValidateCancellation* de la librería, el cual solicita el número de operación de la venta y regresa el monto a cancelar, monto con el cual podemos activar el dspread como se realiza en una venta. Este método se implementa en una opción del lado del desarrollo de la aplicación, no dentro de la clase que mantiene toda la operatividad con el dspread (QposFunctions).

1. Consumir el servicio *ValidateCancellation* el cual regresa el parámetro *mount* del objeto ObjectResponse, haciendo referencia al monto de la venta que se desea cancelar.
2. Activar el flujo operativo de lectura de tarjetas (como se hace en la venta) con el método *qpos.doCheckCard(30)*.
3. En el método *qpos.setAmount()* solicita cuatro parámetros:
   1. parámetro: Va el monto que devuelve el servicio *ValidateCancellation* como cadena de texto, sin comas, puntos y símbolos de moneda.
   2. parámetro: Se ingresa el número 0 como cadena de texto.
   3. parámetro: Se ingresa el número 484 como cadena de texto.
   4. parámetro: El tipo de transacción a realizar, en este caso que es una cancelación el tipo es *TransactionType.REFUND*.
4. El flujo operativo es tal cual como la venta, entra al método *OnDoTradeResult()* donde valida que tipo de tarjeta se está detectando con el parámetro *result* de tipo *DoTradeResult*:

Si es tarjeta CHIP el caso será ICC donde se seguirá llamando el método *qpos.doEmvApp()* y en el método *OnRequestOnlineProcess()* se debe validar si la acción que se está realizando es “venta” o “cancelación”, dicha validación puede ser por medio de una bandera que indiquen en su desarrollo. Una vez definida la acción y esta sea ”cancelación”, dentro de ese mismo método se consume el servicio *servicesBPC.cancel()*.

Si es tarjeta BANDA el caso será MCR y solo se validará la acción a realizar “venta” o “cancelación”, en caso de cancelación se aplicará el método *servicesBPC.cancel()*.



**Figura 15.** Implementación del método validateCancellation.

Retail Cancelación

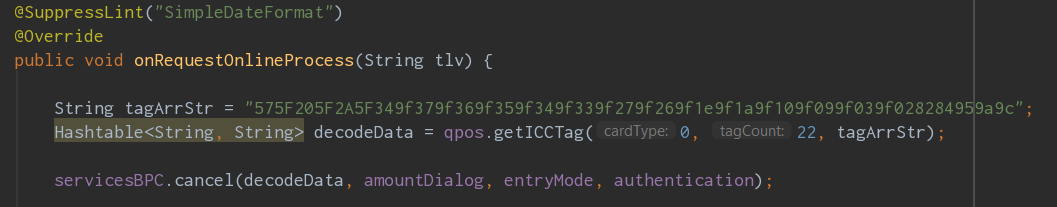
Para culminar la cancelación dependiendo el tipo de tarjeta que se detectó en los lectores del dspread, aplica lo siguiente:

Cuando es tarjeta CHIP: En el método *OnRequestOnlineProcess()* se debe extraer cierta información para la transacción, como se muestra en la figura 12. Primero se debe asignar una variable de tipo HashMap la cual se llena con la función *qpos.getICCTag(0,22, 575F205F2A5F349f379f369f359f349f339f279f269f1e9f1a9f109f099f039f028284959a9c)*, en seguida se llena el método *servicesBPC.cancel()* que termina por realizar la cancelación.

Cuando es tarjeta con BANDA: En el caso MCR del método *OnDoTradeResult()* se debe aplicar el método *servicesBPC.cancel()* siempre y cuando la acción sea cancelación.

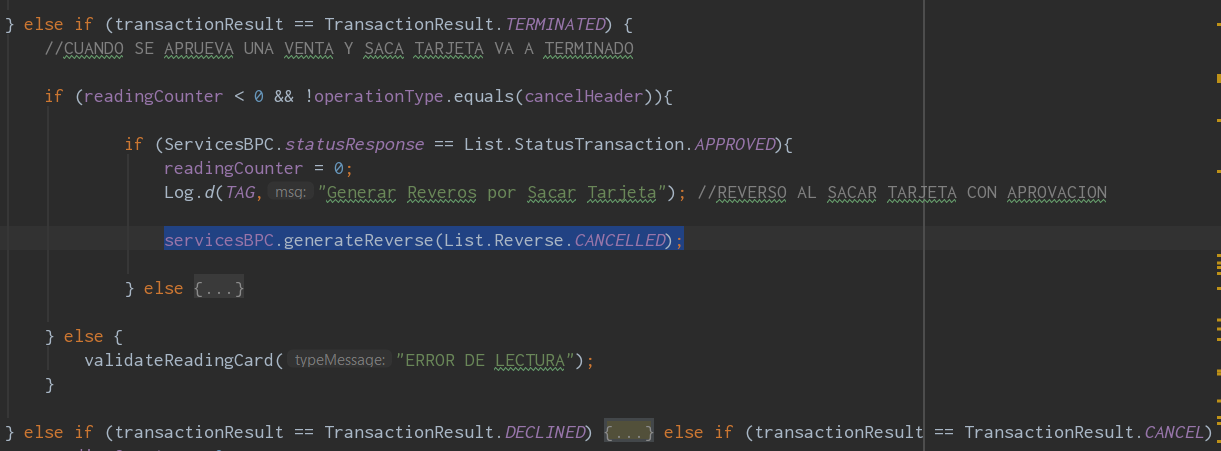
El método *servicesBPC.cancel()* solicita cuatro parámetros:

1. parámetro: Variable HashMap que se llenó anteriormente con la información necesaria para la transacción.
2. parámetro: el monto de la transacción en cadena de texto, sin símbolo de moneda y comas.
3. parámetro: El modo de entrada de la tarjeta List.EntryMode.CHIP ó List.EntryMode.MAGNETIC\_STRIPE.
4. parámetro: La autenticación que se utilizó para la tarjeta List.Authentication.SIGNATURE ó List.Authnetication.PIN.



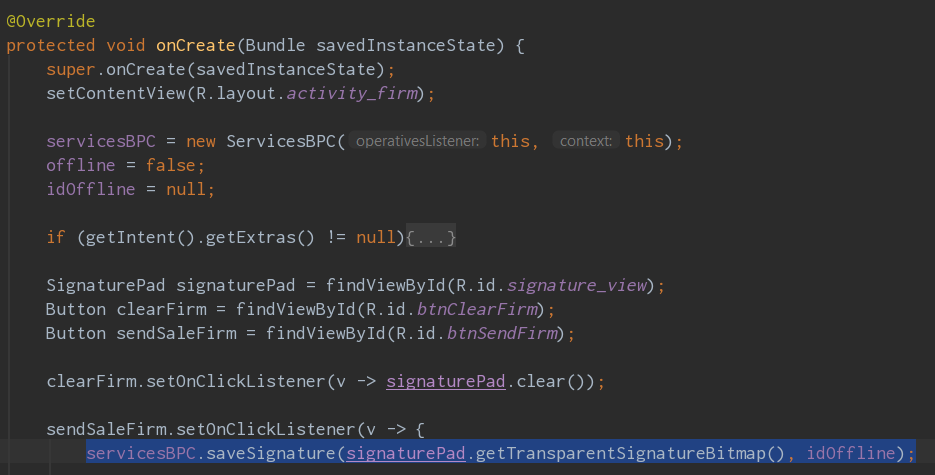
**Figura 16.** Instancia del método cancel que realiza la cancelación/devolución de una venta.

Reverso



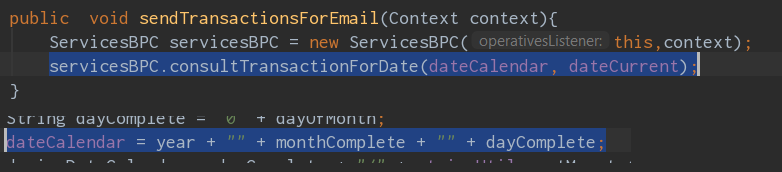
**Figura 17.** Implementación del método generateReverso para generar el reverso de una transacción en curso el cual requiere un parámetro especificando el tipo de reveso.

Almacenar firma



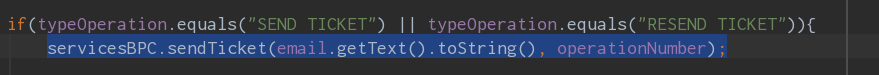
**Figura 18.** Implementación de método saveSignature el cual recibe un bitmap de la firma y el id de una transacción offline en caso de que aplique, si no puede ir nulo.

Consultar transacciones por rango de fecha vía correo electrónico



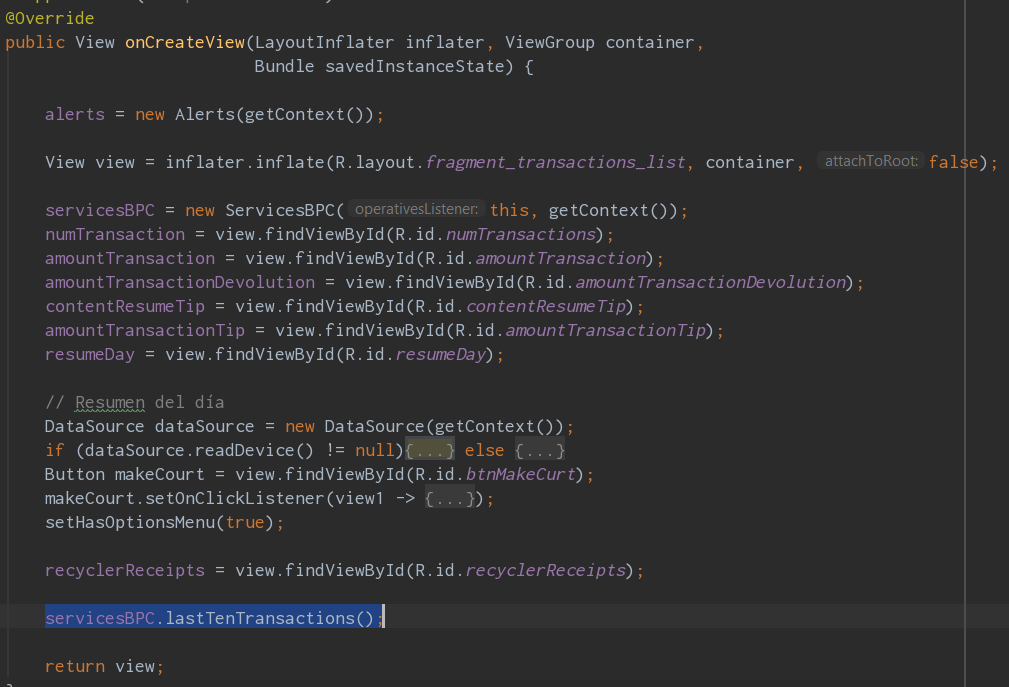
**Figura 19.** Implementación de consultTransactionsForDate donde cada parámetro vale año, mes y día concatenados sin espacios y símbolos de separación.

Enviar ticket por correo electrónico



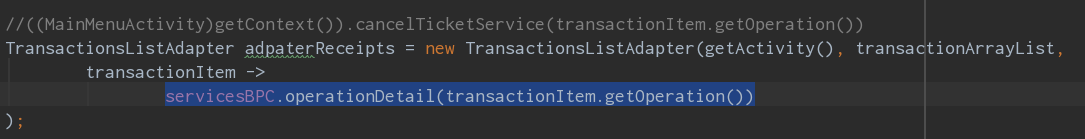
**Figura 20.** Ejemplo de implementación del método sendTicket el cual requiere el correo electrónico al que será enviado el ticket y el número de operación de dicha transacción.

Lista de las ultimas 10 transacciones realizadas



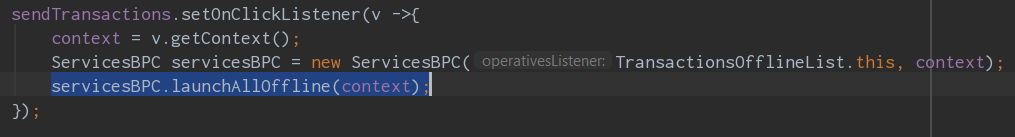
**Figura 21.** Consulta de transacciones el cual solo devuelve el número de operación, monto, código de autorización, los últimos 4 dígitos de la tarjeta y fecha de cada transacción.

Detalle de una transacción en específico



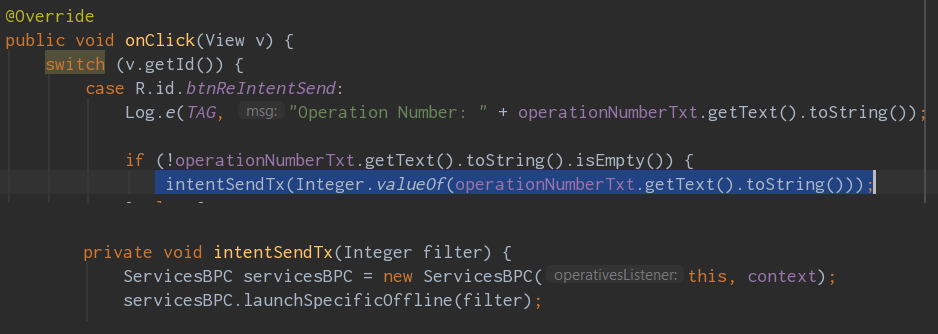
**Figura 22.** Implementación del método operationDetail el cual requiere el número de operación de la transacción a consultar.

Realizar todas las ventas offline



**Figura 23.** Por medio de una acción se activa el lanzamiento de todas las transacciones offline realizadas anteriormente siempre y cuando haya internet, si no regresa un mensaje el cual indica si el dispositivo sigue sin servicio de internet.

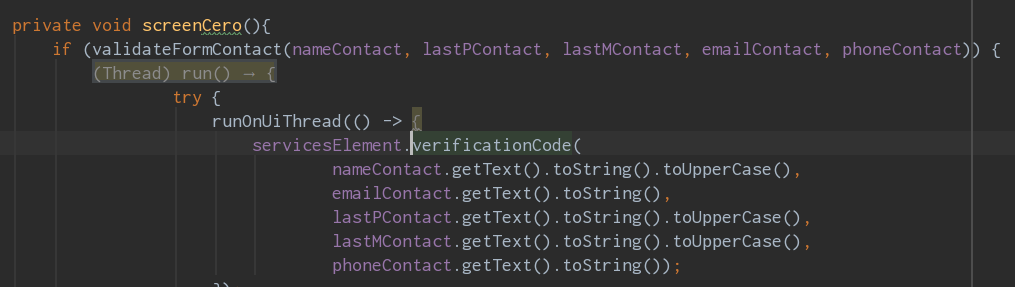
Reintentar una venta offline en específico



**Figura 24.** Lanzar una venta offline en específico con solo el número de operación de la transacción.

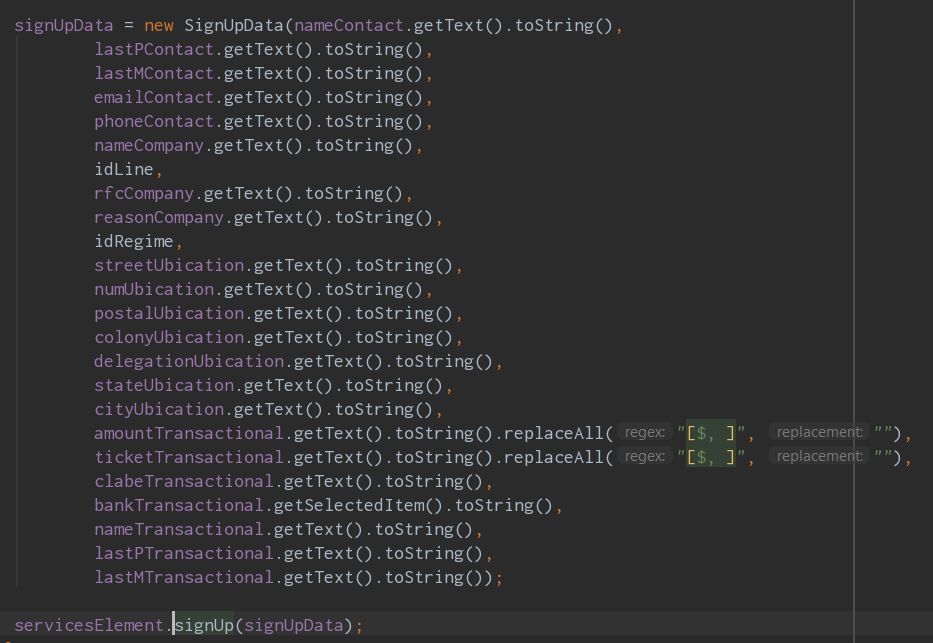
Controlador de servicios Element

Código de verificación



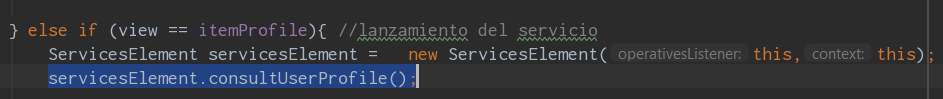
**Figura 25.** Ejecución del método verificationCode para obtener el código de verificación y validar el flujo del registro.

Registro completo



**Figura 26.** Desarrollo del método signUp para el registro completo.

Consulta de perfil del usuario

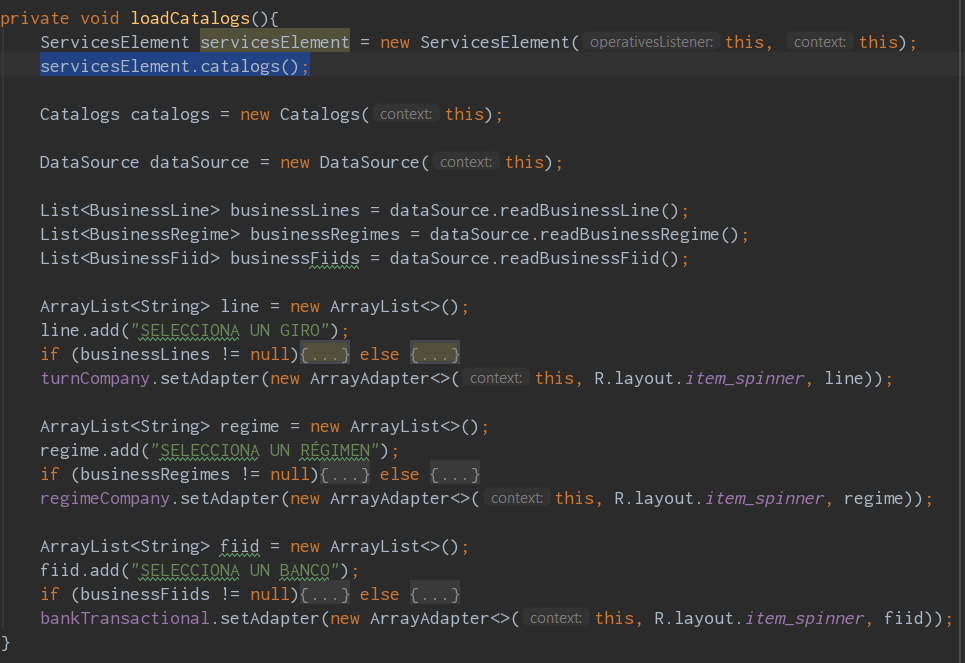


**Figura 27.** Ejecución del método consultUserProfile.

Carga de catálogos

Para el registro completo se requiere el **id** del las opciones de los catálogos de régimen, giro y banco, que se almacena temporalmente en la base de datos del teléfono ya que es necesario la actualización de los mismos.

***Nota:*** *El metodo loadCatalogs es un metodo creado por Blumon Pay para separar el código de carga de catalogos en los combo box.*



**Figura 28.** Implementación de la carga de catálogos por medio del método catalogs