Manual de Integração Android Moderninha Smart

Sumário

Versionamento deste documento 1

Introdução 3

Observações Importantes 4

Importando a biblioteca PlugPagService Wrapper 5

AndroidManifest.xml 7

Permissões 7

Intent-filter 7

Classes 8

Interfaces 10

API 12

PlugPag 12

Constantes 12

Construtores 13

Métodos 15

PlugPagAbortResult 23

Construtores 23

Métodos 23

PlugPagActivationData 23

Construtores 23

PlugPagAppIdentification 24

Construtores 24

Métodos 24

PlugPagBeepData 24

Constantes 24

Construtores 26
PlugPagException 26
Construtores 26
PlugPagInitializationResult 27
Construtores 27
PlugPagEventData 27
Constantes 27
Construtores 31
Métodos 31
Métodos estáticos 31
PlugPagLedData 31
Constantes 32
Construtores 32
PlugPagNFCAuth 33
Construtores 33
PlugPagNearFieldCardData 33
Constantes 33
PlugPagSimpleNFCData 34
Construtores 34
PlugPagNFCInfosResult 34
Construtores 35
PlugPagPaymentData 35
Construtores 35
Métodos 36

PlugPagPaymentData.Builder 37
Construtores 37
Métodos 37
PlugPagTransactionResult 39
Construtores 40
Métodos 40
PlugPagTransactionResult.Builder 43
Construtores 43
Métodos 43
PlugPagVoidData 47
Construtores 47
Métodos 47
PlugPagVoidData.Builder 48
Construtores 48
Métodos 48
PlugPagCustomPrinterLayout 49
Construtor 49
Métodos 49
PlugPagNFCResult 50
Construtor 50
Métodos 50
PlugPagPrintResult 52
Construtor 52
Métodos 52

PlugPagPrinterData 52
Construtor 52
Métodos 53
PreferredNetwork 53
Constantes 53
TerminalCapabilities 54
Constantes 54
PlugPagPrinterListener 56
Métodos 56
PlugPagEventListener 57
Métodos 57
PlugPagPaymentListener 57
Métodos 57
PlugPagNFCListener 58
Métodos 58
PlugPagLastTransactionListener 59
Métodos 59
PlugPagIsActivatedListener 59
Métodos 59
PlugPagInstallmentsListener 60
Métodos 60
PlugPagActivationListener 60
Métodos 60
PlugPagAbortListener 61

Métodos 61
Exemplos 62
Pagamentos 63
Estornar um pagamento 71
Verificar autenticação 72
Invalidar autenticação 73
Solicitar ativação 74
Obter versão da biblioteca 76
Customizar dialog de impressão da via do cliente 80
Exemplos assíncronos 88
Pagamentos 88
Estornar um pagamento 97
Verificar autenticação 99
Invalidar autenticação 100
Solicitar ativação 102
Códigos de retorno 112
Códigos de erro de impressão 116

Versionamento deste documento

Versão doc.	Data	Autor	Descrição	Versão PlugPag Service
1.0.0	28/01/2019	Carlos Vaccari	Criação do documento.	1.0
1.0.1	06/03/2019	Cássio Corrêa	Ajuste layout da capa	1.0
1.0.2	06/03/2019	Cássio Corrêa	Movido seção observações	1.0
1.0.3	14/03/2019	Hugo Yamashita	Correção de exemplos	1.0.7
1.0.4	15/03/2019	Hugo Yamashita	Revisão do documento	1.0.7
1.0.5	19/03/2019	Carlos Vaccari	Adição de NFC	1.1.0
1.0.6	21/03/2019	Carlos Vaccari	Adição de dialog de impressão customizável	1.0.7
1.0.7	05/04/2019	Carlos Vaccari	Correção de exemplo. Adição de observação.	1.1.0
1.0.8	08/04/2019	Carlos Vaccari	Adição de impressão livre.	1.1.0
1.0.9	11/04/2019	Hugo Yamashita	Inclusão da classe PlugPagNearFieldCardData	1.1.1
1.0.10	11/04/2019	Lucas Silva	Adição dos eventos de digitação de PIN na PlugPagEventData	1.2.0

1.0.11	14/05/2019	Carlos Vaccari	Inclusão de exemplo de busca de última transação	1.3.0
1.0.12	28/05/2019	Carlos Vaccari	Inclusão de abort do NFC	1.3.3
1.0.13	01/11/2019	Carlos Vaccari	Inclusão de novos códigos de eventos	1.5.1
1.0.14	01/11/2019	Carlos Vaccari	Inclusão de métodos assíncronos no wrapper	1.6.0
1.0.15	27/11/2019	Lucas Silva	Atualizando listeners, data, terminal capabilities, prefered network, custom printer, activation data	1.6.0
1.0.16	02/12/2019	Carlos Vaccari	Adicionando listeners dos métodos assíncronos	1.6.0

Introdução

Este documento destina-se a integradores que utilizarão o terminal **Moderninha Smart** do PagSeguro como solução de pagamento integrada através do serviço **PlugPagService**.

O serviço **PlugPagService** pode ser acessado diretamente ou por meio da biblioteca **PlugPagService Wrapper**. Este documento destina-se à explicação do uso da biblioteca **PlugPagService Wrapper**.

Observações Importantes

A biblioteca **PlugPagService** para o sistema operacional Android possui algumas restrições para seu uso.

- A biblioteca PlugPag possui suporte da API level 25 (7.1.1 Nougat) à 28 (9.0 Pie), devido a versão de Android presente no terminal **Moderninha Smart**.
- Apenas uma única instância do PlugPag deve existir durante o uso do aplicativo. A existência de múltiplas instâncias pode fazer com que o comportamento seja indeterminado.
- As chamadas dos métodos da classe `PlugPag` devem ser feitas em uma `Thread` que execute em background pois podem demorar para finalizar a execução. Caso a execução seja feita na `Thread` principal (`UI Thread`), o aplicativo pode apresentar um ANR (Application Not Responding). Além disso, alguns métodos executam transações utilizando chamadas remotas pela internet, o que impossibilita suas chamadas na `Thread` principal.
- Eventos que chamem duas ou mais vezes o serviço de pagamento ou estorno antes da operação ser finalizada, podem ocasionar comportamento anormal no serviço e requerer que a aplicação seja fechada para realizar o *unbind* do serviço.

Importando a biblioteca PlugPagService Wrapper

Para importar a biblioteca **PlugPagService Wrapper** na sua aplicação nativa Android basta seguir os passos descritos abaixo:

1- Inserir no arquivo *build.gradle* do projeto a URL do repositório Maven do PlugPag:

```
allprojects {
 repositories {
  ...
  maven {
   url 'https://github.com/pagseguro/PlugPagServiceWrapper/raw/master'
  }
  ...
 }
}
```

2- Inserir as dependências no arquivo *build.gradle* da aplicação:

```
dependencies {
    ...
    implementation 'com.android.support:design:28.0.0'

    implementation 'br.com.uol.pagseguro.plugpagservice.wrapper:wrapper:1.6.0'
    ...
}
```

A versão da dependência `com.android.support:design` deve ser a mesma utilizada para as demais dependências `com.android.support`. A versão `28.0.0` é a mais recente no momento da edição desse documento.

AndroidManifest.xml

Permissões

Para integrar a biblioteca a biblioteca PlugPagService em aplicativos para Android é necessário adicionar a seguinte permissão ao AndroidManifest.xml.

```
<permission android:name="br.com.uol.pagseguro.permission.MANAGE_PAYMENTS"/>
```

Essa permissão permite à biblioteca realizar o bind ao **PlugPagService**, serviço embarcado da **Moderninha Smart**, que gerencia todas as transações de pagamento.

Intent-filter

Para que seu aplicativo possa ser escolhido como aplicativo padrão de pagamento e receber `Intent`s de inserção de cartão, é necessário adicionar o seguinte código em seu *AndroidManifest.xml* dentro da sua `Activity` principal.

```
<intent-filter>
    <action android:name="br.com.uol.pagseguro.PAYMENT"/>
    <category android:name="android.intent.category.DEFAULT"/>
</intent-filter>
```

Classes

A biblioteca **PlugPagService** é composta de um conjunto de classes.

A classe principal chama-se `PlugPag`, mas é necessário utilizar classes auxiliares para configurações e trocas de informações.

Segue abaixo uma lista com classes que compõem a biblioteca.

Classe	Descrição
PlugPag	Classe principal da biblioteca. Essa classe é responsável pelas transações.
PlugPagAbortResult	Resultado obtido ao solicitar um cancelamento de operação, enquanto a operação está em andamento.
PlugPagActivationData	Dados de ativação do terminal
PlugPagAppIdentification	Identificação do aplicativo.
PlugPagBeepData	Dados de frequência e duração
PlugPagCustomPrinterLayout	Classe para customização da dialog de impressão da via do cliente.
PlugPagEventData	Dados de eventos gerados durante transações para atualização de eventos no aplicativo.
PlugPagException	Tipo principal de exceções geradas pelo PlugPag.
PlugPagInitializationResult	Resultado da inicialização do terminal
PlugPagLedData	Dados de cor do led

PlugPagNearFieldCardData	Slots a serem lidos/escritos pelo NFC.
PlugPagNFCAuth	Dados de autenticação do NFC
PlugPagNFCInfosResult	Resultado de uma requisição para obter informações de NFC
PlugPagNFCResult	Resultado de uma leitura/escrita NFC
PlugPagPaymentData	Informações de um pagamento a ser realizado.
PlugPagPrinterData	Informações de uma impressão a ser realizada
PlugPagPrintResult	Resultado de uma requisição de impressão.
PlugPagSimpleNFCData	Dados para escrita em um cartão NFC
PlugPagTransactionResult	Resultado de uma transação.
PlugPagVoidData	Informações de um estorno a ser realizado.
PreferedNetwork	Enum com valores de conexão
TerminalCapabilities	Constantes do capabilities

Interfaces

As interfaces visam facilitar e padronizar algumas chamadas de métodos de forma assíncrona.

Interface	Descrição
PlugPagAbortListener	Interface com métodos que são chamados durante uma transação assíncrona de abort
PlugPagActivationListener	Interface com métodos que são chamados durante uma ativação e/ou durante uma desativação assíncrona
PlugPagInstallmentsListener	Interface com métodos que são chamados ao realizar o cálculo de parcelas de forma assíncrona
PlugPagIsActivatedListener	Interface com métodos que são chamados durante uma verificação se está ou não ativado assíncrona
PlugPagLastTransactionListener	Interface com métodos que são chamados ao tentar obter de forma assíncrona a última transação aprovada
PlugPagNFCListener	Interface com métodos que são chamados durante uma leitura ou uma escrita assíncrona em um cartão NFC
PlugPagPaymentListener	Interface com métodos que são chamados durante uma transação assíncrona de pagamento
PlugPagEventListener	Interface com método chamado quando um evento é enviado durante uma transação.
PlugPagPrinterListener	Listener que retornar informações de erro durante uma impressão

API

Abaixo segue a descrição da interface pública da biblioteca **PlugPagService**.

PlugPag

Essa é a classe principal da biblioteca.

É por meio dessa classe que é possível realizar transações na **Moderninha Smart**.

Constantes

int	RET_OK Código utilizado para indicar sucesso nas operações. Valor: 0
int	REQUEST_CODE_AUTHENTICATION Código utilizado para iniciar a `Activity` de autenticação. Valor: 46981
int	TYPE_CREDITO Tipo de pagamento: crédito. Valor: 1
int	TYPE_DEBITO Tipo de pagamento: débito Valor: 2

int	TYPE_VOUCHER Tipo de pagamento: voucher (vale refeição) Valor: 3
int	INSTALLMENT_TYPE_A_VISTA Forma de parcelamento: à vista Valor: 1
int	INSTALLMENT_TYPE_PARC_VENDEDOR Forma de parcelamento: parcelamento vendedor Valor: 2
int	ERROR_REQUIREMENTS_MISSING_PERMISSIONS Código de retorno para indicar erro de falta de permissões do aplicativo. Valor: -3000
int	ERROR_REQUIREMENTS_ROOT_PERMISSION Código de retorno para indicar que o aparelho possui permissões de root. Valor: -3001

Construtores

PlugPag(Context context, PlugPagAppIdentification appIdentification)

Cria uma instância do `PlugPag` utilizando `context` para acessar dados e recursos do dispositivo e identificando as transações com os dados do aplicativo fornecidos em `appIdentification`.

Gera uma exceção se `context` ou `appIdentification` forem nulos.

Métodos

Tipo de retorno	Método e descrição
PlugPagAbortResult	abort() Solicita o cancelamento da operação atual. O cancelamento da transação **não** ocorre instantaneamente, pois depende do fluxo da transação. Retorna o resultado da solicitação de cancelamento.
PlugPagNFCResult	abortNFC() Aborta uma operação de leitura/escrita NFC.
Int	authNFCCardDirectly(authData: PlugPagNFCAuth) Realiza a autenticação do sistema NFC usado. Retorna sucesso com código 1 e falha com código -1.
Void	asyncAbort(listener: PlugPagAbortListener) Solicita o cancelamento da operação atual de forma assíncrona.
Void	asyncAbortNFC(listener: PlugPagAbortListener) Aborta uma operação de leitura/escrita NFC de forma assíncrona.
Void	asyncCalculateInstallments(saleValue: String, listener: PlugPagInstallmentsListener) Calcula o valor das parcelas de forma assíncrona.

Void	asyncDeactivate(activationData: PlugPagActivationData, listener: PlugPagActivationListener) Realiza a desativação do terminal de forma assíncrona.
Void	asyncGetLastApprovedTransaction(listener: PlugPagLastTransactionListener) Obtém a última transação aprovada de forma assíncrona.
Void	asyncIsAuthenticated(isActivatedListener: PlugPagIsActivatedListener) Verifica se há um usuário autenticado de forma assíncrona.
Void	asyncReadNFC(cardData: PlugPagNearFieldCardData, listener: PlugPagNFCListener) Realiza leitura do conteúdo de um cartão NFC de forma assíncrona.
Void	asyncReprintCustomerReceipt(listener: PlugPagPrinterListener) Realiza a reimpressão da via do cliente de forma assíncrona.
Void	asyncReprintEstablishmentReceipt(listener: PlugPagPrinterListener) Realiza a reimpressão da via do estabelecimento de forma assíncrona.
Void	asyncWriteNFC(cardData: PlugPagNearFieldCardData, listener: PlugPagNFCListener) Realiza escrita em um cartão NFC de forma assíncrona.

Int	beep(beepData: PlugPagBeepData) Toca um beep Retorna sucesso com código 1 e falha com código -1.
Array<String>	calculateInstallments(saleValue: String) Calcula o valor das parcelas Retorna os valores das parcelas
PlugPagTransactionResult	doPayment(PlugPagPaymentData paymentData) Efetua um pagamento. Retorna o resultado da transação.
PlugPagInitializationResult	deactivate(activationData: PlugPagActivationData) Realiza a desativação do terminal Retorna o resultado de uma desativação
Void	disposeSubscriber() Realiza o dispose do subscriber atual.
Void	doAsyncInitializeAndActivatePinpad(activationData: PlugPagActivationData, listener: PlugPagActivationListener) Realiza a inicialização e a ativação do terminal para uso através do código de ativação de forma assíncrona.
Void	doAsyncVoidPayment(voidData: PlugPagVoidData, listener: PlugPagPaymentListener) Efetua um estorno de forma assíncrona de um pagamento identificado pelos dados contidos em `voidData`

PlugPagAppIdentification	getAppIdentification() Retorna a identificação do aparelho definido no construtor da classe.
String	getApplicationCode() Retorna o código da aplicação. Esse código é uma constante da biblioteca.
PlugPagTransactionResult	getLastApprovedTransaction() Obtém a última transação aprovada.
String	getLibVersion() Retorna a versão da biblioteca PlugPagService.
PlugPagNFCInfosResult	getNFCInfos(cardType: Int) Obtém as informações de um cartão NFC Retorna o as informações de cartão NFC
Boolean	hasCapability(capability: Int) Verifica se o terminal tem uma funcionalidade especifica. Retorna `true` se houver a funcionalidade, `false` caso contrário.
int	initBTConnection(PlugPagDevice deviceInformation) Configura a conexão bluetooth utilizando os dados de `deviceInformation`. Retorna `PlugPag.RET_OK` em caso de sucesso.

PlugPagInitializationResult	initializeAndActivatePinpad(activationData: PlugPagActivationData) Realiza a inicialização e a ativação do terminal para uso através do código de ativação. Retorna o resultado da inicialização.
void	invalidateAuthentication() Invalida uma autenticação. Equivalente a realizar um logout.
boolean	isAuthenticated() Verifica se há um usuário autenticado. Retorna `true` se houver um usuário autenticado, `false` caso contrário.
PlugPagPrintResult	printFromFile(printerData: PlugPagPrinterData) Realiza a impressão de um arquivo
PlugPagPrintResult	reprintCustomerReceipt() Realiza a reimpressão da via do cliente
PlugPagPrintResult	reprintStablishmentReceipt() Realiza a reimpressão da via do estabelecimento
void	requestAuthentication(PlugPagAuthenticationListener listener) Solicita autenticação. O resultado da autenticação é notificado ao listener que é passado no parâmetro `listener`.

void	setEventListener(PlugPagEventListener listener) Armazena a referência de uma instância de interface que receberá os eventos gerados durante as transações. Os eventos são gerados apenas para transações feitas utilizando um leitor.
Int	setLed(ledData: PlugPagLedData) Acede o Led Retorna sucesso com código 1 e falha com código -1.
void	setPlugPagCustomPrinterLayout(PlugPagCustomPrinterLayout). Permite customizar elementos da dialog de impressão da via do cliente.
Boolean	setPreferedNetwork(preferedNetwork: Int) Configura a tipo de rede Preferido com 1 (4G/3G/2G), 2 (3G/2G), 3 (2G) Retorna sucesso com true e falha com false.
void	setPrinterListener(listener: PlugPagPrinterListener) Armazena a referência de uma instância de interface que receberá os eventos gerados durante uma impressão.

int	setVersionName(String appName, String appVersion) Define o nome e a versão do aplicativo que está integrando com o `PlugPagService`. `appName` pode ter no máximo 25 caracteres. `appVersion` pode ter no máximo 10 caracteres. Retorna um código de erro se um dos parâmetros for nulo ou vazio.
Int	startNFCCardDirectly() Inicia o sistema NFC para uso Retorna sucesso com código 1 e falha com código -1.
Int	stopNFCCardDirectly() Finaliza o sistema de NFC em uso Retorna sucesso com código 1 e falha com código -1.
PlugPagNFCResult	readFromNFCCard(PlugpagNearFieldCardData cardData) Realiza leitura do conteúdo de um cartão NFC. Retorna sucesso com código 1 e falha com código -1.
Int	readNFCCardDirectly(cardData: PlugPagSimpleNFCData) Realiza leitura do conteúdo de um cartão NFC diretamente. Retorna sucesso com código 1 e falha com código -1.

PlugPagTransactionResult	voidPayment(PlugPagVoidData voidData) Efetua um estorno de um pagamento identificado pelos dados contidos em `voidData`. Retorna o resultado da transação.
PlugPagNFCResult	writeToNFCCard(PlugpagNearFieldCardData cardData) Realiza escrita em um cartão NFC. Retorna sucesso com código 1 e falha com código -1.
Int	writeToNFCCardDirectly(cardData: PlugPagSimpleNFCData) Realiza escrita em um cartão NFC diretamente. Retorna sucesso com código 1 e falha com código -1.

PlugPagAbortResult

Essa classe contém dados resultantes de uma solicitação de cancelamento de operação.

Construtores

PlugPagAbortResult(int result)

Cria um container de dados resultantes de um cancelamento de operação com o código `result`.

Métodos

int	getResult() Retorna o código de resultado da solicitação de cancelamento de operação.

PlugPagActivationData

Essa classe contém dados para realizar uma ativação

Construtores

PlugPagActivationData(String activationCode)

Cria um conjunto de informações necessários para iniciar uma ativação

PlugPagAppIdentification

Essa classe representa a identificação de um aplicativo.

Construtores

PlugPagAppIdentification(String name, String version)

Cria uma identificação do aplicativo, definindo seu nome e sua versão com os valores de `name` e `version`, respectivamente.

Gera uma exceção se `name` ou `version` forem nulos ou vazios.

Métodos

String	getName() Retorna o nome do aplicativo.
String	getVersion() Retorna a versão do aplicativo.

PlugPagBeepData

Essa classe representa os dados de um beep

Constantes

byte	FREQUENCE_LEVEL_0 Código utilizado para frequência 0. As frequências variam de 0 (grave) a 6 (agudo). Valor: 0

---

byte	FREQUENCE_LEVEL_1 Código utilizado para frequência 1. As frequências variam de 0 (grave) a 6 (agudo). Valor: 1
byte	FREQUENCE_LEVEL_2 Código utilizado para frequência 2. As frequências variam de 0 (grave) a 6 (agudo). Valor: 2
byte	FREQUENCE_LEVEL_3 Código utilizado para frequência 3. As frequências variam de 0 (grave) a 6 (agudo). Valor: 3
byte	FREQUENCE_LEVEL_4 Código utilizado para frequência 4. As frequências variam de 0 (grave) a 6 (agudo). Valor: 4
byte	FREQUENCE_LEVEL_5 Código utilizado para frequência 5. As frequências variam de 0 (grave) a 6 (agudo). Valor: 5

byte	FREQUENCE_LEVEL_6 Código utilizado para frequência 5. As frequências variam de 0 (grave) a 6 (agudo). Valor: 6

Construtores

PlugPagBeepData(byte frequency, int duration)

Cria um conjunto de informações necessários para executar um beep

Gera uma exceção se frequency não for uma das constantes.

Gera uma exceção se duration for menor que 0.

PlugPagException

Essa classe representa uma exceção PlugPag

Construtores

PlugPagException()

Cria um conjunto de informações necessários para uma exceção

PlugPagException(String message)

Cria um conjunto de informações necessários para uma exceção

PlugPagException(Throwable cause)

Cria um conjunto de informações necessários para uma exceção

PlugPagException(String message, String errorCode)

Cria um conjunto de informações necessários para uma exceção

PlugPagException(String message, Throwable cause, String errorCode)

Cria um conjunto de informações necessários para uma exceção

PlugPagInitializationResult

Essa classe contém os dados de um resultado de uma ativação ou desativação.

Construtores

PlugPagInitializationResult(int result, String errorCode, String errorMessage)

Cria um objeto para armazenar um conjunto de informações resultantes de uma ativação ou desativação.

PlugPagEventData

Essa classe representa um evento gerado pela biblioteca `PlugPag` para o aplicativo de integração.

Constantes

int	EVENT_CODE_CUSTOM_MESSAGE
	Código de evento indicando mensagem customizada pela PlugPag
	Valor: -2

int	EVENT_CODE_DEFAULT Código padrão de evento. Utilizado quando nenhum evento foi enviado. Valor: -1
int	EVENT_CODE_WAITING_CARD Código de evento indicando que o leitor está aguardando o usuário inserir o cartão. Valor: 0
int	EVENT_CODE_INSERTED_CARD Código de evento indicando que o cartão foi inserido. Valor: 1
int	EVENT_CODE_PIN_REQUESTED Código de evento indicando que o leitor está aguardando o usuário digitar a senha. Valor: 2
int	EVENT_CODE_PIN_OK Código de evento indicando que a senha digitada foi validada com sucesso. Valor: 3
int	EVENT_CODE_SALE_END Código de evento indicando o fim da transação. Valor: 4

int	EVENT_CODE_AUTHORIZING Código de evento indicando que o terminal está aguardando autorização da senha digitada para prosseguir com a transação. Valor: 5
int	EVENT_CODE_INSERTED_KEY Código de evento indicando que a senha foi digitada. Valor: 6
int	EVENT_CODE_WAITING_REMOVE_CARD Código de evento indicando que o terminal está aguardando o usuário remover o cartão. Valor: 7
int	EVENT_CODE_REMOVED_CARD Código de evento indicando que o cartão foi removido do terminal. Valor: 8
int	EVENT_CODE_CVV_REQUESTED Código de evento indicando que foi solicitado o CVV. Valor: 9
int	EVENT_CODE_CVV_OK Código de evento indicando que o CVV foi inserido corretamente. Valor: 10

int	EVENT_CODE_CAR_BIN_REQUESTED Código de evento indicando que foi solicitado o BIN. Valor: 11
int	EVENT_CODE_CAR_BIN_OK Código de evento indicando que o BIN foi inserido corretamente. Valor: 12
int	EVENT_CODE_CAR_HOLDER_REQUESTED Código de evento indicando que foi solicitado o CVV. Valor: 13
int	EVENT_CODE_CAR_HOLDER_OK Código de evento indicando que o HOLDER foi inserido corretamente. Valor: 14
int	EVENT_CODE_ACTIVATION_SUCCESS Código de evento indicando que a ativação foi feita corretamente. Valor: 15
int	EVENT_CODE_DIGIT_PASSWORD Código de evento indicando que a um número da senha foi digitado. Valor: 16
int	EVENT_CODE_NO_PASSWORD Código de evento indicando que a senha foi apagada. Valor: 17

int	EVENT_CODE_SALE_APPROVED Código de evento indicando a venda foi aprovada. Valor: 18
int	EVENT_CODE_SALE_NOT_APPROVED Código de evento indicando que a venda não foi aprovada. Valor: 19

Construtores

PlugPagEventData(int eventCode) Cria um identificador de evento gerado pela biblioteca para o aplicativo de integração, com o código `eventCode`.

Métodos

int	getEventCode() Retorna o código do evento gerado.

Métodos estáticos

String	getDefaultMessage(int eventCode) Retorna uma mensagem padrão para um dado código de evento. Se o código de evento for inválido, retorna uma mensagem padrão.

PlugPagLedData

Essa classe representa os dados de um Led

Constantes

byte	LED_BLUE Código utilizado para o LED azul. Valor: 8
byte	LED_YELLOW Código utilizado para o LED amarelo. Valor: 4
byte	LED_GREEN Código utilizado para o LED verde. Valor: 2
byte	LED_RED Código utilizado para o LED vermelho. Valor: 1
byte	LED_OFF Código utilizado para desligar o LED. Valor: 0

Construtores

PlugPagLedData (byte led) Cria um conjunto de informações necessários para aceder ou apagar um LED Gera uma exceção se led não for uma das constantes.

PlugPagNFCAuth

Essa classe representa os dados para realizar uma autenticação NFC.

Construtores

PlugPagNFCAuth (int type, byte blockNumber, byte[] password) Cria um conjunto de informações necessários para autenticar NFC.

PlugPagNearFieldCardData

Essa classe representa os dados de cartões NFC

Constantes

int	ISO14443_AB Código utilizado para o tipo ISO14443_AB. Valor: 0
int	EMV_AB Código utilizado para o tipo EMV_AB. Valor: 1
int	ONLY_A Código utilizado para o tipo ONLY_A Valor: 2

int	ONLY_B Código utilizado para o tipo ONLY_B. Valor: 3
int	ONLY_M Código utilizado para o tipo ONLY_M. Valor: 4
String	NFC_PWD Código utilizado para o tipo NFC_PWD. Valor: pwd
String	NFC_DATA Código utilizado para o tipo NFC_DATA. Valor: data

PlugPagSimpleNFCData

Essa classe representa os dados de cartões NFC

Construtores

PlugPagSimpleNFCData(int cardType, int slot, byte[] value) Cria um container de dados de um cartão NFC

PlugPagNFCInfosResult

Essa classe contém dados do resultado das informações de um cartão NFC

Construtores

PlugPagNFCInfosResult(int result, byte cardType, byte cid, byte[] other, Serializable serialNumber)

Cria um container de dados resultantes de uma operação para obter informações de um cartão NFC

PlugPagPaymentData

Essa classe representa os dados de um pagamento.

É nessa classe que são definidas informações de tipo de pagamento, valor a ser pago e parcelas, além e outras informações gerenciais.

Construtores

PlugPagPaymentData(int paymentType, int amount, int installmentType, int installments, String userReference)

Cria um conjunto de informações necessários para iniciar um pagamento. O pagamento configurado será do tipo `paymentType`, com o valor `amount`, com parcelamento do tipo `installmentType`, com `installments` número de parcelas, identificado por `userReference`.

O parâmetro `amount` definido é o valor em centavos a ser pago. Para um pagamento de R$ 1,50, o `amount` deverá ser de `150`.

O valor de `userReference` deve conter apenas letras (não acentuadas) e números. Esse campo é limitado a 10 caracteres.

Gera uma exceção se o `userReference` for nulo ou vazio.

PlugPagPaymentData(int paymentType, int amount, int installmentType, int installments, String userReference, Boolean printReceipt)

Cria um conjunto de informações necessários para iniciar um pagamento. O pagamento configurado será do tipo `paymentType`, com o valor `amount`, com parcelamento do tipo `installmentType`, com `installments` número de parcelas, identificado por `userReference`.

O parâmetro `amount` definido é o valor em centavos a ser pago. Para um pagamento de R$ 1,50, o `amount` deverá ser de `150`.

O valor de `userReference` deve conter apenas letras (não acentuadas) e números. Esse campo é limitado a 10 caracteres.

O parâmetro `printReceipt` indicará se deverá ser impresso os comprovantes da transação.

Gera uma exceção se o `userReference` for nulo ou vazio.

Métodos

int	getAmount() Retorna o valor a ser pago, em centavos.
int	getInstallments() Retorna o número de parcelas do pagamento.
int	getInstallmentType() Retorna o tipo de parcelamento. Valores: `PlugPag.INSTALLMENT_TYPE_A_VISTA` ou `PlugPag.INSTALLMENT_TYPE_PARC_VENDEDOR`

int	getType() Retorna o tipo de pagamento. Valores: `PlugPag.TYPE_CREDITO`, `PlugPag.TYPE_DEBITO` ou `PlugPag.TYPE_VOUCHER`.
String	getUserReference() Retorna o código de venda.

PlugPagPaymentData.Builder

Construtor de objetos `PlugPagPaymentData`.

Construtores

Builder() Cria um construtor de objetos `PlugPagPaymentData`.

Métodos

PlugPagPaymentData	build() Cria um `PlugPagPaymentData` com os dados armazenados no `Builder`.

Builder	setAmount(int amount) Define o valor a ser pago. Retorna a referência do próprio `Builder` para chamadas encadeadas. O valor de `amount` deve ser fornecido em centavos. Por exemplo, se o valor desejado é de R$1,50, deve-se passar o valor `150`. Gera uma exceção se `amount` não for maior do que zero.
Builder	setInstallments(int installments) Retorna a quantidade de parcelas do pagamento. Retorna a referência do próprio Builder para chamadas encadeadas. Se `installments` for igual a 1, o tipo de parcelamento é automaticamente definido para `PlugPag.INSTALLMENT_TYPE_A_VISTA`. Gera uma exceção se `installments` não for maior do que zero.
Builder	setInstallmentType(int installmentType) Define o tipo de parcelamento. Valores válidos para `installmentType` são `PlugPag.INSTALLMENT_TYPE_A_VISTA` e `PlugPag.INSTALLMENT_TYPE_PARC_VENDEDOR`. Retorna a referência do próprio `Builder` para chamadas encadeadas. Gera uma exceção se `installmentType` for inválido.

Builder	setType(int type) Define o tipo de pagamento. Valores válidos para `type` são `PlugPag.TYPE_CREDITO`, `PlugPag.TYPE_DEBITO` e `PlugPag.TYPE_VOUCHER`. Retorna a referência do próprio `Builder` para chamadas encadeadas. Gera uma exceção se `type` for inválido.
Builder	setUserReference(String userReference) Define o código de venda. Retorna a referência do próprio `Builder` para chamadas encadeadas. O valor de `userReference` deve conter apenas letras (não acentuadas) e números. Esse campo é limitado a 10 caracteres. Gera uma exceção se `userReference` for nulo ou vazio.

PlugPagTransactionResult

Essa classe representa o resultado de uma transação.

Construtores

PlugPagTransactionResult(String message, String transactionCode, String transactionId, String date, String time, String hostNsu, String cardBrand, String bin, String holder, String userReference, String terminalSerialNumber, String amount, String availableBalance, String cardApplication, String cardCryptogram, String label, String holderName, String extendedHolderName) Cria um objeto para armazenar um conjunto de informações resultantes de uma transação.
PlugPagTransactionResult(String message, String errorCode, String transactionCode, String transactionId, String date, String time, String hostNsu, String cardBrand, String bin, String holder, String userReference, String terminalSerialNumber, String amount, String availableBalance, String cardApplication, String cardCryptogram, String label, String holderName, String extendedHolderName, int result) Cria um objeto para armazenar um conjunto de informações resultantes de uma transação, adicionando o código de resultado `result`.

Métodos

String	getAmount() Retorna o valor transacionado.
String	getAvailableBalance() Retorna o saldo da conta, caso o método de pagamento seja `PlugPag.TYPE_VOUCHER`.
String	getBin() Retorna os 4 (quatro) últimos dígitos do cartão utilizado.

String	getCardApplication() Retorna a aplicação do cartão.
String	getCardBrand() Retorna a bandeira do cartão utilizado.
String	getCardCryptogram() Retorna o criptograma do cartão.
String	getDate() Retorna a data da transação.
String	getErrorCode() Se um erro ocorreu durante a transação, retorna o código de erro.
String	getExtendedHolderName() Retorna o nome completo do titular do cartão utilizado.
String	getHolder() Retorna os 4 últimos dígitos do cartão utilizado.
String	getHolderName() Retorna o nome do titular do cartão utilizado.
String	getHostNsu() Retorna um identificador único do host (servidor).
String	getLabel() Retorna o label do cartão utilizado.

String	getMessage() Retorna uma mensagem do resultado da transação, definida pela biblioteca.
int	getResult() Retorna o código do resultado.
String	getTerminalSerialNumber() Retorna o número de série do terminal ou leitor utilizado para efetuar o pagamento.
String	getTime() Retorna o horário da transação.
String	getTransactionCode() Retorna o código da transação.
String	getTransactionId() Retorna o ID da transação.
String	getUserReference() Retorna o código de venda o pagamento efetuado.

PlugPagTransactionResult.Builder

Construtor de objetos `PlugPagTransactionResult`.

Construtores

Builder() Cria um construtor de objetos `PlugPagTransactionResult`.	

Métodos

PlugPagTransactionResult	build() Constrói uma instância da classe `PlugPagTransactionResult` utilizando os dados armazenados.
Builder	setAmount(String amount) Define o valor da transação. Retorna a referência do próprio `Builder` para chamadas encadeadas.
Builder	setAvailableBalance(String availableBalance) Define o saldo disponível. Retorna a referência do próprio `Builder` para chamadas encadeadas.
Builder	setBin(String bin) Define o BIN do cartão utilizado na transação. Retorna a referência do próprio `Builder` para chamadas encadeadas.

Builder	setCardApplication(String cardApplication) Define a aplicação do cartão utilizado na transação. Retorna a referência do próprio `Builder` para chamadas encadeadas.
Builder	setCardBrand(String cardBrand) Define a bandeira do cartão utilizado na transação. Retorna a referência do próprio `Builder` para chamadas encadeadas.
Builder	setCardCryptogram(String cardCryptogram) Define o criptograma do cartão utilizado na transação. Retorna a referência do próprio `Builder` para chamadas encadeadas.
Builder	setDate(String date) Define a data da transação. Retorna a referência do próprio `Builder` para chamadas encadeadas.
Builder	setExtendedHolderName(String extendedHolderName) Define o nome completo do titular do cartão utilizado na transação. Retorna a referência do próprio `Builder` para chamadas encadeadas.

Builder	setHolder(String holder) Define o nome do titular do cartão. Retorna a referência do próprio `Builder` para chamadas encadeadas.
Builder	setHolderName(String holderName) Define o nome do titular do cartão. Retorna a referência do próprio `Builder` para chamadas encadeadas.
Builder	setHostNsu(String hostNsu) Define o NSU do host que executou a transação. Retorna a referência do próprio `Builder` para chamadas encadeadas.
Builder	setLabe(String label) Define o label do cartão utilizado. Retorna a referência do próprio `Builder` para chamadas encadeadas.
Builder	setMessage(String message) Define a mensagem do resultado da transação que será construído. Retorna a referência do próprio `Builder` para chamadas encadeadas.

Builder	setTerminalSerialNumber(String terminalSerialNumber) Define o número de série do terminal ou leitor utilizado na transação. Retorna a referência do próprio `Builder` para chamadas encadeadas.
Builder	setTime(String time) Define o horário da transação. Retorna a referência do próprio `Builder` para chamadas encadeadas.
Builder	setTransactionCode(String transactionCode) Define o código da transação. Retorna a referência do próprio `Builder` para chamadas encadeadas.
Builder	setTransactionId(String transactionId) Define o ID da transação. Retorna a referência do próprio `Builder` para chamadas encadeadas.
Builder	setUserReference(String userReference) Define o código de venda da transação. Retorna a referência do próprio `Builder` para chamadas encadeadas.

PlugPagVoidData

Essa classe representa os dados de um estorno.

É nessa classe que são definidos dados necessários para solicitar o estorno de um pagamento.

Construtores

PlugPagVoidData(String transactionCode, String transactionId, Boolean printReceipt) Cria um conjunto de informações para solicitar o estorno de um pagamento identificado pelo `transactionCode` e `transactionId` fornecidos. O parâmetro `printReceipt` é opcional e indicará se deverá ser impresso os comprovantes da transação. Gera uma exceção se `transactionCode` for nulo ou vazio.
PlugPagVoidData(String transactionCode, String transactionId) Cria um conjunto de informações para solicitar o estorno de um pagamento identificado pelo `transactionCode` e `transactionId` fornecidos. Gera uma exceção se `transactionCode` for nulo ou vazio.

Métodos

String	getTransactionCode() Retorna o código da transação que será estornada.
String	getTransactionId() Retorna o ID da transação que será estornada.

PlugPagVoidData.Builder

Construtor de objetos `PlugPagVoidData`.

Construtores

Builder() Cria um construtor de objetos `PlugPagVoidData`.	

Métodos

PlugPagVoidData	build() Constrói uma instância da classe `PlugPagVoidData` utilizando os dados armazenados.
Builder	setTransactionCode(String transactionCode) Define o código da transação. Retorna a referência do próprio `Builder` para chamadas encadeadas. Gera uma exceção se `transactionCode` for nulo ou vazio.
Builder	setTransactionId(String transactionid) Define o ID da transação. Retorna a referência do próprio `Builder` para chamadas encadeadas.

PlugPagCustomPrinterLayout

Essa classe representa os elementos a serem customizados da dialog de impressão da via do cliente.

Construtor

PlugPagCustomPrinterLayout()

Cria um construtor de objetos `PlugPagCustomPrinterLayout`.

Métodos

void	setButtonBackgroundColor(String hexaCodeColor) Modifica a cor de fundo dos botões da dialog.
void	setButtonBackgroundColorDisabled(String hexaCodeColor) Modifica a cor de fundo dos botões da dialog quando desativados.
void	setCancelTextColor(String hexaCodeColor) Altera a cor do texto do botão de cancelar da dialog.
void	setSendSMSTextColor(String hexaCodeColor) Altera a cor do texto do botão de enviar sms da dialog.
void	setConfirmTextColor(String hexaCodeColor) Altera a cor do texto do botão de confirmar da dialog.
void	setTitle(String titleText) Seta o texto a ser mostrado na dialog.

void	setTitleColor(String hexaCodeColor) Altera a cor do texto mostrado na dialog.
void	setWindowBackgroundColor(String hexaCodeColor) Modifica a cor de fundo da dialog.

Todos os itens são opcionais. Caso não sejam setados, obeterão seus valores defaults.

PlugPagNFCResult

Essa classe representa o retorno de uma leitura ou escrita a um cartão NFC.

Construtor

PlugPagNFCResult(int startSlot, int endSlot, HashMap<String, Byte[]>[] slots, int result)

Métodos

int	getStartSlot() Retorna o primeiro slot a ser escrito/lido
int	getEndSlot() Retorna o último slot a ser escrito/lido.
HashMap<String, Byte[]>[]	getSlots() Retorna as informações as serem escritas/lidas de cada slot. No total, são 64 slots onde, cada slot, possui um HashMap contendo duas informações: data e pwd. Pwd retorna a senha de deste slot e data retorna o valor que foi lido/escrito naquele slot.

int	getResult() Retorna 1 para sucesso e –1 para falha.

PlugPagPrintResult

Essa classe representa o retorno de uma requisição de impressão pelo PlugPagService.

Construtor

PlugPagPrintResult(int result, String message, String errorCode)

Métodos

int	getResult() Retorna PlugPag.*RET_OK* quando sucesso.
int	getMessage() Retorna a mensagem de erro da operação.
String	getErrorCode() Retorna o código de erro da operação.

PlugPagPrinterData

Essa classe representa os dados de uma impressão a ser realizada.

Construtor

PlugPagPrinterData(String filePath, Int printerQuality, Int step)

Métodos

String filePath	getFilePath() Retorna o caminho de arquivo a ser impresso.
int printerQuality	getPrinterQuality() Retorna a qualidade da impressão. Os valores podem variar de 1 a 4, onde 4 indica a maior qualidade da impressão.
int step	getStep() Retorna o espaçamento a ser feito após a impressão terminar.

PreferredNetwork

Essa classe contém as constantes de preferências de conexão.

Constantes

int	NETWORK_ALL Código utilizado para todos os tipos de conexões. Valor: 1
int	NETWORK_3G Código utilizado para conexão 3G Valor: 2
int	NETWORK_2G Código utilizado para conexão 2G Valor: 3

TerminalCapabilities

Essa classe contém as constantes com as funcionalidades do terminal.

Constantes

int	MODULE_MAG Código utilizado para modulo de Leitor tarja Valor: 1
int	MODULE_ICC Código utilizado para modulo de Leitor de chip Valor: 2
int	MODULE_PICC Código utilizado para modulo de ContacLess Valor: 3
int	MODULE_PED Código utilizado para modulo de Teclado seguro Valor: 4
int	MODULE_KEYBOARD Código utilizado para modulo de Teclado externo Valor: 5

int	MODULE_PRINTER Código utilizado para modulo de impressora Valor: 6
int	MODULE_BT Código utilizado para modulo de bluetooth Valor: 7
int	MODULE_CASH_BOX Código utilizado para modulo de Caixa registradora Valor: 8
int	MODULE_CUSTOMER_DISPLAY Código utilizado para modulo de Tela auxiliar Valor: 9
int	MODULE_ETHERNET Código utilizado para modulo de Rede Valor: 10
int	MODULE_FINGERPRINT_READER Código utilizado para modulo de Leitor de digital Valor: 11
int	MODULE_G_SENSOR Código utilizado para modulo de acelerômetro Valor: 12

int	MODULE_HDMI Código utilizado para modulo de Saída hdmi Valor: 13

PlugPagPrinterListener

Essa classe contém os métodos chamados no sucesso ou falha de uma impressão.

Métodos

void	onError(PlugPagPrintResult data) Callback com as informações da impressão no caso de falha.
void	onSuccess(PlugPagPrintResult data) Callback com as informações da impressão no caso de sucesso.

PlugPagEventListener

Essa classe contém o método chamado quando existem novos eventos a serem informados ao integrador sobre o processo de pagamento, estorno, desativação ou ativação.

Métodos

void	onEvent(PlugPagEventData data) Callback que envia ao integrador informações importantes e ações a serem tomadas no processo de pagamento, estorno, desativação ou ativação.

PlugPagPaymentListener

Essa classe contém os métodos chamados durante o processo de pagamento assíncrono.

Métodos

void	onError(PlugPagTransactionResult result) Callback com as informações da pagamento no caso de sucesso.
void	onSuccess(PlugPagTransactionResult result) Callback com as informações da pagamento no caso de sucesso.
void	onPaymentProgress(PlugPagEventData eventData) Callback que envia ao integrador informações importantes e ações a serem tomadas no processo de pagamento, estorno, desativação ou ativação.

void	onPrinterSuccess(PlugPagPrintResult printerResult) Callback com as informações da impressão no caso de sucesso.
void	onPrinterError(PlugPagPrintResult printerResult) Callback com as informações da impressão no caso de falha.
void	onPrinterError(PlugPagPrintResult printerResult)

PlugPagNFCListener

Essa classe contém os métodos chamados no sucesso ou falha de uma escrita/leitura NFC assíncrona.

Métodos

void	onSuccess(PlugPagNFCResult plugPagNFCResult) Callback com as informações da leitura/escrita NFC no caso de sucesso.
void	onError(String errorMessage) Callback com a mensagem de erro da leitura/escrita NFC.

PlugPagLastTransactionListener

Essa classe contém os métodos chamados na requisição assíncrona de última transação aprovada.

Métodos

void	onRequestedLastTransaction(PlugPagTransactionResult result) Callback com as informações da última transação realizada, seja pagamento ou estorno.
void	onError(String errorMessage) Callback com a mensagem de erro da busca pela última transação realizada.

PlugPagIsActivatedListener

Essa classe contém os métodos chamados na verificação assíncrona de status do terminal.

Métodos

void	onIsActivated(Boolean isActivated) Callback com a informação se o terminal está ou não ativado.
void	onError(String errorMessage) Callback com a mensagem de erro ao buscar status do terminal.

PlugPagInstallmentsListener

Essa classe contém os métodos chamados no cálculo de parcelas.

Métodos

void	onCalculateInstallments(Array<String> installments) Callback com informações de cada parcela, seguindo o padrão: “R$ ##,##”.
void	onError(String errorMessage) Callback com a mensagem de erro ao calcular parcelas.

PlugPagActivationListener

Essa classe contém os métodos chamados no processo assíncrono de ativação ou desativação do terminal.

Métodos

void	onActivationProgress(PlugPagEventData eventData) Callback que envia ao integrador informações importantes e ações a serem tomadas no processo de pagamento, estorno, desativação ou ativação.
void	onSuccess(PlugPagInitializationResult result) Callback com as informações da ativação/desativação no caso de sucesso.

void	onError(String errorMessage) Callback com as informações da ativação/desativação no caso de falha.

PlugPagAbortListener

Essa classe contém os métodos chamados no processo assíncrono de abort.

Métodos

void	onAbortRequested(Boolean abortRequested) Callback com informação se o abort foi solicitado com sucesso.
void	onError(String errorMessage) Callback com as informações do erro na requisição de abort.

Exemplos

Seguem abaixo alguns exemplos de camadas dos métodos do **PlugPagService Wrapper** para realizar transações.

As formas de fazer as chamadas e de tratar os valores retornados vão depender da implementação do seu aplicativo, porém é importante que sua aplicação controle ações de duplo clique para prevenir que o serviço seja chamado duas vezes.

Pagamentos

Pagamento de R$250,00, no crédito, à vista:

```
public void startPayment(Context context) {
  // Define os dados do pagamento
  PlugPagPaymentData paymentData =
      new PlugPagPaymentData(
          PlugPag.TYPE_CREDITO,
          25000,
          PlugPag.INSTALLMENT_TYPE_A_VISTA,
          1,
          "CODVENDA");

  // Cria a identificação do aplicativo
  PlugPagAppIdentification appIdentification =
      new PlugPagAppIdentification("MeuApp", "1.0.7");

  // Cria a referência do PlugPag
  PlugPag plugpag = new PlugPag(context, appIdentification);

  // Ativa terminal e faz o pagamento
  int initResult = plugpag.initializeAndActivatePinpad(new
  PlugPagActivationData("SEU_CODIGO_DE_ATIVAÇÃO")));

  if (initResult == PlugPag.RET_OK) {
    PlugPagTransactionResult result = plugpag.doPayment(paymentData);

    // Trata o resultado da transação
    ...
```

```
    }
}
```

Pagamento de R$300, no crédito, parcelado em 3 parcelas:

```
public void startPayment(Context context) {
  // Define os dados do pagamento
  PlugPagPaymentData paymentData =
      new PlugPagPaymentData(
          PlugPag.TYPE_CREDITO,
          30000,
          PlugPag.INSTALLMENT_TYPE_PARC_VENDEDOR,
          3,
          "CODVENDA");

  // Cria a identificação do aplicativo
  PlugPagAppIdentification appIdentification =
      new PlugPagAppIdentification("MeuApp", "1.0.7");

  // Cria a referência do PlugPag
  PlugPag plugpag = new PlugPag(context, appIdentification);

  // Ativa terminal e faz o pagamento
  int initResult = plugpag.initializeAndActivatePinpad(new
  PlugPagActivationData("SEU_CODIGO_DE_ATIVAÇÃO")));

  if (initResult == PlugPag.RET_OK) {
    PlugPagTransactionResult result = plugpag.doPayment(paymentData);

    // Trata o resultado da transação
    ...
  }
}
```

Pagamento de R$150,00, no débito:

```
public void startPayment(Context context) {
  // Define os dados do pagamento
  PlugPagPaymentData paymentData =
      new PlugPagPaymentData(
          PlugPag.TYPE_DEBITO,
          15000,
          PlugPag.INSTALLMENT_TYPE_A_VISTA,
          1,
          "CODVENDA");

  // Cria a identificação do aplicativo
  PlugPagAppIdentification appIdentification =
      new PlugPagAppIdentification("MeuApp", "1.0.7");

  // Cria a referência do PlugPag
  PlugPag plugpag = new PlugPag(context, appIdentification);

  // Ativa terminal e faz o pagamento
  int initResult = plugpag.initializeAndActivatePinpad(new
  PlugPagActivationData("SEU_CODIGO_DE_ATIVAÇÃO")));

  if (initResult == PlugPag.RET_OK) {
    PlugPagTransactionResult result = plugpag.doPayment(paymentData);

    // Trata o resultado da transação
    ...
  }
}
```

Pagamento de R$50, no voucher:

```
public void startPayment(Context context) {
  // Define os dados do pagamento
  PlugPagPaymentData paymentData =
      new PlugPagPaymentData(
          PlugPag.TYPE_VOUCHER,
          5000,
          PlugPag.INSTALLMENT_TYPE_A_VISTA,
          1,
          "CODVENDA");

  // Cria a identificação do aplicativo
  PlugPagAppIdentification appIdentification =
      new PlugPagAppIdentification("MeuApp", "1.0.7");

  // Cria a referência do PlugPag
  PlugPag plugpag = new PlugPag(context, appIdentification);

  // Ativa terminal e faz o pagamento
  int initResult = plugpag.initializeAndActivatePinpad(new
  PlugPagActivationData("SEU_CODIGO_DE_ATIVAÇÃO")));

  if (initResult == PlugPag.RET_OK) {
    PlugPagTransactionResult result = plugpag.doPayment(paymentData);

    // Trata o resultado da transação
    ...
  }
}
```

Estornar um pagamento

```
public void voidPayment(Context context) {

  // Cria a identificação do aplicativo
  PlugPagAppIdentification appIdentification =
      new PlugPagAppIdentification("MeuApp", "1.0.7");

  // Cria a referência do PlugPag
  PlugPag plugpag = new PlugPag(context, appIdentification);

  // Ativa terminal e faz o pagamento
  int initResult = plugpag.initializeAndActivatePinpad(new
  PlugPagActivationData("SEU_CODIGO_DE_ATIVAÇÃO")));

  if (initResult == PlugPag.RET_OK) {
    PlugPagTransactionResult result = plugpag.voidPayment(
                "transactionCode",
                "transactionId");

    // Trata o resultado do estorno
    ...
  }
}
```

Verificar autenticação

```
public void checkAuthentication(Context context) {
  // Cria a identificação do aplicativo
  PlugPagAppIdentification appIdentification =
      new PlugPagAppIdentification("MeuApp", "1.0.7");

  // Cria a referência do PlugPag
  PlugPag plugpag = new PlugPag(context, appIdentification);

  // Verifica autenticação
  boolean authenticated = plugpag.isAuthenticated();

  if (authenticated) {
    // Usuário autenticado
    ...
  } else {
    // Usúario não autenticado
    ...
  }
}
```

Invalidar autenticação

```
public void logout(Context context) {
  // Cria a identificação do aplicativo
  PlugPagAppIdentification appIdentification =
      new PlugPagAppIdentification("MeuApp", "1.0.7");

  // Cria a referência do PlugPag
  PlugPag plugpag = new PlugPag(context, appIdentification);

  // Invalida a autenticação existente
  plugpag.invalidateAuthentication();
  ...
}
```

Solicitar ativação

```
public void requestAuth(Activity activity) {
  // Cria a identificação do aplicativo
  PlugPagAppIdentification appIdentification =
      new PlugPagAppIdentification("MeuApp", "1.0.7");

  // Cria a referência do PlugPag
  PlugPag plugPag = new PlugPag(activity, appIdentification);


  // Cria objeto com informação do código de ativação
  PlugPagActivationData plugPagActivationData =
        new PlugPagActivationData("SeuCodigoDeAtivação");


  // Solicita autenticação
  PlugPagInitializationResult result =
plugPag.initializeAndActivatePinPad(plugPagActivationData);


  if(result.getResult() == PlugPag.RET_OK) {
    //Sucesso
  } else {
    //Erro
  }
  ...
}
```

O resultado da autenticação será retornado através da classe `PlugPagInitializationResult`.

Se a autenticação for efetuada com sucesso, o método `getResult()` irá retornar valor igual a `PlugPag.RET_OK`. Caso contrário, mais informações do erro podem ser encontradas nos métodos `getErrorCode()` e `getErrorMessage()`.

Obter versão da biblioteca

```
public void getLibVersion(Context context) {
  // Cria a identificação do aplicativo
  PlugPagAppIdentification appIdentification =
      new PlugPagAppIdentification("MeuApp", "1.0.7");

  // Cria a referência do PlugPag
  PlugPag plugpag = new PlugPag(context, appIdentification);

  // Obtém a versão da biblioteca
  String version = plugpag.getLibVersion();
}
```

`Reimpressão da via do estabelecimento`

```
public void reprintStablishmentReceipt() {
  // Cria a identificação do aplicativo
  PlugPagAppIdentification appIdentification =
      new PlugPagAppIdentification("MeuApp", "1.0.7");

  // Cria a referência do PlugPag
  PlugPag plugpag = new PlugPag(context, appIdentification);

  // Obtém a versão da biblioteca
  PlugPagPrintResutlt result = plugpag.reprintStablishmentReceipt();
}
```

```
Reimpressão da via do cliente

public void reprintCustomerReceipt() {

  // Cria a identificação do aplicativo
  PlugPagAppIdentification appIdentification =
      new PlugPagAppIdentification("MeuApp", "1.0.7");

  // Cria a referência do PlugPag
  PlugPag plugpag = new PlugPag(context, appIdentification);

  // Obtém a versão da biblioteca
  PlugPagPrintResutlt result = plugpag.reprintCustomerReceipt();
}
```

`Calcular parcelas`

```
public void calculateInstallments() {

  // Cria a identificação do aplicativo
  PlugPagAppIdentification appIdentification =
      new PlugPagAppIdentification("MeuApp", "1.0.7");

  // Cria a referência do PlugPag
  PlugPag plugpag = new PlugPag(context, appIdentification);

  // Obtém a versão da biblioteca
  Array<String> installments = plugpag.calculateInstallments(saleValue);
}
```

O resultado retornado será uma `String` contendo a quantidade de parcelas permitidas mais o valor de cada parcela, seguindo o padrão “<quantidadeDeParcelas>x R$ <valorDaParcela>”.

Customizar dialog de impressão da via do cliente

```
public void startPayment(Context context) {
  // Define os dados do pagamento
  PlugPagPaymentData paymentData =
      new PlugPagPaymentData(
          PlugPag.TYPE_DEBITO,
          15000,
          PlugPag.INSTALLMENT_TYPE_A_VISTA,
          1,
          "CODVENDA");

  // Cria a identificação do aplicativo
  PlugPagAppIdentification appIdentification =
      new PlugPagAppIdentification("MeuApp", "1.0.7");

  // Cria a referência do PlugPag
  PlugPag plugpag = new PlugPag(context, appIdentification);


  // Cria a customização da dialog de impressão da via do cliente
  PlugPagCustomPrinterLayout customDialog = new
PlugPagCustomPrinterLayout();
  customDialog.setTitle("Imprimir via do client?");
  customDialog.setButtonBackgroundColor("#00ff33");
  customDialog.setConfirmText("Yes");
  customDialog.setCancelText("No");
```

```
    // Ativa terminal e faz o pagamento
    int initResult = plugpag.initializeAndActivatePinpad(new
    PlugPagActivationData("SEU_CODIGO_DE_ATIVAÇÃO")));

    if (initResult == PlugPag.RET_OK) {
        plugPag.setPlugPagCustomPrinterLayout(customDialog);
        PlugPagTransactionResult result = plugpag.doPayment(paymentData);

        // Trata o resultado da transação
        ...
    }
}
```

`Ler cartão NFC`

```
public void readNFCCard() {
  // Cria a referência do PlugPag
  PlugPag plugpag = new PlugPag(context);
  PlugPagNearFieldCardData dataCard = new PlugPagNearFieldCardData();

  dataCard.setStartSlot(1);

  dataCard.setEndSlot(1);


  // Lê um cartão NFC

  PlugPagNFCResult result = plugpag.readFromNFCCard(dataCard);
}
```

O resultado da leitura será retornado através da classe `PlugPagNFCResult`.

Se a leitura for efetuada com sucesso, o método `getResult()` irá retornar valor igual a `1`. Do contrário, o método getResult() retornará -1.

`Escrever no cartão NFC`

```
public void writeToNFCCard() {
  // Cria a referência do PlugPag
  PlugPag plugpag = new PlugPag(context);

   String info = "teste_com16bytes";
   byte[] infoBytes = info.getBytes();

  PlugPagNearFieldCardData dataCard = new PlugPagNearFieldCardData();
  dataCard.setStartSlot(1);
  dataCard.setEndSlot(2);
  dataCard.getSlots()[1].put(“data”, infoBytes);

  // Escreve em um cartão NFC
  int result = plugpag.writeToNFCCard(dataCard);
}
```

O resultado da escrita será retornado através da classe `PlugPagNFCResult`.

Se a escrita for efetuada com sucesso, o método `getResult()` irá retornar valor igual a `1`. Do contrário, o método getResult() retornará -1.

`Abortar operação de leitura/escrita no cartão NFC`

```
public void abortNFC() {
  // Cria a referência do PlugPag
  PlugPag plugpag = new PlugPag(context);

  // Escreve em um cartão NFC
  int result = plugpag.abort();
}
```

O resultado do abort será retornado através da classe `PlugPagNFCResult`.

Se o abort for efetuada com sucesso, o método `getResult()` irá retornar valor igual a `1`. Do contrário, o método getResult() retornará -1.

```
Imprimir arquivo

public void printFile() {
  // Cria a referência do PlugPag
  PlugPag plugPag = new PlugPag(context);


  // Cria objeto com informações da impressão
  PlugPagPrinterData data = new PlugPagPrinterData(
      Environment.getExternalStorageDirectory().getAbsolutePath() +
"/SeuDiretorio/SeuArquivo",
      4,
      10 * 12));


  PlugPagPrinterListener listener = new PlugPagPrinterListener() {
      @Override
      public void onError(@NotNull PlugPagPrintResult
plugPagPrintResult) {
          plugPagPrintResult.getMessage(); // Mensagem de erro
          plugPagPrintResult.getErrorCode(); // Código de erro


      }
  };


   //Seta listener de impressão
  plugPag.setPrinterListener(listener)
```

```
    // Imprime arquivo
    PlugPagPrintResult result = plugPag.printFromFile(plugPagPrinterData)

    if (result == PlugPag.RET_OK) {
      //sucesso
    }
}
```

```
Buscar última transação aprovada

public void getLastApprovedTransaction(Context context) {
  // Cria a referência do PlugPag
  PlugPag plugpag = new PlugPag(context);


  PlugPagTransactionResult lastApprovedTransaction
          = plugpag.getLastApprovedTransaction();
}
```

Exemplos assíncronos

Pagamentos

Pagamento de R$250,00, no crédito, à vista:

```
public void startPayment(Context context) {
  // Define os dados do pagamento
  PlugPagPaymentData paymentData =
      new PlugPagPaymentData(
          PlugPag.TYPE_CREDITO,
          25000,
          PlugPag.INSTALLMENT_TYPE_A_VISTA,
          1,
          "CODVENDA");

  // Cria a identificação do aplicativo
  PlugPagAppIdentification appIdentification =
      new PlugPagAppIdentification("MeuApp", "1.0.7");

  // Cria a referência do PlugPag
  PlugPag plugpag = new PlugPag(context, appIdentification);

  // Ativa terminal e faz o pagamento
  int initResult = plugpag.initializeAndActivatePinpad(new
  PlugPagActivationData("SEU_CODIGO_DE_ATIVAÇÃO")));

  if (initResult == PlugPag.RET_OK) {
    mPlugPag.doAsyncPayment(paymentData, new PlugPagPaymentListener()
```

```
    @Override
    public void onSuccess(@NotNull PlugPagTransactionResult
plugPagTransactionResult) {
      //seu código aqui
    }

    @Override
    public void onError(@NotNull PlugPagTransactionResult
plugPagTransactionResult) {
      //seu código aqui
    }

    @Override
    public void onPaymentProgress(@NotNull PlugPagEventData
plugPagEventData) {
      //seu código aqui
    }

    @Override
    public void onPrinterSuccess(@NotNull PlugPagPrintResult
plugPagPrintResult) {
      //seu código aqui
    }

    @Override
    public void onPrinterError(@NotNull PlugPagPrintResult
plugPagPrintResult) {
     //seu código aqui
```

```
        }
      });
    }
}
```

Pagamento de R$300, no crédito, parcelado em 3 parcelas:

```
public void startPayment(Context context) {
  // Define os dados do pagamento
  PlugPagPaymentData paymentData =
      new PlugPagPaymentData(
          PlugPag.TYPE_CREDITO,
          30000,
          PlugPag.INSTALLMENT_TYPE_PARC_VENDEDOR,
          3,
          "CODVENDA");

  // Cria a identificação do aplicativo
  PlugPagAppIdentification appIdentification =
      new PlugPagAppIdentification("MeuApp", "1.0.7");

  // Cria a referência do PlugPag
  PlugPag plugpag = new PlugPag(context, appIdentification);

  // Ativa terminal e faz o pagamento
  int initResult = plugpag.initializeAndActivatePinpad(new
  PlugPagActivationData("SEU_CODIGO_DE_ATIVAÇÃO")));

  if (initResult == PlugPag.RET_OK) {
    mPlugPag.doAsyncPayment(paymentData, new PlugPagPaymentListener()
     @Override
     public void onSuccess(@NotNull PlugPagTransactionResult
plugPagTransactionResult) {
       //seu código aqui
     }
```

```
        @Override
        public void onError(@NotNull PlugPagTransactionResult
plugPagTransactionResult) {
          //seu código aqui
        }

        @Override
        public void onPaymentProgress(@NotNull PlugPagEventData
plugPagEventData) {
          //seu código aqui
        }

        @Override
        public void onPrinterSuccess(@NotNull PlugPagPrintResult
plugPagPrintResult) {
          //seu código aqui
        }

        @Override
        public void onPrinterError(@NotNull PlugPagPrintResult
plugPagPrintResult) {
         //seu código aqui
        }
       });

    }
   }
}
```

Pagamento de R$150,00, no débito:

```
public void startPayment(Context context) {
  // Define os dados do pagamento
  PlugPagPaymentData paymentData =
      new PlugPagPaymentData(
          PlugPag.TYPE_DEBITO,
          15000,
          PlugPag.INSTALLMENT_TYPE_A_VISTA,
          1,
          "CODVENDA");

  // Cria a identificação do aplicativo
  PlugPagAppIdentification appIdentification =
      new PlugPagAppIdentification("MeuApp", "1.0.7");

  // Cria a referência do PlugPag
  PlugPag plugpag = new PlugPag(context, appIdentification);

  // Ativa terminal e faz o pagamento
  int initResult = plugpag.initializeAndActivatePinpad(new
  PlugPagActivationData("SEU_CODIGO_DE_ATIVAÇÃO")));

  if (initResult == PlugPag.RET_OK) {
    mPlugPag.doAsyncPayment(paymentData, new PlugPagPaymentListener()
     @Override
     public void onSuccess(@NotNull PlugPagTransactionResult
plugPagTransactionResult) {
       //seu código aqui
     }
```

```
        @Override
        public void onError(@NotNull PlugPagTransactionResult
plugPagTransactionResult) {
          //seu código aqui
        }

        @Override
        public void onPaymentProgress(@NotNull PlugPagEventData
plugPagEventData) {
          //seu código aqui
        }

        @Override
        public void onPrinterSuccess(@NotNull PlugPagPrintResult
plugPagPrintResult) {
          //seu código aqui
        }

        @Override
        public void onPrinterError(@NotNull PlugPagPrintResult
plugPagPrintResult) {
          //seu código aqui
        }
       });

     }
    }
}
```

Pagamento de R$50, no voucher:

```
public void startPayment(Context context) {
  // Define os dados do pagamento
  PlugPagPaymentData paymentData =
      new PlugPagPaymentData(
          PlugPag.TYPE_VOUCHER,
          5000,
          PlugPag.INSTALLMENT_TYPE_A_VISTA,
          1,
          "CODVENDA");

  // Cria a identificação do aplicativo
  PlugPagAppIdentification appIdentification =
      new PlugPagAppIdentification("MeuApp", "1.0.7");

  // Cria a referência do PlugPag
  PlugPag plugpag = new PlugPag(context, appIdentification);

  // Ativa terminal e faz o pagamento
  int initResult = plugpag.initializeAndActivatePinpad(new
  PlugPagActivationData("SEU_CODIGO_DE_ATIVAÇÃO")));

  if (initResult == PlugPag.RET_OK) {
    plugPag.doAsyncPayment(paymentData, new PlugPagPaymentListener()
    @Override
    public void onSuccess(@NotNull PlugPagTransactionResult
plugPagTransactionResult) {
      //seu código aqui
    }
```

```
        @Override
        public void onError(@NotNull PlugPagTransactionResult
plugPagTransactionResult) {
          //seu código aqui
        }

        @Override
        public void onPaymentProgress(@NotNull PlugPagEventData
plugPagEventData) {
          //seu código aqui
        }

        @Override
        public void onPrinterSuccess(@NotNull PlugPagPrintResult
plugPagPrintResult) {
          //seu código aqui
        }

        @Override
        public void onPrinterError(@NotNull PlugPagPrintResult
plugPagPrintResult) {
          //seu código aqui
        }
       });

     }
    }
}
```

Estornar um pagamento

```
public void voidPayment(Context context) {

  // Cria a identificação do aplicativo
  PlugPagAppIdentification appIdentification =
      new PlugPagAppIdentification("MeuApp", "1.0.7");

  // Cria a referência do PlugPag
  PlugPag plugpag = new PlugPag(context, appIdentification);

  // Ativa terminal e faz o pagamento
  int initResult = plugpag.initializeAndActivatePinpad(new
  PlugPagActivationData("SEU_CODIGO_DE_ATIVAÇÃO")));

  if (initResult == PlugPag.RET_OK) {

mPlugPag.doAsyncVoidPayment("transactionCode", "transactionId",
                new PlugPagPaymentListener()
    @Override
    public void onSuccess(@NotNull PlugPagTransactionResult
plugPagTransactionResult) {
      //seu código aqui
    }

    @Override
    public void onError(@NotNull PlugPagTransactionResult
plugPagTransactionResult) {
```

```
        //seu código aqui
    }

    @Override
    public void onPaymentProgress(@NotNull PlugPagEventData
plugPagEventData) {
        //seu código aqui
    }

    @Override
    public void onPrinterSuccess(@NotNull PlugPagPrintResult
plugPagPrintResult) {
        //seu código aqui
    }

    @Override
    public void onPrinterError(@NotNull PlugPagPrintResult
plugPagPrintResult) {
        //seu código aqui
    }
    });
  }
}
```

Verificar autenticação

```
public void checkAuthentication(Context context) {
  // Cria a identificação do aplicativo
  PlugPagAppIdentification appIdentification =
      new PlugPagAppIdentification("MeuApp", "1.0.7");

  // Cria a referência do PlugPag
  PlugPag plugpag = new PlugPag(context, appIdentification);

  // Verifica autenticação
  plugPag.asyncIsAuthenticated(new PlugPagIsActivatedListener() {
    @Override
    public void onIsActivated(boolean isActivated) {
      //seu código aqui
    }

    @Override
    public void onError(String message) {
      //seu código aqui

    }
  });
}
```

Invalidar autenticação

```
public void logout(Context context) {
  // Cria a identificação do aplicativo
  PlugPagAppIdentification appIdentification =
      new PlugPagAppIdentification("MeuApp", "1.0.7");

  // Cria a referência do PlugPag
  PlugPag plugpag = new PlugPag(context, appIdentification);

  // Invalida a autenticação existente
  // Solicita autenticação
  plugPag.asyncDeactivate(new PlugPagActivationData(activationCode),
new PlugPagActivationListener() {
    @Override
    public void onActivationProgress(@NotNull PlugPagEventData
                    plugPagEventData) { //seu código aqui
    }

    @Override
    public void onSuccess(@NotNull PlugPagInitializationResult
                plugPagInitializationResult) {
     //seu código aqui
    }

    @Override
    public void onError(@NotNull PlugPagInitializationResult
                plugPagInitializationResult) {
```

```
        //seu código aqui
      }
    });


    ...
}
```

Solicitar ativação

```
public void requestAuth(Activity activity) {
  // Cria a identificação do aplicativo
  PlugPagAppIdentification appIdentification =
      new PlugPagAppIdentification("MeuApp", "1.0.7");

  // Cria a referência do PlugPag
  PlugPag plugPag = new PlugPag(activity, appIdentification);


  // Cria objeto com informação do código de ativação
  PlugPagActivationData plugPagActivationData =
        new PlugPagActivationData("SeuCodigoDeAtivação");


  // Solicita autenticação
  plugPag.doAsyncInitializeAndActivatePinpad(
          new PlugPagActivationData(activationCode), new
PlugPagActivationListener() {
    @Override
    public void onActivationProgress(@NotNull PlugPagEventData
                plugPagEventData) { //seu código aqui
    }

    @Override
    public void onSuccess(@NotNull PlugPagInitializationResult
                plugPagInitializationResult) {
     //seu código aqui
```

```
    }

    @Override
    public void onError(@NotNull PlugPagInitializationResult
                        plugPagInitializationResult) {
     //seu código aqui
    }
  });

  ...
}
```

`Reimpressão da via do estabelecimento`

```
public void reprintStablishmentReceipt() {
  // Cria a identificação do aplicativo
  PlugPagAppIdentification appIdentification =
      new PlugPagAppIdentification("MeuApp", "1.0.7");

  // Cria a referência do PlugPag
  PlugPag plugpag = new PlugPag(context, appIdentification);

  // Reimprime a via do estabelecimento
  plugPag.asyncReprintEstablishmentReceipt(new PlugPagPrinterListener(){
   @Override
   public void onError(@NotNull PlugPagPrintResult plugPagPrintResult) {
    //seu código aqui
   }
```

```
    @Override
    public void onSuccess(@NotNull PlugPagPrintResult
plugPagPrintResult) {
      //seu código aqui
    }
  });
}
```

```
Reimpressão da via do cliente

public void reprintCustomerReceipt() {

  // Cria a identificação do aplicativo
  PlugPagAppIdentification appIdentification =
      new PlugPagAppIdentification("MeuApp", "1.0.7");

  // Cria a referência do PlugPag
  PlugPag plugpag = new PlugPag(context, appIdentification);

  // Reimprime via do cliente
  plugPag.asyncReprintEstablishmentReceipt(new PlugPagPrinterListener(){
   @Override
   public void onError(@NotNull PlugPagPrintResult plugPagPrintResult) {
     //seu código aqui
   }

   @Override
   public void onSuccess(@NotNull PlugPagPrintResult
plugPagPrintResult) {
     //seu código aqui
   }
  });

}
```

`Calcular parcelas`

```
public void calculateInstallments() {

  // Cria a identificação do aplicativo
  PlugPagAppIdentification appIdentification =
      new PlugPagAppIdentification("MeuApp", "1.0.7");

  // Cria a referência do PlugPag
  PlugPag plugpag = new PlugPag(context, appIdentification);

  // Obtém a versão da biblioteca
  plugpag.asyncCalculateInstallments("", new
PlugPagInstallmentsListener() {
  @Override
  public void onCalculateInstallments(@NotNull String[] strings) {
    //seu código aqui
  }

  @Override
  public void onError(@NotNull String s) {
    //seu código aqui
  }
 });
}
```

Ler cartão NFC

```
public void readNFCCard() {
  // Cria a referência do PlugPag
  PlugPag plugpag = new PlugPag(context);
  PlugPagNearFieldCardData dataCard = new PlugPagNearFieldCardData();

  dataCard.setStartSlot(1);

  dataCard.setEndSlot(1);


  // Lê um cartão NFC

  plugpag.asyncReadNFC(dataCard, new PlugPagNFCListener() {
   @Override
   public void onSuccess(@NotNull PlugPagNFCResult plugPagNFCResult) {
        //seu código aqui
   }


    @Override
   public void onError(@NotNull String errorMessage) {
    //seu código aqui

   }
 });
}
```

Escrever no cartão NFC

```
public void writeToNFCCard() {
  // Cria a referência do PlugPag
  PlugPag plugpag = new PlugPag(context);

   String info = "teste_com16bytes";
   byte[] infoBytes = info.getBytes();

  PlugPagNearFieldCardData dataCard = new PlugPagNearFieldCardData();
  dataCard.setStartSlot(1);
  dataCard.setEndSlot(2);
  dataCard.getSlots()[1].put("data", infoBytes);

  // Escreve em um cartão NFC
  plugpag.asyncWriteNFC(dataCard, new PlugPagNFCListener() {
   @Override
   public void onSuccess(@NotNull PlugPagNFCResult plugPagNFCResult) {
        //seu código aqui
   }

     @Override
   public void onError(@NotNull String errorMessage) {
     //seu código aqui
   }
 });
}
```

```
Abortar operação de leitura/escrita no cartão NFC

public void abortNFC() {
  // Cria a referência do PlugPag
  PlugPag plugpag = new PlugPag(context);


  // Escreve em um cartão NFC
  plugpag.asyncAbortNFC(new PlugPagAbortListener() {
   @Override
   public void onAbortRequested(boolean b) {
      //seu código aqui
   }

   @Override
   public void onError(@NotNull String s) {
      //seu código aqui
   }
 });
}
```

```
Buscar última transação aprovada

public void getLastApprovedTransaction(Context context) {
  // Cria a referência do PlugPag
  PlugPag plugpag = new PlugPag(context);


  plugpag.doAsyncVoidPayment(new PlugPagLastTransactionListener() {
  @Override
  public void onRequestedLastTransaction(@NotNull
PlugPagTransactionResult plugPagTransactionResult) {
    //seu código aqui
  }

  @Override
  public void onError(@NotNull String s) {
    //seu código aqui
  }
 });
}
```

Códigos de retorno

Os códigos de retorno descritos abaixo são obtidos ao chamar o método `getResult()` de um `PlugPagTransactionResult` retornado por um dos métodos de transação de um objeto `PlugPag`: `doPayment(PlugPagPaymentData)`, `voidPayment(PlugPagVoidData)` e `getLastApprovedTransaction()`.

Valor	Descrição	Ação
0	Transação concluída com sucesso.	
-1001	Mensagem gerada maior que buffer dimensionado.	Coletar log (se existir) e enviar para o suporte.
-1002	Parâmetro de aplicação inválido.	Coletar log (se existir) e enviar para o suporte.
-1003	Terminal não está pronto para transacionar.	Tente novamente.
-1004	Transação não realizada.	Verificar mensagem retornada.
-1005	Buffer de resposta da transação inválido ao obter as informações de resultado da transação.	Realizar consulta de última transação.
-1006	Parâmetro de valor da transação não pode ser nulo.	Verificar implementação da chamada da biblioteca.
-1007	Parâmetro de valor total da transação não pode ser nulo.	Verificar implementação da chamada da biblioteca.

-1008	Parâmetro de código de venda não pode ser nulo.	Verificar implementação da chamada da biblioteca.
-1009	Parâmetro de resultado da transação não pode ser nulo.	Verificar implementação da chamada da biblioteca.
-1010	Driver de conexão não encontrado.	Verificar se todos os arquivos estão no diretório correto.
-1011	Erro ao utilizar driver de conexão.	Reinstalar os arquivos do driver de conexão.
-1012	Formato do valor da venda inválido.	Valor deve ser um número inteiro sem vírgula.
-1013	Comprimento do código de venda superior a 10 dígitos.	Truncar código de venda para no máximo 10 dígitos.
-1014	Buffer de recepção corrompido.	Refaça a transação.
-1015	Nome da aplicação maior que 25 caracteres.	Limitar nome da aplicação a 25 caracteres.
-1016	Versão da aplicação maior que 10 caracteres.	Limitar versão da aplicação em 10 caracteres.
-1017	Necessário definir nome da aplicação.	Definir nome e versão da aplicação com `setVersionName(String, String)`
-1018	Não existem dados da última transação.	Refaça a transação.

-1019	Erro de comunicação com terminal (resposta inesperada).	Realizar consulta de última transação.
-1024	Erro na carga de tabelas.	Refazer inicialização (carga de tabelas).
-1030	Token não encontrado	Refazer autenticação.
-1031	Valor inválido	Verificar o valor configurado para pagamento e tentar novamente. Valor mínimo: R$ 1,00
-1032	Parcelamento inválido	Verificar o número de parcelas e tentar novamente.
-2001	Porta COM informada não encontrada.	Informar uma porta COM válida.
-2002	Não foi possível obter configurações da porta COM informada.	Informar uma porta COM válida.
-2003	Não foi possível configurar a porta COM informada.	Informar uma porta COM válida.
-2005	Não foi possível enviar dados pela porta COM informada.	Informar uma porta COM válida.
-2022	Java – Adaptador Null.	Verificar implementação.
-2023	Java – erro em DeviceToUse.	Coletar log (se existir) e enviar para o suporte.
-2024	Java – erro no serviço RfcommSocket.	Coletar log (se existir) e enviar para o suporte.
-2026	Java – Close exception.	Coletar log (se existir) e enviar para o suporte.

-3001	Permissão de root	Remover permissão de root do aparelho.
-4046	Não existe dados de autenticação	Efetuar a autenticação.

Códigos de erro de impressão

Os códigos de retorno descritos abaixo são obtidos ao chamar o método `getResult()` de um `PlugPagTransactionResult` retornado por um dos métodos de transação de um objeto `PlugPag`: `doPayment(PlugPagPaymentData)`, `voidPayment(PlugPagVoidData)` e `getLastApprovedTransaction()`.