Ane

Aneline Gyuviyska

Aneline.gyuviyska@gmail.com

Document

Lamas in Pajamas

Test  
Payment Transaction API

0.1

# Project

Maven, Java project for testing Payment Transaction API. Includes jUnit 4.

# Models

## Package

com.example.models

## BaseResponse Class

### Description:

The BaseResponse class is a model class that represents the response of a transaction. It contains information such as the unique identifier of the transaction, the status, the usage, the amount, the transaction time, and a message.

### Fields

* uniqueID (String): A unique identifier for the transaction.
* status (String): The status of the transaction.
* usage (String): The usage of the transaction.
* amount (String): The amount of the transaction.
* transactionTime (String): The time of the transaction.
* message (String): A message related to the transaction.

### Methods

* getUniqueID(): Returns the unique identifier of the transaction.
* setUniqueID(uniqueID: String): Sets the unique identifier of the transaction.
* getStatus(): Returns the status of the transaction.
* setStatus(status: String): Sets the status of the transaction.
* getUsage(): Returns the usage of the transaction.
* setUsage(usage: String): Sets the usage of the transaction.
* getAmount(): Returns the amount of the transaction.
* setAmount(amount: String): Sets the amount of the transaction.
* getTransactionTime(): Returns the time of the transaction.
* setTransactionTime(transactionTime: String): Sets the time of the transaction.
* getMessage(): Returns the message related to the transaction.
* setMessage(message: String): Sets the message related to the transaction.

## SaleRequest Class

### Description:

The SaleRequest class is a model class that is used to represent a sale request in a payment processing system.

### Fields

* cardNumber: The credit card number associated with the sale request.
* cvv: The CVV number associated with the credit card.
* expirationDate: The expiration date of the credit card.
* amount: The amount being requested in the sale.
* usage: The usage for the sale, for example for what purpose the amount is being requested.
* transactionType: The type of transaction being requested.
* cardHolder: The name of the card holder.
* email: The email address associated with the card holder.
* address: The address associated with the card holder.

### Methods

* getCardNumber: Returns the card number associated with the sale request.
* setCardNumber: Sets the card number associated with the sale request.
* getCvv: Returns the CVV number associated with the credit card.
* setCvv: Sets the CVV number associated with the credit card.
* getExpirationDate: Returns the expiration date of the credit card.
* setExpirationDate: Sets the expiration date of the credit card.
* getAmount: Returns the amount being requested in the sale.
* setAmount: Sets the amount being requested in the sale.
* getUsage: Returns the usage for the sale.
* setUsage: Sets the usage for the sale.
* getTransactionType: Returns the type of transaction being requested.
* setTransactionType: Sets the type of transaction being requested.
* getCardHolder: Returns the name of the card holder.
* setCardHolder: Sets the name of the card holder.
* getEmail: Returns the email address associated with the card holder.
* setEmail: Sets the email address associated with the card holder.
* getAddress: Returns the address associated with the card holder.
* setAddress: Sets the address associated with the card holder.

## SaleResponse Class

### Description

The SaleResponse class is a subclass of BaseResponse class and represents the response of a sale transaction.

### Fields

This class inherits all fields from the BaseResponse class.

### Methods

This class inherits all methods from the BaseResponse class.

## VoidRequest Class

### Description

The VoidRequest class is used to represent the request payload for a void transaction in the system.

### Fields

* referenceID (String) : A unique identifier of the original transaction that is being voided.
* transactionType (String) : The type of transaction being performed, which in this case is a void transaction.

### Methods

* getReferenceID() : Returns the referenceID field.
* setReferenceID(String referenceID) : Sets the referenceID field to the given value.
* getTransactionType() : Returns the transactionType field.
* setTransactionType(String transactionType) : Sets the transactionType field to the given value.

## VoidResponse Class

### Description

The VoidResponse class is a subclass of BaseResponse class and represents the response of a sale transaction.

### Fields

This class inherits all fields from the BaseResponse class.

### Methods

This class inherits all methods from the BaseResponse class.

# Main

The class Main is a JUnit test runner class that runs several test classes defined in the package com.example. The tests are executed in the order they are added to the results list. The class uses the JUnitCore class of JUnit framework to run the test classes.

The main method runs 5 test classes: ValidSaleTransactionTest, InvalidVoidTransactionTest, NonExistentReferenceVoidTransactionTest, ValidVoidTransactionTest, and WrongBasicAuthenticationTransactionStatusCode401Test. After each test run, it prints the failure information, if any, and the success status of the test run.

The results of all test runs are stored in an ArrayList of Result type, where each element of the list is the result of a test run. The results of each test run are printed to the console, including any failures and whether the test was successful or not.

BaseResponse Class Documentation

Description:

The BaseResponse class is a model class that represents the response of a transaction. It contains information such as the unique identifier of the transaction, the status, the usage, the amount, the transaction time, and a message.

## Package

com.example

# ConfigProvider Class

## Package

com.example

## Overview

The ConfigProvider class is used to read properties from a properties file called config.properties. The properties file must be located in the classpath of the application.

## Members:

Properties props: A Properties object to store the properties read from the file.

## Methods

### Constructor

The constructor takes no arguments. It initializes the props object and loads the properties from the config.properties file. If the file is not found in the classpath, a FileNotFoundException is thrown.

### getPropValue(String propKey)

This method takes a property key as an argument and returns its corresponding value from the properties file. If the key does not exist, the method returns null.

## Exception Handling

The class handles the IOException thrown when the properties file is read and throws it again if caught. The class also handles any other exception that may be thrown while reading the properties file and prints an error message to the console.

## Usage

To use the ConfigProvider class, instantiate an object of the class and call the getPropValue method with a property key as an argument to retrieve its value. The following is an example usage of the class:

java

ConfigProvider config = new ConfigProvider();

String value = config.getPropValue("propertyKey");

# JsonModelConverter Class

## Package

com.example

## Overview

The JsonModelConverter class is used to convert JSON strings to different model objects and vice versa. The class uses the Jackson library to perform the conversions.

## Members:

None.

## Methods

### getSaleRequestFromJson(String json)

This method takes a JSON string as an argument and returns a SaleRequest object by converting the JSON string to a SaleRequest object.

### getSaleResponseFromJson(String json)

This method takes a JSON string as an argument and returns a SaleResponse object by converting the JSON string to a SaleResponse object.

### getVoidRequestFromJson(String json)

This method takes a JSON string as an argument and returns a VoidRequest object by converting the JSON string to a VoidRequest object.

### getVoidResponseFromJson(String json)

This method takes a JSON string as an argument and returns a VoidResponse object by converting the JSON string to a VoidResponse object.

### getJsonFromVoidRequest(VoidRequest voidRequest)

This method takes a VoidRequest object as an argument and returns its JSON representation by converting the VoidRequest object to a JSON string.

## Exception Handling

The class handles JsonProcessingException that may be thrown while converting JSON strings to model objects and vice versa. The error is printed to the console if caught.

## Usage

To use the JsonModelConverter class, create an instance of the class and call the appropriate conversion method to perform the conversion. The following is an example usage of the class:

java

JsonModelConverter converter = new JsonModelConverter();

String json = "{\"payment\_transaction\":{\"amount\":100.0}}";

SaleRequest saleRequest = converter.getSaleRequestFromJson(json);

# PaymentRequestMessageProvider Class

## Package

com.example

## Purpose

This class is responsible for reading and providing the contents of a message file.

## Class Variables

MESSAGES\_FOLDER\_NAME: A string constant representing the name of the folder where the message files are stored.

## Method

### getMessage(String fileName)

This method is used to read and return the contents of a message file.

### Input

A string representing the name of the file to be read.

### Output

A string containing the contents of the message file.

### Exception Handling

If the input file is not found in the classpath, a FileNotFoundException is thrown.

If there is an error in reading the file, an IOException is thrown.

Any other exception encountered is logged and re-thrown.

### Implementation Details

The method uses the Java ClassLoader to retrieve the input stream of the file.

The input stream is read and its contents are returned as a string.

The input stream is closed after it is read.

# PaymentTransactionClient class:

## Package

com.example

This class provides a client, which makes payment transaction requests to a server. It takes in the port, address, username, and password as arguments and constructs a URI to the server. The basic authentication header is also generated using the username and password provided.

## Method

### makePaymentTransactionRequest

The main method of this class is makePaymentTransactionRequest, which sends a POST request to the server with the message body and the basic authentication header. It returns the response from the server.

The response from the server is returned as an object of type HttpResponse which contains the status code, headers, and body of the response.

Note: This class uses the java.net.http library introduced in Java 11 for making HTTP requests.

SetUpPaymentClient class

### setUpForValidHttpConnection

This class provides a static method setUpForValidHttpConnection to set up a valid PaymentTransactionClient instance by reading necessary configuration values from a ConfigProvider instance. The configuration values include the authenticationUser, authenticationPassword, apiPort, and address of the API. The setUpForValidHttpConnection method returns a new PaymentTransactionClient instance created with the read configuration values. The SetUpPaymentClient class does not contain any test logic but is useful for initializing the PaymentTransactionClient for testing purposes.

# ValidVoidTransactionTest Class

## Package

com.example

## Overview

This class is a JUnit test case used to test the functionality of a void transaction in a payment processing system.

It first sets up a payment transaction client and creates a sale request message using a "PaymentRequestMessageProvider". It then makes a payment transaction request with the sale request message and retrieves the response, which includes a unique ID that will be used for the void request. The void request is created with the unique ID and a transaction type of "void", and its message is stored in "voidRequestMessage".

### checkValidVoidStatusApproved

The main test case "checkValidVoidStatusApproved" makes a payment transaction request with the void request message and retrieves the void response. It then asserts that the void response's status is "approved".

### JsonModelConverter

This test case uses a "JsonModelConverter" class to convert between JSON and Java objects, and a "PaymentTransactionClient" class to make the payment transaction requests.

# ValidSaleTransactionTest Class

## Package

com.example

## Purpose:

The purpose of this JUnit class is to test the validity of sale transactions made to a payment gateway.

The class uses the PaymentRequestMessageProvider class to retrieve a message from the file named "ValidSaleMessages.json". The makePaymentTransactionRequest() method of the PaymentTransactionClient is used to make a payment transaction request with the retrieved message. The response from the API is then used to test the expected results in the test methods. The class also uses the JsonModelConverter class to convert the JSON response to a SaleResponse object.

## Methods

### @BeforeClass PaymentTransactionClient

Before executing the test methods, the class sets up a PaymentTransactionClient object for a valid HTTP connection using the setUpForValidTransaction() method. This method is annotated with @BeforeClass and is executed only once before executing all the test methods.

### validSaleTransactionStatusCode200()

This method tests that the HTTP status code returned in the response is 200.

### checkValidSaleStatusApproved()

This method tests that the status of the payment transaction in the response is "approved".

### checkAmountsAreEqual()

This method tests that the amount in the request and response are equal.

# InvalidVoidTransactionTest Class

## Package

com.example

## Overview

This is a JUnit test class for the scenario of an invalid void transaction in the payment gateway. The class performs a test on the scenario where a void transaction is performed on another void transaction which should return an HTTP response code 422 (Unprocessable Entity).

The class makes use of the PaymentTransactionClient and JsonModelConverter classes and uses the setUpForValidHttpConnection method from the SetUpPaymentClient class to set up a valid HTTP connection for the PaymentTransactionClient object.

A sale request message is retrieved from the PaymentRequestMessageProvider class and a sale transaction is performed using the PaymentTransactionClient. The sale response is then used to initiate a void transaction stored in the voidRequestMessage variable.

## Members:

VALID\_MESSAGE\_FILE\_NAME: a constant string representing the name of the file that contains the valid sale message payload.

paymentClient: an instance of the PaymentTransactionClient class which makes the payment transactions to the payment gateway.

setUpForValidTransaction(): a method annotated with @BeforeClass that sets up the PaymentTransactionClient instance for making valid payment transactions.

validSaleTransactionStatusCode200(): a method annotated with @Test that tests if the HTTP response code of the payment transaction is 200.

checkValidSaleStatusApproved(): a method annotated with @Test that tests if the status of the payment transaction is "approved".

checkAmountsAreEqual(): a method annotated with @Test that tests if the amount of the payment transaction request and response are equal.

## Method

### checkInvalidVoidPointingVoidReturns422

The test is performed in the checkInvalidVoidPointingVoidReturns422 method, where the PaymentTransactionClient is used to make a request with the voidRequestMessage. The response’s status code is then compared with the expected value of 422.

In case the test fails, an error message will be printed with the actual status code received.

# NonExistentReferenceVoidTransactionTest Class

## Package

com.example

## Purpose

This JUnit class is written to test the response status code for a void transaction request with a non-existent reference ID.

## Members:

paymentClient is an instance of the PaymentTransactionClient class to make the payment transaction request.

voidRequestMessage is a string variable that holds the JSON message of the void request.

## Methods:

### @BeforeClass setUpForValidVoidTransaction():

an annotated static method that sets up the environment for the payment transaction request. It creates an instance of the PaymentTransactionClient and sets the value for the voidRequestMessage variable.

### @Test nonExistentReferenceVoidTransactionStatusCode422():

an annotated method that tests the response status code of the void transaction request with a non-existent reference ID. It asserts that the actual result of the status code should be equal to the expected result (422).

# WrongBasicAuthenticationTransactionStatusCode401Test Class

## Package

com.example

## Overview

Purpose: This class tests the basic authentication mechanism of the payment API, by making a payment request with incorrect credentials. The purpose of this class is to ensure that the API returns a status code of 401 Unauthorized, in case of incorrect authentication.

## Methods:

@BeforeClass etUpWrongBasicAuthenticationTransactionStatusCode401()

This code sets up the PaymentTransactionClient for a test scenario that requires a wrong basic authentication. The method creates a ConfigProvider instance which is used to retrieve the values of apiPort and apiAddress from a configuration file. These values are then used to create an instance of PaymentTransactionClient with the specified values of apiPort, apiAddress, authenticationUser and authenticationPassword. The values of authenticationUser and authenticationPassword are hardcoded in the method as "fff". The method also contains a todo comment which indicates that better names are needed for the System.out.println statements.

### @BeforeClass etUpWrongBasicAuthenticationTransactionStatusCode401

This method sets up the environment for the test. It initializes the PaymentTransactionClient with incorrect credentials.

### @Test wrongBasicAuthenticationTransactionStatusCode401

This method tests the API's response for incorrect authentication. It makes a payment request using the PaymentTransactionClient and asserts that the API returns a status code of 401 Unauthorized.

Notes: The API address, API port, and authentication credentials are taken from a configuration file.

The test message used for the payment request is a valid message taken from ValidSaleMessages.json file.