EJERCICIO

Con la pandemia el mercado electrónico ha tomado un gran impulso y el pequeño comercio local se ve en la necesidad de incorporarse a este modo de trabajo para recuperar clientes. Por ello, la asociación de minoristas, encargan una aplicación que les permita introducir sus tiendas en el mercado online. En este ejercicio se pretende dar solución al pago de productos de venta online.

El servicio va guardando pagos a lo largo del día y dispone de una operación "process" que se usa al final del día y que procesa todos los pagos. Procesar un pago, supone validar que el pago es correcto y declararlo válido.

Modelo de datos

Existen tres tipos de pago: tarjeta de crédito (**credit card**) , pago por **PayPa**l, y pago en metálico (**cash**). Todos los pagos tienen un identificador de transacción (**transaction ID),** la cantidad (**amount)** de la venta, de tipo double, y un campo booleano (**valid**) que determina (una vez que el pago ha sido procesado) si el pago es válido.

Por otra parte, los pagos con tarjeta de crédito (Credit card payment) tendrán además una constante **MERCHANT_ID** que permita al banco identificar cada minorista (retailer) (en nuestro caso es el "1111-22"), el número de tarjeta (**card number)** y su mes (**month**) y año (**year**) de caducidad.

Los pagos con PayPal (PayPal payment) tendrán el **username**, y **password** del cliente que ha hecho el pago.

Los pagos por cash (cash payment) no definen campos extra ni métodos.

Servicio retailer

El sistema almacena los pagos que se van generando, por lo que tendrá las siguientes operaciones:

addPayments, añade un pago al sistema

process(), procesa todos los pagos existentes, lo que implicará que éstos se validen si todo ha ido bien.

getTotalSales(): double. Calcula el importe total de los pagos válidos.

El procesamiento de los pagos depende del tipo de pago:

Procesar pagos de CreditCard

Procesar los pagos de CreditCard se realiza a través de un servicio externo correspondiente a una **pasarela de pago**. Habrá que crear un objeto pasarela y llamar a su operación isValid. Este método para validar necesita recibir los datos del pago en cuestión, pero, como es un sistema externo, no se plantea recibir el objeto creditCarPayment, que es del cliente, sino que para realizar la comunicación establecen un "contrato" que deberán cumplir los CreditCardPayment. **La pasarela define el tipo Transaction (se deberá definir en su proyecto)**, de manera que cualquier cliente que se desee validar deberá hacer que sus objetos sean de este tipo. El tipo Transaction define las siguientes operaciones:

- String getCreditCardNumber ();
- Integer getMonth();
- Integer getYear();

El método isValid() de la pasarela recibe como parámetro una transacción (que será el pago a validar)

Si isValid devuelve true, se ha podido validar la transacción y por tanto el pago guardará la información de que es válido en su atributo valid.

Procesar pagos de PayPal

En el caso de PayPal, la operación process no se realiza de la misma manera. Simplemente el objeto paypalPayment crea un objeto de PayPalAPI y llama al método login con usuario y clave. Este devuelve un token y finalmente se hace un checkOut pasándole dicho token, si el checkOut devuelve true, se considera la transacción válida y se asigna el valor true a su atributo valid.

Procesar pagos en cash

En este caso todos los pagos son válidos, por lo que se modificará el valor de valid para asignarle true.

¿Qué hay que hacer?

1. **Realizar el diseño UML de todo el problema (1 punto)**
2. **Crear las clases del modelo** (3 puntos)
3. **Crear el tipo para establecer comunicación con la pasarela de pago** e implementarlo (1,5 puntos)
4. **Crear la clase servicio Retailer** (2 puntos)
5. **Crear la interfaz de usuario (método run) que realice las siguientes operaciones: (0,5 PUNTOS)**
    - **a.** Crear varios pagos, uno válido y otro inválido de cada tipo,
    - **b.** Calcular el total de ventas y mostrarlo
    - **c.** Procesar todos los pagos
    - **d.** Calcular el total de ventas y mostrarlo

6. **PRUEBAS UNITARIAS** (2 puntos)

   Deben realizarse las pruebas JUnit para los métodos:

    - Retailer:: getTotalSales()
    - Pruebas de CreditCardPayment:: process()

Información adicional para mejor comprensión pero ya está implementado

Funcionamiento interno de login y checkOut en clase PayPalAPI

- *El **método logIn**, recibe username y password del pago por paypal como parámetros. Si esta estos datos están guardados en la lista usuario/clave, se genera un token de sesión aleatorio que se usará en el resto de métodos de la clase PayPalAPI para verificar qué clase de usuario se logea. Este token se almacena en una lista de tokens de sesión dentro del objeto PayPayAPI y también es devuelto por el método. Si el username y password no son correctos se devuelve una cadena PayPalAPI.INVALID_LOGIN.*
- *El **método checkout** recibe el token así como el ID y la cantidad de la transacción. Si el token recibido está en la lista de tokens de la sesión, la transacción es válida y devuelve true; en otro caso, devuelve false para señalar que el pago no podrá hacerse.*