

DOCUMENTO PROYECTO 3: DPOO.

1) Funcionamiento de los nuevos requerimientos:

A continuación, se explicará el funcionamiento de los nuevos requerimientos solicitados en ese proyecto 3.

- **Emision de facturas en pdf:** Comenzamos con la creación de facturas después de haberse creado o realizado una reserva por parte de un cliente.

Para ello, usamos el [jPDFProcess](#). En él se creará todo lo relacionado con los pdfs que serán el nuevo método de factura para la reserva. Este archivo requiere de configurar el build path para poner el archivo en el class path. En la parte lógica de la Empresa se realizaron los siguientes cambios:

Se tiene el método guardarReserva, donde este se encargará de guardar la reserva para luego generar el pdf. Funciona de la siguiente manera:

```
public String guardarReserva(long documento) {
    String resp = "";
    Cliente cliente = buscarClientePorId(documento);
    if (cliente != null) {
        if (cliente.verificarReserva()) {
            resp = creator.createReservationPDF(cliente.getReserva());
        }
        else {
            resp += "El cliente no tiene reserva";
        }
    }
    else {
        resp = "No se encuentra registrado ningún cliente con el documento " + documento;
    }
    return resp;
}
```

Como parámetro se tiene el documento del cliente que está realizando la reserva, para luego en la base de datos del programa buscar si este ya se encuentra registrado. Si lo está, se verifica si tiene reserva alguna para luego llamar a la clase creator que es la encargada de realizar los pdfs con el método createReservationPDF. Si no se genera la reserva, significa que el usuario en realidad no solicitó una o no se encuentra ya inscrito en la empresa. De lo contrario, se retorna la factura del vehículo reservado.

Claro está que en la Empresa se tiene el nuevo atributo:

`pdfCreator creator;`

Donde pdfCreator es la nueva clase que realiza el pdf de la reserva y así poder acceder a sus métodos.




Ahora se va a hablar sobre la nueva clase pdfCreator. Esta es la encargada de crear las facturas de las reservas para los clientes. Para ello, se escribe los datos que se desean ver en la reserva y luego con métodos gráficos se genera la parte estética del pdf, es decir el color, tipo de letra.

Al final, se le muestra al cliente la reserva que realizo y un mensaje de confirmación. Si sucede un error en el proceso, se le mostrara a la persona que hubo un error a la hora de generar el pdf.

Importante: Si no aparece la factura en formato .pdf en eclipse, probablemente aparezca si se va a donde esta la carpeta facturas en el explorador de archivos.

- 2) **Pagos con tarjeta credito:** Para implementar esta nueva funcionalidad, usamos las siguientes pasarelas: PayU y PayPal. Esta se encarga de darle la oportunidad a los clientes de poder elegir la pasarela de pago preferida al momento del pago de la reserva. Esta funcionalidad, esta incluida tambien en la nueva aplicación para clientes.

Para llevar a cabo esta nueva funcionalidad, se uso una Interface llamada Pagos, la cual tiene dos metodos abstractos los cuales son para bloquear el cupo de una tarjeta de credito y de procesar y guardar la informacion del cliente que uso alguna pasarela (claramente estos dos metodos solo pueden ser implementados por clases diferentes a la interface).

>  Pagos.java
>  PayPal.java
>  PayU.java

En la imagen anterior, podemos ver como las clases PayPal y PayU, implementan los metodos de la interface Pagos. Todo lo anterior se hizo con el objetivo de poder implementar mas pasarelas a futuro, y que sea mas facil y que tenga menor acoplamiento al momento de querer hacerlo.

A continuacion, se explicaran los metodos implementados por la clase PayU (tomandola como ejemplo).

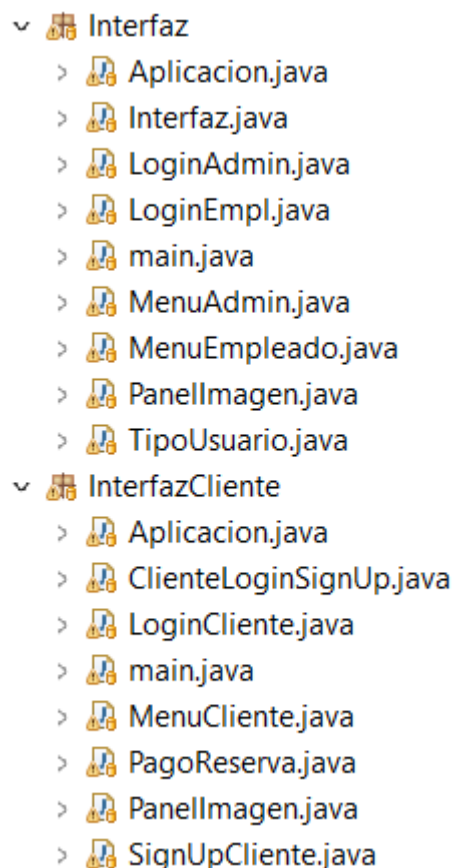
```
@Override
public void pagoProcesado(TarjetaCredito tarjeta, Cliente cliente, double amount) {
    String ruta = "data\\PayU\\clientesPagaron.txt";
    try {
        BufferedWriter archivoClientes = new BufferedWriter(new FileWriter(ruta, true));
        archivoClientes.write(cliente.getNombre() + ";" + Long.toString(tarjeta.numero));
        archivoClientes.newLine();
        archivoClientes.close();
    } catch (IOException e){
        e.printStackTrace();
    }
}
```

pagoProcesado(tarjeta,cliente,cobro)

Este metodo se encarga de poder recibir los datos de la Tarjeta de credito de un cliente, el propietario, y el monto que se quiere cobrar de la tarjeta (teniendo en cuenta una situacion en donde siempre habra suficiente monto en la tarjeta para realizar el pago). Luego, toma los datos que recibio, y escribe una nueva linea en el archivo que se encuentra en la carpeta **...\\data\\PayU** en donde estara el nombre del cliente, y el numero de tarjeta asociado a este. Esto, se hace con el objetivo de tener un registro de que clientes usaron la pasarela al

momento de realizar el pago de la reserva, y así poder llevar un mejor manejo del sistema de pagos.

- 3) **Aplicación para clientes:** Para realizar la nueva aplicación para clientes, lo que se hizo fue lo siguiente: i) Se eliminaron todas las ventanas y funcionalidades que realizaba el cliente en la Interfaz original; ii) Luego, se pasaron a un nuevo paquete llamado InterfazCliente (ver imagen). Esto se hizo con el objetivo de que las aplicaciones se puedan ejecutar por separado. Sin embargo, esto no afecta las funcionalidades entre ambas. Por ejemplo, si se quiere conocer el estado del vehículo que reservó un cliente (en la aplicación nueva), va a aparecer que este estará reservado.



Esta nueva funcionalidad es importante, ya que logramos separar dos aplicaciones para dos tipos de usuarios diferentes. Además de esto, esta nueva aplicación nunca pierde relación con la lógica del mundo, ya que también tiene la clase aplicación la cual hace de Controller, para conectarse con la lógica.

- 4) **Pruebas automáticas:** Por último, empleamos la clase JUnit para verificar si la carga de datos se estaba realizando correctamente. Para ello, implementamos test en cada carga. Los métodos a probar provienen de la lógica que se encuentra en la clase Empresa, dichas funciones son: cargarVehiculos, cargarAdministrador, cargarSedes, cargarClientes, cargarEmpleados, cargarSeguros, cargarCategorias.

La base de verificación las pruebas es saber si la cantidad de elementos en las distintas estructuras de datos son correctas. Por ejemplo, en la carga de vehículos, están organizados dentro de un mapa que tiene longitud de 10. Para ello usamos el siguiente formato de testeo:

```
@Test
void testCargaVehiculo() throws ParseException {
    File fileVehiculos = new File("\\data\\vehiculos.txt");
    if (fileVehiculos.exists()) {
        empresa.cargarVehiculos(fileVehiculos);
        int cantidadEsperada = 10;
        assertEquals(cantidadEsperada, empresa.getVehiculos().size());
    } else {
        System.out.println("El archivo de sedes no se encuentra en la ruta especificada.");
    }
}
```

Primero, creamos el objeto tipo File con la ruta del archivo que contiene la información de los vehículos que tiene la empresa. Si el documento existe, procedemos a probar la carga de vehículos. Luego, creamos una variable de tipo entero donde estará la cantidad esperada de elementos que tiene el hash map para después con el método asserEquals comparar dicha variable con el mapa cargado por la función cargarVehiculos a través del método .size().

Ahora, hablando sobre las pruebas de integración en la reserva tenemos los siguientes casos:

```
if (sedes.get(sedeInicio+"\n").buscarVehiculoCategoria(categoria) == null) {
    resp = "No se encontraron vehiculos de la categoria " + categoria + "en la sede " + sedeInicio;
}
```

Si la categoría ingresada por el usuario no se encuentra en la sede de donde se generó la reserva, le aparecerá el mensaje al usuario para que este verifique bien los datos ingresados.

```
else if (cliente.verificarReserva()) {
    resp = "Ya tienes una reserva activa";
}
```

Si se llega a saber que el usuario ya tiene una reserva, se le mostrará el mensaje de alerta de que ya tiene una reserva activa, por ende, no podrá realizar otra.