

## **Proyecto 3**

Marlon David Forero

202220738

Daniel Mancilla

202221038

Miguel Rey

202112736

### **Contenido:**

- Cambios en las entidades
- Añadidos en la lógica
- Interfaz grafica

### **Cambios en las entidades**

Para esta entrega tuvimos que modificar únicamente la entidad de Compra, puesto que en la entrega anterior no habíamos incluido un atributo de fecha, cosa que fue absolutamente necesaria para esta entrega. Al añadir este atributo no tuvimos que cambiar mucho las cosas, simplemente añadir el atributo, cambiar el método toString() para que incluyera la fecha y el método que se encarga de cargar una Compra para que tenga en cuenta esta nueva parte del archivo txt (que se trata de un nuevo campo separado por coma). Por lo mismo, la estructura de la persistencia para compra es prácticamente la misma.

```
@Override
public String toString() {
    String retorno = "";
    retorno += id + ",";
    for (Integer pieza : piezas) {
        retorno += pieza + "-";
    }
    retorno += ",";
    retorno += medioDePago + ",";
    retorno += valor + ",";
    retorno += mediante + ",";
    retorno += comprador + ",";
    retorno += fechaCompra + ",";
    return retorno;
}
```

Otro cambio menor fue en la persistencia de pieza, que cambiamos el carácter para separar la información de los propietarios, cambiando el carácter “|” por “;” para esto.

## Añadidos en la lógica

Hubo dos tipos de añadidos, el primero se trata de métodos en la lógica para recopilar información (como las ventas por año o piezas según comprador o compra), el segundo fue la implementación de carga dinámica de clases para los métodos de pago.

### Métodos en lógica

Como recién se indicó, son métodos sencillos que van desde controlador hasta inventario e incluso el inventario propio de alguna entidad (como inventarioCompra por ejemplo). Estos métodos se crearon para facilitar la creación de las tablas en las funciones de consulta o para el ultimo requerimiento de crear un gráfico sobre ventas anuales (el cual se detallará en la última sección del documento).

Ejemplo:

```
public ArrayList<Compra> buscarComprasPorComprador(int id) {  
    ArrayList<Compra> comprasComprador = new ArrayList<>();  
    if (comprasSegunComprador.containsKey(id)) {  
        ArrayList<Integer> idsCompras = comprasSegunComprador.get(id);  
        for (Integer idCompra : idsCompras) {  
            comprasComprador.add(compras.get(idCompra));  
        }  
    }  
    return comprasComprador;  
}
```

buscarComprasPorComprador para utilizar su resultado en la consulta de un comprador

### Carga dinámica de clases para métodos de pago

Para esto, tuvimos que modificar un poco la clase de MedioDePago y crear una subclase de esta para cada medio de pago posible. Las que implementamos fueron para PayPal, MasterCard y Nequi. La información que cada una de estas almacena en su debido archivo .txt es la información de la cuenta de la empresa a la cual deben pagar los clientes al utilizar el método de pago.

```

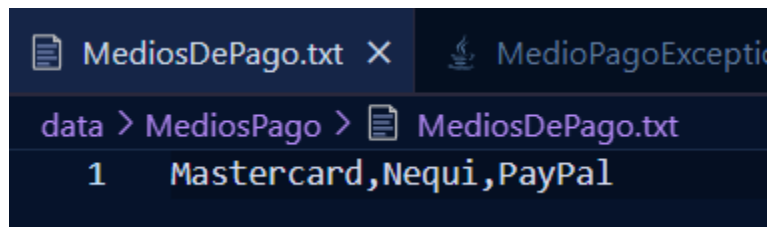
public class PayPal extends MedioDePago{
    private String email;
    private String password;
    public final static String TIPO = "PayPal";

    public PayPal(String email, String password) {
        this.email = email;
        this.password = password;
        this.tipo = TIPO;
    }
}

```

Clase creada para PayPal

Debido a estos cambios, la persistencia tuvo cambios significativos, puesto que ahora para la persistencia se maneja un archivo principal que contiene los nombres de las clases de métodos de pago



The screenshot shows an IDE window with two tabs: 'MediosDePago.txt' and 'MedioPagoException'. The 'MediosDePago.txt' tab is active, showing a file explorer view with the path 'data > MediosPago > MediosDePago.txt'. Below the path, the content of the file is displayed as '1 Mastercard,Nequi,PayPal'.

Archivo donde se guardan los nombres de las clases

Con este archivo, nuestra aplicación sabe cuales son los medios de pago existentes y procede a cargar uno por uno

```

public class PersistenciaMediosPago {
    public InventarioMediosPago cargarMediosDePago(InventarioMediosPago inventarioMediosPago) throws RuntimeException {
        try {
            File medios = new File(archivoBase);
            if (!medios.exists()) {
                Files.createFile(Paths.get(archivoBase));
            }
            String texto = Files.readString(Paths.get(archivoBase));
            String[] mediosSeparados = texto.split(regex:",");
            for (String medio : mediosSeparados) {
                File medioFile = new File(FILE_PATH + medio + ".txt");
                if (!medioFile.exists()) {
                    Files.createFile(Paths.get(FILE_PATH + medio + ".txt"));
                }
                String contenido = Files.readString(Paths.get(FILE_PATH + medio + ".txt"));
                MedioDePago medioDePago = cargarMedioDePago(medio, contenido);
                inventarioMediosPago.agregarMedioPago(medioDePago);
            }
            return inventarioMediosPago;
        } catch (IOException e) {
            System.out.println("Error al cargar los medios de pago: " + e.getMessage());
            throw new RuntimeException(message:"Error al cargar los medios de pago");
        }
    }
}

```

### Método encargado de cargar todos los medios de pago

Con esto, por cada medio de pago carga la información del archivo correspondiente (Nequi.txt por ejemplo) en un String y lo transmite al método responsable de la carga dinámica de clases, que es un método estático en la clase FactoryMediosPago

```

public class FactoryMediosPago {
    public static MedioDePago crearMedioPago(String tipo, String[] contenido) throws ClassNotFoundException, InstantiationException, IllegalAccessException {
        Class<?> clase = Class.forName("uniandes.dpoo.estructuras.model.mediosPago."+tipo);
        MedioDePago medioDePago = (MedioDePago) clase.getDeclaredConstructor(...parameterTypes:null).newInstance();
        medioDePago.cargar(contenido);
        return medioDePago;
    }
}

```

### Clase y métodos encargados de la carga dinámica de clases

Como se puede ver, este método no es directamente el que mete la información en cada clase nueva, sino que se trata de la propia clase. Únicamente crea la instancia vacía del medio de pago para que este mismo cargue la información.

Debido a este tipo de implementación, los cambios que se deben hacer para añadir un nuevo medio de pago son muy sencillos. Simplemente es crear la nueva clase, añadir el nombre de la clase al archivo que los guarda, crear su respectivo método que cargue la información a partir del string del archivo guardado (que generalmente tiene el formato de un csv, pero eso depende meramente del que implemente la clase) y finalmente su método toString para que la persistencia lo guarde automáticamente en el formato deseado.

```

public class Nequi extends MedioDePago{
    private String numeroTelefono;
    public final static String TIPO = "Nequi";

    public Nequi(String numeroTelefono) {
        this.numeroTelefono = numeroTelefono;
        this.tipo = TIPO;
    }

    public Nequi(){
        this.tipo = TIPO;
    }

    public void cargar(String[] contenido){
        this.numeroTelefono = contenido[0];
    }

    public Nequi(String[] contenido){
        this.numeroTelefono = contenido[0];
        this.tipo = TIPO;
    }

    @Override
    public String toString() {
        return numeroTelefono + ",";
    }
}

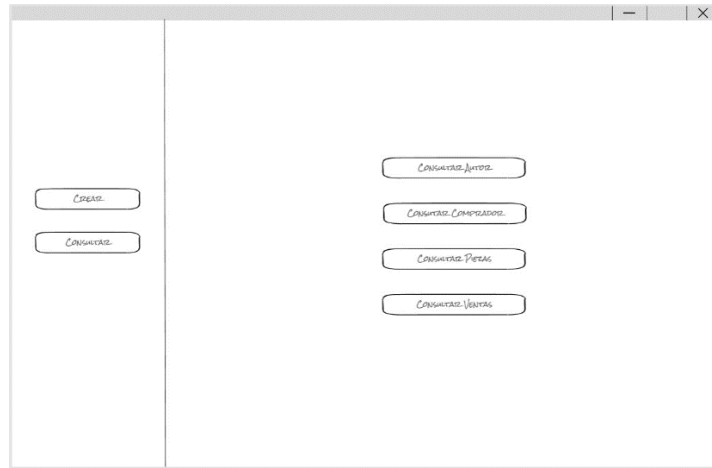
```

Clase Nequi completa, con sus métodos para cargar a partir de un array de strings y su propio toString para guardarlo con el formato deseado

Por lo tanto, es bastante sencillo añadir un método de pago nuevo, lo único que falta es agregar la lógica necesaria para comunicarse con la API o el servicio que se necesite para procesar el pago, pero nosotros no implementamos eso porque no es necesario para el proyecto.

## Interfaz Grafica

Para la interfaz gráfica decidimos utilizar Swing y la bocetamos mediante la herramienta figma, el diseño de la implementación se trata de un JFrame principal el cual va reemplazando el contenido de sus paneles dependiendo del comando.



Boceto de la página principal con el submenú de consultar

Todos los bocetos no se van a incluir en este documento, pero habrá un pdf adjunto con estos.

Para la implementación manejamos una pantalla de login (a la cual se puede acceder utilizando como usuario y contraseña “admin”) que luego da paso al menú principal con dos botones para abrir cada uno un submenú. Estos submenús son para ver las funcionalidades de creación y de consulta, simplemente con darle a cualquiera de estos reemplaza el panel del submenú con el panel necesario para esta funcionalidad.

Para la gran mayoría de funcionalidades, manejamos los ids (que se pueden consultar fácilmente viendo el nombre de los archivos de la persistencia, que son el mismo id de esa instancia de cada tipo de entidad). Están todas las funcionalidades para crear una pieza nueva, vender una pieza, crear un empleado o comprador y para crear una subasta. Igualmente, en el submenú de consultar están los botones para consultar un comprador, una pieza, un autor y las ventas durante el año.

Cada panel es bastante intuitivo para su funcionalidad, ya sea que pide todos los datos necesarios para guardar una información (y en algunos casos, como en el de crear una pieza, crea un panel adicional para guardar información que no se pide en la entidad simple, como la información adicional según un tipo de pieza), mientras

que para los de consulta pide normalmente un único dato (o ninguno para el caso de las ventas).

Respecto a los bocetos, la mayoría de las paginas son bastante iguales, no personalizamos mucho la parte visual puesto que preferimos mantener una estética simple para no complicar el código de la interfaz. Además de esto, hay algunos bocetos que pueden diferir con el producto final, como es el caso de la consulta de ventas que paso de ser un grafico de cuadros a un grafico de barras hecho completamente con swing, debido a que representó una complejidad menor para su implementación. Finalmente hubo casos de paneles y pantallas que inicialmente no pensamos en los bocetos pero que finalmente aplicamos para el proyecto, las cuales generalmente eran necesarias para funciones menores (como moverse entre paneles o funciones adicionales como utilizar el medio de pago seleccionado).

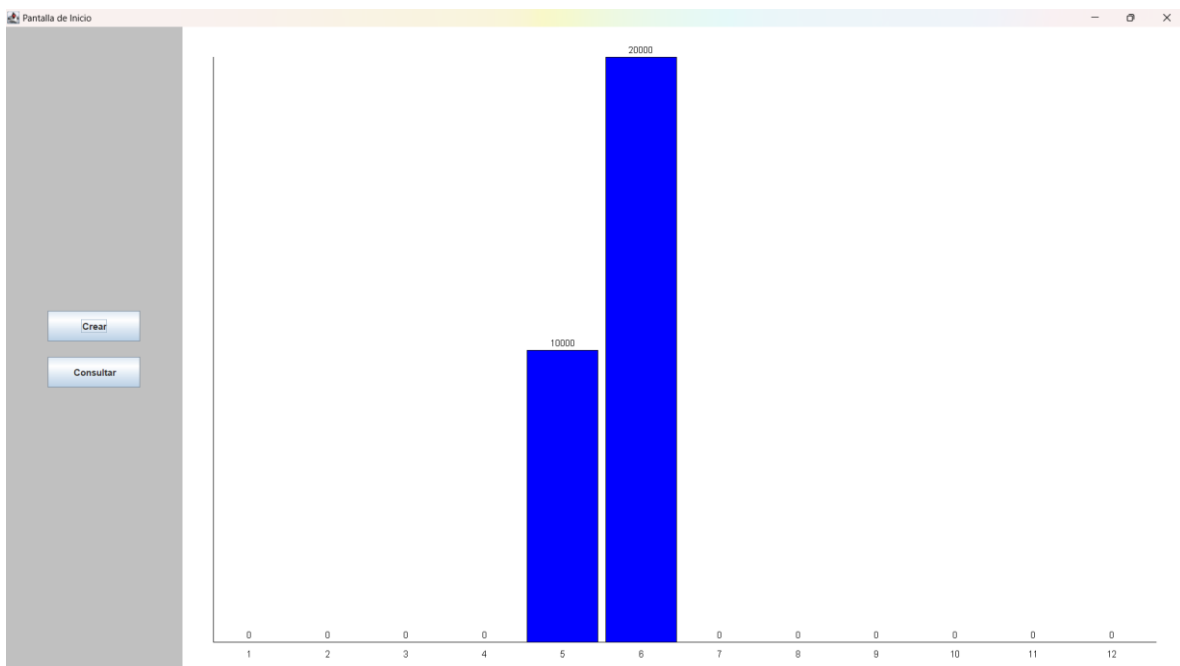


Grafico generado por la consulta de ventas anuales, cada barra representa un mes (no están todos puesto que no alcanzamos a incluir tanta información)

### Tips para utilizar la interfaz

- En el caso de no conocer ningún id para el uso de esta, siempre se puede revisar la carpeta data con la persistencia, que cada una tiene archivos cuyo nombre es [número].txt y ese número es el id de la entidad que representa el archivo.
- Para utilizar el login, manejamos el usuario y contraseña “admin”, puesto que es el que da todas las funcionalidades.



- Para inicializar la interfaz, hay que correr la clase Main.java en la carpeta “Interfaz”, algo importante de notificar puesto que también hay otras clases main en la carpeta de la consola que entregamos para el anterior proyecto.