Documento de diseño: Proyecto 3

Santiago Quiroz Pintor

Miguel Flórez Gutierrez

Carlos Daniel Ramírez Rodríguez

Luis Sebastián Contreras Diaz

Universidad de Los Andes

Diseño y Programación Orientado a Objetos

2 de Junio de 2024

Introducción

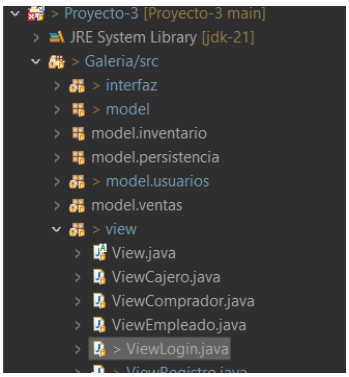
En el proyecto 3 decidimos mantener la implementación realizada anteriormente donde el primer inicio de las interfaces es en consola pero, es solo la configuración del administrador, ya luego cuando se sabe y verifica que es el administrador se pasa a la vista inicial de ViewAdministradorGUI() la cual si es la interfaz gráfica del mismo, lo anterior es lo mismo que ocurre con Comprador pues se necesita saber quien es el que ingresa para saber la vista que se le brindará, en este primer caso de vista del administrador GUI se implementa un frame (desde la creación de la clase) y se crea un panel auxiliar donde irán los botones de opciones para el administrador como se ve en la siguiente imagen:

```
private void initialize() {
    setTitle("Administrador - Galería");
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setSize(400, 300);
    setLocationRelativeTo(null);

    JPanel panel = new JPanel();
    panel.setLayout(new GridLayout(0, 1, 10, 10));

    JLabel welcomeLabel = new JLabel("Bienvenido, Administrador!", JLabel.CENTER);
    welcomeLabel.setFont(new Font("Arial", Font.BOLD, 16));
    panel.add(welcomeLabel);
```

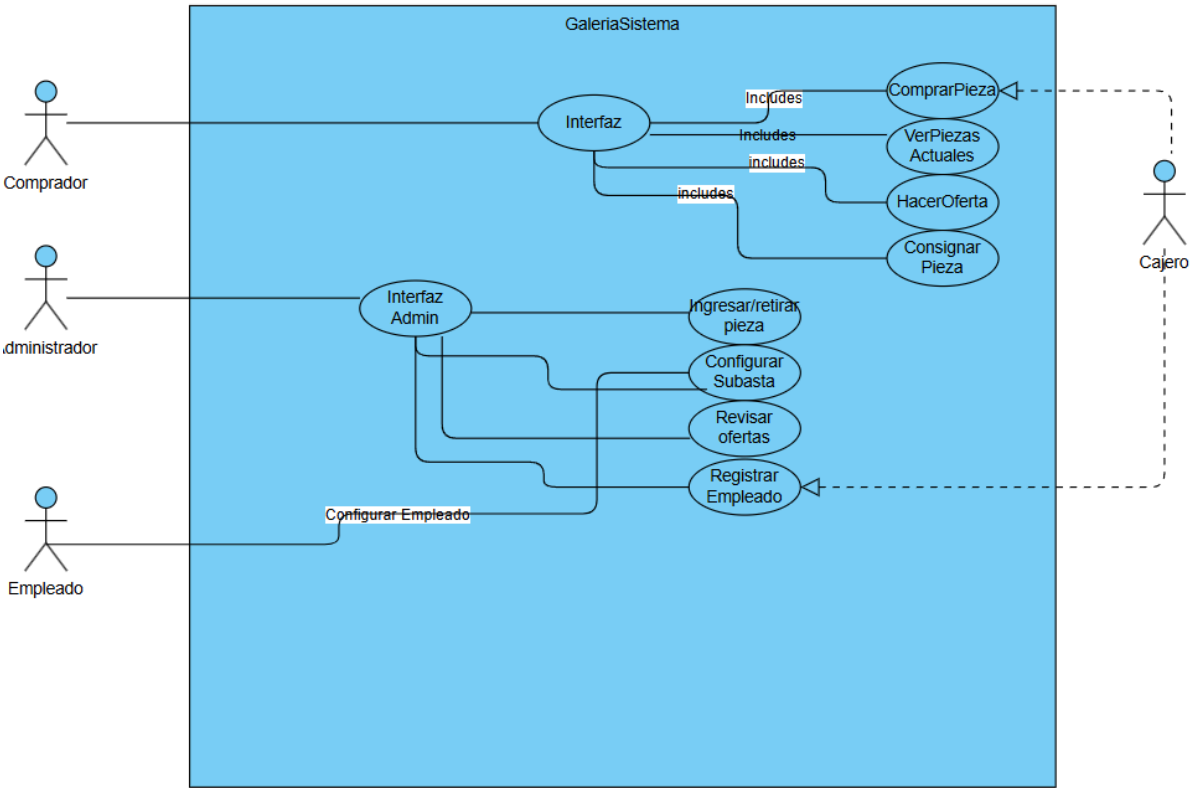
El mismo proceso se repite con las demás vistas GUI como en la del comprador tambien se visualiza a continuación:

```
private void initialize() {
    setTitle("Comprador - Galería");
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setSize(400, 300);
    setLocationRelativeTo(null);

    JPanel panel = new JPanel();
    panel.setLayout(new GridLayout(0, 1, 10, 10));
    JLabel welcomeLabel = new JLabel("Bienvenido " + comprador.getNombre() + "!", JLabel.CENTER);
    welcomeLabel.setFont(new Font("Arial", Font.BOLD, 16));
    panel.add(welcomeLabel);
```

La implementación de estas interfaces se hicieron con JavaSwing y JavaAWT. Se hicieron más cambios para la implementación de los nuevos requerimientos adicionales a el uso del patrón Singleton para la clase Administrador con el fin de setear un único administrador y que sea el único que agregue o quite piezas, y demás métodos. Además se uso la implementación del patrón MVC como se puede observar se tienen paquetes separados de lógica otros de creación y otros de interfaz como se muestra en esta imagen:



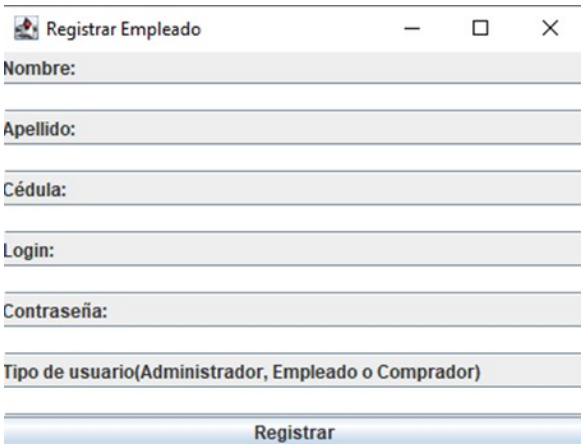
Historias de Usuarios:



ViewLogin: El View para ingresar se ve así:

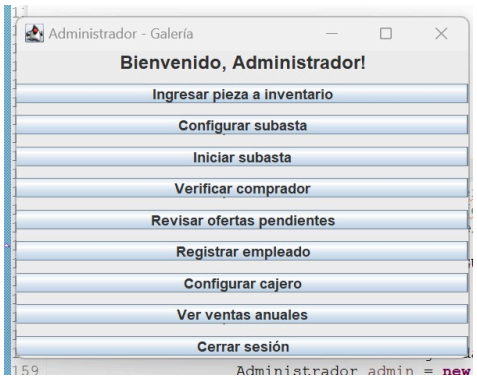
Si el usuario ya se registró a la galería, dependiendo de su tipo de usuario aparecerá su view respectivo.

Registro: Para registrarse sale la siguiente Ventana:



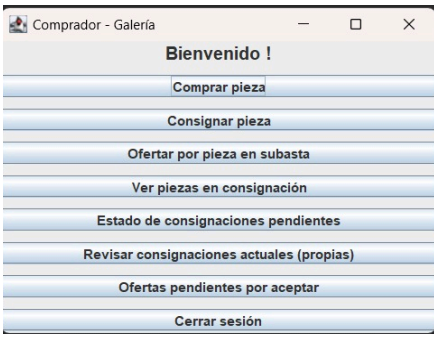
Para que el registro sea válido hay que llenar todas las casillas (si no aparecerá un anuncio indicando que hay que llenar todas las casillas) y escribir el tipo de usuario correctamente(si no aparecerá otro anuncio y el registro no será válido).

Administrador: El administrador tiene en su interfaz la opción de:



Donde cada uno de estos botones tendrá sus propios views.

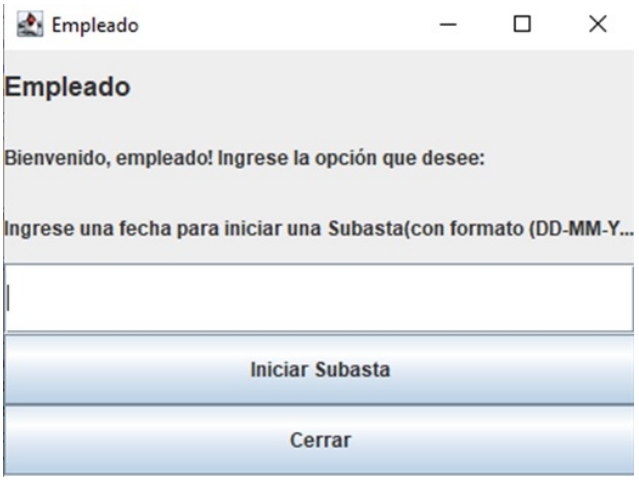
Comprador: tendrá acceso a los siguientes métodos:



/////Continuar///// -> LAS DEMÁS VIEWS

Cajero: El cajero tiene los siguientes métodos

Empleado: El empleado tiene los siguientes métodos



Decisiones de diseño

1. **Manejo de inventario**

Uno de los aspectos más importante del programa es el manejo del inventario de la galería. Para esto, se crearon distintas estructuras con el objetivo de facilitar el acceso y el ordenamiento de la información que se utiliza en el programa. La clase "Galería" es uno de los componentes de más alto nivel de la aplicación, esta actúa como un model en el que se dentro del cual se almacenan las estructuras de datos.

Para las piezas se crearon la estructuras *piezasPasadas, piezasInventario, piezasBodega* y *piezasExhibidas*. Cada estructura se trató de un ArrayList<Pieza> que permitía acceder directamente al grupo de piezas necesarias para resolver los requerimientos correspondientes. Cada vez que se modifica una de estas estructuras, se persistía la información de la galería, para así evitar cualquier pérdida de información.

2. **Acceso al sistema y usuarios**

En cuanto a los usuarios, se manejó *Usuario* como una clase abstracta que contenía todos los comportamientos y atributos necesarios para cualquier tipo de usuario que utilizara el sistema. De esta clase se desprendían dos subclases principales: los usuarios de tipo *Empleado* y los de tipo *Comprador*. Por el lado de los empleados se encontraban todos los usuarios que modificaban el y manejaban el sistema y luego, de esta clase, se desprendían las clases *Administrador* y *Cajero*, debido a que contaban con tareas más específicas que los demás empleados no realizaban.

La clase Administrador tiene una única instancia, y esta se crea solo cuando el sistema de la galería es utilizado por primera vez: cuando el directorio de datos no existe. Luego el sistema solicita los datos del administrador para crearse así mismo junto con su primer usuario registrado por medio de un *ViewRegistro*.

Una vez que el sistema ya ha sido creado, este siempre se inicia desde una instancia de *ViewLogin*. Aquí se verifica el usuario y la contraseña para dirigir al usuario al menú que le

corresponde dentro del sistema. En caso de que el usuario no exista, este puede registrarse así mismo; sin embargo, este proceso solo crea usuario de tipo *Comprador*.

El registro de los usuarios de tipo *Empleado* lo realiza únicamente el administrador del sistema. El administrador ingresa el nombre, apellido y número de cédula del empleado y el sistema se encarga de crear automáticamente las credenciales del empleado, debido a que no hace mucho sentido que cualquier persona se registre libremente como empleado de la galería.

Para ingresar al sistema, cualquier usuario previamente registrado puede iniciar sesión libremente dentro del sistema.

3. **Subastas**

Historias de Usuarios: Lista las historias de usuarios, que son descripciones cortas y simples de una característica o función del sistema desde la perspectiva del usuario. Ayuda a asegurar que el desarrollo del software se mantenga enfocado en las necesidades del usuario.

Decisión 1: Uso de Swing para la Interfaz Gráfica

Justificación: Swing es una biblioteca de componentes de interfaz gráfica de usuario (GUI) para Java que proporciona un conjunto de componentes personalizables y fáciles de usar, lo que facilita el desarrollo de interfaces de usuario intuitivas y atractivas.

Decisión 2: Organización en GridLayout

Justificación: Se eligió un GridLayout para organizar los botones de acción en un formato limpio y ordenado, lo que facilita la navegación y la usabilidad. Además la facilidad de posición en filas y columnas lo que nos permitió en general añadir formados de columnas personalizados que generaban una visualización mucho mejor a diferencia de los otros Layouts.

Decisión 3: Ventanas Emergentes para Detalles

Justificación: Se utilizan JFrames adicionales para capturar detalles específicos, como ingresar nueva pieza o configurar subastas, lo que mantiene la interfaz principal despejada y centrada en las opciones de navegación.

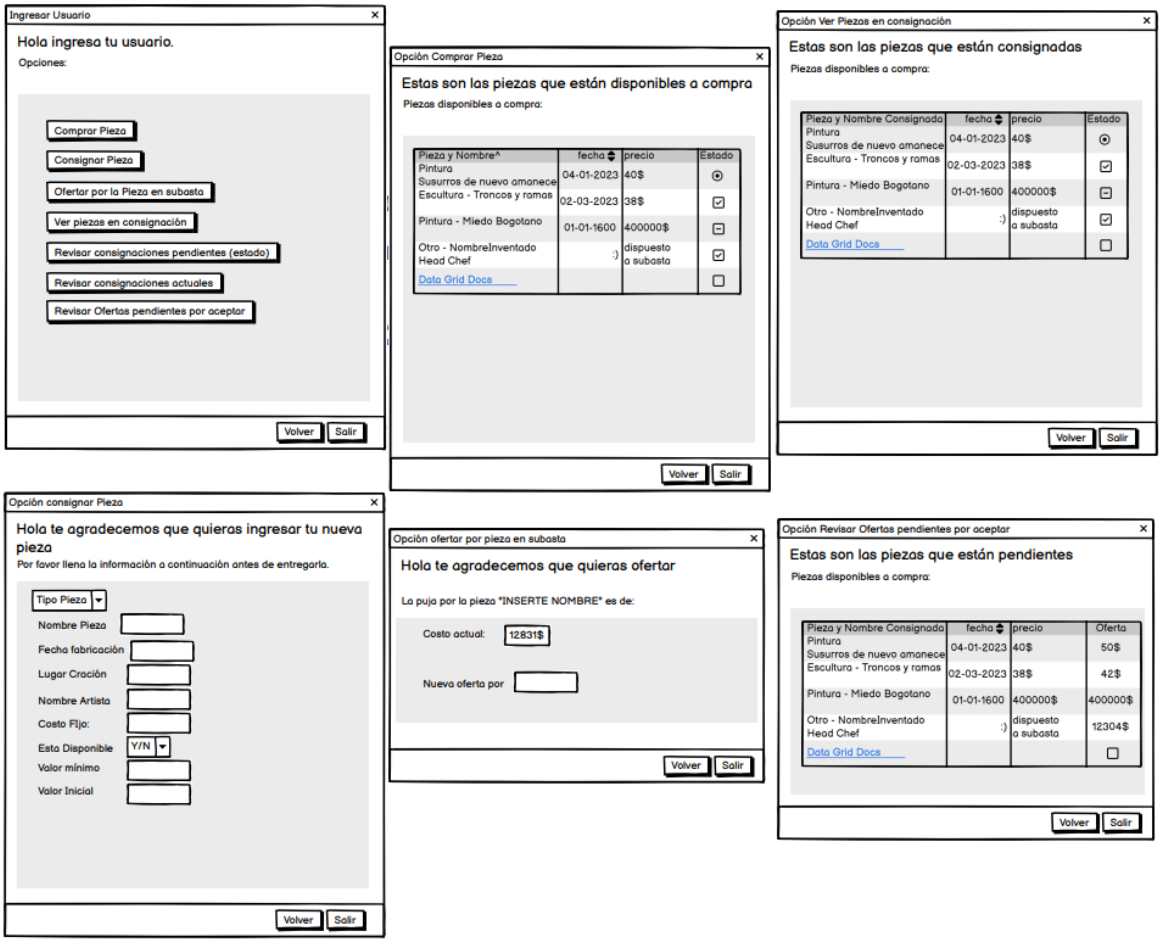
Decisión 4: El uso de singleton para administrador

Justificación: Debido a la necesidad de setear un único administrador en la galería con el fin de delimitar el ingreso por eso se usa el patrón de singleton donde la misma instancia de administrador inicial es la que se distribuye a lo largo del proyecto y las demás clases que usan sus métodos.

Decisión 5: Persistencia en Serializable

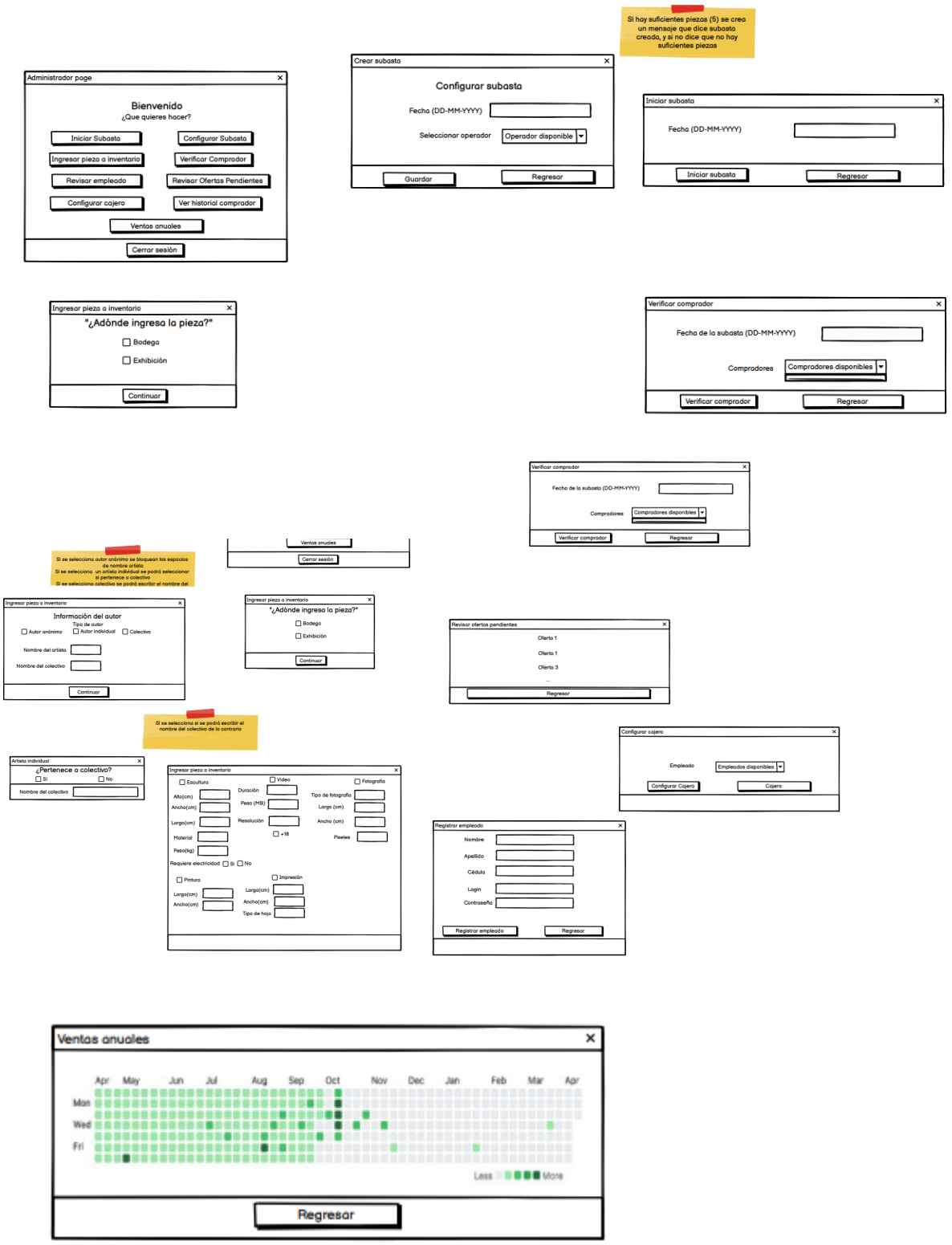
Justificación: la razón principal por la cual se usó Serializable fue debido a que si persistimos con este método la galería(todo) no tendríamos que generar un archivo de texto plano por cada clase por lo anterior usamos la implementación de este tipo de serialización.

Decisión 6: Diseño de la GUI de comprador:



A partir de este borrador se planteó la solución para la implementación de las interfaces.

Decisión 7: Diseño GUI de Administrador



Diseño 8: Diseño GUI demas interfaces

3. Requerimientos del Sistema

Requerimientos Funcionales: Describe todas las funcionalidades que el sistema debe realizar.

Requerimientos No Funcionales: Incluye aspectos como seguridad, rendimiento, plataformas soportadas, etc.

Requerimientos del Sistema, los funcionales serán ingresar piezas al inventario de la mano del administrador, por otro lado también tenemos la creación de subasta donde se programa la fecha específica donde sé creará la misma y se guarda para la inicialización por parte cajero, el cual será nuestro tercer requerimiento funcional, el cual consiste en que el cajero inicie una subasta previamente creada con las piezas que ingresaron para la misma. También se tiene la verificación de un comprador pues es donde el administrador se encarga de darle un estatus específico a un usuario del tipo comprador que le permite acceder a nuevos métodos como por ejemplo la creación de ofertas para una pieza, adicional a la participación en subastas y la liberación del cupo específico para la puja y la oferta en aquellas subastas. Otro requerimiento es la revisión de ofertas pendientes donde al administrador luego de una posible oferta de un cliente para compra una pieza o la solicitud de algún servicio el administrador decide si aceptarla o no, también se tienen los registros de empleados el cual consiste en la creación de los mismos y a su vez se tiene el requerimiento de la configuración de un cajero. Para este segundo proyecto se implementaron los nuevos requerimientos y se corrigieron los anteriormente nombrados, en primer lugar se implementó el acceso a la historia de una pieza, la pieza posee un atributo que es su historia, allí va añadiendo los dueños que va teniendo y el precio por el que es comprada, así como su propio nombre y su tipo, además de la fecha por la que fue vendida, de esta manera se hace más fácil acceder a ella, y se va almacenando sin que se pierda, además el administrador y empleados pueden

verla según sea el caso, en segundo lugar se abordó de la siguiente manera el recuperar la historia de un artista:

```
public HashMap<String, ArrayList<Pieza>> procesarHistoriaArtista(String nombreArtista){
    HashMap<String, ArrayList<Pieza>> historiaArtista = new HashMap<String, ArrayList<Pieza>>();
    int precioAcumulado = 0;
    ArrayList<Pieza> piezasPasadas = galeria.getPiezasPasadas();
    for(Pieza p : piezasPasadas) {
        ArrayList<Artista> artistasSub =   p.getArtistas();
        for (Artista a : artistasSub ) {
            String nombreArtista1 = a.getNombre();
            if(historiaArtista.containsKey(a.getNombre())){
                ArrayList<Pieza> arreglo = historiaArtista.get(a.getNombre());
                arreglo.add(p);
                precioAcumulado += p.getCostoFijo();

            }
            else {
                ArrayList<Pieza> arrayPiezas = new ArrayList<Pieza>();
                historiaArtista.put(nombreArtista1, arrayPiezas);
                ArrayList arreglo = historiaArtista.get(a.getNombre());
                arrayPiezas.add(p);
                precioAcumulado += p.getCostoFijo();

            }
        }
    }
    System.out.println(precioAcumulado);
    return historiaArtista;
}
```

Donde en primer lugar se crea un HashMap donde se guardaran a partir de llaves (Artistas) y un ArrayList como valor donde primero se recorre y se sacan los valores de nombre del artista y se empiezan a recorrer la lista de piezas que se encuentra en la estructura recién creada para así crear parejas individuales y únicas, para que así si encuentra una coincidencia entre el creador y la pieza, la ingrese al hashmap, a su vez se guardan los costosFijos para verificar que obra es la de mayor ganancia, a su vez la información de venta, como lugar y fecha del mismo estará en la información de la pieza.

En tercer lugar, la implementación del último requerimiento pedimos como parámetro el login del usuario comprador de cual queremos obtener la historia, a través de los métodos que tiene el administrador sacamos cuál es el comprador específico, si se da el caso en el que el login no pertenezca al de un comprador lanza un throw de opción inválida, aquí está la imagen del código del requerimiento:

```
}
public void historialDeUnComprador(String login) {
    Usuario usuario=galeria.getUsuario(login);
    try {
    if (usuario instanceof  Comprador) {
        double valorColeccion=0;
        HashMap<String, Pieza> piezasCompradas=new HashMap<String, Pieza>();
        Comprador comprador = (Comprador) usuario;
        ArrayList<Pieza>piezasDuenio=comprador.getPiezasActuales();
        ArrayList<Venta>ventas=new ArrayList<>(galeria.getVentas().values());
        for (Venta venta:ventas) {
            if(venta.getComprador().equals(comprador)) {
            piezasCompradas.put(venta.getFecha(), venta.getPieza());
            }
            if(piezasDuenio.contains(venta.getPieza())) {
                valorColeccion=valorColeccion+venta.getvalorOferta();
            }
        }
        Set<String> Llaves = piezasCompradas.keySet();
        List<String> fechas = new ArrayList<>(Llaves);
        for(String fecha:fechas) {
            if(piezasCompradas.get(fecha).getPropietario().equals(comprador)) {
                System.out.println("Esta pieza: "+piezasCompradas.get(fecha).getTituloPieza()+" fue comprada en "+fecha+" y actualmente es dueño" );
            }else {
                System.out.println("Esta pieza: "+piezasCompradas.get(fecha).getTituloPieza()+" fue comprada en "+fecha+" y actualmente es no dueño" );
            }
        }
        System.out.println("El valor de la colección es"+ valorColeccion);
        }
    else {
        throw new IllegalArgumentException("Opción inválida.");
    }
    }catch (IllegalArgumentException e) {
        System.out.println(e.getMessage());
    }
    }
```

Luego creamos un mapa que tenga como llave la fecha en la que el comprador adquirió la pieza, y como valor la pieza en sí, este mapa lo llenamos recorriendo la lista de ventas la cual me da la información que necesito. Después de que el mapa ya se haya llenado, recorremos los valores de este (las piezas que alguna vez compró el usuario de comprador) y rectificamos que estas aún sean propiedad de esta y así sabemos de cual es propietario o no cumpliendo con el requerimiento.

Para los requerimientos no funcionales se tiene la persistencia la cual ya se corrigió a diferencia de la primera entrega, se implementó un serializable debido a la facilidad que brinda a la hora de persistir la clase que almacena a las demas (Galería) si hacemos un implements de serializable en las demás clases será sencillo persistir todo el programa, por ello cambiamos el primer intento de persistencia por medio de un txt plano (fichero). También se generó un delay a medida local para el manejo de varios usuarios en la subasta, y generar así un view específico en un tiempo específico para mostrar una pieza en un tiempo definido. Por último pero no menos importante la implementación de una seguridad importante, donde se crean unos passwords automáticos de empleados donde siguen los mecanismos de necesidad de un carácter en mayúscula uno en minúscula y un carácter especial “#$%!”&”.

*** Administrador**

Gestión de Inventarios: El administrador puede ingresar nuevas piezas de arte al inventario.

Configuración y Gestión de Subastas: El administrador puede configurar, iniciar y gestionar subastas.

Verificación de Compradores: El administrador puede verificar compradores para las subastas.

Revisión de Ofertas: El administrador puede revisar y aceptar o rechazar ofertas de piezas.

Registro de Empleados: El administrador puede registrar nuevos empleados en el sistema.

Configuración de Cajeros: El administrador puede designar empleados como cajeros.

Visualización de Ventas Anuales: El administrador puede visualizar las ventas anuales de la galería.

***Comprador**

*** Como se implemento en el proyecto 3 ***

```
JButton btnComprarPieza = new JButton("Comprar pieza");
JButton btnConsignarPieza = new JButton("Consignar pieza");
JButton btnOfertarPieza = new JButton("Ofertar por pieza en subasta");
JButton btnPiezasConsignadas = new JButton("Ver piezas en consignación");
JButton btnPiezasEstado = new JButton("Estado de consignaciones pendientes");
JButton btnConsignacionesActual = new JButton("Revisar consignaciones actuales (propias)");
JButton btnRevisarOfertasPendientes = new JButton("Ofertas pendientes por aceptar");
JButton btnCerrarSesion = new JButton("Cerrar sesión");

btnComprarPieza.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        interfazCompraPieza();
    }
});



btnConsignarPieza.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        interfazConsignarPieza();
    }
});
```

Donde a partir de los botones se llama a los métodos específicos a través del patrón MVC donde el view comunica con el controles en este caso interfazCompraPieza();

Se repite el patrón para los demás botones.

Como se abordó la plataforma de pagos se implementó en la clase comprador inicialmente se añadieron 3 atributos a la clase comprador los cuales son tarjetaCredito la cual será un string que el usuario tendrá por defecto, también el cvcTarjetaCredito, y por ultimo saldoTarjetaCredita el cual será un int, donde a la hora de hacer el pago por el atributo metodoPago si es igual a “Tarjeta Crédito” se inicializa luego de la creación de la pestaña auxiliar.

```java
public void comprarPieza(String tipoPieza, String idPieza, int valorOferta, String peticion, String metodoPago) {
    Pieza pieza = galeria.getPiezaPorID(tipoPieza, idPieza);
    Oferta oferta = new Oferta(pieza, this, valorOferta, peticion, metodoPago);

    if (peticion != null) {
        galeria.getAdminstrador().getOfertasARevisar().put(oferta.getIdOferta(), oferta);
        oferta.getPieza().setPropietario(this);
    } else {
        galeria.getCajero().getOfertasAceptadas().add(oferta);
        oferta.getPieza().setPropietario(this);


    if (metodoPago == "Tarjeta Credito") {
        JFrame ventanaPago = new JFrame();
        ventanaPago.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        ventanaPago.setSize(400, 300);
        ventanaPago.setLocationRelativeTo(null);
        JPanel panelTarjeta = new JPanel();
        JLabel etiquetaNumeroTarje = new JLabel("Ingrese su numero de tarjeta: ");
        JLabel  etiquetaCVC = new JLabel("Ingrese el cvc de la tarjeta (Numeros de atras)");
        JTextField numeroTarjeta = new JTextField();
        JTextField numeroCVC = new JTextField();
        GridLayout layoutTop = new GridLayout(0,4);
        JButton botonComprar = new JButton("Comprar");
        panelTarjeta.setLayout(layoutTop);
        panelTarjeta.add(etiquetaNumeroTarje);
        panelTarjeta.add(numeroTarjeta);
        panelTarjeta.add(etiquetaCVC);
        panelTarjeta.add(numeroCVC);
        ventanaPago.add(panelTarjeta);
        ventanaPago.add(botonComprar);

        botonComprar.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                // TODO Auto-generated method stub
                String numeros =  numeroTarjeta.getText();
                String numerosCVC = numeroCVC.getText();
                if (numeros== tarjetaCredito) {
                    if (numerosCVC == cvcTarjetaCredito) {
                        if (valorOferta >=saldoTarjetaCredito ) {
                        saldoTarjetaCredito -= valorOferta;
                        JOptionPane.showMessageDialog(ventanaPago, "Su transacción fue aprobada por: " + valorOferta );
                        galeria.getCajero().getOfertasAceptadas().add(oferta);
                        ventanaPago.dispose();
                        String[] options = {"JSON", "TXT", "XML"};
                        String formato = (String) JOptionPane.showInputDialog(ventanaPago, "Seleccione el formato del archivo:","Formato de Arc

                        if (formato != null) {
                            switch (formato) {
                                case "TXT":
                                    generarTXT(oferta);
                                    break;
                                case "XML":
                                    generarXMLe(oferta);
                                    break;
                            }
                        }

                    } else {
                        JOptionPane.showMessageDialog(ventanaPago, "Su transacción no fue aprobada por falta de cupo en la tarjeta");
                    }
                } else {
                    JOptionPane.showMessageDialog(ventanaPago, "Su transacción no fue aprobada por error en el CVC de la tarjeta");
                }
            } else {
                JOptionPane.showMessageDialog(ventanaPago, "Su transacción no fue aprobada por errores al ingreso de los datos");
            }
        }
    });
  }}
}
```

donde primero se comparan si los datos ingresados en la GUI son los verdaderos datos del comprador, si se verifica que es el número de la tarjeta, el número del CVC de la tarjeta y además que el cupo de la tarjeta es mayor al valor de la oferta se ejecuta el descuento y se pasa a guardar la información en uno de los formatos disponibles (no se uso JSON debido a la necesidad de descarga de la biblioteca específica) donde el usuario decide el formato en el que quiere guardar la información de factura.

Cajero:

El cajero puede ver las ofertas Aceptadas al darle al botón ver “ofertas aceptadas” aparecerá un panel con las ofertas aceptadas, si la lista de ofertas aceptadas está vacía aparecerá una

ventana diciendo que esta lista está vacía, mas abajo hay otro botón que dice “registrar ventas” que convierta las ofertas aceptadas a ventas de la galería.

Empleado:

El empleado solo puede iniciar subastas, pero es necesario llenar el “textField” con una fecha para poder realizar esta función.

5. Diseño Detallado

Los diagramas serán subidos a parte para una visualización mucho mejor.

Diagrama de clases UML:

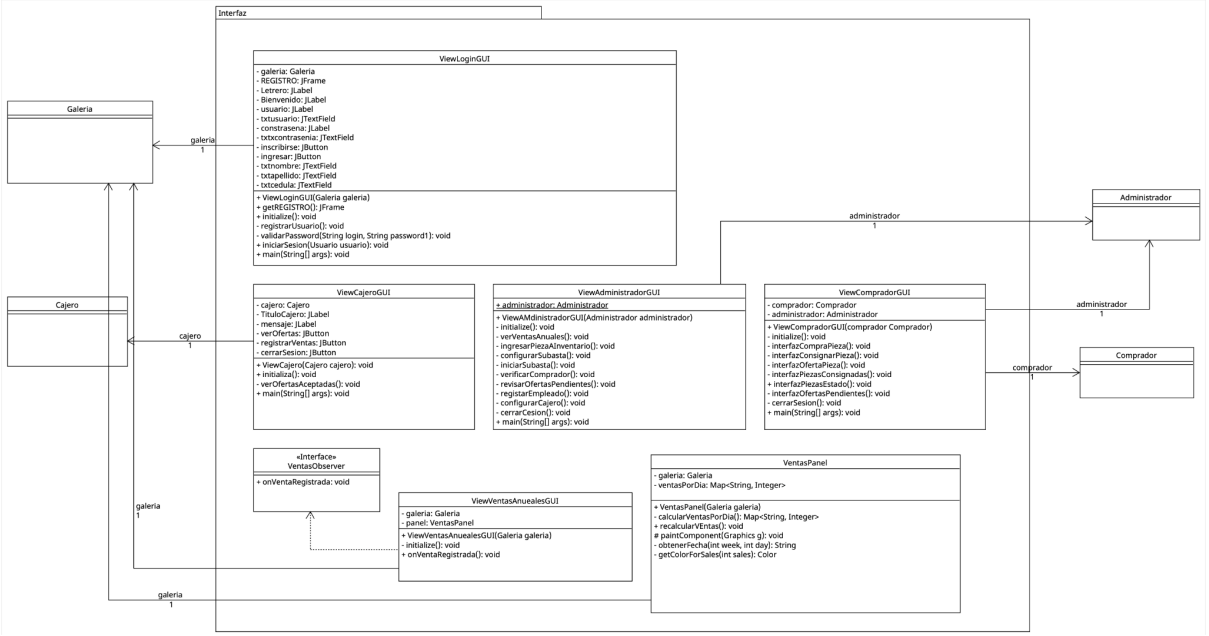


Diagrama de clase UML de alto nivel: