Universidad de los Andes Colombia | Departamento de Ingeniería de Sistemas y Computación

Diseño y Programación O.O. Proyecto 3

Sergio David Ferreira Universidad de los Andes 202210819

Orlando Suárez Universidad de los Andes 202221593

Jorge Andrés Caro Universidad de los Andes 20202807

Juan Pablo Bedoya Universidad de los Andes 202214493

I. Nivel de Diseño

I-A. *Componentes candidatos y estereotipos*

Deben existir componentes que agrupen las siguientes funcionalidades:

1. Un componente que agrupe toda la información respecto a las piezas, incluyendo la diferenciación entre las mismas, dicho componente es un *information holder*.
2. Un componente que maneje todo el tema de usuarios y cómo van a interactuar con la aplicación, dicho componente es un *controller* e *interfaz*.
3. Un componente que maneje todo el proceso de la compra, de los propietarios y los compradores, Dicho componente es a su vez *controller* e *information holder*.
4. Otro componente que maneje el proceso previo al de la compra que es el de la venta, incluyendo la oferta hecha y la diferenciación que hay entre hoy los tipos de venta, dicho componente es un *estructurador* ya que remite este proceso posteriormente a la compra.

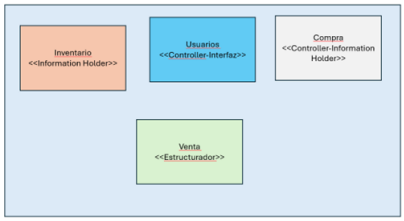

Figura 1: Componentes Generales

I-B. *Responsabilidades*

Se asignaron las responsabilidades a los diferentes componentes identificados, incluyendo responsabilidades generales necesarias para el funcionamiento del sistema.

N°	Responsabilidad	Componente
1	Iniciar el programa	Inventario
2	Registrar Pieza	
3	Historia de la pieza	
4	Cambio de Exhibida/Bodega	
5	Confirmar devolución	
6	Consultar comprador	Usuarios
7	Verificar comprador	
8	Modificar comprador	
9	Crear perfil comprador	
10	Registrar pago	
11	Consultar Historia del Artista	
12	Consignar pieza	Compras
13	Historial piezas	
14	Ofertar compra directa	Ventas
15	Confirmar venta	
16	Ofertar compra en subasta	
17	Realizar subasta	
18	Registrar subasta	

I-C. *Colaboraciones*

- El inventario debe colaborar con comprar y ventas para las transferencias de piezas.
- Subasta debe interactuar con venta y compra al momento de realizar una subasta.
- Compras y venta interactúan al momento de confirmar una venta, inicializando compra.
- Al encontrar el historial de piezas, el elemento propietario de compras interactúa con inventario.

I-D. *Interfaz Grafica*

La interfaz gráfica de la aplicación para la Casa de Subastas se diseñó utilizando `Swing` en Java. El objetivo principal fue crear una estructura clara y fácil de navegar que permita a los usuarios interactuar con las diferentes funcionalidades del sistema de manera eficiente. A continuación, se detalla el código utilizado y se explica la decisión detrás del diseño:

```
import javax.swing.*;
import java.awt.*;

public class AuctionHouseGUI {

    public static void main(String[] args) {
        JFrame frame = new JFrame("Casa de Subastas");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(800, 600);

        JPanel mainPanel = new JPanel(new BorderLayout());

        JTabbedPane tabbedPane = new JTabbedPane();

        // Panel de inventario
        JPanel inventoryPanel = new JPanel();
        inventoryPanel.setLayout(new BoxLayout(inventoryPanel, BoxLayout.Y_AXIS));
        inventoryPanel.add(new JLabel("Inventario"));
        inventoryPanel.add(new JButton("Registrar Pieza"));
        inventoryPanel.add(new JButton("Historial de Pieza"));
        inventoryPanel.add(new JButton("Cambio de Exhibida/Bodega"));
        tabbedPane.addTab("Inventario", inventoryPanel);

        // Panel de usuarios
        JPanel usersPanel = new JPanel();
        usersPanel.setLayout(new BoxLayout(usersPanel, BoxLayout.Y_AXIS));
        usersPanel.add(new JLabel("Usuarios"));
        usersPanel.add(new JButton("Consultar Comprador"));
        usersPanel.add(new JButton("Verificar Comprador"));
        usersPanel.add(new JButton("Modificar Comprador"));
        usersPanel.add(new JButton("Crear Perfil Comprador"));
        usersPanel.add(new JButton("Registrar Pago"));
        usersPanel.add(new JButton("Consultar Historia del Artista"));
        tabbedPane.addTab("Usuarios", usersPanel);

        // Panel de ventas
        JPanel salesPanel = new JPanel();
        salesPanel.setLayout(new BoxLayout(salesPanel, BoxLayout.Y_AXIS));
        salesPanel.add(new JLabel("Ventas"));
        salesPanel.add(new JButton("Consignar Pieza"));
        salesPanel.add(new JButton("Historial Piezas Compras"));
        salesPanel.add(new JButton("Ofertar Compra Directa"));
        salesPanel.add(new JButton("Confirmar Venta"));
        salesPanel.add(new JButton("Ofertar Compra en Subasta"));
        salesPanel.add(new JButton("Realizar Subasta"));
        salesPanel.add(new JButton("Registrar Subasta"));
        tabbedPane.addTab("Ventas", salesPanel);

        mainPanel.add(tabbedPane, BorderLayout.CENTER);

        frame.add(mainPanel);

        frame.setVisible(true);
    }
}
```

Decisiones de Diseño

- **Uso de JTabbedPane:** La decisión de utilizar `JTabbedPane` permite organizar las diferentes secciones de la aplicación (Inventario, Usuarios, Ventas) en pestañas, lo cual facilita la navegación y mejora la usabilidad al agrupar funcionalidades relacionadas en un solo lugar.
- **Panel de Inventario:** Este panel incluye botones para registrar piezas, consultar el historial de piezas y cambiar el estado de las piezas entre exhibida y bodega. Estas funcionalidades son esenciales para la gestión de inventario en una casa de subastas.
- **Panel de Usuarios:** El panel de usuarios proporciona opciones para consultar, verificar, modificar y crear perfiles de compradores, así como para registrar pagos y consultar la historia de los artistas. Esto permite una gestión integral de los usuarios y sus interacciones con el sistema.
- **Panel de Ventas:** Este panel incluye funcionalidades para consignar piezas, consultar el historial de compras, realizar ofertas directas, confirmar ventas, y gestionar subastas. Todas estas acciones son cruciales para las operaciones de venta en una casa de subastas.
- **Diseño Modular:** La estructura modular del diseño facilita la expansión y el mantenimiento de la aplicación. Cada panel es independiente, lo que permite agregar o modificar funcionalidades sin afectar otras partes de la interfaz.
- **Uso de BoxLayout:** El uso de `BoxLayout` en los paneles individuales asegura que los componentes se alineen verticalmente, proporcionando una interfaz limpia y organizada.

Aplicación de la Casa de Subastas

El diseño planteado mostrado en 2 3 y 4 proporciona una estructura clara y bien organizada que cubre las necesidades principales de gestión de inventario, usuarios y ventas. La interfaz es intuitiva y permite a los usuarios realizar las tareas necesarias de manera eficiente. Además, la modularidad del diseño facilita la adaptación a futuros requerimientos y la integración con otras funcionalidades del sistema.

*I-E. **Diseño preliminar de la interfaz gráfica de la aplicación***

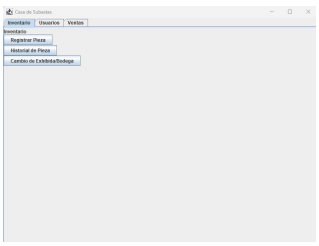

Figura 2: Diseño preliminar de la interfaz gráfica, apartado para inventario

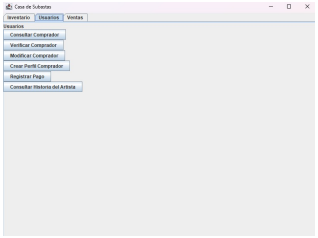

Figura 3: Diseño preliminar de la interfaz gráfica, apartado para usuarios

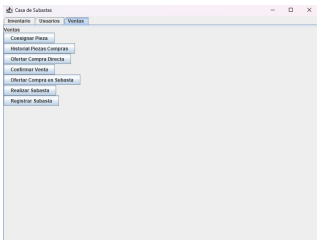

Figura 4: Diseño preliminar de la interfaz gráfica, apartado para ventas

II. Nivel 2 de diseño

II-A. Componentes candidatos y estereotipos

Inventario:
Se compone de la clase inventario, ya que fue la decisión tomada para el manejo de las piezas presentes en la galería por la facilidad para diferenciar el estado de las diferentes piezas. A su vez se tomó la decisión de manejar el hecho de que hay diferentes piezas mediante una clase piezas con los tributos generales y clases que hereden de la misma para diferenciar entre los tipos de piezas.

Usuarios:
Dentro de los usuarios, se tomó la decisión de diferenciar entre los usuarios presentes en el enunciado del problema, a través de la herencia, permitiendo a todos los usuarios gozar de características generales, mas diferenciarse en las específicas. El usuario interactúa con el programa a través de estar clases, haciendo de estos componentes *controller* e *interfaz* para el programa.

Compra:
Compra es un elemento con clases propietario, comprador y compra. Se tomó esta decisión porque en este proceso de compra, son los elementos que controlan esta transacción, y a su vez la mejor forma de modelar la información presente en los mismos elementos, haciendo de estos elementos *controller* e *information holder*.

Venta:
Finalmente, se tomó la decisión de componer el elemento venta con las clases oferta, lista de subasta, subasta individual, venta y venta fija viene del hecho que se busca un diseño competente entre estas clases, más no sobrecargar una sola con varias responsabilidades.

II-B. *Colaboraciones*

Respecto a las colaboraciones, se toma la decisión de tratar el núcleo del programa, las ventas y subastas, a través de los componentes de venta y compra. Por lo tanto, las clases en estas áreas van a tener una alta interacción entre ellas. A su vez, los usuarios interactuarán con el inventario.

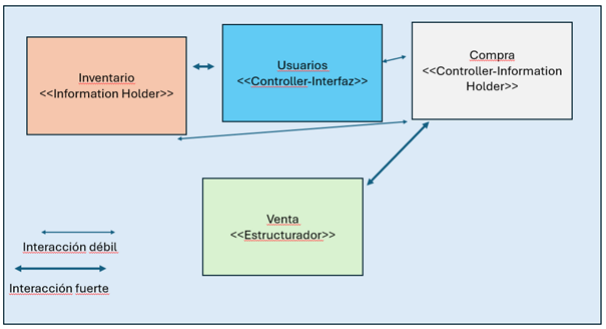

Figura 5: Colaboraciones planteadas

III. Nivel 3 de diseño

Al llegar a este punto, se definieron las responsabilidades como métodos, las colaboraciones como relaciones entre clases y se mantuvieron los estereotipos encontrados en el nivel anterior. De esta manera, el proceso ya puede ser estandarizado a elementos como los que se van a mostrar a continuación. Además, se incluyen ciertas descripciones de métodos importantes para complementar los resultados de las decisiones tomadas.

III-A. *Ventas*

III-A1. ListaDeSubastas: Clase que gestiona una colección de subastas y sus participantes dentro de la galería.

Atributos y Métodos

- **Atributos**
  - `subastas`: Lista que almacena las subastas activas.
  - `participantes`: Lista de *Compradores* que participan en cualquier subasta.
- **Métodos**
  - `agregarSubasta(Subasta subasta)`: Agrega una nueva subasta a la lista.

III-A2. Oferta: Clase que representa una oferta hecha por un comprador en una subasta.

Atributos y Métodos

- **Atributos**
  - `ofertador`: *Comprador* que realiza la oferta.
  - `valor`: Entero que representa el valor monetario de la oferta.
- **Métodos**
  - `nuevaOferta(int valor, Comprador ofertador, Subasta subasta)`: Registra una nueva oferta en una subasta si el valor supera el valor inicial y la oferta actual.

III-A3. Subasta: Clase que extiende a *Venta*, diseñada para manejar el proceso de subastar una pieza de arte. La subasta permite a múltiples compradores hacer ofertas en tiempo real hasta que la subasta sea cerrada por el operador. Cada subasta tiene un valor inicial y un valor mínimo que debe ser alcanzado para que la venta sea considerada válida.

Atributos y Métodos

- **Atributos**
  - `valorInicial`: Entero que representa el precio de partida de la subasta. Todas las ofertas deben ser superiores a este valor.
  - `valorMinimo`: Entero que indica el precio mínimo aceptable para que la subasta sea exitosa.
  - `ofertaActual`: Entero que refleja la oferta más alta hecha hasta el momento en la subasta.
  - `pieza`: Objeto de tipo *Pieza* que está siendo subastada. Incluye detalles como título, descripción, artista y fecha de creación.
  - `ofertas`: *HashMap* que almacena las ofertas realizadas durante la subasta, asociando el valor de cada oferta con el *Comprador* que la realizó.
  - `abierta`: Booleano que indica si la subasta está actualmente abierta (true) o cerrada (false).
- **Métodos**
  - `getOfertas()`: Devuelve el mapa de ofertas realizadas en la subasta.
  - `getValorInicial()`: Getter para el valor inicial de la subasta.
  - `getPieza()`: Devuelve la pieza de arte que está siendo subastada.
  - `setPagoHecho(boolean pagoHecho)`: Setter que actualiza el estado del pago realizado por el comprador ganador.
  - `iniciarSubasta()`: Método que abre la subasta permitiendo a los compradores empezar a hacer ofertas.
  - `cerrarSubasta()`: Cierra la subasta, impidiendo que se realicen más ofertas y preparando el proceso de venta para el ganador.

III-A4. Venta: Clase con un método para crear una instancia de la clase *Compra*. Crea una copia de la instancia de clase *Comprador* en una instancia de la clase *Propietario* en caso de que el usuario no tenga una instancia de dicha clase; y en caso de que ya sea Propietario, se agrega la instancia de la *Compra* a una lista con el historial de compras. Finalmente, retira la pieza del Inventario y asigna un nuevo propietario mediante el método `setPropietario` en la clase

Pieza de la pieza correspondiente. La clase *Venta* tiene asociadas una o varias piezas.

Atributos y Métodos

- **Atributos**
  - `pagoHecho`: Atributo tipo booleano que es modificado por la clase Cajero, y se verifica que sea verdadero antes de llamar al método `realizarCompra`.
- **Metodos**
  - `crearCompra()`: Este método crea una nueva instancia de la clase Compra y registra los detalles de la transacción.
  - `reclasificarComprador()`: Verifica si el comprador es un Propietario o no. Si no lo es, crea una instancia de Propietario y actualiza las referencias adecuadas.
  - `agregarHistorialCompra(Compra compra)`: Agrega la compra al historial de compras del Propietario si ya existe como tal.
  - `sacarPiezaDelInventario(Pieza pieza)`: Llama al método `sacarDelInventario` del Inventario para remover una pieza y la asocia al nuevo propietario utilizando `setPropietario()` en la clase Pieza.
  - `realizarVenta`: Tras verificar que el atributo `pagoHecho` es True, llama a todos los métodos anteriores para realizar una venta.

III-A5. VentaFija: Clase que extiende a *Venta*, diseñada específicamente para manejar ventas de piezas de arte a un precio fijo establecido previamente. Esta modalidad de venta asegura una transacción rápida donde el primer comprador que acepte el precio fijo puede adquirir la pieza inmediatamente.

Atributos y Métodos

- **Atributos**
  - `precioFijo`: Entero que representa el precio fijo al que se ofrece la pieza. Es el único precio aceptado para la transacción.
  - `comprador`: Objeto de tipo *Comprador* que representa a la persona que realiza la compra. Este atributo se asigna cuando un comprador acepta el precio fijo y completa la transacción.
- **Métodos**
  - `venta(int monto, Comprador aspirante)`: Realiza la venta si el monto ofrecido coincide con el precio fijo. Asigna el comprador y llama al método `realizarVenta()` para finalizar la transacción.
  - `compra()`: Registra la transacción de compra, creando un objeto *Compra* que incluye detalles como el identificador del comprador, la fecha y hora de la transacción, y el precio pagado.
  - `reclasificarComprador()`: Evalúa si el comprador debe ser reclasificado como un *Propietario*, basado en su historial de compras. Si es su primera compra, se crea una instancia de *Propietario*.
  - `agregarHistorialCompra(Compra compra)`: Añade la transacción de compra recién creada al historial de compras del comprador.
  - `sacarPiezaDelInventario(Inventario inventario, Pieza pieza)`: Retira la pieza del inventario y actualiza el propietario de la pieza al nuevo comprador, asegurando que la pieza no esté disponible para futuras ventas.
  - `realizarVenta()`: Verifica que el pago haya sido realizado, luego procede con la secuencia de métodos que finalizan la venta, incluyendo registrar la compra, actualizar el estado del comprador, y retirar la pieza del inventario.

III-B. Inventario

III-B1. Pieza: Clase abstracta que sirve como base para representar diferentes tipos de piezas de arte en el inventario de la galería. Cada pieza de arte es creada por un artista y puede ser vendida, exhibida o almacenada.

Atributos y Métodos

- **Atributos**
  - `titulo`: Cadena que representa el título de la pieza.
  - `anio`: Entero que indica el año de creación de la pieza.
  - `lugarCreacion`: Cadena que indica el lugar donde la pieza fue creada.
  - `autores`: Lista de cadenas que contiene los nombres de los autores de la pieza.
  - `exhibida`: Booleano que indica si la pieza está actualmente en exhibición.
  - `disponible`: Booleano que indica si la pieza está disponible para la venta.
  - `propietario`: Objeto de tipo *Propietario* que posee la pieza.
  - `precioVenta`: Doble que indica el precio al que se vendió la pieza.
  - `fechaVenta`: Cadena que representa la fecha en la que se vendió la pieza.
  - `creador`: Objeto de tipo *Artista* que creó la pieza.
  - `tipoPieza`: Cadena que especifica el tipo de la pieza (e.g., escultura, pintura).
- **Métodos**
  - `venderPieza(double precio, String fecha)`: Registra la venta de la pieza, actualizando el precio de venta, la

fecha de venta y marcando la pieza como no disponible.
- `getPrecioVenta()`, `getFechaVenta()`, `getCreador()`, `getTitulo()`, `getAnio()`, `getLugarCreacion()`, `getTipoPieza()`, `getAutores()`, `isExhibida()`, `isDisponible()`: Getters para obtener los atributos de la pieza.
- `setCreador(Artista creador)`, `setExhibida(boolean exhibida)`, `setDisponible(boolean disponible)`, `setPropietario(Propietario propietario)`: Setters para modificar los atributos de la pieza.

III-B2. Inventario: Clase que gestiona el almacenamiento y control de las piezas de arte dentro de la galería. Inventario maneja diferentes estados de las piezas, como disponibles, bloqueadas y exhibidas, y proporciona funcionalidades para modificar estos estados según las operaciones realizadas.

Atributos y Métodos

- **Atributos**
  - `piezasDisponibles`: Lista de piezas que están actualmente disponibles para la venta o exhibición.
  - `piezasBloqueadas`: Lista de piezas que están bloqueadas, posiblemente reservadas o en proceso de venta y no disponibles para transacciones adicionales.
  - `piezasExhibidas`: Lista de piezas que están actualmente en exhibición en la galería.
- **Métodos**
  - `agregarPieza(Pieza pieza)`: Añade una nueva pieza al inventario en la lista de disponibles, lanzando una excepción *PiezaDuplicadaException* si la pieza ya existe.
  - `contienePieza(Pieza pieza)`: Verifica si una pieza ya existe en cualquier lista del inventario.
  - `bloquearPieza(String titulo)`: Transfiere una pieza de la lista de disponibles a la lista de bloqueadas, indicando que la pieza no está disponible para operaciones normales.
  - `sacarDelInventario(String titulo)`: Remueve completamente una pieza de todas las listas del inventario, indicando que ya no es gestionada por la galería.
  - `exhibirPieza(String titulo)`: Mueve una pieza de la lista de disponibles a la lista de exhibidas, lanzando una excepción *PiezaInexistenteException* si la pieza no está en la lista de disponibles.
  - `guardarPiezaEnBodega(String titulo)`: Mueve una pieza de la lista de exhibidas a la lista de disponibles, preparándola para almacenamiento o venta futura, también con manejo de excepciones si la pieza no se encuentra.
- **Métodos Privados**
  - `buscarPiezaPorTitulo(String titulo, ArrayList<Pieza>listaPiezas)`: Método auxiliar para buscar una pieza por su título en una lista específica, retornando la pieza si la encuentra o null si no.

III-B3. Escultura: Clase que extiende a *Pieza*, diseñada para representar esculturas dentro del inventario de la galería. Las esculturas tienen características tridimensionales específicas y pueden requerir instalaciones especiales.

Atributos y Métodos

- **Atributos**
  - `ancho`, `alto`, `profundidad`: Dimensiones de la escultura en metros.
  - `material`: Cadena que describe el material de la escultura.
  - `peso`: Peso de la escultura en kilogramos.
  - `necesitaElectricidad`: Booleano que indica si la escultura requiere electricidad para su exhibición.
  - `detallesInstalacion`: Detalles adicionales sobre cómo debe ser instalada la escultura.
- **Métodos**
  - `getAncho()`, `getAlto()`, `getProfundidad()`, `getMaterial()`, `getPeso()`, `isNecesitaElectricidad()`, `getDetallesInstalacion()`: Métodos para obtener los detalles de la escultura.

III-B4. Fotografía: Clase que extiende a *Pieza*, diseñada para representar obras de fotografía. Incluye detalles sobre la cámara utilizada y las dimensiones de la imagen.

Atributos y Métodos

- **Atributos**
  - `ancho`, `alto`: Dimensiones de la fotografía en metros.
  - `camara`: Cadena que indica el tipo de cámara utilizada para tomar la fotografía.
- **Métodos**
  - `getAncho()`, `getAlto()`, `getCamara()`: Métodos para obtener las especificaciones de la fotografía.

III-B5. Pintura: Clase que extiende a *Pieza*, diseñada para representar pinturas. Incluye detalles sobre la técnica y el estilo de la pintura.

Atributos y Métodos

- **Atributos**
  - `ancho`, `alto`: Dimensiones de la pintura en metros.
  - `tecnica`: Cadena que describe la técnica de pintura utilizada.
  - `estilo`: Cadena que describe el estilo artístico de la pintura.
- **Métodos**
  - `getAncho()`, `getAlto()`, `getTecnica()`, `getEstilo()`: Métodos para obtener los detalles de la pintura.

III-B6. Video: Clase que extiende a *Pieza*, diseñada para representar obras de video. Incluye detalles sobre el idioma y la duración del video.

Atributos y Métodos

- **Atributos**
  - `idioma`: Cadena que indica el idioma del video.
  - `duracion`: Duración del video en minutos.
- **Métodos**
  - `getIdioma()`, `getDuracion()`: Métodos para obtener las especificaciones del video.

III-C. Compradores

III-C1. Compra: Clase que representa una transacción de compra de una pieza de arte. Registra detalles como el identificador de la compra, la fecha, el monto pagado, y la pieza adquirida.

Atributos y Métodos

- **Atributos**
  - `id`: Entero que representa el identificador único de la compra.
  - `fecha`: Objeto *LocalDateTime* que almacena la fecha y hora exactas de la compra.
  - `monto`: Entero que representa el monto pagado por la pieza.
  - `pieza`: Objeto *Pieza* que representa la pieza de arte comprada.
- **Métodos**
  - No se detallan métodos específicos en la descripción proporcionada. Supone que existen getters para cada atributo.

III-C2. Comprador: Clase que representa a un comprador en la galería. Mantiene información de contacto, credenciales de acceso, y un registro de compras realizadas.

Atributos y Métodos

- **Atributos**
  - `verificado`: Booleano que indica si el comprador ha sido verificado.
  - `nombre`: Cadena que representa el nombre del comprador.
  - `identificador`: Entero que representa el identificador único del comprador.
  - `telefono`: Entero que representa el número de teléfono del comprador.
  - `login`, `password`: Cadenas que representan las credenciales de inicio de sesión del comprador.
  - `compras`: Lista de objetos *Compra* que representan las compras realizadas por el comprador.
- **Métodos**
  - `isVerificado()`, `getNombre()`, `getIdentificador()`, `getTelefono()`, `getLogin()`, `getPassword()`, `getCompras()`: Getters para obtener la información del comprador.
  - `setTelefono(int telefono)`: Setter para actualizar el número de teléfono del comprador.
  - `verificarPasswd(String psswd)`: Método para verificar la contraseña del comprador.

III-C3. Propietario: Clase que extiende a *Comprador*, específicamente diseñada para representar a un propietario de piezas de arte. Hereda todos los atributos y métodos de *Comprador*.

Atributos y Métodos

- **Atributos**
  - Hereda todos los atributos de *Comprador*.
- **Métodos**
  - Hereda todos los métodos de *Comprador*.

III-D. Usuarios

Usuarios es una superclase y las siguientes clases son subclases de la misma, por lo tanto heredan las propiedades básicas de login y contraseña.

III-D1. Admin: Clase que extiende a *Usuario*, diseñada para gestionar actividades administrativas como registrar piezas, confirmar ventas y gestionar empleados dentro de una galería.

Atributos y Métodos

- **Atributos**
  - `nombre`: Cadena que representa el nombre del administrador.
  - `id`: Entero que identifica de manera única al administrador.
  - `empleados`: Diccionario que almacena objetos *Empleado*, indexados por su login.
- **Métodos**
  - `registrarPieza(Pieza pieza, Inventario inventario)`: Intenta agregar una pieza al inventario. Captura y maneja la excepción *PiezaDuplicadaException* si la pieza ya existe.
  - `confirmarVenta(Venta venta)`: Verifica si el pago de una venta ha sido realizado antes de proceder a finalizar la venta.

- `agregarEmpleado(String nombre, int identificador, String login, String password, String cargo)`: Crea y agrega un nuevo empleado al diccionario de empleados, pudiendo ser un *Operador* o un *Cajero* según el cargo.
- `consultarComprador(int id, List<Comprador>compradores)`: Busca y retorna un comprador por su identificador dentro de una lista proporcionada.
- `verificarComprador(Comprador comprador)`: Retorna un booleano indicando si un comprador está verificado.

III-D2. Artista: Clase que extiende a *Usuario*, representando a un artista que puede ser también un vendedor de piezas de arte dentro de la galería.

Atributos y Métodos

- **Atributos**
  - `nombre`: Cadena que representa el nombre del artista.
  - `idEmpleado`: Entero que identifica de manera única al artista como empleado.
  - `cargo`: Cadena que describe el rol específico del artista dentro de la galería.
- **Métodos**
  - `crearObra(String titulo, String descripcion)`: Permite al artista crear una nueva obra de arte, especificando su título y descripción.
  - `exponerObra(Obra obra, SalaDeExhibicion sala)`: Asigna una obra de arte a una sala de exhibición específica.
  - `getNombre()`, `setNombre(String nombre)`: Getters y setters para el nombre del artista.
  - `getIdEmpleado()`: Getter para el identificador del empleado.
  - `getCargo()`: Getter para el cargo del empleado.

III-D3. Cajero: Clase que extiende a *Empleado*, diseñada para gestionar las transacciones financieras dentro de la galería, incluyendo el manejo de pagos y el registro de ventas.

Atributos y Métodos

- **Atributos**
  - `cajaRegistradora`: Objeto que representa la caja registradora utilizada para manejar transacciones monetarias.
- **Métodos**
  - `procesarPago(Venta venta)`: Verifica el pago realizado por una venta y actualiza el estado de la venta como completada.
  - `emitirRecibo(Venta venta)`: Genera un recibo detallando la transacción realizada durante una venta.
  - `abrirCaja()`, `cerrarCaja()`: Métodos para manejar la apertura y cierre de la caja registradora.

III-D4. Empleado: Clase abstracta que extiende a *Usuario*, diseñada para ser la base de diferentes tipos de empleados dentro de la galería, como cajeros, operadores, entre otros.

Atributos y Métodos

- **Atributos**
  - `nombre`: Cadena que representa el nombre del empleado.
  - `idEmpleado`: Entero que identifica de manera única al empleado.
  - `cargo`: Cadena que describe el rol específico del empleado dentro de la galería.
- **Métodos**
  - `asistirCliente(Cliente cliente)`: Método para proporcionar asistencia a los clientes de la galería.
  - `reportarIncidencia(String incidencia)`: Método para reportar cualquier incidencia o irregularidad observada dentro de la galería.
  - `getNombre()`, `setNombre(String nombre)`: Getters y setters para el nombre del empleado.
  - `getIdEmpleado()`: Getter para el identificador del empleado.
  - `getCargo()`: Getter para el cargo del empleado.

III-D5. Operador: Clase que extiende a *Empleado*, específicamente encargada de manejar subastas y eventos relacionados con las piezas de arte dentro de la galería.

Atributos y Métodos

- **Atributos**
  - No hay atributos adicionales únicos para *Operador* más allá de los heredados de *Empleado*.
- **Métodos**
  - `registrarSubasta(Subasta subasta, ListaDeSubastas listaDeSubastas)`: Registra una nueva subasta en la lista de subastas y anuncia su creación.
  - `realizarSubasta(Subasta subasta)`: Inicia la subasta de una pieza, manejando el proceso y anuncia el inicio de la misma.

III-D6. Usuario: Clase abstracta que sirve como base para todos los tipos de usuarios dentro del sistema de la galería, incluyendo administradores, empleados y artistas.

Atributos y Métodos

- **Atributos**
  - `login`: Cadena que representa el identificador único de inicio de sesión del usuario.

- `password`: Cadena que almacena la contraseña del usuario.

- **Métodos**
  - `verificarPassword(String password)`: Verifica si la contraseña proporcionada coincide con la almacenada.
  - `cambiarPassword(String nuevoPassword)`: Permite al usuario cambiar su contraseña, siempre que la nueva contraseña no sea nula o vacía.
  - `getLogin()`: Getter para el identificador de inicio de sesión del usuario.
  - `getPassword()`: Getter para la contraseña del usuario.

III-E. Tests

III-E1. AdminTest: Clase de prueba para la clase *Admin*, que verifica la funcionalidad de los métodos críticos como registrar piezas, confirmar ventas y agregar empleados.

Métodos de Prueba

- `testRegistrarPieza()`: Verifica que una pieza se registre correctamente en el inventario sin lanzar excepciones.
- `testConfirmarVentaPagoHecho()`: Asegura que una venta se confirme correctamente cuando el pago ha sido realizado.
- `testConfirmarVentaPagoNoHecho()`: Asegura que una venta no se confirme si el pago no ha sido realizado.
- `testAgregarEmpleado()`: Prueba que un empleado se agregue correctamente al registro de empleados del administrador.
- `testConsultarCompradorEncontrado()`: Verifica que la consulta de un comprador por identificador retorne el comprador correcto cuando existe.

III-E2. ArtistaTest: Clase de prueba para la clase *Artista*, enfocada en verificar la correcta creación de obras y su asignación a exhibiciones.

Métodos de Prueba

- `testCrearObra()`: Prueba que una obra de arte sea creada correctamente con todos los detalles necesarios.
- `testExponerObra()`: Asegura que una obra de arte se asigne correctamente a una sala de exhibición especificada.

III-E3. CajeroTest: Clase de prueba para la clase *Cajero*, que evalúa el registro de pagos para ventas fijas y subastas, asegurando que los pagos se manejen correctamente.

Métodos de Prueba

- `testRegistrarPagoVentaFijaCorrecta()`: Verifica que el pago de una venta fija se registre correctamente cuando el monto es exacto.
- `testRegistrarPagoVentaFijaInsuficiente()`: Asegura que el pago de una venta fija no se registre si el monto es insuficiente.
- `testRegistrarPagoSubastaCorrecta()`: Prueba que el pago de una subasta se registre correctamente cuando el monto es igual al valor de la oferta más alta.
- `testRegistrarPagoSubastaInsuficiente()`: Verifica que el pago de una subasta no se registre si el monto es inferior al valor de la oferta más alta.

III-E4. UsuarioTest: Clase de prueba para la clase *Usuario*, que evalúa las credenciales y contraseñas de los usuarios.

Métodos de Prueba

- `testConfirmarCambioPassword()`: Verifica que la contraseña del usuario fue cambiada con éxito.
- `testCambioPasswordVacio()`: Verifica que el programa no permita que se establezaca una contraseña vacía.

III-E5. NuevosReqsTest: Clase de prueba para los nuevos requerimientos funcionales, que evalúa el correcto funcionamiento de estos.

Métodos de Prueba

- `testVerificarInclusiónDeElementoPieza()`: Verifica al obtener una historia de una pieza se añadan los elementos correctamente y la lista no esté vacía.
- `testCambioPasswordVacio()`: Verifica al obtener una historia de un artista se añadan los elementos correctamente y la lista no esté vacía.

III-F. Interfaz

III-F1. Interfaz: Clase que representa la interfaz gráfica del usuario para la aplicación de subastas. Esta clase incluye componentes visuales y controladores de eventos para manejar interacciones del usuario.

Atributos y Métodos

- **Atributos**
  - `ventanaPrincipal`: Objeto que representa la ventana principal de la aplicación.
  - `botones`, `etiquetas`, etc.: Componentes gráficos utilizados para la interacción con el usuario.
- **Métodos**
  - `inicializarComponentes()`: Método que inicializa los componentes gráficos de la interfaz.
  - `configurarEventos()`: Configura los manejadores de eventos para las acciones del usuario, como clics de botón y entradas de teclado.

III-F2. InterfazAdmin: Clase que extiende de *JFrame* y representa la interfaz gráfica del administrador para la aplicación de subastas. Esta interfaz permite la gestión y administración de componentes clave como artistas, compradores, y piezas de arte.

Atributos y Métodos

- **Atributos**

- `banner`: *JPanel* utilizado para mostrar información visual relevante en la parte superior de la interfaz.
- `menu`: *PanelMenuAdmin* que contiene las opciones y funcionalidades específicas de la administración.

- **Métodos**
  - `InterfazAdmin()`: Constructor de la clase que configura la interfaz, incluyendo el diseño y la configuración inicial de los componentes.
  - `actionPerformed(ActionEvent e)`: Método implementado de la interfaz *ActionListener* para manejar eventos de acción, como clics en los botones.

III-F3. PanelMenuAdmin: Clase que extiende de *JPanel* y configura el menú administrativo de la aplicación. Diseñada para ofrecer un conjunto de botones y opciones que permiten a los administradores gestionar diferentes partes de la galería.

Atributos y Métodos

- **Atributos**
  - `botones`: Colección de *JButton* que facilitan la navegación y la ejecución de funciones administrativas.
  - `layout`: *GridBagLayout* utilizado para organizar los componentes dentro del panel.
  - `bordes`: Personalizaciones de *Border* para mejorar la estética de los componentes.
- **Métodos**
  - `configurarBotones()`: Método que inicializa y configura los botones con sus respectivos manejadores de eventos.
  - `dibujarComponentes()`: Método que personaliza la visualización de los componentes del panel.

III-F4. RoundedBorder: Clase que implementa la interfaz *Border* para crear un borde redondeado personalizable para componentes de la interfaz gráfica. Se utiliza para mejorar la estética visual de los componentes de la aplicación.

Atributos y Métodos

- **Atributos**
  - `radius`: Entero que define el radio de la curvatura del borde.
  - `color`: Objeto *Color* que define el color del borde.
  - `thickness`: Entero que define el grosor del borde.
- **Métodos**
  - `RoundedBorder(int radius, Color color, int thickness)`: Constructor que inicializa un nuevo borde redondeado con los parámetros especificados.
  - `paintBorder(Component c, Graphics g, int x, int y, int width, int height)`: Método para dibujar el borde alrededor del componente dado, utilizando las dimensiones y la posición especificadas.
  - `getBorderInsets(Component c)`: Devuelve un objeto *Insets* que indica el espacio que el borde ocupa alrededor del componente.
  - `isBorderOpaque()`: Método que indica si el borde es opaco o no, lo cual afecta cómo se debe pintar el fondo del componente.

III-F5. VentanaRegistrar: Clase que extiende de *JFrame* y se utiliza para la interfaz gráfica donde los usuarios pueden registrarse en la aplicación. Esta ventana podría incluir campos de entrada para datos personales y detalles de cuenta, así como botones para enviar la información o cancelar el registro.

Atributos y Métodos

- **Atributos**
  - `camposDeTexto`: Objetos *JTextField* para la entrada del nombre, correo electrónico, contraseña, etc.
  - `botonRegistrar`: *JButton* que el usuario presiona para completar el registro.
  - `botonCancelar`: *JButton* para cerrar la ventana sin guardar los cambios.
- **Métodos**
  - `configurarGUI()`: Método para configurar los componentes gráficos de la ventana.
  - `accionRegistrar()`: Método que maneja el evento del botón registrar, incluyendo la validación de los datos ingresados y su almacenamiento.
  - `accionCancelar()`: Método para cerrar la ventana cuando el usuario presiona el botón cancelar.

IV. Diagramas de Clases de la Aplicación

A continuación se proporcionan los diagramas de clases de alto nivel que permite entender las relaciones entre las clases descritas anteriormente

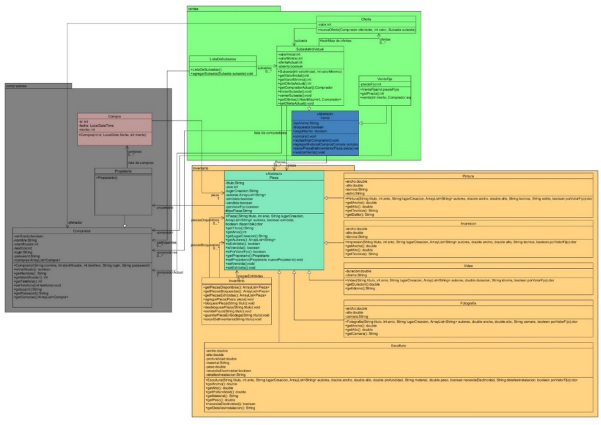

Figura 6: Diagrama de clases

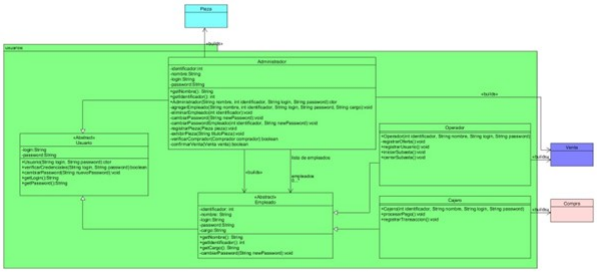

Figura 7: Diagrama de clases

Finalmente, para el apartado de la interfaz planteada, teniendo en cuenta como esta se integraria con la lógica de dominio, tenemos el siguiente diagrama de clases

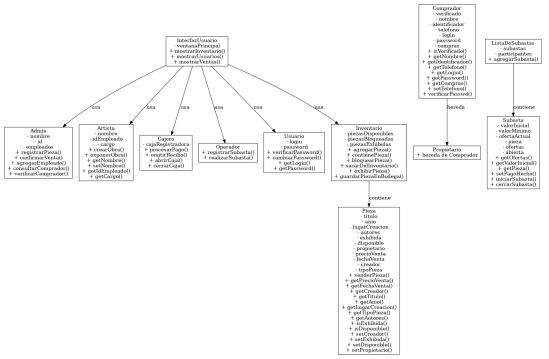

Figura 8: Diagrama de clases para la interfaz

IV-A. Diagrama de clases de alto nivel del sistema

Figura 9: Diagrama de clases de alto nivel del sistema

Relaciones entre clases de Usuarios:

- **Admin**:
  - Hereda de *Usuario*
- **Artista**:
  - Hereda de *Usuario*
- **Cajero**:
  - Hereda de *Empleado*
- **Operador**:
  - Hereda de *Empleado*
- **Empleado**:
  - Hereda de *Usuario*

Estas relaciones indican que *Admin* y *Artista* son tipos específicos de *Usuario*, mientras que *Cajero* y *Operador* son tipos específicos de *Empleado*, que a su vez hereda de *Usuario*.

Relaciones entre clases de Inventario:

- **Pieza**:
  - Hereda de *Escultura*, *Fotografia*, *Pintura*, y *Video*
- **Inventario**:
  - Contiene *Pieza*

Estas relaciones indican que *Pieza* es una clase base para diferentes tipos específicos de piezas (*Escultura*, *Fotografia*, *Pintura*, y *Video*). *Inventario* es una clase que contiene objetos de tipo *Pieza*.

Relaciones entre clases de Compradores:

- **Comprador**:
  - Hereda de *Propietario*
- **Propietario**:
  - Crea *Compra*

Estas relaciones indican que *Propietario* es un tipo específico de *Comprador*, y que *Propietario* tiene una relación de creación con *Compra*, es decir, los propietarios realizan compras.

Relaciones entre clases de Ventas:

- **ListaDeSubastas**:
  - Contiene *Subasta*
- **Subasta**:
  - Contiene *Oferta*
- **Venta**:
  - Hereda de *VentaFija*
  - Crea *Compra*

IV-B. Diagrama de clases de alto nivel de la interfaz

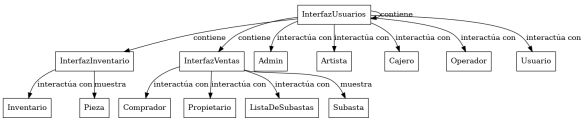

Figura 10: Diagrama de clases de alto nivel para la interfaz

El diagrama de clases mostrado en 10 contiene los siguientes elementos:

InterfazUsuarios:

- **Contiene**:
  - *InterfazInventario*
  - *InterfazVentas*
- **Interactúa con**:
  - *Admin*
  - *Artista*
  - *Cajero*
  - *Operador*
  - *Usuario*

Donde la **InterfazUsuarios** es la clase principal de la interfaz de usuario, que gestiona y organiza todas las sub-interfazes. Contiene las interfaces específicas para Inventario y Ventas, y se comunica con las diferentes clases de usuarios (*Admin*, *Artista*, *Cajero*, *Operador* y *Usuario*).

***InterfazInventario**:*

- **Interactúa con**:
  - *Inventario*: La **InterfazInventario** se encarga de mostrar y gestionar la información del inventario.
- **Muestra**:
  - *Pieza*: La **InterfazInventario** también gestiona la visualización y manipulación de los objetos *Pieza*.

***InterfazVentas**:*

- **Interactúa con**:
  - *Comprador*: La **InterfazVentas** interactúa con la clase *Comprador* para gestionar la información de los compradores.
  - *Propietario*: También interactúa con la clase *Propietario* para manejar la información de los propietarios.
  - *ListaDeSubastas*: La **InterfazVentas** se comunica con la *ListaDeSubastas* para gestionar y visualizar las subastas.
- **Muestra**:
  - *Subasta*: La **InterfazVentas** es responsable de mostrar y gestionar las subastas individuales.

***Relaciones con Clases de Usuarios**:*

- *Admin*: La **InterfazUsuarios** interactúa con *Admin* para gestionar las tareas administrativas.
- *Artista*: La **InterfazUsuarios** interactúa con *Artista* para gestionar las acciones relacionadas con la creación y exhibición de obras.
- *Cajero*: La **InterfazUsuarios** se comunica con *Cajero* para manejar las transacciones y pagos.
- *Operador*: La **InterfazUsuarios** se relaciona con *Operador* para gestionar las subastas.
- *Usuario*: La **InterfazUsuarios** interactúa con *Usuario* para manejar las credenciales y autenticación de los usuarios.