Andrés Felipe Romero - 202312399

Juan José Penha - 202312307

**Documentación del programa (proyecto 3)**

**Contexto:**

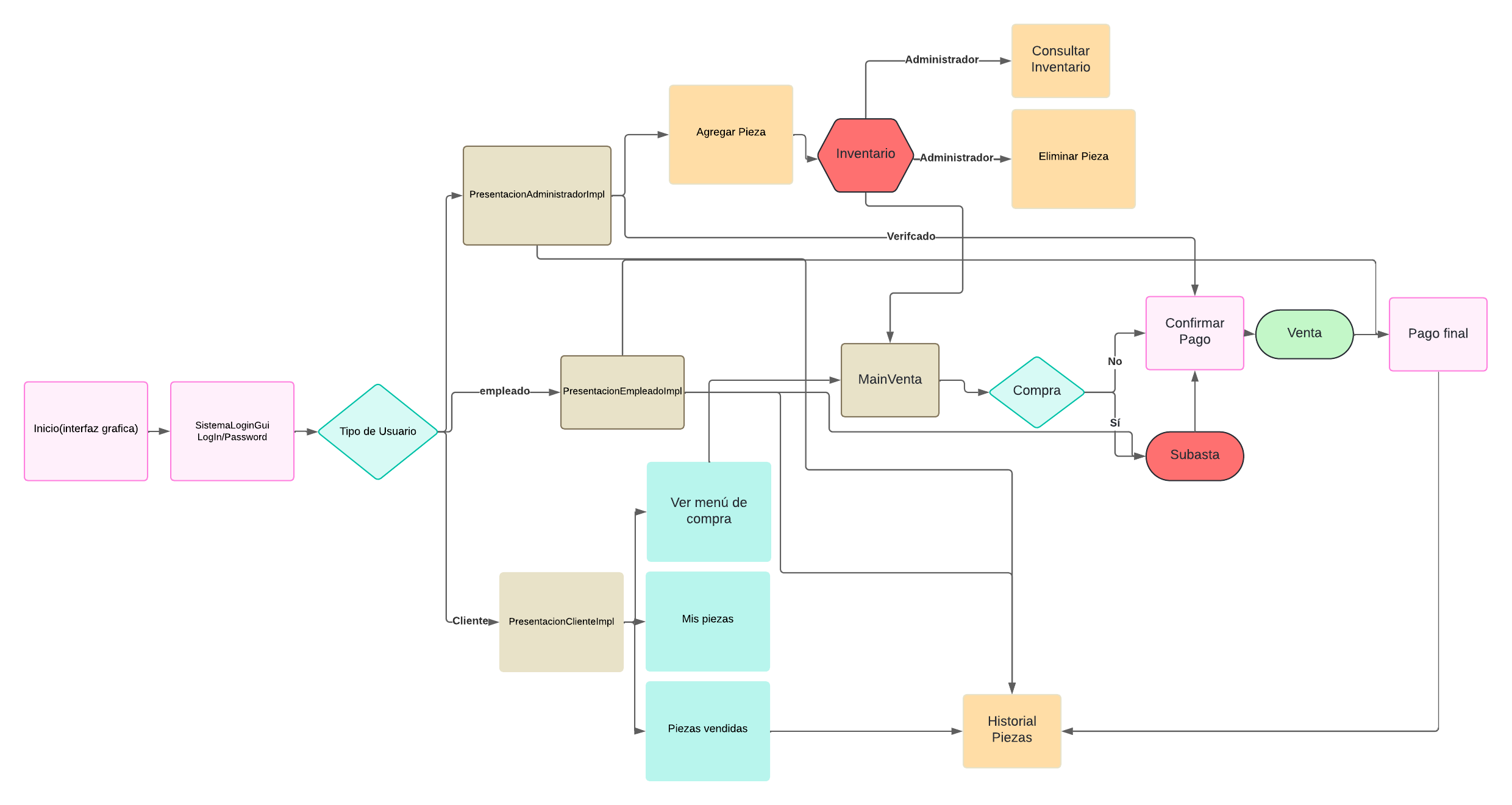
En el desarrollo del tercer proyecto, nos enfrentamos a la tarea de implementar interfaces gráficas para que el modelo de subasta tenga un contenido mucho más comprensible para el usuario. Después de un análisis exhaustivo, llegamos a la conclusión de que teníamos que generar un paquete de presentaciones donde se implementarían todas las interfaces gráficas, entre ellas están “SistemaLoginGui”, “PresentacionEmpleadoImpl”, “PresentacionClienteImpl” y “PresentacionAdministradorImpl” estas clases funcionan de la misma manera que las clases “Main…” del anterior proyecto, lo que realmente se cambió es que a estas nuevas clases se les implemento una parte de interfaces graficas. Esto se ilustra en el diagrama de secuencia (Figura 1) y el diagrama de clases (Figura 2). (se recomiendo hacer zoom)

Figura 1

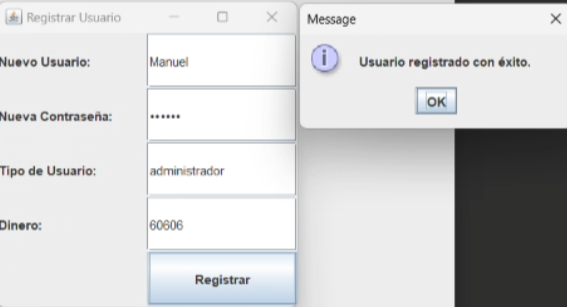
**Visualización Interfaces gráficas:**

Interfaz de usuario gráfica, Texto, Aplicación

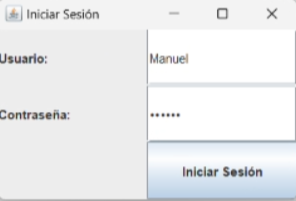
Descripción generada automáticamenteAl principio, nos centramos en implementar métodos de autenticación a través de Login/Contraseña (inicio de sesión). Durante la creación de un nuevo usuario, se le asigna un rol específico que determina su acceso a diferentes menús dentro del sistema. Por ejemplo, un administrador tiene acceso únicamente al "PresentacionAdministradorImpl". En este proyecto implementamos la interfaz gráfica del inicio de sesión que se ve de este modo:

En la figura 0.1 podemos ver como se ve la ventana la cual se titula “Sistema de Login” y tiene dimensiones de 600x400, en esta ventana decidimos implementar el BorderLayOut con el fin de tener mas orden en la ventana, en la parte norte, este y oeste no hay información puesto que solo requerimos de una parte central e inferior, dentro de la parte central encontramos una imagen que hace alusión a las subastas y en la parte sur de la ventana encontramos 3 botones organizados en GridLayout, los botones son IniciarSesion, CerrarSesion y Salir. Cada uno tiene un ActionListener y cumple con su respectiva función.

(Figura 0.1)

En la figura 0.2 podemos evidenciar lo que sucede cuando se le da click al botón de “Registrar Usuario” en donde se surge una ventana emergente la cual hace preguntas necesarias para poder guardar a una persona en la base de datos (el archivo txt). Al final, después de agregar toda la información, se le da al botón de registrar y sale una ventana emergente con la confirmación del registro “Usuario registrado con éxito”.

(Figura 0.2)

La figura 0.3 es como se ve el inicio de sesión una vez ya registrados, en este caso ponemos mismo usuario y contraseña que en el registro y como ya el sistema sabe que “Manuel” es administrador, el programa nos mandará a el menú de administrador, como se puede ver en la figura 0.4, Dentro de este menú podemos evidenciar diferentes opciones que pueden hacer los administradores, desde la figura 0.5 hasta la figura 0.15 se ven los métodos y ejemplos de su funcionalidad

(Figura 0.3)

METODOS DESPUES DE INICIAR SESIÓN EN ADMINISTRADOR: (ejemplo de administrador)

Menu Administrador: Agregar Pieza a Inventario: Consultar Inventario:

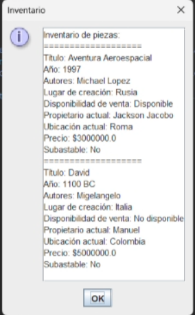
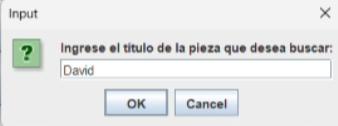
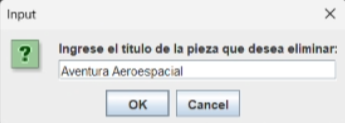


FOTO DE AGREGAR PIEZA

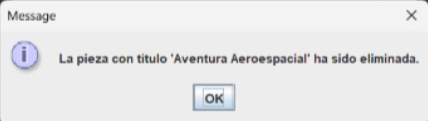
Figura 0.6

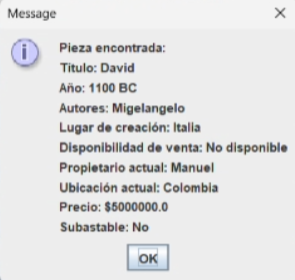
(Figura 0.4)

Eliminar Pieza de Inventario: Buscar Pieza por título: Ver historial de clientes:



(Figura 0.7) (Figura 0.8) (Figura 0.9)

 **Resultado del método**



**Resultado del método**

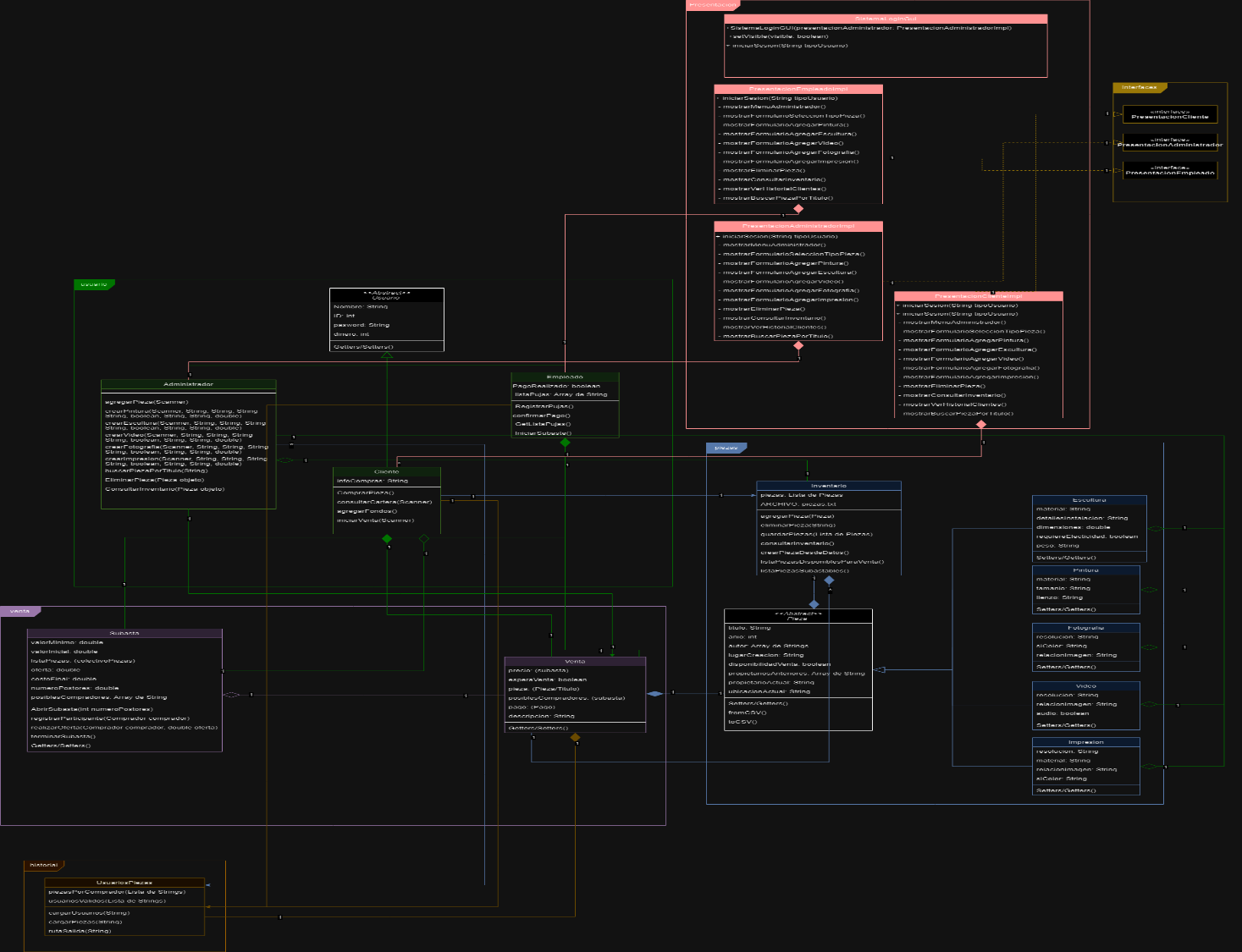
Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

Los distintos “main” (que actalmente se cambiaron a los Impl) que ofrecemos son los siguientes: "PresentacionAdministradorImpl", " PresentacionClienteImpl "y “PresentacionEmpleadoImpl”. Y estas son sus interfaces gráficas, donde se muestran sus menús y opciones. (Vea la parte de arriba, desde la Figura 0.4 hasta la 0.9).

Dentro del " PresentacionAdministradorImpl ", se pueden llevar a cabo acciones como agregar nuevas piezas, consultar el inventario, eliminar piezas. Sin embargo, es importante destacar que las opciones de consulta y eliminación solo están disponibles después de que se hayan agregado piezas al inventario, ya que sin piezas no hay datos que consultar o eliminar.

En el " PresentacionClienteImpl ", se ofrecen tres funcionalidades principales: "Ver el menú de Compra", "PiezasVendidas" y "MisPiezas". La opción "Ver el menú de Compra" es donde se verifica la disponibilidad de la pieza para su venta o subasta. En caso de estar disponible para la venta directa, se procede a una confirmación de pago por parte de un administrador, asegurándose así de que el usuario tenga los fondos suficientes para adquirirla. Si la pieza no está disponible, se inicia un proceso de subasta donde el precio puede variar y es administrado por un empleado en el " PresentacionEmpleadoImpl ", quien gestiona las

ofertas y el valor de la subasta. Una vez confirmado el pago, se lleva a cabo la venta final y por último se efectúa directamente el pago final.

(Figura 2)

El diagrama de clases UML se compone de 5 paquetes que contienen varias clases, todas interrelacionadas entre sí mediante diversas paridades y tipos de relaciones, como composición, herencia, asociación y agregación.

El primer paquete ya no es "consola", sino “presentación”, este consta de 4 clases llamadas “SistemaLoginGui”,“PresentacionEmpleadoImpl”,“PresentacionClienteImpl”y“PresentacionAdministradorImpl”. Esta clase tiene los atributos mostrados en la figura 2.1. Como se puede observar, las clases se relaciona directamente con todo el paquete de "usuario", ya que una vez que se completa el atributo "IniciarSesion", dirige al usuario al "Impl" correspondiente a su rol previamente seleccionado. La mayor diferencia respecto al proyecto anterior es que logramos unificar el paquete de consola y menú, pues con la interfaz grafica bien implementada solo necesitábamos de un paquete de presentación que contuviera todas las clases mencionadas previamente, haciendo que el programa se “ahorrara” varios pasos para hacer sus responsabilidades.

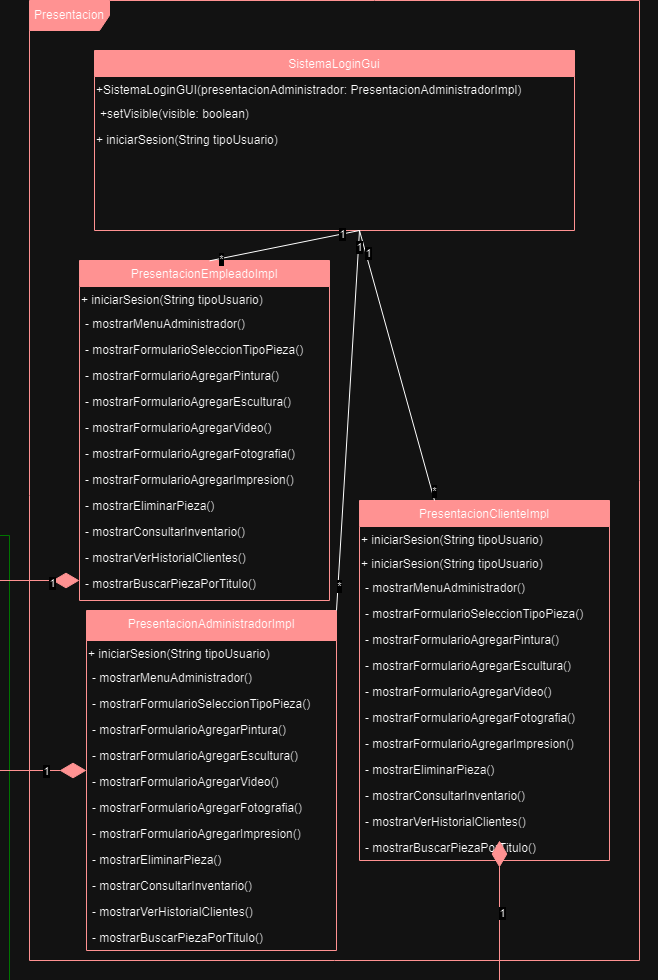


Figura 2.1

Una vez que los usuarios ingresan a cualquiera de los "IMPL", el sistema se dirige al paquete "usuario", el cual contiene todos los métodos que se detallan en la figura 2.2. La relación entre las clases del paquete "presentacion" y el paquete "usuario" es de composición, ya que la única forma de acceder al paquete "usuario" es a través de las clases del paquete "presentacion", y si se eliminara el paquete "usuario", el paquete "presentacion" perdería su utilidad. En cuanto a las paridades, son de 1 a 1, puesto que cada vez que se llama a un método específico, se accede una vez al "Main" del usuario correspondiente al método.

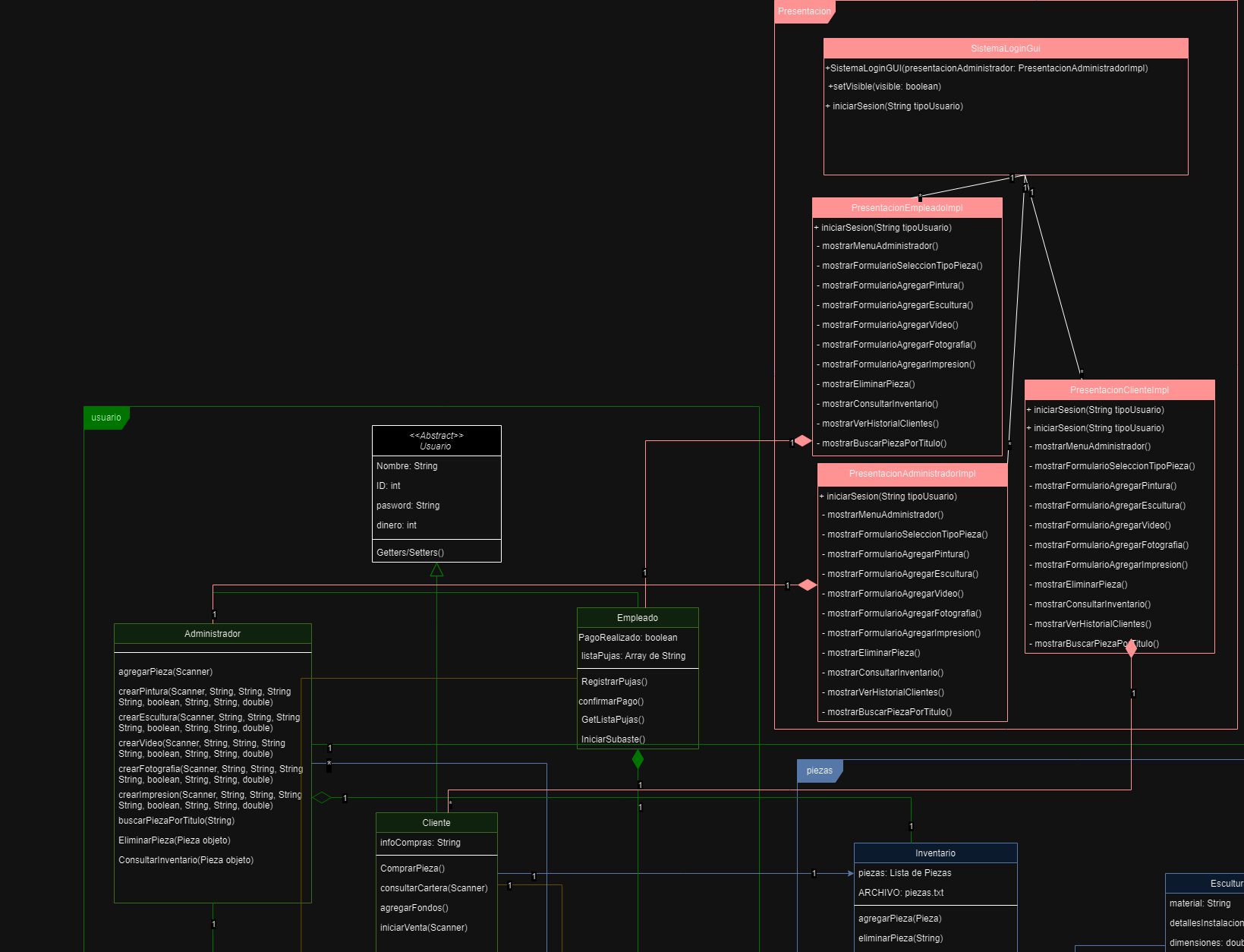


Figura 2.2

Ahora, una vez el sistema quiera ejecutar correctamente los métodos en el paquete “usuario” este paquete necesitará traer información de diferentes paquetes, es por esto por lo que surgen las relaciones entre el paquete “usuarios” a “piezas” y “venta”. Estas relaciones se establecen debido a que el sistema, tiene que tomar los datos de las piezas para poder modificarlas en el paquete de “usuarios”, y el resultado de la modificación se guarda en la clase inventario para luego poder subastarla o venderla para posteriormente hacer el pago final.

**Relaciones:**

Administrador: tiene varias relaciones respecto a todas las piezas (Escultura, Pintura, Fotografía, Video, Impresión), estas relaciones son de agregación y su paridad es de 1 a 1. Además, administrador también se relaciona con “inventario” (del paquete piezas), esta relación es de agregación, ya que inventario se compone de lo que administrador le mande, pero si administrador es destruido, inventario no tendría por qué quedar inútil o destruido. Finalmente, administrador se conectará con venta, eso gracias a que este usuario debe confirmar el pago para que la persona pueda realizar su transacción, la relación entre la clase “Administrador” y la clase “Venta” es de asociación y su paridad es 1 a muchos pues un solo administrador puede realizar varias ventas.

Empleado: Esta clase se relaciona con “subasta”, “venta” y “UsuariosPiezas”, las primeras dos mencionadas se relacionan por composición y su paridad es 1 a 1, mientras que en “UsuariosPiezas”, proveniente del paquete “historial” la relación es de agregación, pues el empleado tiene como función adicionar atributos de ventas en esta clase, por lo que su paridad es de 1 a 1. Además, tiene relación con venta, pues el empleado se encarga de agregar todos los datos necesarios para realizar la venta, por ende la relación de empleado con venta es de composición, ya que sin un empleado no hay manera de llevar a cabo una venta.

Cliente: La clase “Cliente” se relaciona con el inventario, su relación es de asociación, pues necesita ver todas las piezas disponibles para poder comprar, es por esto que la paridad es de 1 a 1, ya que un solo cliente puede ver únicamente un inventario (ya que solamente hay un inventario en todo el sistema), además el cliente se relaciona 1 a 1 con “Venta”(del paquete “venta”), pues solo un cliente puede realizar un pago, por lo que hace que esta relación sea de composición pues sin un cliente no hay venta, finalmente cliente se relaciona con subasta, pues necesita estar en la subasta si el usuario lo desea, es por esto que al igual que en “venta” la subasta también tiene una relación de composición y su paridad es de 1 a 1.

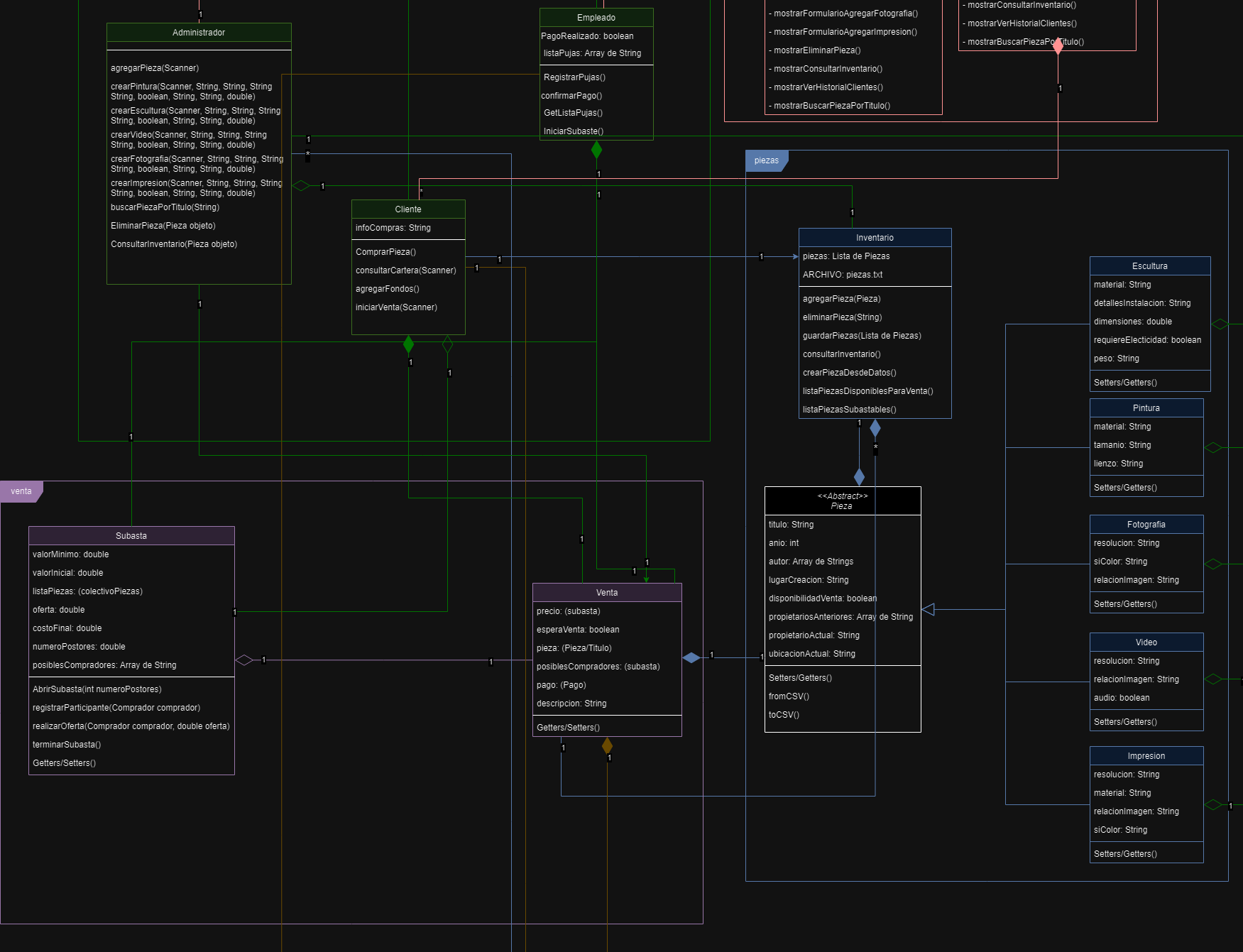


Figura 2.3

Finalmente se encuentra el último paquete “historial” que tiene como única clase “UsuariosPieza” y es la clase encargada de reunir información entre el nombre del usuario y los títulos de las piezas que sean de su posesión. Esta clase se relaciona con "cliente” y “venta”. La relación entre todas las relaciones que tiene son de agregación pues la clase de UsuariosPiezas es realmente un “historial” donde se guarda todos los datos para poder obtener un historial de cada cliente que compre o venda, finalmente la paridad entre todas estas relaciones es de mucho a 1, donde ese 1 es UsuariosPiezas, pues pueden haber varios empleados y ventas.

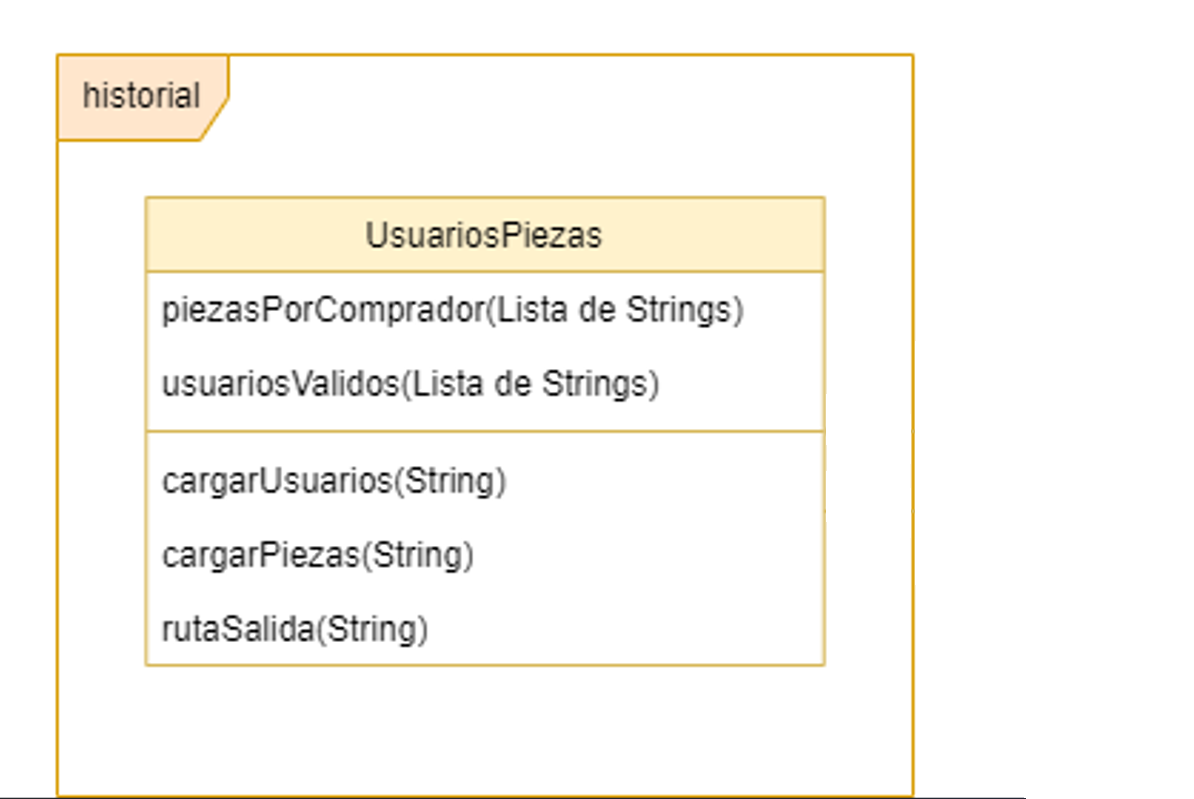


Figura 2.4

**Pruebas:**

Para las pruebas se creó un paquete titulado “pruebas” el cual se compone de 2 clases las cuales son “TestAdministrador” y “TestCliente”.

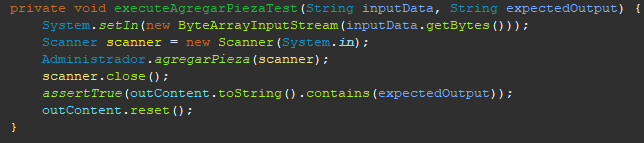
TestAdministrador: Dentro de este test se ejecuta un método y muchos test, el método es de vital importancia pues nos permite ejecutar los test de una manera más simple y rápida. El método es el siguiente (Figura 3).

Figura 3

Esta función tiene como parámetros “inputData” el cual recibe el string data que está en la Figura 3.1, y “expectedOutput” que recibe de parámetros el resultado de lo que debería dar la función (se encuentra en la última línea de la figura 3.1), en la segunda línea, encontramos System.setIn el cual se encarga de guardar la información mostrada en consola, para poder compararla con el resultado esperado, en Administrador.agregarPieza se agregan los datos que están en la figura 3.1 para al final comprobar que el resultado sea el “expectedOutput”(Esto se logra con el assertTrue…).Y al final del todo “outContent.rest();” se encarga de resetear el valor de todo para que se puedan ejecutar todas las pruebas

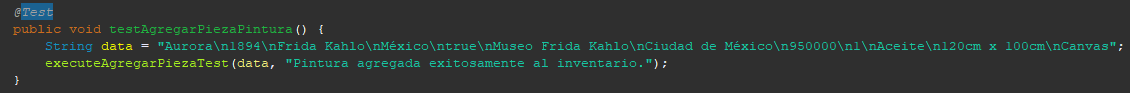


Figura 3.1

Una vez aclarado el método a usar para las pruebas de administrador, se pueden determinar las siguientes pruebas:

1. Test para agregar adecuadamente una pieza de categoría Pintura.
2. Test para agregar adecuadamente una pieza de categoría Escultura.
3. Test para agregar adecuadamente una pieza de categoría Video.
4. Test para agregar adecuadamente una pieza de categoría Fotografía.
5. Test para agregar adecuadamente una pieza de categoría Impresión.
6. Test para eliminar adecuadamente una pieza.
7. Test para consultar que el inventario esté vacío.
8. Test para consultar que el inventario esté con piezas.
9. Test para buscar la pieza por título existente.
10. Test para buscar la pieza por título no existente.

Por otra parte, está “TestCliente”, en esta clase de pruebas se encuentran la siguiente serie de pruebas:

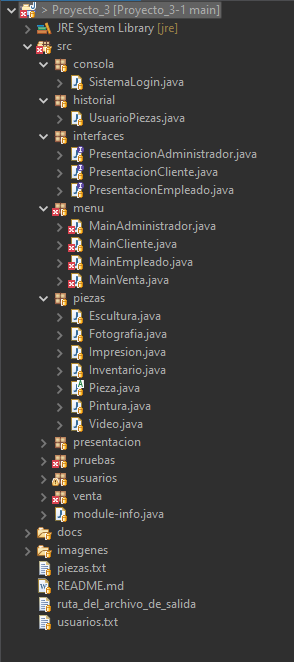
1. Test para confirmar los atributos de cliente
2. Test para confirmar el historial de usuarios (se mira la clase UsuarioPiezas)

Finalmente está “TesthistorialUsuarios”, en esta clase se encuentran la siguiente serie de pruebas:

1. Test para cargar Usuarios Válidos
2. Test cargar Piezas de manera adecuada
3. test cargar archivo

**Estructura:**

El programa actualizado está compuesto de varios IMPL los cuales nos dirigen a diferentes partes dependiendo del usuario que sea la persona; las clases de usuarios son las que contienen todos los métodos necesarios para el correcto funcionamiento del código.



Esta es la organización principal del programa, en donde se tiene el paquete “consola" este es el encargado de contener el “SistemaLogin” el cual tiene como métodos: “iniciarSesion”, “registrarUsuario” y se encarga de crear el primer archivo de persistencia llamado “usuarios.txt” (Este archivo se halla en la parte inferior de la imagen). Después del paquete “consola” se encuentra el paquete de “historial” el cual se encarga de generar el segundo archivo persistente del programa, que se llama “comprador.txt”, este archivo de texto plano se encarga de almacenar la información del nombre de los usuarios y las piezas que cada uno posee, esto con el fin de hacer posible el método de historial, para que el administrador pueda saber el historial de compras y ventas de los propietarios y compradores.

Para este proyecto decidimos crear dos nuevos paquetes los cuales tendrían como función implementar las interfaces gráficas, estos dos paquetes se llaman “interfaces” y “presentacion”. “Interfaces” contiene la conexión entre la presentacion y los usuarios, y el paquete “presentacion”, se encarga de cargar toda la información visual para que el programa sea más “amigable” con el usuario (ver las figuras 0.4-0.9). Siguiendo con el paquete “menu” nos encontramos con todos los main de todos los usuarios; los cuales se encargan de cargar las funcionalidades de cada tipo de usuario. Después nos encontramos con piezas, las cuales todas están en el mismo paquete por lo que usamos herencia, ya que la mayoría de los atributos se comparten entre sí, atributos como “Titulo”, “Año”, “Autores”, etc. Después de “piezas” está el paquete de “pruebas” el cual se encarga de realizar todas las pruebas necesarias para tener una cobertura cercana a 60%, este porcentaje fue tomado después de un debate grupal, debido a que no podemos emplear tanto tiempo en el código como para llegar al 100% de cobertura, esto se debe a limitaciones de tiempo para hacer la entrega del proyecto; Después, sigue el paquete de “usuarios” el cual funciona de la misma manera que el paquete de piezas, pues también hay herencia, ya que todos comparten ciertos atributos; las clases que más resaltan aquí son las clases de administrador y cliente, pues son las que más interacciones tienen con otras clases (esto se explicará más adelante) además de que la clase “Administrador” crea el tercer archivo txt, el cual se titula “piezas.txt” y se encarga de almacenar toda la información de una pieza, pues se dice en las instrucciones que administrador es el usuario que registra y elimina piezas del inventario. Finalmente se encuentran los paquetes de “venta” y “docs”; “venta” se encarga de gestionar todo lo relacionado con las subastas y ventas (más adelante se hablará más a detalle) y finalmente “docs” es donde se guardan todos los archivos txt necesarios, decidimos dejarlos afuera del todo para que sea de fácil uso, pues es más fácil acceder a ellos desde cualquier parte.

**Restricciones:**

1. No se puede tener dos piezas con el mismo nombre
2. Una pieza no puede estar expuesta y en bodega al mismo tiempo.
3. La pieza se bloquea cuando se va a realizar una compra.
4. No pueden existir dos usuarios con el mismo nombre.
5. Un usuario solo puede tener un rol dentro del programa.
6. Solo el administrador tiene acceso a las piezas de la galería.
7. El administrador es el único que puede autorizar una venta.
8. El empleado es el único que registra lo que pasa en una subasta.
9. El comprador no puede cambiar el estado de compra de una pieza.
10. No se pueden subastar más de una pieza a la vez.
11. Hasta que no se realice el pago final de una pieza no se puede empezar la compra de la siguiente pieza.
12. El comprador de una pieza se elige hasta que se acabe la subasta.
13. En caso de dos pujas iguales la que primero se realice gana.
14. Para completar la subasta la puja tiene que superar el valor mínimo.
15. Los empleados no pueden participar en la compra o venta de una pieza.

**Entidades:**

**Usuario (Abstract Class):**

● Atributos:

○ nombre (String): Nombre del usuario

○ dinero (Date): Fecha de nacimiento del usuario

○ contrasenia (String): Contrasenia con la que el usuario se identifica

● Métodos:

○ Getters

○ Setters

**Empleado:**

● Atributos:

○ listaPujas (Array de double): Lista donde se almacenan los datos de la puja en una subasta hasta que se confirme la venta.

○ subastaActual(Subasta): es una subasta.

○pieza(Pieza): es una pieza de inventario que se va a subastar.

○ Pago realizado (Boolean): Estado que verifica que el pago se haya realizado correctamente.

○ Participantes Puja(Lista de Strings): Donde se guardan los nombres descendientes de los participantes de la puja.(mayor a menor)

● Métodos:

○ CambiarComprador: en caso de que no pueda pagarlo el primero, este método buscará el segundo en la puja

○inciarSubasta

○ registrarPujas:Crea un array de tuplas que captura la segunda posición (La cual contiene el dinero que se está pujando) y organiza los números de mayor a menor.

○ Constructor

**Cliente:**

● Atributos:

○ infoCompras (Array de String): Lista con los nombres de las piezas que ha comprado el usuario.

● Métodos:

○ Getters

○ Setters

○ comprarPieza: Redirigir hacia mainVenta

○ consultarDinero: Acceder a Usuario.dinero

○ agregarFondos: Sumar en el atributo dinero de usuario

○consultarHistorialCompras:Accede al txt de Historial

○vender: Da la pieza a otro propietario a través de inventario ☼

○piezasVendidas:Acceder a la lista de nombres de piezas vendidas

○ Constructores

**Administrador:**

● Atributos:

no hay ☻

● Métodos:

○ ConfirmarPago : Determina si el comprador tiene el dinero suficiente para comprar la pieza

○ agregarPieza: llamar a crear pintura, escultura, video etc.

○crearPintura: Crea una clase pintura y la agrega a inventario

○crearEscultura: Crea una clase escultura y la agrega a inventario

○crearVideo: Crea una clase video y la agrega a inventario

○crearFotografia: Crea una clase fotografia y la agrega a inventario

○crearImpresion: Crea una clase impresión y la agrega a inventario

○buscarPiezaPorTitulo: Accede a “Piezas.txt” y busca por titulo solo una pieza

○ ConsultarInventario: Accede a todo el inventario y lo muestra en consola

○ eliminarPieza: Se busca por el título y se elimina dicha pieza

**Pieza (Abstract Class):**

● Atributos:

○ título (String): Titulo de la pieza

○ anio (String): Anio de creación de una pieza

○ autores (Array de string): Lista con los autores de una pieza

○ lugarCreación (String); Lugar de creación de la pieza

○ DisponivilidadVenta (Boolean): Parámetro que muestra si la pieza se encuentra en venta o no

○ PropietariosAnteriores (Array de string): Lista con todos los autores que ha tenido una pieza desde su creación

○ PropietarioActual (String): Nombre del propietario actual

○ ubicaciónActual (String): Muestra si la pieza se encuentra en bodega o en exposición.

○subastable(boolean): Si es “True”: se puede subastar, en caso de “False”, la pieza se irá a venta

○precio(double):Precio de la pieza

● Métodos:

○ HacerDisponibleParaVenta:Cambiar el caso “False” del atributo subastable a “True”

○ ActualizarPrecio:Cambia el valor del atributo precio

○ Getters

○Setters

**Pintura:**

● Atributos:

○ material (String):

○ tamanio (String):

○ lienzo (String):

● Métodos:

○ Getters

○ Constructor

○Setters

**Video:**

● Atributos:

○ resolución (String):

○ audio (boolean):

○ relacionImagen (String):

○tieneColor(boolean):

● Métodos:

○ Getters

○ Constructor

○Setters

**Escultura**:

● Atributos:

○ material (String):

○ detallesInstalacion (String):

○dimensiones (String):

○ requiereElecticidad (boolean):

○ peso (String):

● Métodos:

○ Getters

○ Constructor

○Setters

**Fotografía:**

● Atributos:

○ resolucion (String):

○ tieneColor (String):

○ relacionImagen (String):

○esDigital(boolean):

● Métodos:

○ Getters

○ Constructor

○Setters

**Impresión:**

● Atributos:

○ resolucion (String):

○ material (String):

○ relacionImagen (String):

○ tieneColor (String):

● Métodos:

○ Getters

○ Constructor

○Setters

**Inventario:**

● Atributos:

(todos se llaman en otras partes del sistema)

● Métodos:

○agregarPieza: Adiciona una pieza en el archivo “Piezas.txt”

eliminar

○eliminarPieza: Se busca por el título y se elimina dicha pieza

○guardarPiezas: Es el formato adecuado para almacenar en el archivo “Piezas.txt”

○ consultarInventario: Accede a todo el inventario y lo muestra en consola

○crearPiezaDesdeDatos: Crea una pieza para consultar inventario (donde se ingresa la información a una lista)

○listaPiezasDisponibleParaventa:devuelve una lista que tienen todas las piezas con valor “True” en el atributo “disponibilidadVenta”

○listaPiezasSubastables:devuelve una lista que tienen todas las piezas con valor “True” en el atributo “Subastable”

**Venta:**

● Atributos:

○ precio (Int):

○ esperaVenta(Boolean):

○ piezaTitulo (String):

○ posiblesCompradores(Array de Strings):

○ pago(Pago):

● Métodos:

○ Getters

○Setters

**Subasta:**

● Atributos:

○ valorMinimo (double):

○ valor Inicial (double):

○ ofertas (double):

○ pieza(Pieza):

○ costoFinal (double):

○ posiblesCompradores (Array de Strings):

○ numeroPostores (double):

○ofertaActual(double):

● Métodos:

○ Getters

○Setters

○ abrirSubasta: Da inicio a la subasta (comprueba antes que hayan piezas para subastar)

○ RegistrarParticipantes: Crear una lista con nombres de los participantes de la subasta

○realizarOferta: Se realiza una puja inicial mayor a “valorInicial”

○ TerminarSubasta: Da fin a la subasta (tiene que venderse mínimo una pieza para poder cerrarla)