# Documento de diseño (Proyecto 2)

Grupo 6: Diseño y programación orientada a objetos

## 1. Contexto del problema

A nivel de requerimientos no funcionales, la aplicación debe actuar de diferentes formas dependiendo del usuario que esté utilizándola. Sin embargo, se hace una primera aproximación de la composición mediante la figura 1.

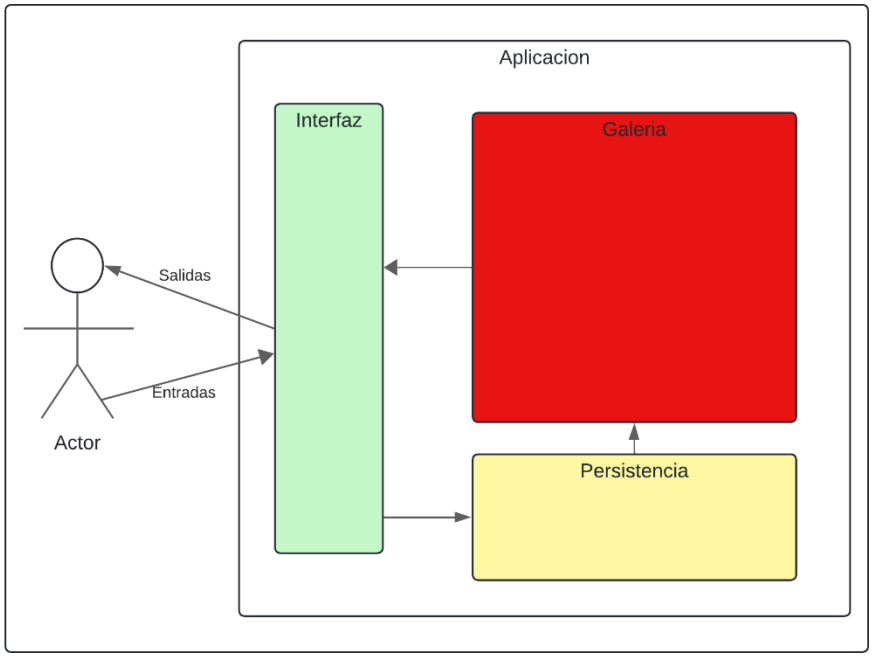



Figura 1: Aplicación para el proyecto 2. Se tiene una interfaz que es la que interactúa con el usuario. Esta es la que le indica a la persistencia para que cargue los datos o los guarde. Toda la lógica compleja debería estar en el componente galería.

Se quiere hacer hincapié en las solicitudes que el usuario puede hacer al sistema y las solicitudes que el sistema le puede hacer al usuario. Para ello, se muestran los dos siguientes cuadros:

Cuadro I: Solicitudes que los distintos usuarios pueden realizarle al sistema de acuerdo con los requerimientos funcionales.

| Actor | Entradas |
|---|---|
| Administrador de la galería | registrarPieza(), confirmarVenta(), devolverPieza(), verificarUsuario(), reestablecerMaximo(). |

| | |
|---|---|
| Cajero | registrarPago(). |
| Comprador | comprar(), ofertar(). |
| Operador | planearSubasta(), agregarOferta(). |
| Propietario | cederPieza(), consultarPiezas(), consultarHistorial(). |
| Usuario | autenticarse(). |

Cuadro II: Solicitudes que el sistema le puede realizar al usuario dependiendo de su rol.

| Actor | Salidas |
|---|---|
| Usuario | pedirInformacionContacto(), pedirUsuarioContraseña(), |

Para entender los pseudo-nombres que se usaron en el cuadro I es necesario leer el documento de análisis que precede a este documento de diseño. Por otro lado, se entiende como salida toda solicitud directa que se le hace al usuario, no se tiene en cuenta las responsabilidades que tienen los trabajadores.

A continuación, solo se mostrará el diseño de la galería. Para la documentación de la persistencia y la interfaz, diríjase a los apéndices A y B al final de este documento.

## 2. Nivel 1

### *2.1. Componentes candidatos Galería:*

I. Dentro de la galería es importante el componente del inventario, que va a tener la información de las obras y manejará esta información. Por ello se decide que dentro de la galería habrá un componente con el estereotipo *information holder* llamado Inventario.

*II.* Es necesario mantener la información de los usuarios relacionada con sus datos o lo que los compongan de acuerdo con el tipo. Entonces, se deduce que los usuarios deben ser de tipo *structurer*, para que así la información se mantenga relacionada de acuerdo con el usuario.

III. Cuando un usuario hace cambios en el sistema e interactúa con este, es necesario que los procedimientos realizados sigan las reglas del dominio y tengan una lógica. Por lo que se plantea un componente *controller* llamado controlador galería.

IV. En la anterior sección y el documento de análisis, es claro que los usuarios tienen muchas interacciones entre sí, pero que el sistema es un intermediario entre ellos. Reaccionar a las peticiones del usuario coordinándolas con otros elementos de la galería hace que sea necesario un componente *coordinator llamado* Coordinador Usuarios.

De esta forma, para la primera abstracción se definió que la galería se conforma de la siguiente manera:

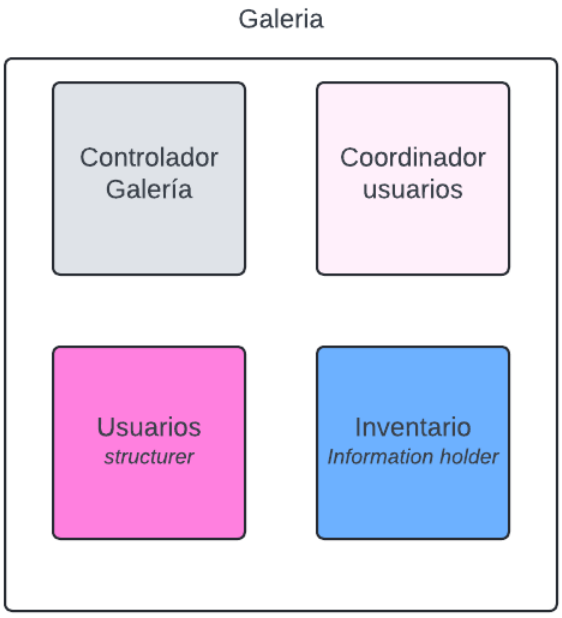



Figura 2: Resultado de la primera abstracción para el caso de la galería

### *2.2. Responsabilidades:*

La siguiente tabla muestra el detalle de las responsabilidades asociadas a cada componente:

Cuadro III: Responsabilidades clasificadas según el componente encargado.

| Responsabilidad | Componente |
|---|---|
| Decidir qué puede hacer un usuario según su rol dentro de la galería | Controlador Galería |
| Accionar las decisiones disponibles que haya elegido el usuario | |
| Crear un usuario | |
| Ejecutar las acciones que no requieran la interacción entre dos o más usuarios. | |

| | |
|---|---|
| Permitir el inicio de sesión | |
| Coordinar las peticiones del controlador con los usuarios o el inventario para concretar una acción que depende de muchos roles dentro de la galería. | Coordinador Usuarios |
| Estructurar organizadamente la información de cada usuario. | Usuarios |
| Relaciona al usuario varias contenencias relevantes de acuerdo con el documento de diseño | |
| Contiene la información de las obras y cada parámetro que se relaciona con ellas. | Inventario |

### *2.3. Colaboraciones:*

- **Crear y almacenar un usuario:** Implica el trabajo conjunto del controlador galería y el *structurer* usuarios.
    1. El usuario le pide al controlador crear un usuario.
    2. El controlador le pide la información básica de contacto al usuario
    3. El usuario se la envía al controlador galería
    4. El controlador se la envía al *structurer* Usuarios para que cree el usuario

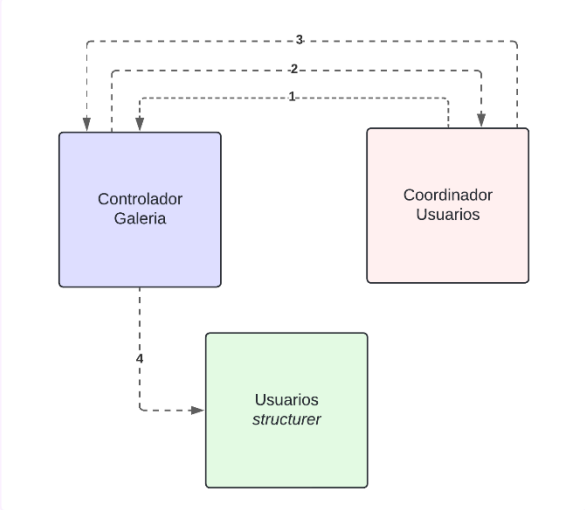



Figura 3: Representación gráfica de la colaboración que crea y almacena un usuario

- **Ofertar en una subasta:** Esta colaboración se lleva a cabo cuando la sesión activa es la de un comprador.
    1. El comprador decide participar en una subasta.
    2. El *controller* Galería busca en el inventario las obras que están disponibles para subastar.

3. El *controller* Galería muestra las obras que se están subastando y la última oferta que se ha realizado.
4. El comprador selecciona una obra y realiza una oferta.
5. El *controller* Galería verifica que la oferta cumpla con los requisitos para no ser rechazada y le comunica al *coordinator* Usuarios la oferta.
6. El coordinador Usuarios se comunica con el *structurer* Usuarios para dejar al operador la oferta.

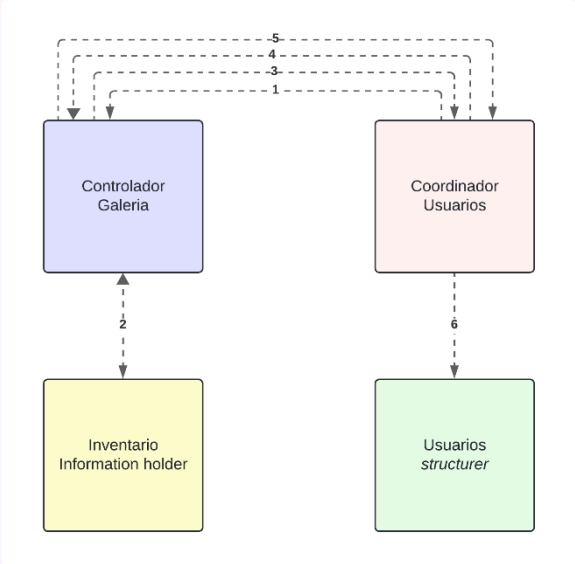



Figura 4: Representación gráfica de la colaboración para realizar una oferta en una subasta.

Realmente casi toda todas las interacciones que haga un usuario con el sistema van a implicar una colaboración entre el *controller* y otros componentes del sistema. Sin embargo, por el gran nivel de abstracción del diseño, las colaboraciones específicas no tienen relevancia.

Se puede observar un patrón constante en cualquier interacción que tenga un actor con el sistema. Principalmente ocurre la siguiente secuencia: El usuario le pide algo al *controller* Galería, luego el controller Galería corrobora si necesita del apoyo del *coordinator* Usuarios para que él termine la tarea o *controller* lo termina solo.

## 3. Nivel 2

### *3.1. Controlador galería*

Las funciones que asociamos a este controlador consisten en las siguientes:

- Decidir qué puede hacer un usuario según su rol dentro de la galería
- Accionar las decisiones disponibles que haya elegido el usuario
- Crear un usuario

- Ejecutar las acciones que no requieran la interacción entre dos o más usuarios.
- Permitir el inicio de sesión

### *3.1.1. Componentes candidatos y estereotipos*

Debido a la gran responsabilidad de tareas que se le delegó a este componente y siguiendo el principio: “Dividir y conquistar” se tomó la decisión de dividir el controlador en 2 componentes. Donde habrá un controlador para los usuarios que trabajan en la galería (Controlador Internos) y un controlador para aquellos que son clientes de la galería (Controlador Externos). Finalmente, se decidió asignar un coordinador que delegue cuál controlador se ejecutará dependiendo del usuario que este interactuando con la aplicación.

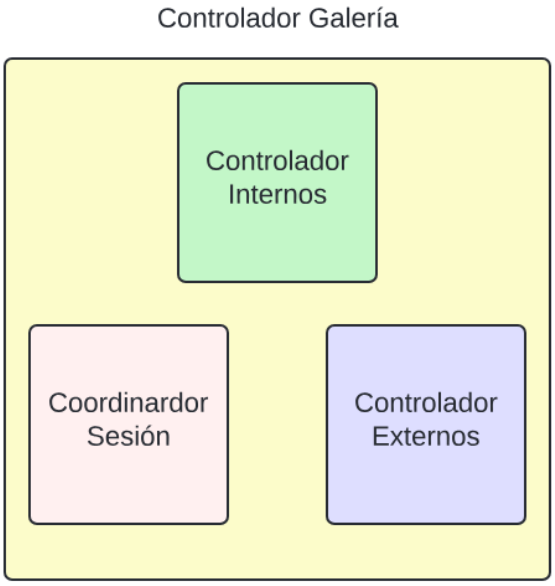



Figura 5: Descomposición en componentes del controlador Galería.

### *3.1.2. Responsabilidades*

Nuevamente se hace uso de una tabla para hacer más clara las responsabilidades que ocupan a cada descomposición:

| Responsabilidad: | Componente: |
|---|---|
| Accionar las operaciones que desee el funcionario de acuerdo con las reglas del dominio. | Controlador Internos |
| A partir del inicio de sesión, crear las estructuras correspondientes al usuario elegido. | Controlador Externos |
| Realizar las acciones solicitadas por el usuario de acuerdo con su rol específico. | |

| | |
|---|---|
| Corresponder el respectivo usuario con el cliente de forma adecuada. | Coordinador Sesión |
| Hacer correctamente el inicio de sesión para el correspondiente usuario de un empleado. | |
| Delegar las tareas de la sesión al respectivo controlador. | |
| Crear un nuevo usuario externo. | |

A partir de estas responsabilidades, es importante destacar que la diferenciación entre un propietario y un comprador estará en las estructuras que contenga su usuario. De esta forma la aplicación no diferencia entre un comprador y un propietario, sino que ambos se identifican como usuarios externos.

### *3.1.3. Colaboraciones*

Las colaboraciones en este segmento de la aplicación son fundamentales y muy recurrentes en el inicio de sesión de un usuario.  De esta forma se tiene que:

- **Iniciar sesión:** El coordinador sesión hace la autenticación del usuario para iniciar la sesión.
    1. El usuario ingresa sus datos de ingreso
    2. El coordinador establece de qué usuario se trata por medio de la información del *structurer* Usuarios
    3. Si el usuario ya está previamente registrado y no es empleado, entonces se envía a controlador externos.
    4. Si se trata de un usuario que es un empleado, entonces se envía a controlador internos.
    5. En caso de no estar registrado se crea una excepción.

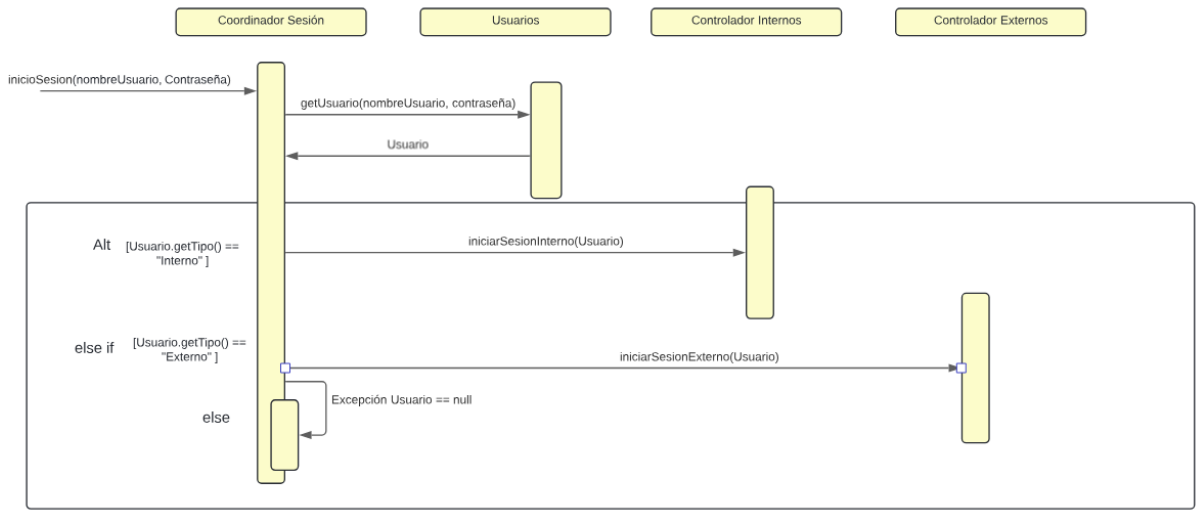



Figura 6: Diagrama de secuencia para un posible inicio de sesión, nótese que, aunque Usuarios hace parte de otro componente del sistema su uso es fundamental para acceder a la información y así dejar que coordinador usuarios tome la decisión.

1. **Crear un nuevo usuario externo:** El coordinador Galería
    1. recibe la petición por parte del cliente de que se quiere crear un nuevo usuario externo.
    2. El coordinador redirige la petición al controlador externos.
    3. El *controller* Externos termina su operación

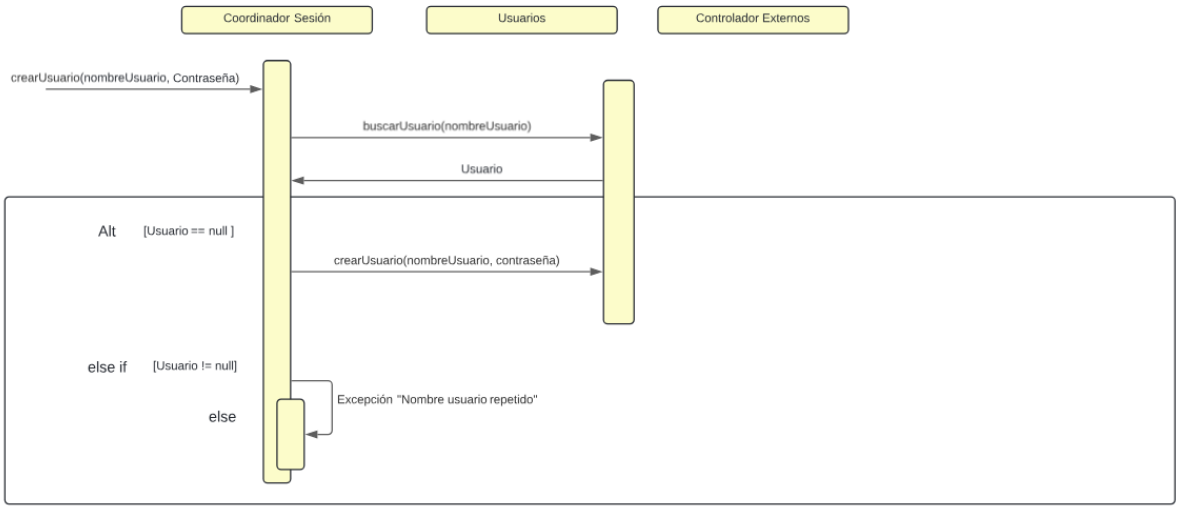



Figura 7: Diagrama de secuencia para la creación de un usuario que no es trabajador de la galería (externo). Se puede observar la interacción únicamente entre el coordinador sesión y el *structurer*

Usuarios, responsabilizando únicamente al *coordinator* Sesión de las tareas de inicio de sesión y creación de usuarios.

Así, el sistema divide las peticiones en dos, asignando desde el primer momento (inicio de sesión) una distribución equitativa de las tareas que debe realizar cada controlador. De esta forma hay menos acople, y mayor flexibilidad a los cambios que el sistema pueda tener.

### *3.2. Inventario*

Como se aclaró en la sección 2, la responsabilidad de este componente es contener la información de las obras y cada parámetro relacionado con ello. Esta responsabilidad es mucho más compleja de lo que puede parecer, por lo que se tomó la decisión de convertir este componente a un *structurer*. De esta forma se mantiene la información más organizada y relacionada entre sí.

#### *3.2.1. Roles*

Así pues, es necesario descomponer el inventario en las piezas elementales que la galería puede tener. De esta forma se hace necesaria la herencia de atributos para las piezas específicas como video, pintura, fotografía, impresión y escultura que comparten la información básica de una pieza genérica. Cada una de estas piezas específicas es un *information holder* debido a que su importancia está en la información que contiene la obra. Además, la información de las subastas, ofertas y ventas hechas son otros componentes que contienen información y no la relacionan de forma específica. Así pues, cada una de estas potenciales clases deben establecerse como *information holder*. Finalmente, debe existir un *structurer* que relacione todos los *information holder* en listas o estructuras relacionales que permitan organizar y clasificar los contenidos del inventario. Además, se añadieron distintos atributos en la clase pieza y una nueva clase artista para cumplir los requerimientos adicionales de este proyecto. De esta forma surge el siguiente diagrama de clases:

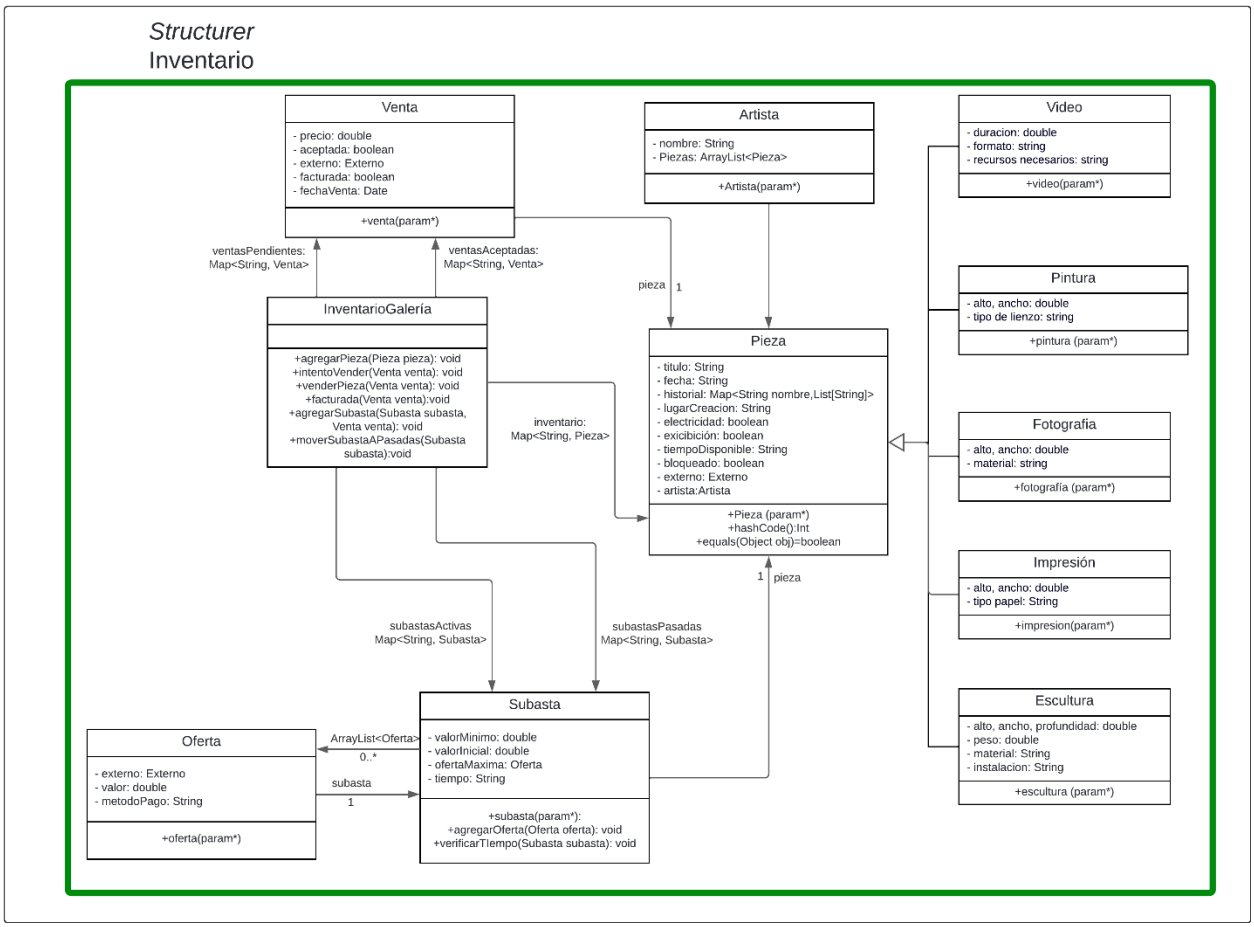



Figura 8: Diagrama de clases actualizado para el proyecto 2 del *structurer* inventario.

### *3.2.2. Responsabilidades*

La responsabilidad de cada clase resulta clara, pues es contener la información de cada relación, atributos o contenencias que se observan en la figura 8. Para el caso especial del inventario galería, sus responsabilidades son agregar ítems o remover ítems al inventario de la galería y agregar una nueva subasta a la estructura *subastasActivas*.

### *3.2.3. Colaboraciones*

No hay una relación de trabajo conjunto para realizar una acción específica en este componente. Resulta factible este hecho ya que sus componentes se basan en contenedoras cuyo método más complejo es editar sus propios parámetros.

## *3.3. Usuarios*

Las responsabilidades del *structurer* Usuarios son las siguientes:

- Estructurar organizadamente la información de cada usuario.
- Relaciona al usuario varias contenencias relevantes de acuerdo con el documento de diseño.

### *3.3.1. Componentes candidatos y estereotipos*

La distribución de responsabilidades es lo suficientemente concreta como para llevarla a la implementación de clases. Sin embargo, es importante considerar una separación entre los usuarios externos e internos, pues de esta forma se facilita el inicio de sesión al separar a los usuarios entre trabajadores (internos) y clientes(externos).

Una decisión muy importante es que no habrá diferenciación con un usuario externo que sea solo propietario o comprador, así se evita la creación de dos usuarios para una misma persona que quiera tener ambos roles. Por cada usuario externo habrá la posibilidad de que este tenga un contenedor llamado “propietario” o “comprador”, de esta forma se almacenará la información relacionada con los roles que pueda tener un mismo cliente. De esta forma, los anteriores componentes cumplirían el estereotipo de *structurer*.

Respecto a los usuarios que trabajan dentro de la galería, se decidió que siempre habrá uno de cada tipo desde la creación de la galería. Además, cada uno tendrá un atributo final y público que mencione el tipo de usuario que es y de qué tipo de interno se trata. Dentro del administrador es necesario también tener una lista que contenga todos los usuarios sin verificar, esto para cumplir el requerimiento funcional mencionado en el documento de análisis. Nuevamente, cada uno de estos usuarios internos es también un *structurer*, ya que tiene una forma específica y compleja de organizar la información.

Finalmente, los usuarios serán almacenados en un contenedor llamado Usuarios Galería, con lo que cualquier persona que se haya registrado en el sistema se encontrará en esta contenedora, así pues, se concluye que este es un *information holder*.

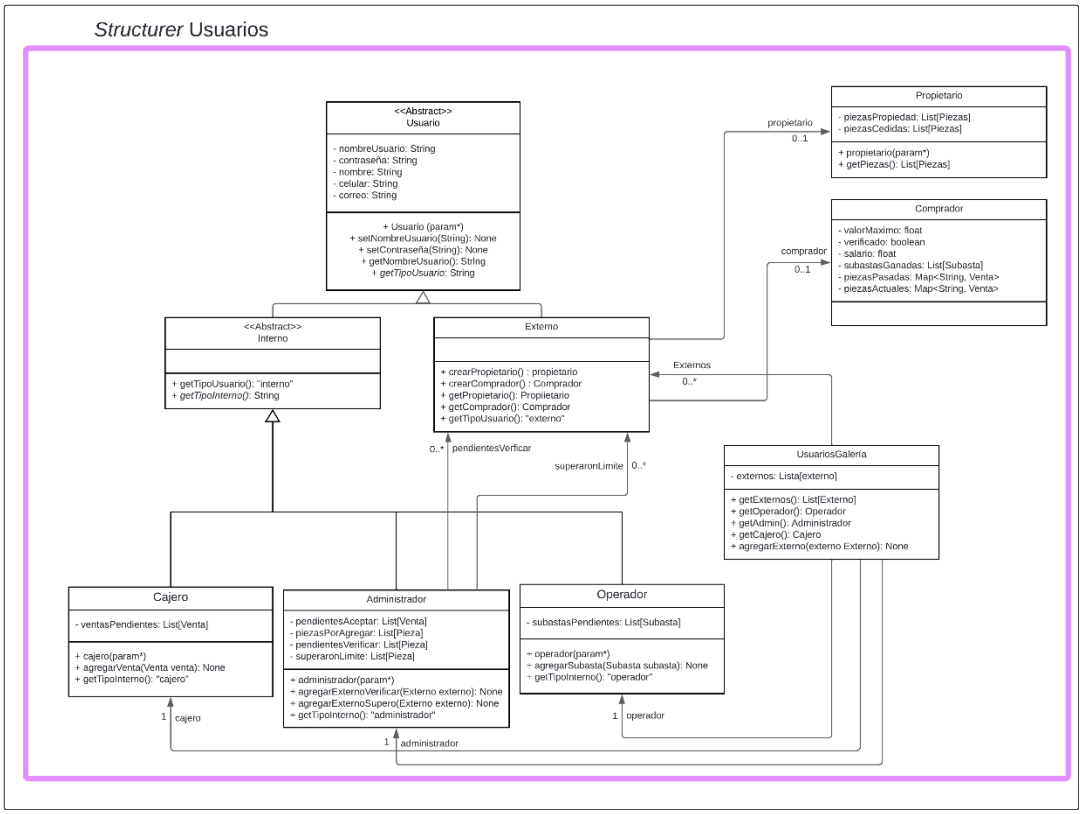



Figura 9: Diagrama de clases actualizado del proyecto 2 para el componente *structurer* Usuarios.

### *3.3.2. Responsabilidades*

Debido a que estas clases son en su mayoría *structurer*, sus responsabilidades se basan en la contención, devolución y agregación de más información de acuerdo con el tipo de usuario que se trate.

### *3.3.3. Colaboraciones*

No hay una relación de trabajo conjunto para realizar una acción específica en este componente. Resulta factible este hecho ya que sus componentes se basan en contenedoras cuyo método más complejo es editar la información de sus propios parámetros.

## *3.4. Coordinador usuarios*

La responsabilidad asignada a este componente fue la de coordinar las peticiones del controlador con los usuarios o el inventario para concretar una acción que depende de muchos roles dentro de la galería.

### *3.4.1. Componentes candidatos y estereotipos, responsabilidades y Colaboraciones:*

Aunque desde un primer instante se pensó que este componente era útil para disminuir la complejidad del controlador galería, al momento de construir su diagrama de clases se vio

que este era prescindible y que con la división de los controladores ya era suficiente la distribución de la carga de responsabilidades. Agregar este componente solo entorpecería las colaboraciones entre los distintos paquetes de componentes al agregar pasos extra.

## 4. Nivel 3

En este nivel, los únicos componentes que hacen falta por descomponer en clases son aquellos que surgieron de controller Galería.

### *4.1. Controlador Internos*

La responsabilidad de este controlador es accionar las operaciones que desee el funcionario de acuerdo con las reglas del dominio.

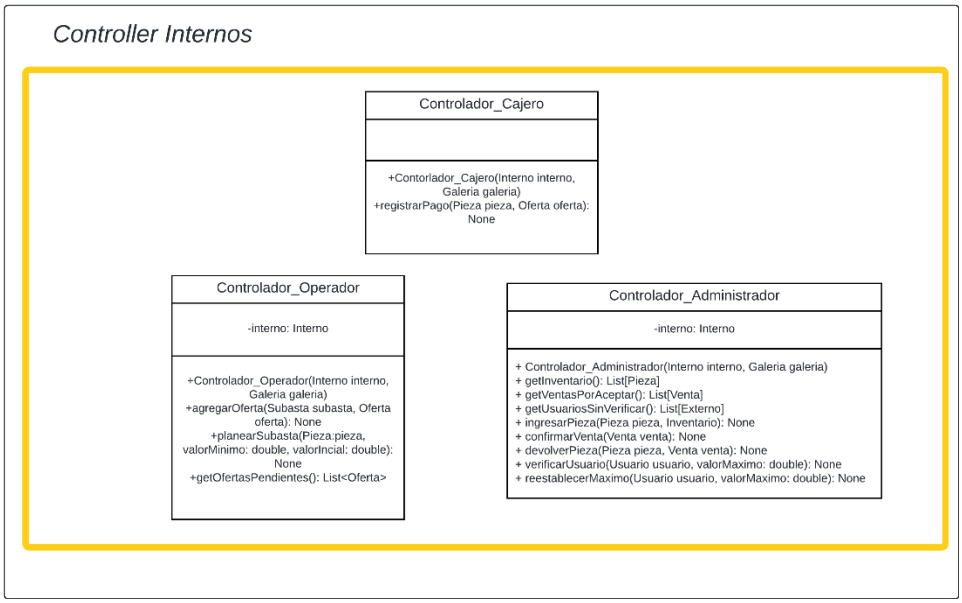



Figura 11: Diagrama de clases para el componente *controller* Internos.

#### *4.1.1. Componentes candidatos y estereotipos*

Como ya se llegó a un nivel de abstracción muy alto, se decidió que este controlador se iba a dividir en controladores para las acciones correspondientes dependiendo del tipo de trabajador. Con ello se tiene un mayor orden y entendimiento de las acciones de cada usuario en el sistema.

#### *4.1.2. Responsabilidades:*

2. Accionar las operaciones correspondientes para cada tipo de empleado.

3. Cada tipo de empleado posee los métodos para realizar los requerimientos funcionales mencionados anteriormente en el documento de análisis.

### *4.1.3. Colaboraciones*

Puesto que el controlador decide qué funciones va a tener acceso el usuario dependiendo su tipo entonces hay una colaboración simple entre esta para proporcionar las tareas específicas a un empleado determinado

## *4.2. Controlador Externos*

Las funciones asignadas a este controlador son el control de las acciones de los usuarios que no forman parte del grupo de empleados de la galería.

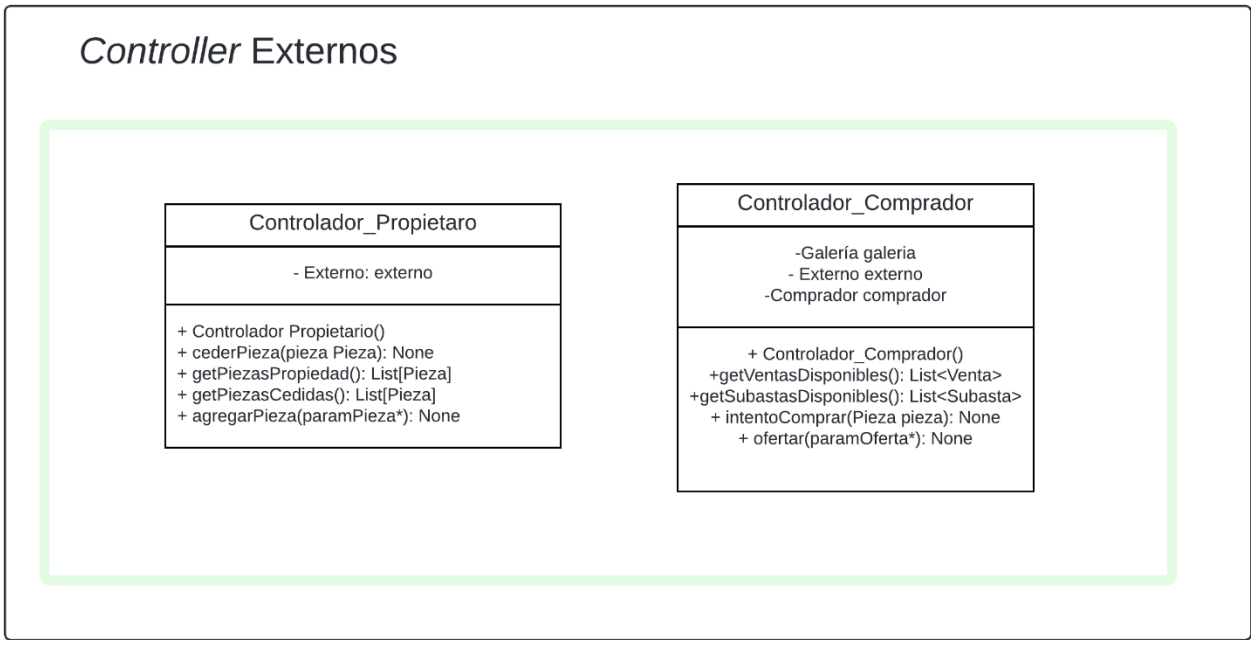



Figura 12: Diagrama de clases para el componente *controller* Externos.

### *4.2.1. Componentes candidatos y estereotipos*

Similarmente al controlador interno, este controlador se dividiría en controladores para las acciones correspondientes del cliente. Con ello se tiene un mayor orden y división de las acciones de cada usuario en el sistema.

### *4.2.2. Responsabilidades:*

- Realizar las acciones solicitadas por el usuario según su rol específico.

- Cada cliente posee los métodos mencionados anteriormente en el documento de análisis.

### *4.2.3. Colaboraciones*

En este caso nuevamente vemos una colaboración similar al controlador internos pues le permite al usuario realizar ciertas acciones según solicite, es decir, dependiendo de si

quiere entrar como comprador o propietario puede acceder a ciertas funciones con el mismo nombre de usuario y contraseña.

### *4.3. Coordinador Sesión*

Su función es la de manejar el sistema de inicio de sesión para que cada usuario acceda al sistema dependiendo de si es interno o externo, y si es un comprador, propietario, administrador, operador o cajero.

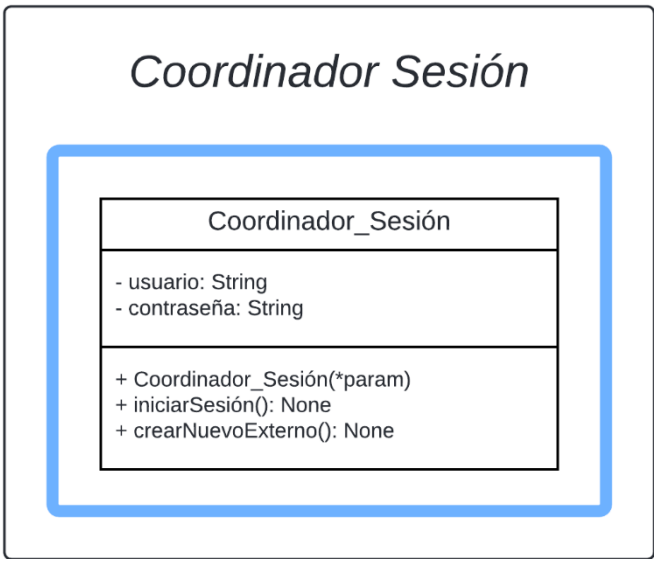



Figura 13: Clase descompuesta del anterior componente coordinador sesión.

#### *4.3.1. Componentes candidatos y estereotipos*

Como este se encarga de repartir el trabajo dependiendo de que usuario se trata, entonces no se cuenta con múltiples clases, siendo que solo una puede encargarse de ello.

#### *4.3.2. Responsabilidades:*

- Corresponder el respectivo usuario con el actor que inicia sesión.

- Si el usuario es externo, redirigir al controlador que debe utilizar el usuario si ingreso como comprador o propietario.

- Crear un usuario externo.

- Delegar las tareas de la sesión al respectivo controlador.

#### *4.3.3. Colaboraciones*

Al solo tener una clase no se tiene colaboraciones internas.

## 5. Diseño final

Según el desarrollo mostrado, se puede establecer la perspectiva general de la aplicación Galería con un diagrama de clases de alto nivel. Si se quiere más detalle sobre los métodos de cada clase, se recomienda ver los diagramas mostrados antes. Tener un diagrama UML con todos los métodos no sería tan útil por la complejidad de los espacios en la imagen y el poco aporte que hace en el entendimiento de la estructuración del programa.

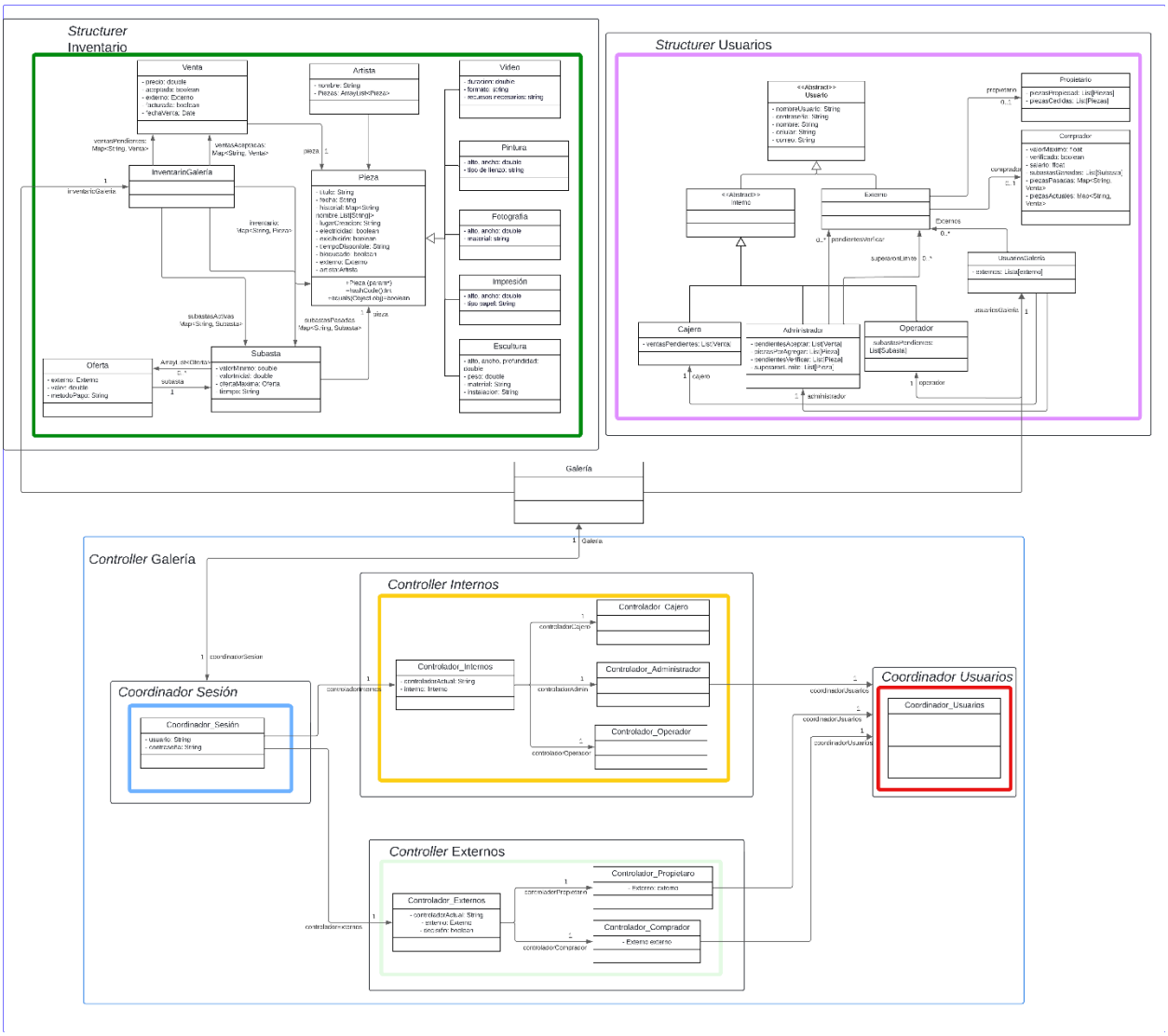
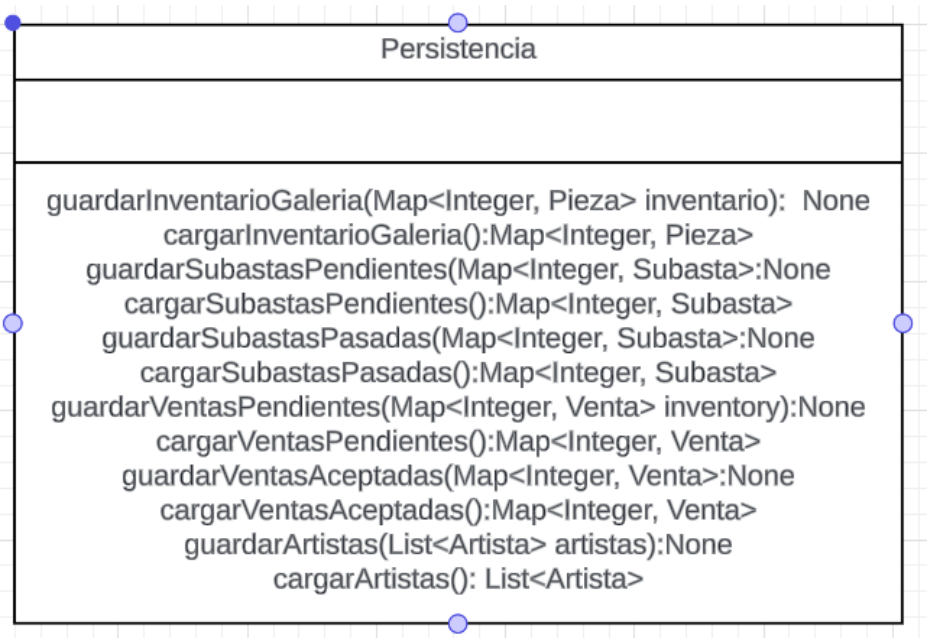



Figura 14: Resultado final del diseño como un diagrama de clases de alto nivel.

## A. Apéndice: Diseño de la persistencia.

Dado que se hizo la separación de estructuras por usuarios e inventario resulta sencillo hacer una estructura únicamente una estructura para el inventario. Guardando cada uno

de los hashmaps y listas más importantes de forma individual para así facilitar su carga y guardado.




## B. Apéndice: Diseño de la interfaz

Siguiendo el hilo del documento, nótese que el trabajo de la interfaz está muy facilitado gracias a que los controladores ofrecen los métodos para cualquier requerimiento funcional que quiera realizar un actor. Debido a que en este proyecto se pidió realizar una interfaz aparte por cada actor con un método *main* en cada uno, el proceso de iniciar sesión es ineficiente y complejo. Esto es porque en el diseño se pensó que a partir de un usuario y contraseña se debería redirigir a una interfaz en específico, lo cual tiene mucho más sentido en una aplicación en la que se necesita iniciar sesión. Por este motivo, aunque se podría diseñar con menor complejidad el inicio de sesión, se decidió que la aplicación no se cambiaría.

Así, se espera tener una clase por cada uno de los actores requeridos (administrador, comprador, operador, cajero) que contenga la siguiente estructura genérica:

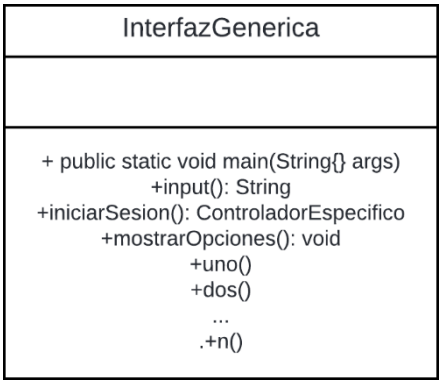



Figura B1: Clase modelo para cada interfaz a realizar. Se espera que por cada una de las opciones en *mostrarOpciones()* haya un método que lo ejecute: + *uno()*, + *dos()*... donde el método +*n()* permitiría al usuario salir de la aplicación. Además, el método +*input*() se encarga de leer las respuestas del usuario para las opciones.

En el primer momento en el que el método *main* es llamado, se invoca a la persistencia para que esta devuelva la galería cargada. A partir de la galería, se hace el llamado de *iniciarSesion*(), método que devuelve el controlador específico si el inicio de sesión fue exitoso. Una vez con el controlador correspondiente, el trabajo de la interfaz se basa en desplegar los getters() del controlador, recibir y transformar los inputs del usuario, y hacer llamados a los métodos del controlador. Al final, cuando el usuario seleccione la opción n se cierra la sesión y se hace un llamado a la persistencia para que se guarden todos los cambios.

De esta forma se completan todos los requerimientos funcionales que la aplicación debe cumplir, donde el diagrama de clases del paquete de *interfaz* resultó así:

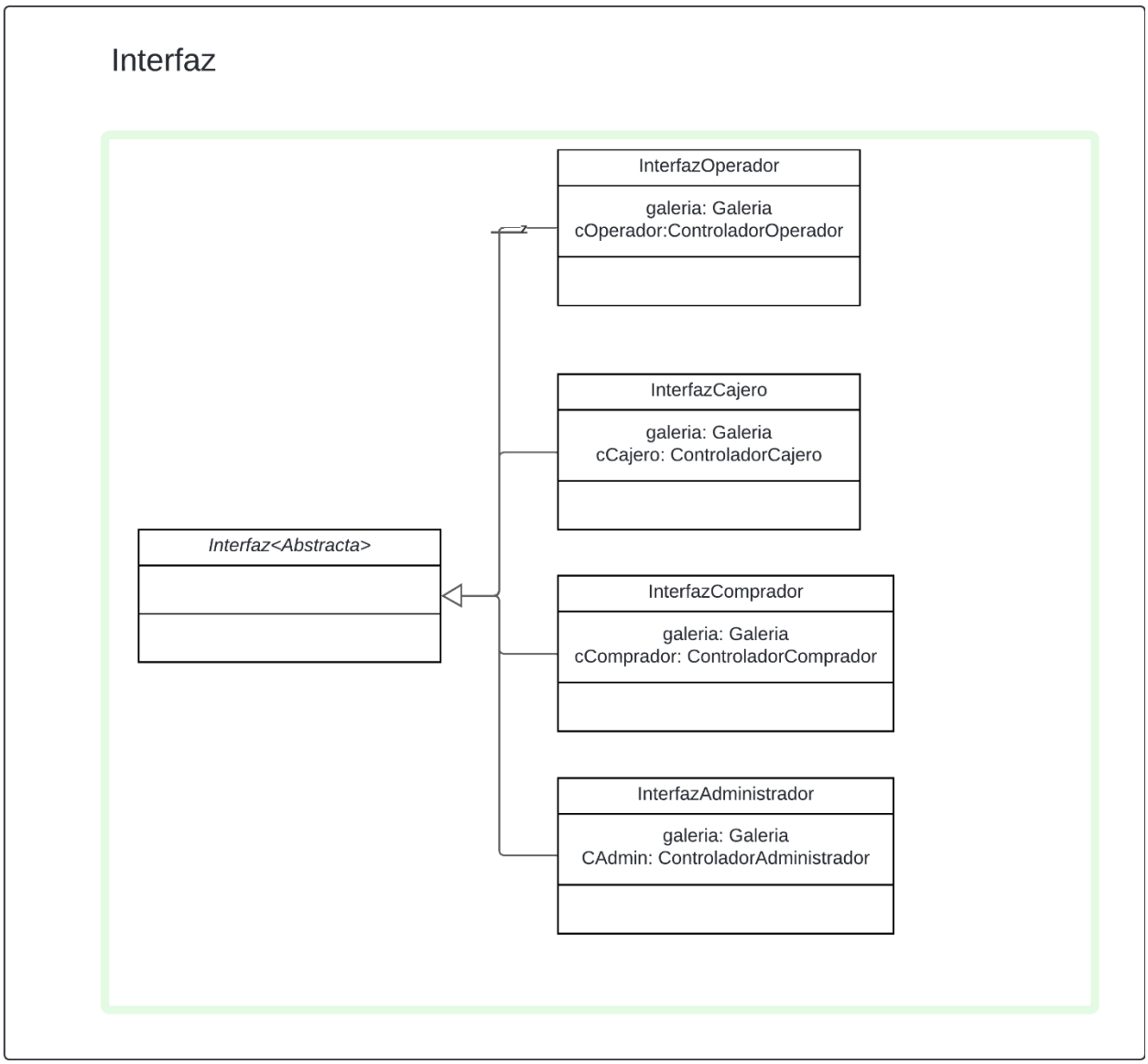



Figura B2: Diagrama de clases de la interfaz final. La interfaz abstracta sólo tiene métodos de inputs para generalizar las entradas que se le solicitan al usuario.