**Documento de diseño: Proyecto 2**

**Santiago Quiroz Pintor**

**Miguel Flórez Gutierrez**

**Carlos Daniel Ramírez Rodríguez**

**Luis Sebastián Contreras Diaz**

**Universidad de Los Andes**

**Diseño y Programación Orientado a Objetos**

**9 de mayo de 2024**

**Introducción**

Para la segunda entrega del proyecto del curso, se buscaba mejorar el diseño de un programa capaz de manejar la información de una galería de arte.

**Historias de Usuarios**

registro: primer registro (admin), de empleado y comprador

ingreso de pieza

creacion de subasta

compra piea costo fijo).

**Decisiones de diseño**

1. **Manejo de inventario**

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

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

1. **Acceso al sistema y usuarios**

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

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

Una vez que el sistema ya ha sido creado, este siempre se inicia desde una instancia de *ViewLogin*. Aquí se verifica el usuario y la contraseña para dirigir al usuario al menú que le corresponde dentro del sistema. En caso de que el usuario no exista, este puede registrarse así mismo; sin embargo, este proceso solo crea usuario de tipo *Comprador*.

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

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

1. **Subastas**

La subasta es configurada por el adminsitrador, se verifica que existan al menos cinco piezas disponibles. Si se cumple este requisito, se seleccionan de forma aleatoria y se asigna un empleado como operador de la subasta y, por último, el administrador debe verificados compradores registrados en el sistema para que participen dentro de la subasta y luego se pasa a guardar la información de la subasta en el sistema.

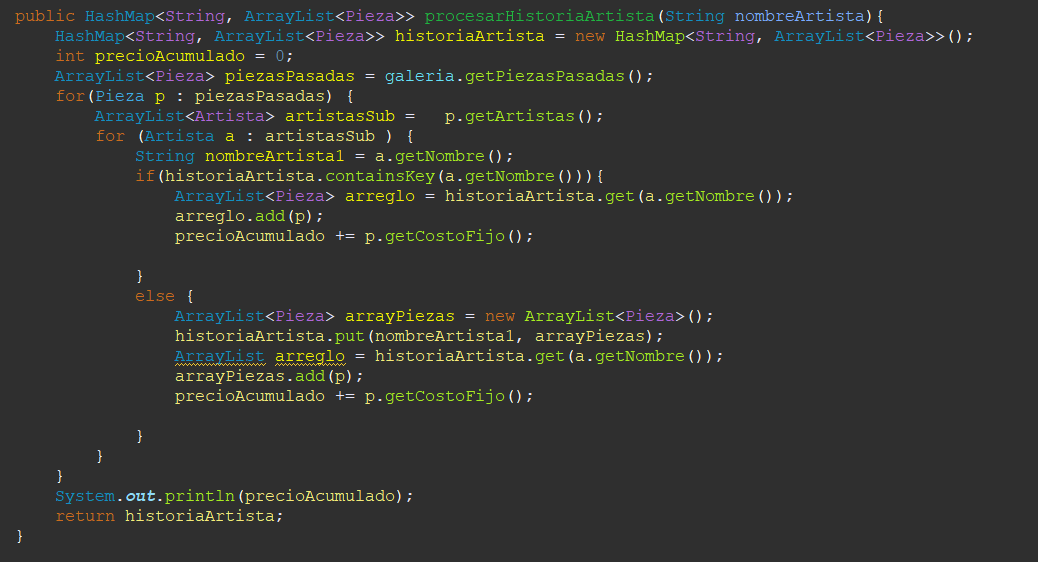
Una vez creada, la subasta se iniciarSubasta en la clase Subasta, donde se notifica a compradores verificados y se comienzan a mostrar las piezas en intervalos regulares, permitiendo que se realicen ofertas. Este proceso se mantiene activo hasta que todas las piezas hayan sido ofrecidas, momento en el cual se finaliza la subasta. Durante este tiempo, las ofertas son gestionadas y actualizadas continuamente, asegurando que el valor más alto sea registrado. Al final, se cierra la subasta, registrando los resultados y restableciendo el estado de los participantes, asegurando que el sistema esté listo para la próxima subasta.

1. **Requerimientos del Sistema**

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

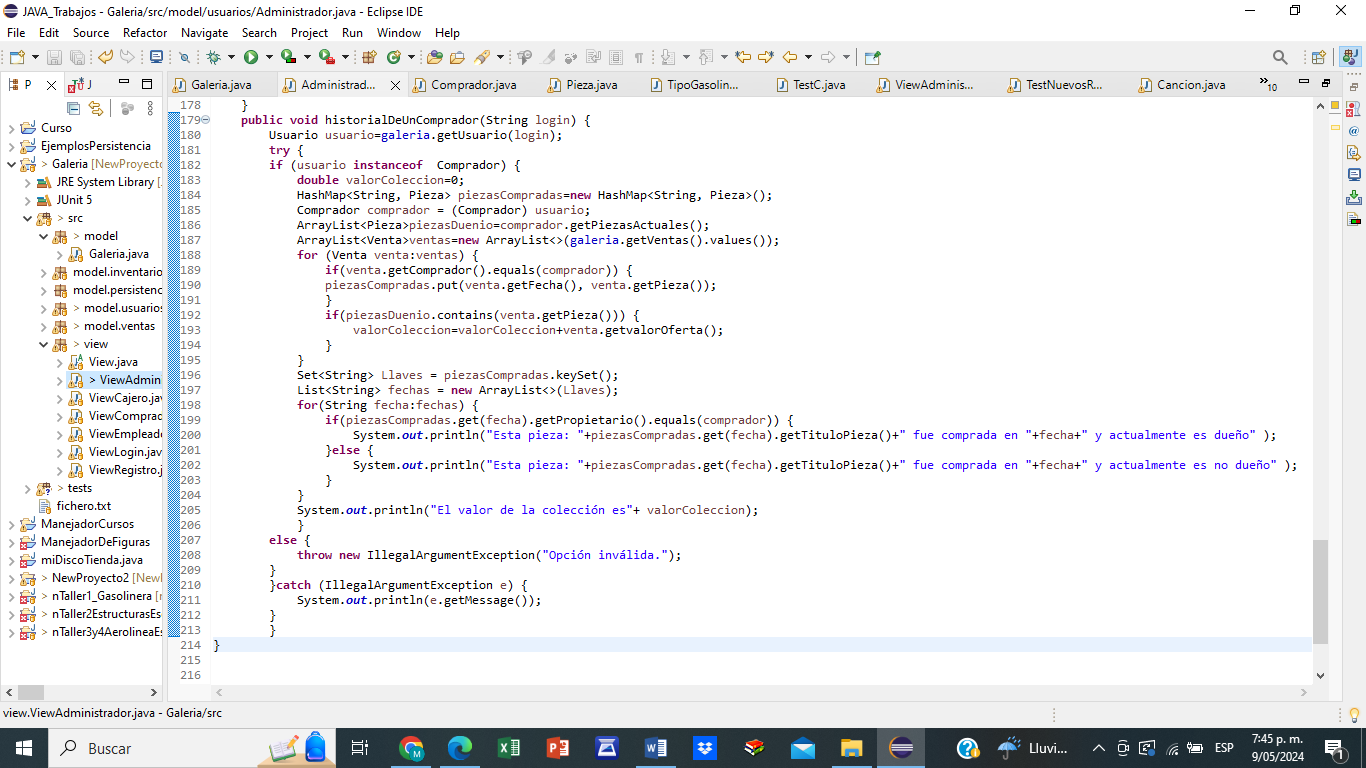
Requerimientos No-Funcionales: Incluye aspectos como seguridad, rendimiento, plataformas soportadas, etc.

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



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

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



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

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

1. **Diseño de Arquitectura**

Para el diseño del sistema utilizamos un estilo de control delegado, esto ya que los primeros elementos del sistema con los que interactúa el usuario son views, los views que aparecen dependen del tipo de usuario(aquí se aplica el principio de diseño de segregación de interfaces.). los view actúan como coordinadores. ya que delegan responsabilidades al programa que este debe cumplir Si se está ingresando al sistema por primera vez se le pide al usuario que cree un administrador, porque sin este la galería no podría aplicar la mayoría de funciones que esta ofrece. Una vez creado el administrador se inicializa la galería. Un aspecto importante es que tanto la galería como el administrador son controladores, esto ya que manejan gran parte de la lógica del programa, toman decisiones importantes y manejan la información de varios componentes con la intención de cumplir las funcionalidades de la galería. El resto de las clases se encargan de cumplir una tarea en específico(Single Responsability). que a punta de colaboraciones con las otras clases hace que la aplicación funcione correctamente.

1. **Persistencia**

**Diagrama de clases UML**

Para llegar al UML final pasamos por distintos diagramas UML que discutimos y analizamos frecuentemente para poder lograr hacer un diagrama UML que cubra todo el proyecto de la galería. Este primer diagrama fue un aspecto general de cómo manejaríamos las diferentes clases de la aplicación, a grandes rasgos se hizo un cambio grande con respecto al diagrama de análisis utilizado, este diagrama lo utilizamos para orientarnos de lo que teníamos que hacer. Sin embargo, fue altamente discutido y provocó los siguientes diagramas más complejos y estructurados.

Finalmente, llegamos a este diagrama en el que nos basamos para realizar el desarrollo del programa

**Diagrama de clases de diseño**



Con el desarrollo de diagramas de secuencia y clases de alto nivel, el equipo pudo visualizar y refinar la estructura del sistema, asegurando una implementación eficiente y bien organizada. A través de iteraciones y análisis continuo, se logró un diagrama UML final que abarcaba todos los aspectos del proyecto, desde la persistencia de datos hasta la experiencia del usuario en la galería.