Proyecto # 2

Marlon David Forero-202220738

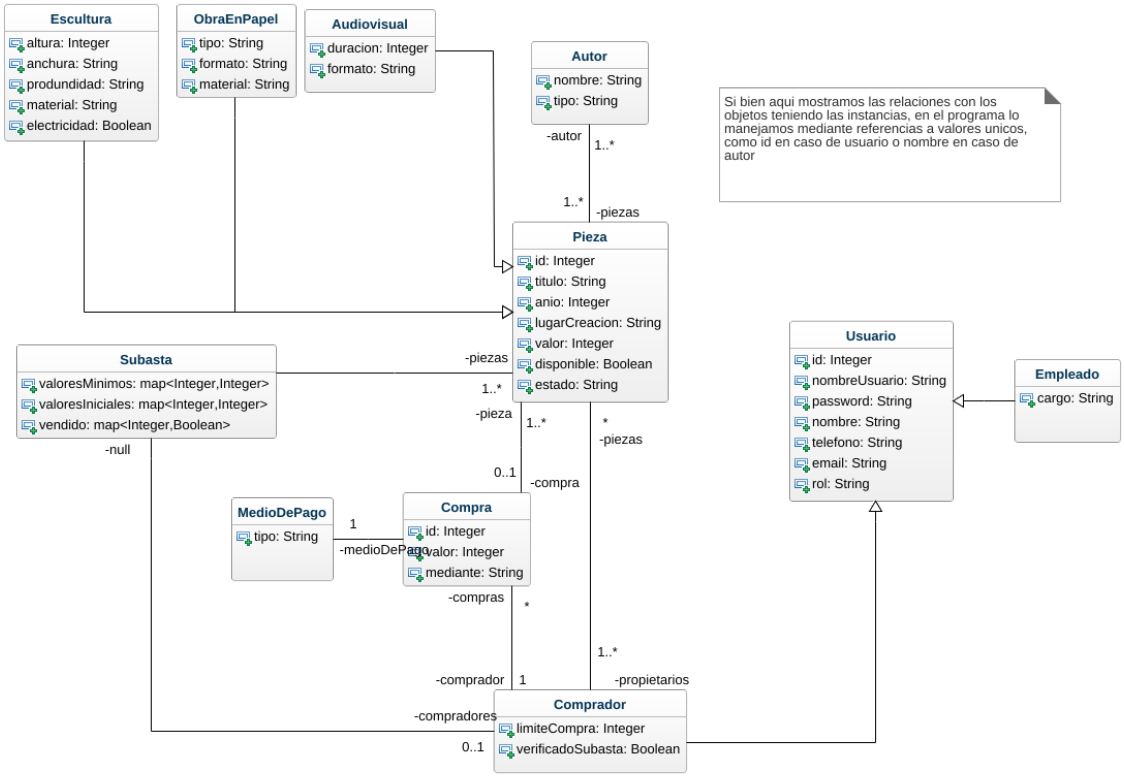
Miguel Rey – 202112736

Daniel Mancilla- 202221038

Contenido:

- **Entidades y relaciones**
- **Lógica y decisiones de diseño**
- **Restricciones y funciones actuales en la consola**

Entidades y relaciones

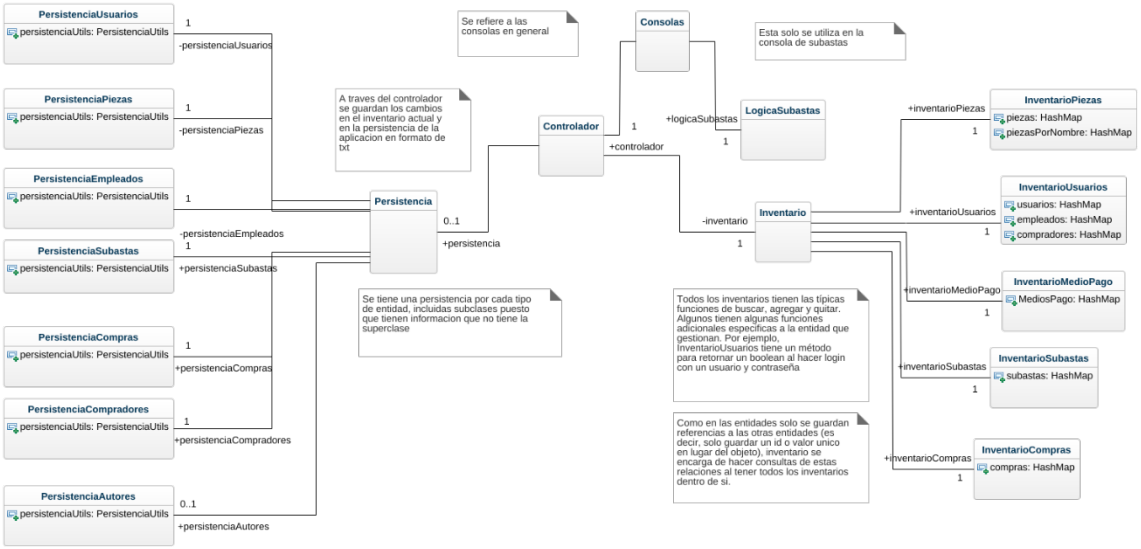


Las entidades las organizamos y relacionamos tal como muestra el diagrama, tuvimos en cuenta principalmente que objeto podría contener a otro (pues, por ejemplo, una subasta debe contener si o si las piezas que se van a subastar) y las relacionamos. No obstante, estas relaciones no son todo bidireccionales todo el tiempo ya que se podría complicar mucho para persistencia y gasto de memoria. Además de esto, hay que tener en cuenta algunas aclaraciones:

- Si bien están organizadas de esta forma, tal como indica la anotación, no conservan las instancias de las demás entidades dentro de si puesto que (como se explicará en la sección de lógica y decisiones de diseño) sería algo complejo manejar de forma adecuada la persistencia. Por esto, decidimos simplemente dejar referencias a la entidad a algún tipo de valor único de la entidad, como un id en el caso de la mayoría o un nombre que no se repite, cosa que pasa en MedioPago y Autor
- Las entidades las pensamos para poderse ampliar y agregar requerimientos adicionales en el futuro (Como en el punto anterior, se explicará más detalladamente en la siguiente sección), por lo que hay entidades que no contienen mucha información. Como mayor ejemplo, está la entidad MedioPago, que simplemente tiene como atributo su nombre, otro ejemplo es Empleado, que además de la información heredada de Usuario, solo tiene un atributo que es el cargo.

El diagrama se va a encontrar en la carpeta Docs del proyecto en una mayor resolución

Lógica y decisiones de diseño



El diagrama se encuentra en mejor calidad en la carpeta Docs del proyecto

Esta sección se encuentra dividida en las siguientes partes:

- Persistencia
- Lógica interna

En las cuales explicaremos los distintos aspectos del diseño de nuestro proyecto y las decisiones que tomamos, pensadas para permitir la escalabilidad y adaptabilidad del proyecto.

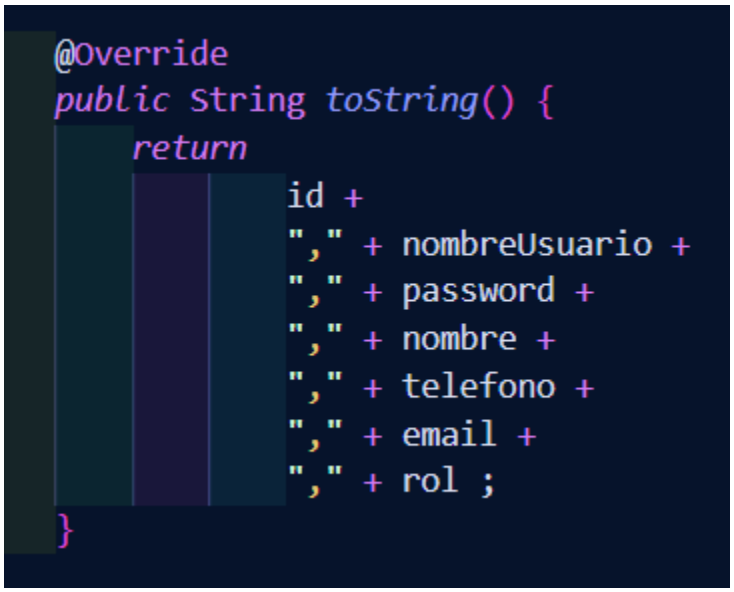
Persistencia

Para la persistencia, decidimos utilizar archivos de texto (.txt) en donde guardar la información de las entidades. La estructura de cada archivo es similar a un csv, pero con varios cambios para almacenar de forma efectiva la información de ciertas estructuras especiales pero importantes para el funcionamiento de la aplicación.

La estructura de las carpetas y archivos es sencilla, un archivo por entidad, elegido por el sencillo motivo de que es relativamente fácil de cambiar para actualizar una entidad ya existente (puesto que simplemente reemplazamos el archivo que la contiene, en lugar de sobrescribir un archivo que contenga todas las entidades del mismo tipo). Gracias a esta decisión, el guardar archivos también es un proceso rápido puesto que simplemente necesitamos guardar o actualizar una entidad en el momento que es creada o actualizada, en lugar de guardar todas las entidades del mismo tipo o actualizar el archivo que las contiene todas.

No obstante, este modelo de persistencia puede facilitar el generar errores en el programa al introducir o no borrar apropiadamente un archivo, algo que intentamos suprimir todo lo posible en las clases encargadas de la persistencia, pero que no podemos suprimir totalmente en el caso de que alguien interfiera con las carpetas de forma externa.

La estructura de cada archivo, tal como indicamos anteriormente, es básicamente un csv que depende de cada entidad (ver las funciones toString() de cada entidad para entender a detalle la estructura de cada una), por lo que es relativamente fácil de editar y cargar en la aplicación.



Método toString() de Usuario

Debido a la estructura que elegimos, es fácil añadir nuevas entidades, sean totalmente nuevas o subclases de las que ya tenemos. Por ejemplo, Usuario tiene dos subclases, Empleado y Comprador, que tienen información adicional del cargo y piezas de las que es propietario el usuario respectivamente, pero al almacenarse en carpetas distintas, podemos editar cualquiera de las dos clases y no afectara directamente a la otra, ni siquiera si cambiamos la estructura de Usuario serán afectada su persistencia.

Lógica

Para la lógica, la mayoría de los métodos son internos de las clases, para que se puedan verificar rápidamente varias restricciones y reglas en la consola o interfaz. Por ejemplo, no poner dos veces la misma pieza en la misma compra o subasta. Fuera de esto, tratamos de hacer la aplicación lo mas modular posible, para que pueda ser cambiada o ampliada sin mucho problema, por este motivo, las clases para la persistencia están separadas por cada entidad a pesar de que los inventarios no lo estén del todo. Decidimos separar de esta forma la persistencia para que verifique exactamente y de ser necesario, agregar funcionalidades adicionales al momento de guardar cada clase en específico, pero que todas estén conectadas a una clase principal que sirve de conexión con el controlador, que es la clase que se comunica con la interfaz que interactúa con el usuario. Debido a esto, podemos editar cosas en la base de la persistencia y solo necesitaríamos cambiar la estructura de métodos simples (o añadir adicionales).

Por otro lado, los inventarios si los simplificamos un poco respecto a las clases de la persistencia, decidimos dejar en el mismo inventario a las clases que están profundamente relacionados, como lo son las superclases y sus subclases (el caso de Pieza y Usuario), puesto que en un solo requerimiento puede ser necesario utilizar varias de estas. Por ejemplo, al hacer login, inicialmente se consulta si el usuario existe, para luego verificar que exista como empleado o comprador (según la consola). Igual que con las persistencias, todos los inventarios están conectados a uno solo, donde se pueden llegar a conectar varios inventarios para alguna operación, como verificar si existe una pieza para inmediatamente agregarla a una compra o subasta.

Cabe aclarar que si los métodos no se ven en el uml es porque serian muchos y muy repetitivos, puesto que los inventarios tienen funciones para agregar, quitar y buscar bastante simples e inventario principal tiene un método que conecta con cada uno de los métodos en los inventarios, sin agregar mucho más. Caso similar con las persistencias, pero simplemente para cargar y guardar datos, al igual que con Controlador, que tiene los métodos que conectan con los métodos de inventario y persistencia.

Finalmente, está el controlador, la clase que conecta la persistencia y el inventario. Esta clase se encarga de hacer las operaciones indicadas por la interfaz y que inmediatamente guarda en la persistencia los cambios hechos. Por ejemplo, al momento de crear una nueva pieza (sin importar de que tipo sea), llama al inventario para agregarla (y así este llame a InventarioPiezas) y justo después, guardarla como archivo txt, siendo igual si se quiere actualizar o borrar, pero con sus respectivos cambios. Debido a esta estructura, si queremos añadir funcionalidades, simplemente debemos editar los métodos en los inventarios para que hagan algo nuevo, incluso añadiendo estructuras internas nuevas (En el caso de InventarioPiezas, manejamos algunos mapas adicionales para búsquedas mas rápidas, como lo sería buscar por nombre una pieza).

Hay una clase adicional, que es LogicaSubastas y la utilizamos para ir imprimiendo más fácilmente los datos de cada pieza y comprador de una subasta, para evitar introducir toda esta lógica en la interfaz o el controlador.

Como se puede ver, tratamos de hacer nuestra aplicación de la forma más modular posible, para que se facilite agregar nuevas clases o funcionalidades a clases ya existentes (como un nuevo tipo de usuario o un nuevo tipo de pieza), lo que, si bien puede generar que tengamos una cantidad muy grande de clases distintas, facilita el trabajo para los próximos desarrolladores que deban modificar el programa.

Restricciones y funciones actuales en la consola

En la consola, añadimos todas las funcionalidades solicitadas, pero no logramos hacerla del todo amistosa con el usuario debido a la complejidad que representaba. No implementamos todas las opciones con simplemente un input numérico debido a que podría requerir imprimir menús bastante grandes y complicar la implementación de cada requerimiento. Además de esto no alcanzamos a implementar que en cada input se impida avanzar al usuario si pone una opción incorrecta o invalida, pero que le permita volverla a poner en esos casos debido a la complejidad que representaba.

No obstante, sigue siendo una consola relativamente sencilla de utilizar pero que requiere saber los id de cada pieza (o sus nombres en casos particulares), lo bueno es que los id se van generando de 1 en 1, así que si hay una cantidad interesante de estas entidades (sean usuarios o piezas), es fácil encontrarlos poniendo números pequeños.

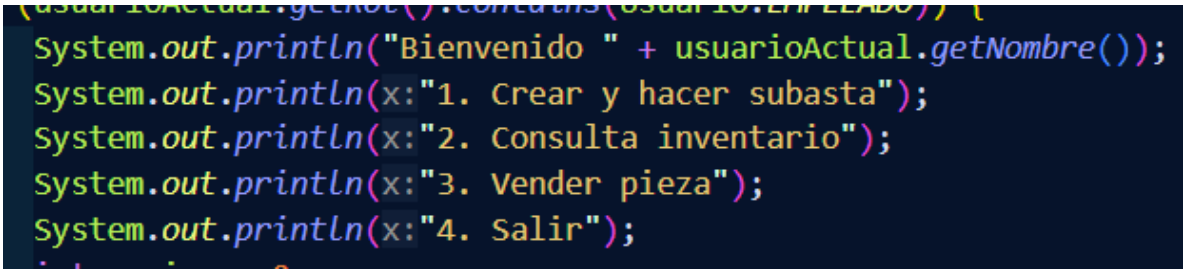
En uso, es una interfaz bastante intuitiva con lo que se puede hacer, explica directamente la funcionalidad de cada opción y pide el menor numero de inputs posibles para utilizarla.

```
System.out.println("Bienvenido " + usuarioActual.getNombre());
System.out.println(x:"1. Menu piezas");
System.out.println(x:"2. Consultar compradores");
System.out.println(x:"3. Crear y hacer subasta");
System.out.println(x:"4. Añadir usuarios");
System.out.println(x:"5. Consulta inventario");
System.out.println(x:"6. Vender pieza");
System.out.println(x:"7. Salir");
```

Menú principal en MainAdmin

```
(usuarioActual.getRol().contains(Usuario.COMPRADOR)) {
System.out.println("Bienvenido " + usuarioActual.getNombre());
System.out.println(x:"1. Consultar mis piezas");
System.out.println(x:"2. Consulta inventario");
System.out.println(x:"3. Salir");
```

Menú principal en MainComprador



Menú principal en MainEmpleado

Como se puede ver, los menús explican bastante bien que historias se implementaron por consola. En el caso de MainAdmin es posible acceder a todas las funcionalidades de la aplicación, que incluyen el crear usuarios, compradores, empleados, subastas, piezas, etc. Además de la consulta de todo un comprador y el historial de sus piezas. Para MainComprador simplemente dejamos que pueda hacer las consultas de la historia de una pieza, un autor y sus propias piezas. Finalmente, para MainEmpleado, permitimos que pueda vender piezas únicamente a usuarios autorizados por el admin, además de crear una consulta y consultar la historia de una pieza o autor.

Los resultados de cada una son simplemente un mensaje de confirmación o, en el caso de las consultas, la información organizada de cada compra y/o pieza (pero sin mostrar todos los datos, como la lista de piezas de una compra), igual con los autores.

No se pueden consultar todas las piezas ni tampoco usuarios o entidades del estilo puesto que es información algo complicada de ver en una consola, pero las consultas indicadas en este proyecto #2 son posibles (y accesibles desde sus respectivos main).

El diseño de la interfaz es modular también, puesto que funciones especificas tienen su propia clase la cual es llamada desde el main correspondiente, así podemos limitar que opciones tiene cada tipo de usuario sin necesitar un main monumental, de hecho, siendo los main de las clases de consola con menos código.