**Taller 5 – Patrones de diseño**

**Patrón:** Data Access Object (DAO)

El patrón DAO proporciona una abstracción de la lógica de acceso e interacción de datos, separándola de lógica de la aplicación logrando así un código más simple y mantenible.

El patrón consta de 3 elementos claves:

* ObjetoDAO: Una interfaz o clase abstracta en la que se definen los métodos que se emplearan para acceso y manipulación de datos, generalmente se definen las operaciones básicas CRUD (Create – Read – Update - Delete), esto permite que se interactúe con los datos sin necesidad de saber la implementación.
* Objetos de domino: Los objetos relacionados con la aplicación y de los cuales se almacenan datos, como usuarios o productos. Esta clase contiene la lógica asociada a la app.
* DAOobjetoImpl: Esta es la clase que implementa de la interfaz, se emplea para establecer la lógica, según la tecnología de almacenamiento de datos, como archivos o bases de datos.

El patrón DAO contribuye a la evolución y el mantenimiento de la aplicación, debido a que los cambios a diferentes fuentes de datos o tecnologías de almacenamiento no afectan o comprometen directamente la lógica de la app. Asimismo, mejora el rendimiento y la seguridad, ya que permite estrategias como el uso de cachés, la gestión de transacciones y la validación de permisos para el obtener acceso a los datos.

**Proyecto para analizar:** Tienda online

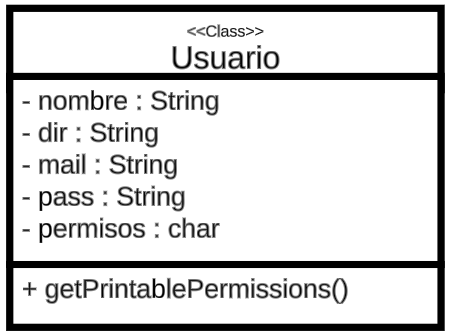
Repositorio de GitHub: <https://github.com/JuanDYB/tiendaonlinelpijdyb.git>

El proyecto es una tienda online dividida en dos grandes funcionalidades: las del usuario y las del administrador. El administrador tiene la capacidad de gestionar productos, usuarios y comentarios, además de acceder a un historial de ventas. Los usuarios, por su parte, pueden realizar compras y pagos, y también tienen la posibilidad de dejar comentarios sobre los productos ofrecidos.

El proyecto presenta diversos desafíos, como la persistencia de datos, que se aborda utilizando el patrón DAO. Otro desafío importante es garantizar la seguridad durante los pagos, ya que la información bancaria es extremadamente delicada. Además, se debe abordar la sincronización de datos, ya que podría darse el caso de que dos personas intenten comprar la única unidad disponible de un producto al mismo tiempo, lo que generaría un conflicto. Asimismo, en el apartado de comentarios, es necesario implementar un sistema que filtre y maneje los comentarios inapropiados.

La estructura del proyecto se basa en el patrón Modelo-Vista-Controlador (MVC). En el modelo, se encuentran las clases:

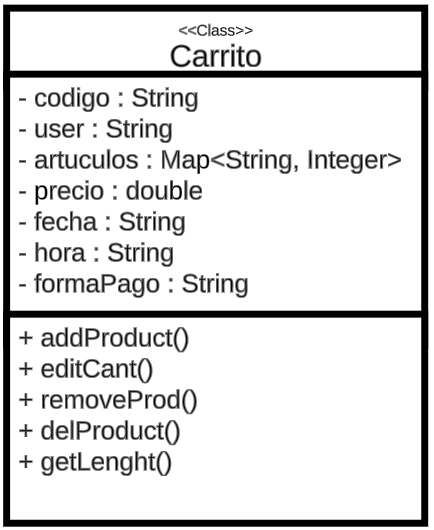
* Usuario:



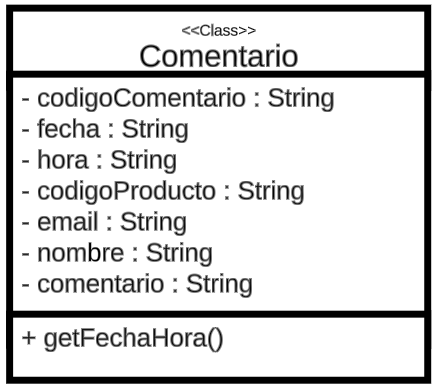
* Producto:



* Carrito:



* Comentario:



Cabe recalcar que en las clases producto y carrito se hace uso de la sobre carga de métodos, aplicada en el constructor.

En cuanto al rol que desempeña el patrón en el proyecto, se centra en el acceso e interacción de datos. A continuación, se explicará el código que demuestra la aplicación del patrón DAO.

El código se encuentra dentro del paquete "Persistencia". En este paquete, se encuentra la clase "PersistenceFactory", la cual se basa en el patrón Factory y se encarga de crear y devolver instancias de objetos. Aquí es donde se realiza la elección entre el almacenamiento de datos en archivos o a través de una base de datos. Además, en el paquete se encuentra la interfaz "PersistenceInterface", que sirve como base del DAO. En esta interfaz se definen los métodos de acceso y manipulación de datos que serán implementados por las clases concretas.

Por último, las clases "PersistenceFile" y "PersistenceBD" son las encargadas de implementar el código necesario para el acceso y manipulación de datos, según la tecnología de almacenamiento elegida. Estas clases representan las opciones de almacenamiento en archivo y base de datos, respectivamente.

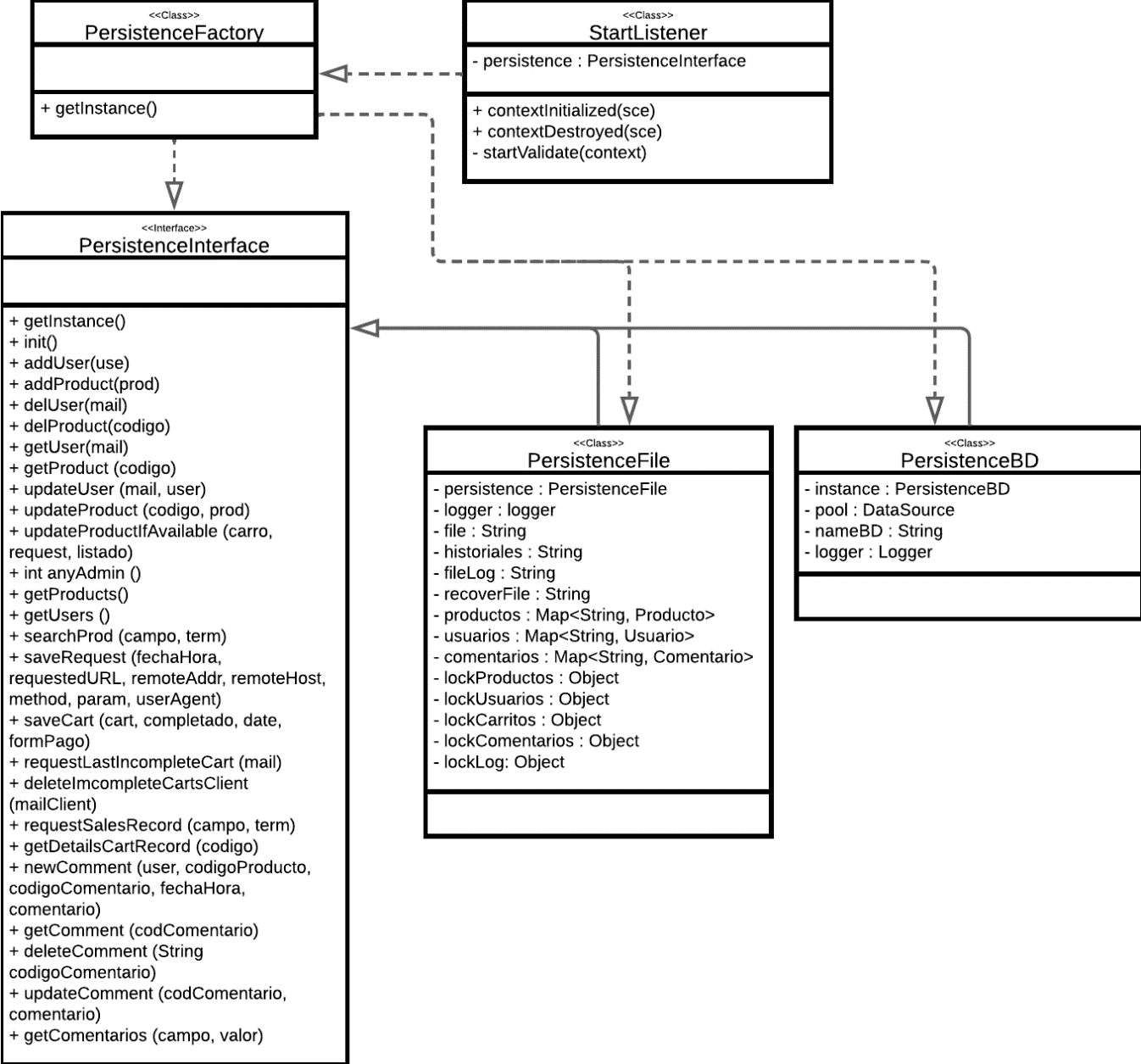


Diagrama UML que representa la relación de las clases e interfaz del package “Persistencia”, adicionalmente la relación de la clase que ejecuta o invoca los métodos disponibles para el acceso y manipulación de datos.

En lo expuesto anteriormente se evidencian las ventajas del Patrón DAO, ya que no solo permite el uso de archivos o base de datos para almacenar los datos importantes de la aplicación, sino que también brinda la flexibilidad de adoptar otros métodos de almacenamiento de datos. Simplemente se requiere instaurar una clase con la lógica necesaria, y el código fuente no se verá afectado en caso de ser necesario cambiar el método de almacenamiento. Sin embargo, en proyectos de mayor envergadura, la integración del patrón puede volverse un poco tediosa.

En el caso del proyecto analizado, en mi opinión, habría sido factible abstraer aún más algunas funcionalidades comunes entre los objetos manipulados. Esto se podría lograr creando una interfaz que acepte un objeto genérico y que herede una interfaz específica para cada tipo de objeto (Usuario, Producto, Carrito de compras, etc.). Esta abstracción adicional también contribuiría a mejorar la comprensión y organización del código.

@Override

    public boolean addUser(Usuario user) {

        synchronized (lockUsuarios) {

            if (usuarios.containsKey(user.getMail())) {

                return false;

            } else {

                usuarios.put(user.getMail(), user);

                return true;

            }

        }

    }

    @Override

    public boolean addProduct(Producto prod) {

        synchronized (lockProductos) {

            if (productos.containsKey(prod.getCodigo())) {

                return false;

            } else {

                productos.put(prod.getCodigo(), prod);

                return true;

            }

        }

    }

Parte del código donde se aprecia la repetición de operaciones comunes entre los objetos.