Gabriela García Suárez 202210869

Patrón: Singleton

Repositorio: <https://github.com/giovanniramos/PDO4You.git>

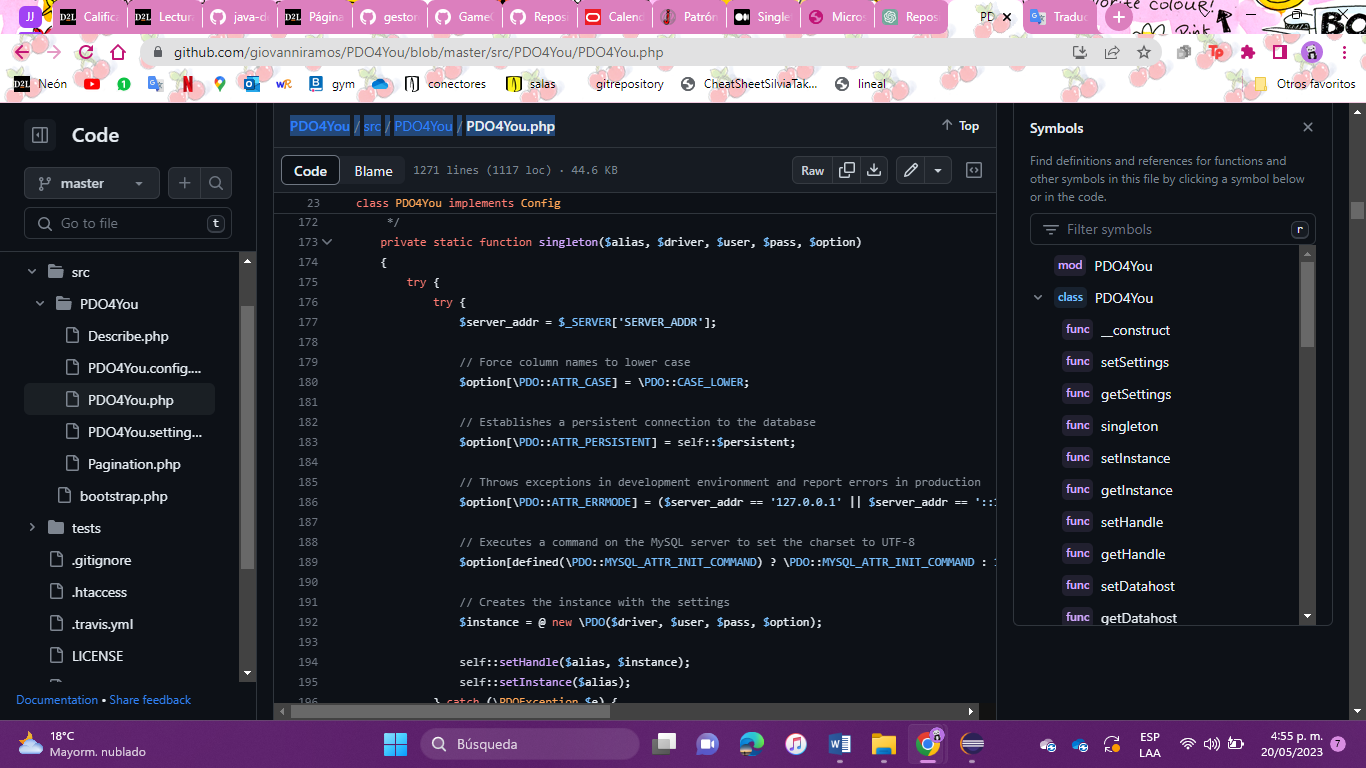
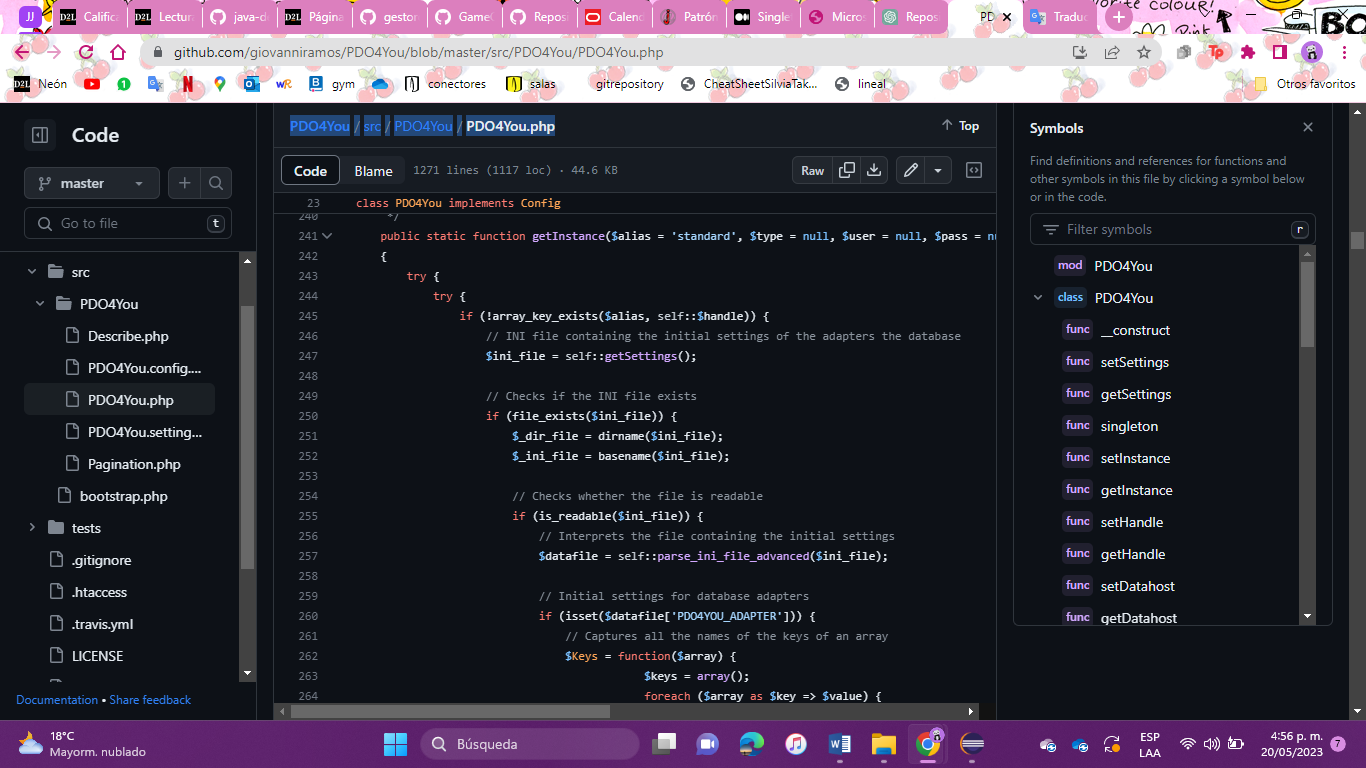
El proyecto implementa PDO, una extensión de PHP, para la creación de “código portátil para atender las bases de datos más populares”. El código proporciona facilidad para el acceso de los datos, sin importar la base de datos que se use, ya sea para consultar u obtener datos. El repositorio utiliza Singleton para “optimizar la conexión, asegurando una sola instancia del objeto de conexión”. Este patrón permite abstracción de conexión, protección contra inyección SQL, múltiples conexiones de base de datos, manejo de errores con seguimiento de pila, entre otros.

El patrón Singleton limita la creación de objetos pertenecientes a una clase para garantizar que solo exista una instancia u objeto de una clase para que esta pueda proporcionar un punto de acceso global a la misma. De hecho, el patrón es muy común en la conexión de bases de datos como es el caso actual. También permite una persistencia de datos, como lo implementamos en nuestro proyecto a lo largo del semestre, ya que al mantener una única instancia permitía que sus atributos no se reiniciaran y fueran constante a lo largo de la ejecución del programa.

En este caso, la instancia se crea en la siguiente dirección:

[PDO4You](https://github.com/giovanniramos/PDO4You/tree/master)/[src](https://github.com/giovanniramos/PDO4You/tree/master/src)/[PDO4You](https://github.com/giovanniramos/PDO4You/tree/master/src/PDO4You)

Solo agregare un pantallazo ya que el código donde se implementa es extenso.



Se evidencia como se crea la instancia de la clase, y se genera el getter, para que únicamente cuando se necesite se llame y genere la conexión necesaria sin reiniciar datos. Por otro lado, se ve como en otras clases, como en la clase Describe, se utiliza dicha la instancia y así obtener solo la conexión necesaria.

Se evidencia que se pone en privado el constructor y el atributo de la instancia es estático. Con esta estrategia le damos a la clase el control de sus instancias y asimismo permitimos un acceso global a esta desde cualquier parte de la aplicación con el getter que es a su vez, estático.

Además de lo anteriormente mencionado, el patrón Singleton es útil para que una clase determinada tenga un único objeto; la clase solo tiene instancias por lo que no se reservan nombre para las variables globales. Al usar Singleton, se garantiza un control al acceso de la instancia, y que esta sea única

El patrón tiene bastantes desventajas. Según el blog (<https://medium.com/dise%C3%B1o-de-software/singleton-el-patr%C3%B3n-del-mal-f3fdab0e16a2> ), afirma que las principales desventajas son que genera más acoplamiento ya que la referencia al Singleton no puede ser cambiada según el ambiente, además que impide realizar cambios en la clase ya que afectarían directamente a gran parte del proyecto. Nos obliga a invocar una clase que se acoplara al usar la instancia creada, generando una relación de dependencia que, al intentar romperla, nuevamente tendría consecuencias en todo el proyecto. Además, hace que las pruebas unitarias sean complicadas y casi imposibles ya que si hay un único objeto de una clase que se utiliza en diversas clases, genera problemas a la hora de realizar pruebas, ya que como no se tiene control de las instancias. Por ultimo gasta mucha memoria, ya que todo lo que sea realizado o llamado a dicha instancia creada por un Singleton siempre será utilizada, porque será la única al que llamen, variables y elementos basura, no serán eliminados.

Otra forma para solucionar la persistencia de datos de una clase en lugar de utilizar Singleton, puede ser generar archivos donde se almacene la persistencia de datos. También crear un controlador, donde se almacene constantemente la confirmación requerida a lo largo del proyecto, disminuyendo así el acoplamiento.

El proyecto contiene pocas clases, no obstante, en su implementación prima la instancia creada por el Singleton, como se muestra en el diagrama UML:

