Assignment: Task 1: Refactoring to Adhere to SOLID Principles

Course: Software Refactoring
Student: Ming Xin
Score: 3.0
Account: ECE

Leonardo da Silva Sousa: The refactoring for STEP 5 is incomplete; thus, the violation of the Dependency Inversion Principle persists (-1). As the README outlines, it's necessary to 'use dependency injection to remove the violation.'

The UserService should receive a reference to the DatabaseDriver instead of hardcoding a specific implementation (like PostgresDriver). In this way, UserService (an abstraction) still relies on a concrete implementation. Therefore, it still violates the DIP. To complete the refactoring, the UserService constructor should have DatabaseDriver as a parameter. This refactoring offers several benefits that enhance flexibility, maintainability, testability, and adherence to design principles. For example, the DI promotes decoupling between components by ensuring that UserService depends on an abstraction (DatabaseDriver) rather than a concrete implementation (PostgresDriver), which is what we expect from a code that adheres to the dependency inversion principle.

While your refactoring is almost complete, you must create a subpackage as requested in the Readme and only move the concrete implementation of the drivers to it (-1). The instruction was to 'Implement the PostgresDriver Class by extending the new interface and moving it to a newly created drivers subpackage.' This would have helped isolate the concrete implementations (PostgresDriver) from the contract (DatabaseDriver). There are several benefits to that:

* Separation of Concerns: Placing the interface in a higher-level package (persistence) and its concrete implementation in a subpackage (persistence.drivers) clearly separates the contract (interface) from the implementation details.

* Encapsulation and Modularity: By isolating the implementation in a subpackage, you can hide the implementation details and only expose the interface.

* Improved Code Organization: Keeping interfaces and implementations in separate packages improves the overall organization of your code. It lets developers quickly locate all interfaces and understand the application's architecture by reviewing the higher-level packages.

* Facilitates Dependency Injection: When interfaces are kept separate from implementations, it is easier to use Dependency Injection (DI) frameworks (like Spring or CDI) to manage dependencies. (Sep 15, 2024 at 3:39pm)