# Assignment: Task 1: Refactoring to Adhere to SOLID Principles

Course: Software Refactoring
Student: Yang Liu
Score: 2.5
Account: ECE

**Leonardo da Silva Sousa**: While your refactoring is correct, it is not complete: you must create a subpackage as requested in the README and separate contract/interface from implementation (-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.

The refactoring for STEP 5 is also incomplete; thus, the violation of the Dependency Inversion Principle persists (-1.5). As the README outlines, it's necessary to 'use dependency injection to remove the violation.'

It would help if you had used Dependency Injection (DI) to provide UserService with a reference to DatabaseDriver through its constructor, rather than hardcoding a specific implementation (like PostgresDriver). 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.

The incomplete refactored version also violates the Single Responsibility principle (SRP). When UserService is responsible for creating its own dependencies, it violates the SRP because it takes on the additional responsibility of knowing how to instantiate a DatabaseDriver. With DI, UserService is only responsible for its primary function (e.g., managing user-related operations) and does not concern itself with the details of creating or managing its dependencies.

P.S. Next time, submit the entire Java Project, not only the src folder. So I can compile and run your project! (Sep 15, 2024 at 3:37pm)