Dakota Carter-Prestage

COMP – 3081

Midterm Project Solid Analysis

The Package Delivery Tracking System demonstrates strong adherence to the SOLID principles of object-oriented design. Each class and interface in the system has been structured to promote modularity, reusability, and easy maintenance, while ensuring that future extensions can be made with minimal changes to existing code.

The Single Responsibility Principle states that each class should have one reason to change — in other words, it should perform only one well-defined role. This project follows SRP by ensuring that each class handles a distinct concern. The Package and its subclasses (StandardPackage, ExpressPackage, and OvernightPackage) are responsible solely for calculating and describing package costs. The TrackingEvent class exists only to represent a single immutable record of shipment progress, while the Shipment class manages tracking and lifecycle logic, delegating cost calculations and notifications to other components. Each notifier (ConsoleNotifier, SMSNotifier, and EmailNotifier) focuses purely on how updates are communicated. This clear division of responsibility ensures that changes to one part of the system do not risk introducing bugs elsewhere.

The Open/Closed Principle emphasizes that classes should be open for extension but closed for modification. In this project, the design allows for easy extension without altering core components. For example, new package types or notifier classes can be introduced without modifying the existing Shipment class or other base structures. Because Shipment relies on the Notifier interface and the abstract Package class, it remains unaffected when new implementations are added. Through polymorphism and dependency injection, the system achieves flexibility and scalability without sacrificing stability in existing code.

The Liskov Substitution Principle requires that subclasses remain fully interchangeable with their parent class without altering program correctness. This system meets that standard through the design of the Package hierarchy. Each subclass — StandardPackage, ExpressPackage, and OvernightPackage — correctly implements the expected methods (cost and description) with behavior consistent with the abstract Package interface. Any of these subclasses can be passed to a Shipment object without breaking functionality or changing expected outcomes. This consistent behavior ensures that subclasses extend functionality safely and predictably.

The Interface Segregation Principle states that clients should not be forced to depend on interfaces they do not use. The project applies this principle by defining a single, lightweight Notifier interface containing only one required method: send\_update(shipment, event). Each notifier class implements only this method, avoiding unnecessary complexity. This ensures that classes depend only on what they need and that the notifier interface remains clear, focused, and easy to maintain. The design avoids “fat” interfaces that could require implementing unrelated methods.

The Dependency Inversion Principle dictates that high-level modules should depend on abstractions rather than concrete implementations. This project applies DIP by having Shipment depend on the Notifier abstraction instead of a specific notification type. Through dependency injection, any notifier can be passed to a Shipment instance, whether it is console-based, SMS, or email. This inversion of control decouples high-level logic from low-level implementation details, making the system more flexible and easier to test. New notifier types can be integrated seamlessly without requiring modifications to the Shipment class.