S:

My code follows the Single Responsibility Principle because each class has one clear responsibility. In my code, the Package class handles the shipping cost logic, the TrackingEvent class represents the immutable tracking data, the Shipment class coordinates the package's status updates, and the notifiers manage communication. There are no classes in my code where they mix unrelated concerns. An example of mixing unrelated concerns would be if Shipment printed messages rather than the notifiers. Therefore, my code keeps responsibilities separated.

O:

My code follows the Open/ Closed Principle because it allows extension without modification. In my code adding a new shipping type or notifier wouldn't break the original code. They can just be added using the same interface. None of my existing classes need to be changed in order to support addition into the code. Therefore, my code is open for extension but closed for modification.

L:

My code follows the Liskov Substitution Principle because my subclasses can be used anywhere the main class is expected. Each subclass of Package (Standard, Express, and Overnight) can use the cost(method consistently without altering the expected behavior. Therefore, any subclass can substitute for the base Package without breaking the program.

I:

My code follows the Interface Segregation Principle because my Notifier interface is very minimal and focused. The Notifier class only defines send_update. Each of the notifiers in the interface implements exactly what it needs to do. Therefore, the interface is lightweight and avoids forcing any notifier to implement unnecessary methods.

D:

My code follows the Dependency Inversion Principle because shipment depends on Notifier (abstraction). It receives the notifier by dependency injection in its constructor. This allows shipment to be able to use any object that implements send_update. Therefore, my code is flexible and decoupling high-levl business logic from details like printing/ messaging.