Joseph Gareri

jgareri@my.athens.edu

00099183

January 10, 2024

CS 418

Assignment 01

1.) S.O.L.I.D

a.) S = Single Responsibility Principle. "A class should have one and only one reason to change, meaning that a class should have only one job." This would be connected to the UNIX principle 2: "one tool to do one task;" because, it should have only one responsibility.

b.) O = Open Closed Principle. "Objects or entities should be open for extension but closed for modification." This would be connected to UNIX principle 7: "Modular, designed to be repurposed by others;" because, UNIX is designed to be repurposed by others, but not be designed for modification.

c.) L = Liskov Substitution Principle. "The Liskov Substitution Principle states that derived classes must be able to completely substitute for their base class." This does not correspond to any UNIX principles.

d.) I = Interface Segregation Principle. "The ISP suggests splitting up interfaces so that implementer can pick and choose depending upon need." This would be connected to UNIX principle 8: "Provide the mechanism, not the policy," because the users of an operating system just require a mechanism, and not the policy.

e.) D = Dependency Inversion Principle. "Abstractions should not depend upon details and details should depend on abstractions." This could potentially be connected to

UNIX principle 4: "Combine tools seamlessly;" because this would allow tools to work together without much dependency.

2.) The LSP explains that derived class must be able to completely substitute for their base class. The substitution: `class B: public A {:` is correct; however, class B is only written to throw an exception. Thus, this causes a malfunction and is not a proper substitution.

3.) This is an example of LSP, because the function `def aFancyHelperMethod(a)` is successfully substituted into the function `def aFancyMethod(a).` However, the issue is not with substitution, but with errors in design. This will cause errors; therefore, this fails LSP. For example, if one of the objects is missing then it will cause an error.

4.) a.) ISP states splitting up interfaces so that implementer can pick and choose depending upon need. Currently, the way the class is set up triggers orderBurger, orderFries, and orderCombo, but a person might not want to order all three. Thus, this fails ISP.

b.) To split up the class OrderService:

```
class BurgerOrder {
      virtual void orderBurger(int quantity) = 0; };
class FriesOrder {
      virtual void orderFries(int quantity) = 0; };
class ComboOrder {
      virtual void orderCombo(int quantity, int fries) = 0;
      };
```

c.) Even though the original code failed ISP, the code was more cohesive because the definition of cohesion refers to the degree to which elements inside a module belong together. Thus, the new code has bad cohesion because they all refer to the same action but have separate instances. In comparison to coupling which is the degree of interdependence between software modules; a measure of how closely connected two routines or modules are. This new code has bad coupling because it's now a higher degree since it has 3 separate instances.

5.) a.) This code shows ISP, however, as the code above it does not show good cohesion and coupling. I would mitigate this by combining the Café and CreditCard class together.

b.)