**Class:** ENSF 607

**Assignment:** Lab #4

**By:** Yajur Vashisht, Balkarn Gill, Satchytan ASJDENBFSJNBFSAJNBF, Momin Muhammad

**Due Date:** October 27th, 2023

Exercise One:

SOLID Principles are:

• Single Responsibility

• Open/Close

• Liskov Substitution

• Interface Segregation

• Dependency Inversion

For our project we have went with an E-commerce Product Management software in Java. We will be providing class diagrams, code, and appropriate use case for all the exercises in relation to our Product Management software theme.

A screenshot of a computer

Description automatically generated

|  |  |
| --- | --- |
| **Figure 1:** | Single Responsibility Principle Class Diagram |

A diagram of a product

Description automatically generated

|  |  |
| --- | --- |
| **Figure 2:** | Open/Close Principle Class Diagram |

**A screenshot of a diagram

Description automatically generated**

|  |  |
| --- | --- |
| **Figure 3:** | Liskov Substitution Principle Class Diagram |

A screenshot of a computer

Description automatically generated

|  |  |
| --- | --- |
| **Figure 4:** | Interface Segregation Principle Class Diagram |

A diagram of a product

Description automatically generated

|  |  |
| --- | --- |
| **Figure 5:** | Dependency Inversion Principle Class Diagram |

Exercise Two:

Exercise Three:

**Single Responsibility:**

The single responsibility principle is demonstrated in the Figure 1 because both classes product and product catalog are responsible for their specific tasks. Product only receives the name of the product and the price of the product as a string and double respectively whereas product catalog allows you to add and remove a product because products is an array list.

**Open/Close:**

Figure 2 for the open closed principle demonstrates that we should keep as much of the existing code unchanged as possible when implementing new classes and new functionality we should add them with minimum changes.

**Liskov Substitution:**

The provided plant UML diagram in Figure 3 for the Liskov substitution principle illustrates that class hierarchies are created during application development. Classes are extended and derived in this process. In the e-commerce platform example shown, the subtypes can be substitutable for their base types.

**Interface Segregation:**

To demonstrate the interface segregation principle, Figure 4 uses an interface called displayable that will display the products, cart, and sale products that their customers have chosen. We have avoided using fat interfaces by limiting the number of methods the actual class needs. The interface has been broken out to satisfy client needs exactly.

**Dependency Inversion:**

The dependency inversion principle is displayed in Figure 5. The figure displays the change of low level implementations as well as the use of abstraction via interfaces and abstract classes. The interface product allows us to achieve loose coupling.