

**Usman Institute of Technology**

**Department of Computer Science**

**Course Code: SE308**

**Course Title: Software Design and Architecture**

# Summer 2024

**Lab 05**

**OBJECTIVE: To Understand and Implement the SOLID Design Principles**

**Student Information**

|  |  |
| --- | --- |
| Student Name | **Jawwad Bhatti** |
| Student ID | **20B-011-SE** |
| Date | **22-07-2024** |

**Assessment**

|  |  |
| --- | --- |
| Marks Obtained |  |
| Remarks |  |
| Signature |  |

|  |  |
| --- | --- |
| **Software Design Principle (SOLID)**  Several principles have been identified throughout the literature that help in making component-level design decisions, including (SOLID). These design principles intended to make object-oriented designs more understandable, flexible, and maintainable. The principles are a subset of many principles promoted by American software engineer and instructor Robert C. Martin [1] also called "Uncle Bob".   1. Single Responsibility Principle 2. The open–closed principle (OCP) 3. The Liskov substitution principle (LSP) 4. The interface segregation principle (ISP) 5. Dependency Inversion Principle   ***1. Single Responsibility Principle***  The single responsibility principle (SRP) states that every class, method, and function should have only one job or one reason to change.     |  | | --- | | *Game.py --* ***SRP Violation***    *Class Game : def play (self) def saveScore (self) def hardLevel (self)* |   Consider any game application that do gaming activities as well as to track the scores, so the Game class kept the responsibility to keep game playing activities, and the Scorer class got the responsibility to calculate the score and keep all score as stats section.     * The Three functions, it declares are certainly functions belonging to a game. * The *play* and *hardlevel* functions   manage the playing activities   * while the *saveScore* keep all score as stats     **Rectification :** To make the Game class conforms to the single responsibility principle, you’ll need to create another class that is in charge of storing game scores in database    **Exercise 1:** Use the single responsibility principle to separate classes, methods, implement your own scenario.            [1] Martin, Robert C. (2000). "Design Principles and Design Patterns" (PDF). Retrieved 2022-11-10 http://www.objectmentor.com/resources/articles/Principles\_and\_Patterns.pdf |

|  |  |  |
| --- | --- | --- |
| 1. ***The open*–*closed principle (OCP)***   The open-closed principle states that a class, method, and function should be open for extension but closed for modification.   * + Consider the four functions, it declares certainly functions belonging to a ITEmployee.   + What will you do when new types of Employees come?   + This if-elif chain eventually will become hell and very hard to maintain.   + You can create a new class as a subclass of the Employee     Appy SPR first then follow the rectification instruction     |  | | --- | | *Emp.py --* ***OCP Violation***    *Class ITEmployee : def work (self) def Analysis (self) def develope (self) def test (self)* |   **Rectification:** Now Employee is an abstract class and it has an abstract method called work. All subclasses of this class have to implement a work function. Developer calls its develop method and Tester calls its test method. In the company, all we had to do is calling the work() method of the given employee. If I need to add a new Employee like Maintenance  Engineer, all you need to do is implement the work() method    **Exercise 2:** Use the Open/Close principle implement your own scenario.       1. ***The Liskov substitution principle (LSP***   The Liskov substitution principle states that a child class must be substitutable for its parent class. Liskov substitution principle aims to ensure that the child class can assume the place of its parent class without causing any errors.  This principle was coined by Barbar Liskov [1] in her work regarding data abstraction and type theory. It also derives from the concept of Design by Contract (DBC) by Bertrand Meyer [1].   * + - Consider the four Class  |  | | --- | | class Employee(ABC): def save (self):    class Students (Employee): def save (self)    class Teacher (Employee) def save (self)    class Manager (Employee) def save (self) |      * + - Let’s modify the code and add one more abstract method salary() in each class.      * + - Assume that students did not get paid      * + - Student class will throw exceptions or not work as expected. |

|  |
| --- |
| **Rectification :** Remove the salary() method from Student and create a new abstract class Payment    **Exercise 3:** Use the Liskov Substitute principle and implement it with your own scenario.       1. ***The interface segregation principle (ISP***   The interface segregation principle states that an interface should be as small a possible in terms of cohesion. In other words, it should do ONE thing   1. ***Dependency Inversion Principle***   The dependency inversion principle states that, high-level modules should not depend on lowlevel modules. Both should depend on abstractions. Abstractions should not depend on details. Details should depend on abstractions        **Exercise 4:** Use the Interface Sagregation principle and implement it with the given scenario  [(Web Resource)](https://www.pythontutorial.net/python-oop/python-interface-segregation-principle/).          **Exercise 5:** Use the Dependency Inversion principle and implement it with given scenario  [(Web Resource)](https://www.pythontutorial.net/python-oop/python-dependency-inversion-principle/). |