**Laboratory Report: Implementation of SOLID Principles in Python**

**Objective:**

The objective of this laboratory work is to implement two principles from the **SOLID** design principles in Python. Specifically, we will focus on the **Single Responsibility Principle (SRP)** and the **Open/Closed Principle (OCP)**. The aim is to create a simple project for managing orders, calculating costs, and generating reports while adhering to these principles.

**Introduction to SOLID Principles:**

The **SOLID** principles are a set of design guidelines in object-oriented programming that promote more maintainable, scalable, and flexible code.

* **S**: **Single Responsibility Principle (SRP)** — A class should have only one reason to change, i.e., it should have only one job.
* **O**: **Open/Closed Principle (OCP)** — A class should be open for extension but closed for modification. This ensures that new functionality can be added without altering existing code.

For this lab, we will implement these principles using Python classes to handle a store's orders and reports.

**Implementation**

**Classes Implemented:**

1. **Order**: This class holds the order details such as items, quantities, and prices. It follows **SRP** by focusing solely on storing order-related data.
2. **CostCalculator**: This is an abstract base class that handles cost calculation. It adheres to **OCP** by allowing extension via different cost strategies, such as regular cost calculation or discounted cost calculation.
3. **ReportGenerator**: This is an abstract class responsible for generating reports in various formats (e.g., text or HTML). It follows **SRP** as it only generates reports and does not handle any other business logic.

**Code Implementation**

from abc import ABC, abstractmethod

# Single Responsibility Principle (SRP): Order class only stores order details.

class Order:

def \_\_init\_\_(self, order\_id, items):

self.order\_id = order\_id

self.items = items # List of tuples (item\_name, quantity, price\_per\_item)

# Open/Closed Principle (OCP): Abstract class for cost calculation.

class CostCalculator(ABC):

@abstractmethod

def calculate\_total(self, order):

pass

# Regular cost calculation (extension).

class RegularCostCalculator(CostCalculator):

def calculate\_total(self, order):

total = 0

for item in order.items:

total += item[1] \* item[2] # quantity \* price\_per\_item

return total

# Discounted cost calculation (extension).

class DiscountedCostCalculator(CostCalculator):

def \_\_init\_\_(self, discount\_percentage):

self.discount\_percentage = discount\_percentage

def calculate\_total(self, order):

total = 0

for item in order.items:

total += item[1] \* item[2]

discount = total \* self.discount\_percentage / 100

return total - discount

# Single Responsibility Principle (SRP): Abstract class for generating reports.

class ReportGenerator(ABC):

@abstractmethod

def generate\_report(self, order, total\_cost):

pass

# Text report generation (extension).

class TextReportGenerator(ReportGenerator):

def generate\_report(self, order, total\_cost):

report = f"Order ID: {order.order\_id}\nItems:\n"

for item in order.items:

report += f"{item[0]} - {item[1]} @ {item[2]} each\n"

report += f"Total Cost: {total\_cost}\n"

return report

# HTML report generation (extension).

class HTMLReportGenerator(ReportGenerator):

def generate\_report(self, order, total\_cost):

report = f"<h1>Order ID: {order.order\_id}</h1>\n<ul>\n"

for item in order.items:

report += f"<li>{item[0]} - {item[1]} @ {item[2]} each</li>\n"

report += f"</ul>\n<p>Total Cost: {total\_cost}</p>\n"

return report

# Example usage

order = Order(1, [("Laptop", 1, 1200), ("Mouse", 2, 50)])

# Regular cost calculator

regular\_calculator = RegularCostCalculator()

total\_cost = regular\_calculator.calculate\_total(order)

text\_report\_generator = TextReportGenerator()

print(text\_report\_generator.generate\_report(order, total\_cost))

# Discounted cost calculator (10% discount)

discounted\_calculator = DiscountedCostCalculator(10)

discounted\_total\_cost = discounted\_calculator.calculate\_total(order)

html\_report\_generator = HTMLReportGenerator()

print(html\_report\_generator.generate\_report(order, discounted\_total\_cost))

**Principles Demonstrated:**

**1. Single Responsibility Principle (SRP):**

* **Order Class**: The Order class focuses solely on managing order-related data such as items and their quantities. It is not responsible for calculating the cost or generating reports.
* **ReportGenerator** Class: The responsibility of generating reports in different formats is separated into the ReportGenerator class, with specific report formats handled by subclasses (TextReportGenerator and HTMLReportGenerator).

**2. Open/Closed Principle (OCP):**

* **CostCalculator** Class: The CostCalculator class is designed to allow extension for different calculation strategies (e.g., regular and discounted calculations) without modifying the base class. New calculation strategies can be added as separate subclasses.
* **ReportGenerator** Class: This class is extendable for different report formats. By adding new subclasses, such as HTMLReportGenerator, new formats can be generated without changing existing logic.

**Conclusion:**

In this laboratory work, we demonstrated the **Single Responsibility Principle (SRP)** and the **Open/Closed Principle (OCP)** through a simple order management system in Python. By adhering to **SRP**, each class was focused on a single responsibility, promoting clarity and maintainability. Using **OCP**, we extended the system with new functionality (different cost calculators and report generators) without modifying the existing codebase. This approach enhances scalability and flexibility, which are key aspects of the SOLID principles.

**References:**

* **SOLID Principles**: Robert C. Martin, *Clean Code: A Handbook of Agile Software Craftsmanship*.