Wenzhou Kean University

SCHOOL OF COMPUTER SCIENCE AND TECHNOLOGY

CPS 5301 Advanced Software Engineering
Assignment 2

▪ Objective

This assignment aims to help you understand common design issues in object-oriented systems and practice refactoring those designs using appropriate design patterns. You will be given a set of classes with suboptimal designs. Your task is to identify the design flaws, explain why they are problematic, and then apply suitable design patterns to improve the system's flexibility, maintainability, and readability.

▪ Assignment Instructions

Analyze the Existing Design: You will be given a set of classes that exhibit poor design practices, such as improper coupling, excessive dependency, lack of scalability, and unclear responsibilities.

▪ Assignment Requirements

You are required to:

1. Identify the design flaws in the provided system.
2. Provide a clear explanation of the design problems you found.
3. Refactor the Design Using Design Patterns: Based on your analysis, apply suitable object-oriented design patterns to resolve the issues.
4. After refactoring, ensure the modified design adheres to the SOLID principles (Single Responsibility, Open/Closed, Liskov Substitution, Interface Segregation, Dependency Inversion).
5. Describe the design pattern(s) you applied and why they were appropriate for resolving the issues
6. No handwriting or hand drawing is accepted. Use Papyrus to draw your new designs.
7. Submit your solution in PDF format by the specified date on Canvas.

The design patterns you may consider include (but are not limited to):

- Singleton Pattern (to ensure a class has only one instance and provides a global point of access)
- Factory Method (to create objects without specifying the exact class of object that will be created)
- Strategy Pattern (to define a family of algorithms, encapsulate each one, and make them interchangeable)

- Observer Pattern (to allow a subject to notify observers about state changes)
- Decorator Pattern (to attach additional responsibilities to an object dynamically)
- Adapter Pattern (to convert one interface to another expected by the client)
- Composite Pattern (to compose objects into tree-like structures to represent part-whole hierarchies)
- Command Pattern (to turn a request into a stand-alone object that contains all information about the request)

To Know more about design patterns, follow this link (https://www.oodesign.com/)

- **Grading Rubric**
  1. Analysis (30%): Thorough identification of design issues and their impact on maintainability and scalability.
  2. Design Patterns Application (70%): Correct and appropriate usage of design patterns, and clear justification for their use.

- **Classes Implementation**

class Product:

```
def __init__(self, name, price):
    self.name = name
    self.price = price
```

class Order:

```
def __init__(self):
    self.products = []
def add_product(self, product):
    self.products.append(product)
def calculate_total(self):
    total = 0
    for product in self.products:
        total += product.price
    return total
```

class Customer:

```
def __init__(self, name):
```

```
    self.name = name
    self.orders = []
def place_order(self, order):
    self.orders.append(order)
```

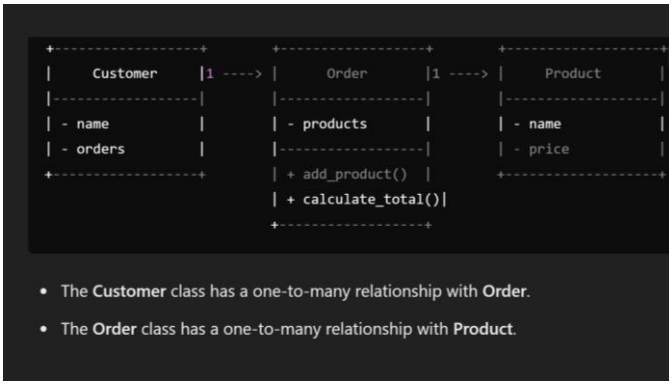

Figure 1: Classes and relationships