Отчёт  
по лабораторной работе №2  
по дисциплине «МДК 01.01 Разработка программного модуля»  
Тема: «Структурные паттерны»

|  |  |  |
| --- | --- | --- |
| Студент группы |  | Липинский К.С. |
| Преподаватель |  | Ремизова В.И. |

## Цель работы

Изучить шаблоны проектирования, относящиеся к классу порождающих, освоить применение шаблонов этого класса при разработке программных систем с применением объектно-ориентированных языков программирования.

## Выполнение работы

**Вариант 1 (1.2)**

Код

import json  
import os  
  
## Установите пакет googletrans через pip, если он не установлен:  
# pip install googletrans==4.0.0-rc1  
from googletrans import Translator  
  
class TextTransformer:  
 *"""  
 Базовый класс для преобразования текста.  
 """* def \_\_init\_\_(self, name):  
 self.name = name  
  
 def transform(self, text):  
 raise NotImplementedError("Этот метод должен быть переопределен в дочернем классе.")  
  
 def reverse\_transform(self, text):  
 raise NotImplementedError("Этот метод должен быть переопределен в дочернем классе.")  
  
  
class TranslationTransformer(TextTransformer):  
 *"""  
 Класс для перевода текста.  
 """* def \_\_init\_\_(self, dest\_lang='en'):  
 super().\_\_init\_\_("translate")  
 self.translator = Translator()  
 self.dest\_lang = dest\_lang  
 self.src\_lang = None  
  
 def transform(self, text):  
 *"""Переводит текст по словам, сохраняя исходный язык для обратного преобразования."""* try:  
 words = text.split()  
 translated\_words = []  
 for word in words:  
 # Определяем исходный язык первого слова, чтобы использовать его для обратного перевода  
 if self.src\_lang is None:  
 detected = self.translator.detect(word)  
 self.src\_lang = detected.lang  
 translated = self.translator.translate(word, dest=self.dest\_lang)  
 translated\_words.append(translated.text)  
 return ' '.join(translated\_words)  
 except Exception as e:  
 print(f"Ошибка при переводе: {e}")  
 return text  
  
 def reverse\_transform(self, text):  
 *"""Возвращает переведенный текст обратно на исходный язык."""* if self.src\_lang is None:  
 print("Не удалось определить исходный язык для обратного перевода.")  
 return text  
 try:  
 words = text.split()  
 original\_words = []  
 for word in words:  
 translated = self.translator.translate(word, dest=self.src\_lang)  
 original\_words.append(translated.text)  
 return ' '.join(original\_words)  
 except Exception as e:  
 print(f"Ошибка при обратном переводе: {e}")  
 return text  
  
 def to\_json(self):  
 return {"name": self.name, "dest\_lang": self.dest\_lang, "src\_lang": self.src\_lang}  
  
 @staticmethod  
 def from\_json(data):  
 transformer = TranslationTransformer(data.get("dest\_lang"))  
 transformer.src\_lang = data.get("src\_lang")  
 return transformer  
  
  
class PermutationTransformer(TextTransformer):  
 *"""  
 Класс для шифрования методом перестановки слов по заданной схеме.  
 """* def \_\_init\_\_(self, scheme=None):  
 super().\_\_init\_\_("encrypt")  
 self.scheme = scheme  
 self.inverse\_scheme = None  
  
 def transform(self, text):  
 *"""Шифрует текст, переставляя слова по заданной схеме."""* words = text.split()  
   
 # Если схема не задана, создаем ее на основе длины текста  
 if not self.scheme:  
 self.scheme = list(range(len(words)))  
 import random  
 random.shuffle(self.scheme)  
   
 # Проверяем, соответствует ли схема длине текста  
 if len(self.scheme) != len(words):  
 print("Схема перестановки не соответствует количеству слов. Пропускаем шифрование.")  
 return text  
   
 # Генерируем обратную схему, если ее нет  
 if not self.inverse\_scheme:  
 self.inverse\_scheme = [0] \* len(self.scheme)  
 for i, j in enumerate(self.scheme):  
 self.inverse\_scheme[j] = i  
  
 encrypted\_words = [None] \* len(words)  
 for i, j in enumerate(self.scheme):  
 encrypted\_words[j] = words[i]  
   
 return ' '.join(encrypted\_words)  
  
 def reverse\_transform(self, text):  
 *"""Расшифровывает текст, используя обратную схему."""* if not self.inverse\_scheme:  
 print("Обратная схема не найдена. Невозможно расшифровать.")  
 return text  
  
 words = text.split()  
 if len(self.inverse\_scheme) != len(words):  
 print("Схема не соответствует количеству слов в тексте. Пропускаем расшифровку.")  
 return text  
  
 decrypted\_words = [None] \* len(words)  
 for i, j in enumerate(self.inverse\_scheme):  
 decrypted\_words[j] = words[i]  
  
 return ' '.join(decrypted\_words)  
   
 def to\_json(self):  
 return {"name": self.name, "scheme": self.scheme, "inverse\_scheme": self.inverse\_scheme}  
  
 @staticmethod  
 def from\_json(data):  
 transformer = PermutationTransformer(data.get("scheme"))  
 transformer.inverse\_scheme = data.get("inverse\_scheme")  
 return transformer  
  
class TextFileProcessor:  
 *"""  
 Класс-тестовое приложение для записи и чтения данных с преобразованиями.  
 """* def \_\_init\_\_(self, file\_path):  
 self.file\_path = file\_path  
 self.transformer\_map = {  
 "translate": TranslationTransformer,  
 "encrypt": PermutationTransformer  
 }  
  
 def write\_with\_transformations(self, text, transformations\_list):  
 *"""Записывает текст в файл, применяя заданную последовательность преобразований."""* print(f"Исходный текст: {text}")  
   
 operations\_data = []  
 current\_text = text  
 for op\_name in transformations\_list:  
 if op\_name not in self.transformer\_map:  
 print(f"Неизвестное преобразование: {op\_name}. Пропускаем.")  
 continue  
  
 # Создаем экземпляр класса для преобразования  
 transformer = self.transformer\_map[op\_name]()  
 current\_text = transformer.transform(current\_text)  
 operations\_data.append(transformer.to\_json())  
 print(f"Текст после '{op\_name}': {current\_text}")  
  
 # Формируем структуру данных для сохранения в файл  
 file\_content = {  
 "data": current\_text,  
 "operations": operations\_data  
 }  
  
 with open(self.file\_path, 'w', encoding='utf-8') as f:  
 json.dump(file\_content, f, ensure\_ascii=False, indent=2)  
  
 print(f"\nДанные успешно записаны в файл: {self.file\_path}")  
  
 def read\_with\_transformations(self):  
 *"""Считывает данные из файла и применяет обратные преобразования."""* if not os.path.exists(self.file\_path):  
 print(f"Файл не найден: {self.file\_path}")  
 return None  
  
 with open(self.file\_path, 'r', encoding='utf-8') as f:  
 file\_content = json.load(f)  
  
 encoded\_text = file\_content.get("data")  
 operations\_data = file\_content.get("operations")  
  
 print(f"\nСчитываем данные из файла: {self.file\_path}")  
 print(f"Закодированный текст: {encoded\_text}")  
 print(f"Последовательность операций: {[op['name'] for op in operations\_data]}")  
   
 # Применяем обратные преобразования в обратном порядке  
 current\_text = encoded\_text  
 for op\_data in reversed(operations\_data):  
 op\_name = op\_data["name"]  
   
 if op\_name == "translate":  
 transformer = TranslationTransformer.from\_json(op\_data)  
 elif op\_name == "encrypt":  
 transformer = PermutationTransformer.from\_json(op\_data)  
 else:  
 print(f"Неизвестная операция в файле: {op\_name}. Невозможно расшифровать.")  
 continue  
  
 current\_text = transformer.reverse\_transform(current\_text)  
 print(f"Текст после обратного '{op\_name}': {current\_text}")  
  
 print("\nОбработка завершена.")  
 return current\_text  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 file\_path = "output.json"  
 initial\_text = "Пример текста для демонстрации работы программы с разными преобразованиями."  
   
 # Создаем экземпляр тестового приложения  
 processor = TextFileProcessor(file\_path)  
  
 # Задаем последовательность преобразований (сначала шифрование, потом перевод)  
 transformations\_to\_apply = ["encrypt", "translate"]  
   
 # 1. Запись в файл с преобразованиями  
 processor.write\_with\_transformations(initial\_text, transformations\_to\_apply)  
  
 # 2. Чтение из файла с обратными преобразованиями  
 decoded\_text = processor.read\_with\_transformations()  
  
 print(f"\nКонечный результат (восстановленный текст): {decoded\_text}")  
 print(f"Проверка: текст полностью восстановлен? {initial\_text == decoded\_text}")

Результат (рис. 1)

Изображение выглядит как текст, снимок экрана, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 1

## Контрольные вопросы

1. Назначение структурных шаблонов проектирования.

Структурные шаблоны проектирования — это группа шаблонов, которые помогают формировать сложные структуры из более мелких объектов. Они определяют, как объекты и классы могут быть скомпонованы в более крупные структуры, чтобы создавать новые функции, сохраняя при этом гибкость и эффективность. Основная цель — обеспечить совместную работу компонентов без ущерба для их целостности.

**Перечень структурных шаблонов:**

* Адаптер (Adapter)
* Мост (Bridge)
* Компоновщик (Composite)
* Декоратор (Decorator)
* Фасад (Facade)
* Приспособленец (Flyweight)
* Заместитель (Proxy)

1. Назначение и структура шаблона «адаптер» (Adapter/Wrapper).

Назначение: Адаптер позволяет объектам с несовместимыми интерфейсами работать вместе. Он преобразует интерфейс одного класса в интерфейс, который ожидают клиенты, выступая в роли посредника.

Структура: Шаблон включает три основных компонента:

* **Target (Цель):** Интерфейс, который клиент ожидает увидеть.
* **Adaptee (Адаптируемый):** Класс, который нужно адаптировать. У него есть несовместимый интерфейс.
* **Adapter (Адаптер):** Класс, который реализует интерфейс Target и содержит экземпляр Adaptee, делегируя ему вызовы.

**Преимущества:**

* Позволяет использовать существующие классы без изменения их кода.
* Повышает гибкость и повторное использование кода.
* Скрывает детали реализации адаптируемого класса от клиента.

**Недостатки:**

* Может усложнить код, так как добавляет новый класс.
* Иногда простая подклассификация может быть более подходящим решением.

1. Назначение и структура шаблона «мост» (Bridge).

Назначение: Мост разделяет большую иерархию классов на две отдельные иерархии — абстракции и реализации, которые можно развивать независимо. Это позволяет избежать «взрывного роста» классов.

Структура: Шаблон состоит из:

* **Abstraction (Абстракция):** Класс, который определяет интерфейс высокого уровня.
* **Refined Abstraction (Уточненная Абстракция):** Расширение Abstraction.
* **Implementor (Реализатор):** Интерфейс, который определяет низкоуровневые операции.
* **Concrete Implementor (Конкретный Реализатор):** Класс, который реализует Implementor.

**Преимущества:**

* Отделяет абстракцию от реализации, позволяя изменять их независимо.
* Повышает расширяемость системы.

**Недостатки:**

* Может усложнить код, так как добавляет много новых интерфейсов и классов.

1. Назначение и структура шаблона «компоновщик» (Composite).

Назначение: Компоновщик позволяет создавать иерархические структуры из объектов, чтобы работать с отдельными объектами и их группами одинаковым образом.

Структура: Ключевые элементы:

* **Component (Компонент):** Общий интерфейс для всех объектов в структуре.
* **Leaf (Лист):** Простой объект, не имеющий дочерних элементов.
* **Composite (Компоновщик):** Объект, который может содержать другие компоненты (как листья, так и другие компоновщики).

**Преимущества:**

* Упрощает код клиента, который может работать с иерархией единообразно.
* Позволяет легко добавлять новые типы компонентов.

**Недостатки:**

* Может усложнить код, если некоторые компоненты не поддерживают операции, определенные в общем интерфейсе.

1. Назначение и структура шаблона «декоратор» (Decorator).

Назначение: Декоратор позволяет динамически добавлять новые функции к объекту, «оборачивая» его в специальный объект-обёртку.

Структура: Элементы шаблона:

* **Component (Компонент):** Общий интерфейс для оборачиваемых объектов.
* **Concrete Component (Конкретный Компонент):** Объект, к которому добавляется новая функциональность.
* **Decorator (Декоратор):** Абстрактный класс, который содержит ссылку на объект Component и реализует его интерфейс.
* **Concrete Decorator (Конкретный Декоратор):** Класс, который добавляет новую функциональность перед или после вызова метода оборачиваемого объекта.

**Преимущества:**

* Более гибкая альтернатива наследованию для расширения функциональности.
* Позволяет добавлять и удалять функциональность во время выполнения.

**Недостатки:**

* Может привести к большому количеству маленьких классов.
* Усложняет отладку.

1. Назначение и структура шаблона «фасад» (Facade).

Назначение: Фасад предоставляет единый упрощенный интерфейс для сложной подсистемы. Он скрывает сложность и обеспечивает удобную точку входа.

Структура: Ключевые части:

* **Facade (Фасад):** Класс, который содержит методы для выполнения сложных операций и делегирует вызовы соответствующим объектам подсистемы.
* **Subsystem classes (Классы подсистемы):** Сложные классы, которые выполняют работу.

**Преимущества:**

* Упрощает использование сложной системы для клиента.
* Слабо связывает клиентский код с подсистемой.

**Недостатки:**

* Фасад не должен скрывать все классы подсистемы; это может ограничить функциональность.
* Может стать «Божественным объектом», если будет содержать слишком много функций.

1. Назначение и структура шаблона «приспособленец» (Flyweight).

Назначение: Приспособленец экономит память, позволяя использовать множество объектов с общими характеристиками, выделяя их в один общий объект.

Структура: Основные компоненты:

* **Flyweight (Приспособленец):** Интерфейс для объектов, которые могут быть разделены.
* **Concrete Flyweight (Конкретный Приспособленец):** Объект, который хранит общие данные.
* **Flyweight Factory (Фабрика Приспособленцев):** Создает и управляет объектами-приспособленцами.

**Преимущества:**

* Значительно снижает потребление памяти, если используется много похожих объектов.

**Недостатки:**

* Может усложнить код из-за разделения состояния на «внутреннее» (общее) и «внешнее» (уникальное).
* Эффективен только в специфических случаях, когда объекты имеют много общего.

1. Назначение и структура шаблона «заместитель» (Proxy).

Назначение: Заместитель предоставляет «заместительный» или «местозаполнительный» объект для другого объекта, чтобы контролировать доступ к нему.

Структура: Шаблон состоит из:

* **Subject (Субъект):** Общий интерфейс для реального объекта и его заместителя.
* **Real Subject (Реальный Субъект):** Объект, к которому осуществляется доступ.
* **Proxy (Заместитель):** Объект, который управляет доступом к Real Subject, выполняя дополнительные действия, такие как кэширование, ленивая инициализация или проверка прав доступа.

**Преимущества:**

* Обеспечивает контролируемый доступ к объекту.
* Позволяет добавить функциональность, не изменяя оригинальный класс.

**Недостатки:**

* Может немного замедлить работу из-за дополнительного слоя посредничества.