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

## Выполнение работы

**Вариант 1 (1.3)**

Код

from \_\_future\_\_ import annotations  
from typing import List, Optional  
  
class ВзлетноПосадочнаяПолоса:  
 def \_\_init\_\_(self, id: int):  
 self.id = id  
 self.занята = False  
  
 def \_\_repr\_\_(self) -> str:  
 return f"ВПП №{self.id}"  
  
class Терминал:  
 def \_\_init\_\_(self, id: int):  
 self.id = id  
 self.занят = False  
  
 def \_\_repr\_\_(self) -> str:  
 return f"Терминал №{self.id}"  
  
class АвиадиспетчерскаяВышка:  
 def \_\_init\_\_(self):  
 self.впп: List[ВзлетноПосадочнаяПолоса] = []  
 self.терминалы: List[Терминал] = []  
  
 def добавить\_впп(self, впп: ВзлетноПосадочнаяПолоса):  
 self.впп.append(впп)  
  
 def добавить\_терминал(self, терминал: Терминал):  
 self.терминалы.append(терминал)  
  
 def разрешить\_взлет(self, самолет: Самолет) -> bool:  
 доступная\_впп = next((п for п in self.впп if not п.занята), None)  
 if доступная\_впп:  
 доступная\_впп.занята = True  
 print(f"Вышка: Разрешаю взлет самолету {самолет.имя} с {доступная\_впп}.")  
 self.впп\_в\_использовании = доступная\_впп  
 return True  
 else:  
 print(f"Вышка: Отказываю в взлете самолету {самолет.имя}. Все ВПП заняты.")  
 return False  
  
 def завершить\_взлет(self, самолет: Самолет):  
 print(f"Вышка: Самолет {самолет.имя} завершил взлет.")  
 if hasattr(self, 'впп\_в\_использовании') and self.впп\_в\_использовании:  
 self.впп\_в\_использовании.занята = False  
 self.впп\_в\_использовании = None  
  
 def разрешить\_посадку(self, самолет: Самолет) -> bool:  
 доступная\_впп = next((п for п in self.впп if not п.занята), None)  
 if доступная\_впп:  
 доступная\_впп.занята = True  
 print(f"Вышка: Разрешаю посадку самолету {самолет.имя} на {доступная\_впп}.")  
 self.впп\_в\_использовании = доступная\_впп  
 return True  
 else:  
 print(f"Вышка: Отказываю в посадке самолету {самолет.имя}. Все ВПП заняты.")  
 return False  
  
 def завершить\_посадку(self, самолет: Самолет):  
 print(f"Вышка: Самолет {самолет.имя} завершил посадку.")  
 if hasattr(self, 'впп\_в\_использовании') and self.впп\_в\_использовании:  
 self.впп\_в\_использовании.занята = False  
 self.впп\_в\_использовании = None  
  
 def разрешить\_парковку\_у\_терминала(self, самолет: Самолет) -> bool:  
 доступный\_терминал = next((т for т in self.терминалы if not т.занят), None)  
 if доступный\_терминал:  
 доступный\_терминал.занят = True  
 print(f"Вышка: Разрешаю парковку самолету {самолет.имя} у {доступный\_терминал}.")  
 return True  
 else:  
 print(f"Вышка: Отказываю в парковке самолету {самолет.имя}. Все терминалы заняты.")  
 return False  
  
 def завершить\_парковку(self, самолет: Самолет, терминал\_id: int):  
 терминал = next((т for т in self.терминалы if т.id == терминал\_id), None)  
 if терминал:  
 терминал.занят = False  
 print(f"Вышка: Самолет {самолет.имя} освободил {терминал}.")  
  
class Самолет:  
 def \_\_init\_\_(self, имя: str, вышка: АвиадиспетчерскаяВышка):  
 self.имя = имя  
 self.вышка = вышка  
 self.статус = "на\_земле"  
 self.терминал = None  
  
 def \_\_repr\_\_(self) -> str:  
 return f"Самолет {self.имя}"  
  
 def запросить\_взлет(self):  
 print(f"{self.имя}: Запрашиваю разрешение на взлет.")  
 if self.вышка.разрешить\_взлет(self):  
 self.статус = "взлетает"  
 print(f"{self.имя}: Взлетаю...")  
 self.вышка.завершить\_взлет(self)  
 self.статус = "в\_полете"  
 print(f"{self.имя}: В воздухе.")  
 else:  
 print(f"{self.имя}: Ожидаю разрешения на взлет.")  
  
 def запросить\_посадку(self):  
 print(f"{self.имя}: Запрашиваю разрешение на посадку.")  
 if self.вышка.разрешить\_посадку(self):  
 self.статус = "заходит\_на\_посадку"  
 print(f"{self.имя}: Захожу на посадку.")  
 self.вышка.завершить\_посадку(self)  
 self.статус = "на\_земле"  
 print(f"{self.имя}: Приземлился.")  
 self.запросить\_парковку()  
 else:  
 print(f"{self.имя}: Ожидаю разрешения на посадку.")  
  
 def запросить\_парковку(self):  
 print(f"{self.имя}: Запрашиваю парковку у терминала.")  
 if self.вышка.разрешить\_парковку\_у\_терминала(self):  
 self.статус = "паркуется"  
 print(f"{self.имя}: Паркуюсь у терминала.")  
 self.статус = "в\_терминале"  
 print(f"{self.имя}: Припаркован.")  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 вышка = АвиадиспетчерскаяВышка()  
  
 for i in range(1, 4):  
 вышка.добавить\_впп(ВзлетноПосадочнаяПолоса(i))  
 for i in range(1, 6):  
 вышка.добавить\_терминал(Терминал(i))  
  
 самолет1 = Самолет("Boeing 747", вышка)  
 самолет2 = Самолет("Airbus A380", вышка)  
 самолет3 = Самолет("Cessna 172", вышка)  
 самолет4 = Самолет("Boeing 777", вышка)  
 самолет5 = Самолет("Boeing 787", вышка)  
  
 print("\n--- Сценарий взлета ---\n")  
 самолет1.запросить\_взлет()  
 самолет2.запросить\_взлет()  
 самолет3.запросить\_взлет()  
 самолет4.запросить\_взлет()  
 print("\n--- Сценарий посадки ---\n")  
 самолет5.запросить\_посадку()  
 самолет4.запросить\_посадку()  
 самолет3.запросить\_посадку()  
 самолет2.запросить\_посадку()  
 самолет1.запросить\_посадку()

Результат (рис. 1)

Изображение выглядит как текст, снимок экрана, меню, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 1

## Контрольные вопросы

1. Назначение поведенческих шаблонов проектирования.

Поведенческие шаблоны описывают способы взаимодействия объектов и распределение обязанностей между ними. Они фокусируются на алгоритмах и передаче управления.

**Перечень поведенческих шаблонов:**

* + Цепочка ответственности
  + Команда
  + Интерпретатор
  + Итератор
  + Посредник
  + Хранитель
  + Наблюдатель
  + Состояние
  + Стратегия
  + Шаблонный метод
  + Посетитель

1. **Шаблон «Цепочка ответственности» (Chain of responsibility).**
   * **Назначение:** Позволяет передавать запросы последовательно по цепочке обработчиков. Каждый обработчик решает, обработать запрос или передать его следующему в цепочке.
   * **Структура:** Состоит из Client (инициирует запрос), Handler (интерфейс обработчика), и ConcreteHandler (конкретные обработчики).
   * **Преимущества:** Снижает связанность между отправителем и получателем запроса, позволяет динамически добавлять/удалять обработчики.
   * **Недостатки:** Запрос может остаться необработанным, сложность отладки длинных цепочек.
2. **Шаблон «Команда» (Command).**
   * **Назначение:** Инкапсулирует запрос как объект, позволяя параметризовать клиентов с различными запросами, ставить их в очередь, логировать или поддерживать отмену операций.
   * **Структура:** Command (интерфейс), ConcreteCommand (реализация), Invoker (инициатор), Receiver (получатель).
   * **Преимущества:** Разделяет инициатора и получателя, упрощает отмену и повтор операций, позволяет создавать очереди команд.
   * **Недостатки:** Усложняет код из-за множества классов для простых операций.
3. **Шаблон «Интерпретатор» (Interpreter).**
   * **Назначение:** Определяет грамматику для языка и предоставляет интерпретатор для обработки предложений на этом языке. Подходит для простых языков.
   * **Структура:** AbstractExpression (интерфейс выражения), TerminalExpression (терминальные выражения), NonterminalExpression (нетерминальные выражения), Context (контекст).
   * **Преимущества:** Упрощает изменение и расширение грамматики.
   * **Недостатки:** Подходит только для простых грамматик, может быть сложным в реализации.
4. **Шаблон «Итератор» (Iterator).**
   * **Назначение:** Предоставляет способ последовательного доступа к элементам коллекции, не раскрывая ее внутреннее устройство.
   * **Структура:** Iterator (интерфейс), ConcreteIterator (реализация), Aggregate (интерфейс коллекции), ConcreteAggregate (конкретная коллекция).
   * **Преимущества:** Единый способ доступа к элементам разных коллекций, скрывает внутреннюю структуру.
   * **Недостатки:** Дополнительная сложность, может быть избыточным для простых коллекций.
5. **Шаблон «Посредник» (Mediator).**
   * **Назначение:** Определяет объект, который инкапсулирует способ взаимодействия набора объектов. Посредник обеспечивает слабое связывание, позволяя объектам взаимодействовать без прямых ссылок друг на друга.
   * **Структура:** Mediator (интерфейс), ConcreteMediator (реализация), Colleague (интерфейс), ConcreteColleague (взаимодействующие объекты).
   * **Преимущества:** Снижает связность между компонентами, упрощает управление взаимодействием, делает систему более гибкой.
   * **Недостатки:** Посредник может стать очень сложным и превратиться в "Божественный объект" (God Object), его логика может быть трудноотлаживаемой.
6. **Шаблон «Хранитель» (Memento).**
   * **Назначение:** Без нарушения инкапсуляции фиксирует и выносит за пределы объекта его внутреннее состояние, чтобы впоследствии можно было его восстановить.
   * **Структура:** Originator (объект-создатель), Memento (объект-хранитель состояния), Caretaker (опекун).
   * **Преимущества:** Позволяет реализовать отмену операций, не нарушая инкапсуляцию.
   * **Недостатки:** Может потреблять много памяти, если состояние объекта большое.
7. **Шаблон «Наблюдатель» (Observer).**
   * **Назначение:** Определяет зависимость «один ко многим» между объектами. Когда один объект (субъект) меняет состояние, все зависящие от него объекты (наблюдатели) автоматически оповещаются и обновляются.
   * **Структура:** Subject (субъект), Observer (интерфейс), ConcreteSubject, ConcreteObserver.
   * **Преимущества:** Слабая связанность, динамическая настройка зависимостей.
   * **Недостатки:** Порядок вызова обновлений может быть непредсказуемым.
8. **Шаблон «Состояние» (State).**
   * **Назначение:** Позволяет объекту изменять свое поведение в зависимости от внутреннего состояния. Поведение выносится в отдельные классы-состояния.
   * **Структура:** Context (контекст, меняющий поведение), State (интерфейс состояния), ConcreteStateA, ConcreteStateB.
   * **Преимущества:** Устраняет большие условные операторы, упрощает добавление новых состояний.
   * **Недостатки:** Может привести к большому количеству классов.
9. **Шаблон «Стратегия» (Strategy).**
   * **Назначение:** Определяет семейство алгоритмов, инкапсулирует каждый из них и делает их взаимозаменяемыми. Позволяет выбирать алгоритм во время выполнения.
   * **Структура:** Context (контекст), Strategy (интерфейс), ConcreteStrategyA, ConcreteStrategyB.
   * **Преимущества:** Позволяет менять алгоритмы на лету, устраняет условные операторы.
   * **Недостатки:** Увеличивает количество объектов, клиент должен знать о различиях между стратегиями.
10. **Шаблон «Шаблонный метод» (Template method).**
    * **Назначение:** Определяет скелет алгоритма в суперклассе, оставляя подклассам возможность переопределить некоторые шаги без изменения общей структуры.
    * **Структура:** AbstractClass (абстрактный класс с шаблонным методом), ConcreteClass (реализует примитивные операции).
    * **Преимущества:** Контроль над общей структурой алгоритма, гибкая реализация отдельных шагов.
    * **Недостатки:** Ограничивает гибкость, так как общая структура остается неизменной.
11. **Шаблон «Посетитель» (Visitor).**
    * **Назначение:** Позволяет добавлять новые операции к классам объектов, не изменяя сами классы.
    * **Структура:** Visitor (интерфейс), ConcreteVisitorA, Element (интерфейс), ConcreteElementA, ObjectStructure (коллекция).
    * **Преимущества:** Добавление новых операций без изменения классов, объединение связанных операций в один класс.
    * **Недостатки:** Нарушает инкапсуляцию, так как посетитель должен иметь доступ к внутренностям посещаемых классов.