МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ

КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«СИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ТЕЛЕКОММУНИКАЦИЙ И ИНФОРМАТИКИ»

(СибГУТИ)

ХАБАРОВСКИЙ ИНСТИТУТ ИНФОКОММУНИКАЦИЙ (ФИЛИАЛ)

(ХИИК СибГУТИ)

СРЕДНЕЕ ПРОФЕССИОНАЛЬНОЕ ОБРАЗОВАНИЕ

ОТЧЁТ

по МДК 01.01 «Разработка программных модулей»

|  |  |
| --- | --- |
| Выполнил | Лабзов Д.В |
| Проверил | Ничипорук Н.Е |
| Группа | ИСП-320 |

Хабаровск, 2025го

**Цель работы, задание, вариант:**

* **Цель работы:** Изучение и практическое применение порождающих шаблонов проектирования на языке Python.
* **Задание:** Реализовать программу для решения задачи согласно варианту с использованием требуемого порождающего шаблона проектирования.
* **Вариант:**
  + Программа 1 (Шаблон Строитель): Информационная система для формирования отчета по лабораторным работам
  + Программа 2 (отложенная инициализация): Ведение журнала событий программы.

**3. Краткие теоретические сведения:**

**3.1. Описание шаблона проектирования:**

Пул одиночек (Singleton Pool) — это порождающий шаблон проектирования, который управляет ограниченным набором уникальных экземпляров класса, гарантируя их повторное использование и предотвращая создание избыточных объектов. Пул одиночек централизует логику создания и распределения экземпляров, обеспечивая их доступность в рамках заданного лимита.

**Шаблон Строитель (Builder):** Шаблон строитель - это порождающий шаблон проектирования, который отделяет конструирование сложного объекта от его представления, так что один и тот же процесс построения может создавать разные представления. Он позволяет создавать объекты пошагово, предоставляя клиенту возможность контролировать процесс построения.

**3.2. Классовая диаграмма для шаблонов проектирования в нотации UML:**

* + **Пул одиночек:**

@startuml

class SingletonPool {

- \_instances: Dict[str, List[DatabaseConnection]]

- \_max\_instances: int = 2

- \_lock: threading.Lock

+ {static} get\_connection(db\_name: str): DatabaseConnection

+ {static} release\_connection(conn: DatabaseConnection): void

}

class DatabaseConnection {

- db\_name: str

- in\_use: bool = False

+ query(sql: str): void

}

SingletonPool o-- DatabaseConnection : contains

Client --> SingletonPool : uses

Client --> DatabaseConnection : uses

@enduml

* + **Шаблон Строитель:**

@startuml

class Director {

+Construct(builder: Builder)

}

abstract class Builder {

+BuildPartA()

+BuildPartB()

+GetResult(): Product

}

class ConcreteBuilder1 {

+BuildPartA()

+BuildPartB()

+GetResult(): Product1

}

class Product {

}

class Product1 {

}

class ConcreteProduct2 {

}

Director -- Builder : manages

Builder -- Product : creates

ConcreteBuilder1 -- Product1 : creates

Product <|-- Product1

Builder <|-- ConcreteBuilder1

@enduml

plantuml

**4. Описание условий и хода работы:**

**4.1. Используемые аппаратные и программные средства:**

* + Операционная система: Windows 11
  + Язык программирования: Python 3.9
  + IDE: PyCharm
  + Библиотеки: (Для задач, использующих внешние библиотеки)
  + Инструмент UML: PlantUML

**4.2. Интерпретация шаблона проектирования для решения задачи:**

**Программа 1 (Фабричный метод):**

* + - Product: Абстрактный класс Document с атрибутами number, date, content.
    - ConcreteProduct: Классы Letter, Order, TravelOrder, наследующие от Document и добавляющие специфичные атрибуты.
    - Creator: Абстрактный класс DocumentFactory с методом create\_document(doc\_type, data).
    - ConcreteCreator: Классы LetterFactory, OrderFactory, TravelOrderFactory, наследующие от DocumentFactory и реализующие метод create\_document, создавая соответствующие типы документов.
    - Клиентский код использует DocumentFactory для создания документов, не зная конкретных классов документов.

**Программа 2 (Шаблон Строитель):**

* + - Product: Класс Report, представляющий отчет по лабораторной работе.
    - Builder: Абстрактный класс ReportBuilder с методами build\_title\_page(), build\_objective(), build\_task(), build\_theory(), build\_experiment(), build\_results(), build\_analysis(), build\_conclusion() и get\_report().
    - ConcreteBuilder: Классы DatabaseReportBuilder, NetworkReportBuilder, ProgrammingReportBuilder, наследующие от ReportBuilder и реализующие методы для построения отчета с учетом особенностей каждой дисциплины.
    - Director: Класс ReportGenerator, управляющий процессом построения отчета с помощью ReportBuilder.
    - Клиентский код использует ReportGenerator для создания отчетов, предоставляя конкретный ReportBuilder в зависимости от дисциплины.

**4.3. Классовая диаграмма для задачи в нотации UML:**

* + **Программа 1 (Фабричный метод):**

@startuml

abstract class Document {

-number: str

-date: str

-content: str

+\_\_init\_\_(number: str, date: str, content: str)

+display(): str

}

class Letter {

-type: str

-correspondent: str

+\_\_init\_\_(number: str, date: str, content: str, type: str, correspondent: str)

+display(): str

}

class Order {

-department: str

-deadline: str

-responsible: str

+\_\_init\_\_(number: str, date: str, content: str, department: str, deadline: str, responsible: str)

+display(): str

}

class TravelOrder {

-employee: str

-period: str

-destination: str

+\_\_init\_\_(number: str, date: str, content: str, employee: str, period: str, destination: str)

+display(): str

}

abstract class DocumentFactory {

+create\_document(doc\_type: str, dict): Document

}

class LetterFactory {

+create\_document(dict): Letter

}

class OrderFactory {

+create\_document(dict): Order

}

class TravelOrderFactory {

+create\_document(dict): TravelOrder

}

Document <|-- Letter

Document <|-- Order

Document <|-- TravelOrder

DocumentFactory -- Document : creates

DocumentFactory <|-- LetterFactory

DocumentFactory <|-- OrderFactory

DocumentFactory <|-- TravelOrderFactory

LetterFactory -- Letter : creates

OrderFactory -- Order : creates

TravelOrderFactory -- TravelOrder : creates

@enduml

plantuml

* + **Программа 2 (Шаблон Строитель):**

@startuml

class Report {

-title\_page: str

-objective: str

-task: str

-theory: str

-experiment: str

-results: str

-analysis: str

-conclusion: str

+\_\_init\_\_()

+display(): str

+to\_html(): str

}

abstract class ReportBuilder {

-report: Report

+\_\_init\_\_()

+build\_title\_page()

+build\_objective()

+build\_task()

+build\_theory()

+build\_experiment()

+build\_results()

+build\_analysis()

+build\_conclusion()

+get\_report(): Report

}

class DatabaseReportBuilder {

+\_\_init\_\_()

+build\_title\_page()

+build\_objective()

+build\_task()

+build\_theory()

+build\_experiment()

+build\_results()

+build\_analysis()

+build\_conclusion()

+get\_report(): Report

}

class NetworkReportBuilder {

+\_\_init\_\_()

+build\_title\_page()

+build\_objective()

+build\_task()

+build\_theory()

+build\_experiment()

+build\_results()

+build\_analysis()

+build\_conclusion()

+get\_report(): Report

}

class ProgrammingReportBuilder {

+\_\_init\_\_()

+build\_title\_page()

+build\_objective()

+build\_task()

+build\_theory()

+build\_experiment()

+build\_results()

+build\_analysis()

+build\_conclusion()

+get\_report(): Report

}

class ReportGenerator {

+\_\_init\_\_(builder: ReportBuilder)

+generate\_report()

+get\_report(): Report

}

ReportBuilder -- Report : builds

ReportGenerator -- ReportBuilder : uses

ReportBuilder <|-- DatabaseReportBuilder

ReportBuilder <|-- NetworkReportBuilder

ReportBuilder <|-- ProgrammingReportBuilder

@enduml

plantuml

**5. Результаты работы:**

**5.1. Текст программы:**

*# Программа 1: Фабричный метод*

class Document:

def \_\_init\_\_(self, number, date, content):

self.number = number

self.date = date

self.content = content

def display(self):

return f"Номер: {self.number}\nДата: {self.date}\nСодержание: {self.content}"

class Letter(Document):

def \_\_init\_\_(self, number, date, content, type, correspondent):

super().\_\_init\_\_(number, date, content)

self.type = type

self.correspondent = correspondent

def display(self):

return f"{super().display()}\nТип: {self.type}\nКорреспондент: {self.correspondent}"

class Order(Document):

def \_\_init\_\_(self, number, date, content, department, deadline, responsible):

super().\_\_init\_\_(number, date, content)

self.department = department

self.deadline = deadline

self.responsible = responsible

def display(self):

return f"{super().display()}\nПодразделение: {self.department}\nСрок выполнения: {self.deadline}\nОтветственный: {self.responsible}"

class TravelOrder(Document):

def \_\_init\_\_(self, number, date, content, employee, period, destination):

super().\_\_init\_\_(number, date, content)

self.employee = employee

self.period = period

self.destination = destination

def display(self):

return f"{super().display()}\nСотрудник: {self.employee}\nПериод: {self.period}\nМесто назначения: {self.destination}"

class DocumentFactory:

def create\_document(self, doc\_type, data):

if doc\_type == "letter":

return LetterFactory().create\_document(data)

elif doc\_type == "order":

return OrderFactory().create\_document(data)

elif doc\_type == "travel\_order":

return TravelOrderFactory().create\_document(data)

else:

raise ValueError("Неизвестный тип документа")

class LetterFactory:

def create\_document(self, data):

return Letter(\*\*data)

class OrderFactory:

def create\_document(self, data):

return Order(\*\*data)

class TravelOrderFactory:

def create\_document(self, data):

return TravelOrder(\*\*data)

documents = [

{"doc\_type": "letter", "data": {"number": "123", "date": "2023-10-26", "content": "Входящее письмо", "type": "входящее", "correspondent": "ООО Ромашка"}},

{"doc\_type": "order", "data": {"number": "456", "date": "2023-10-25", "content": "Приказ о премировании", "department": "Бухгалтерия", "deadline": "2023-11-05", "responsible": "Иванов И.И."}},

{"doc\_type": "travel\_order", "data": {"number": "789", "date": "2023-10-24", "content": "Распоряжение о командировке", "employee": "Петров П.П.", "period": "2023-10-27 - 2023-10-31", "destination": "г. Москва"}}

]

factory = DocumentFactory()

document\_objects = []

for doc in documents:

document\_objects.append(factory.create\_document(doc["doc\_type"], doc["data"]))

print("1. Полный перечень документов:")for document in document\_objects:

print(document.display())

print("-" \* 20)

document\_number = input("Введите номер документа для просмотра содержания: ")

found\_document = Nonefor document in document\_objects:

if document.number == document\_number:

found\_document = document

break

if found\_document:

print("2. Содержание выбранного документа:")

print(found\_document.display())else:

print("Документ с указанным номером не найден.")

*# Программа 2: Шаблон Строитель*

class Report:

def \_\_init\_\_(self):

self.title\_page = None

self.objective = None

self.task = None

self.theory = None

self.experiment = None

self.results = None

self.analysis = None

self.conclusion = None

def display(self):

print(f"Титульный лист: {self.title\_page}")

print(f"Цель работы: {self.objective}")

print(f"Задание: {self.task}")

print(f"Теория: {self.theory}")

print(f"Эксперимент: {self.experiment}")

print(f"Результаты: {self.results}")

print(f"Анализ: {self.analysis}")

print(f"Выводы: {self.conclusion}")

def to\_html(self):

html = "<html><body>"

html += f"<h1>{self.title\_page}</h1>"

html += f"<h2>Цель работы</h2><p>{self.objective}</p>"

html += f"<h2>Задание</h2><p>{self.task}</p>"

html += f"<h2>Теория</h2><p>{self.theory}</p>"

html += f"<h2>Экспериментальная установка</h2><p>{self.experiment}</p>"

html += f"<h2>Результаты работы</h2><p>{self.results}</p>"

html += f"<h2>Анализ результатов</h2><p>{self.analysis}</p>"

html += f"<h2>Выводы</h2><p>{self.conclusion}</p>"

html += "</body></html>"

return html

class ReportBuilder:

def \_\_init\_\_(self):

self.report = Report()

def build\_title\_page(self):

raise NotImplementedError

def build\_objective(self):

raise NotImplementedError

def build\_task(self):

raise NotImplementedError

def build\_theory(self):

raise NotImplementedError

def build\_experiment(self):

raise NotImplementedError

def build\_results(self):

raise NotImplementedError

def build\_analysis(self):

raise NotImplementedError

def build\_conclusion(self):

raise NotImplementedError

def get\_report(self):

return self.report

class DatabaseReportBuilder(ReportBuilder):

def \_\_init\_\_(self):

super().\_\_init\_\_()

def build\_title\_page(self):

self.report.title\_page = "Отчет по лабораторной работе №1. Базы данных"

def build\_objective(self):

self.report.objective = "Изучить основы реляционных баз данных"

def build\_task(self):

self.report.task = "Разработать схему базы данных для предметной области"

def build\_theory(self):

self.report.theory = "Нормализация, реляционная модель данных, SQL"

def build\_experiment(self):

self.report.experiment = "Создание таблиц, запросы, индексы"

def build\_results(self):

self.report.results = "Схема базы данных, запросы SQL"

def build\_analysis(self):

self.report.analysis = "Анализ эффективности запросов"

def build\_conclusion(self):

self.report.conclusion = "Базы данных - основа информационных систем"

class NetworkReportBuilder(ReportBuilder):

def \_\_init\_\_(self):

super().\_\_init\_\_()

def build\_title\_page(self):

self.report.title\_page = "Отчет по лабораторной работе №1. Компьютерные сети"

def build\_objective(self):

self.report.objective = "Изучить основы сетевых протоколов"

def build\_task(self):

self.report.task = "Настроить сетевое соединение между двумя компьютерами"

def build\_theory(self):

self.report.theory = "Модель OSI, протоколы TCP/IP, адресация"

def build\_experiment(self):

self.report.experiment = "Настройка IP-адресов, ping, traceroute"

def build\_results(self):

self.report.results = "Скриншоты настроек, результаты ping"

def build\_analysis(self):

self.report.analysis = "Анализ сетевых задержек"

def build\_conclusion(self):

self.report.conclusion = "Компьютерные сети обеспечивают передачу данных"

class ProgrammingReportBuilder(ReportBuilder):

def \_\_init\_\_(self):

super().\_\_init\_\_()

def build\_title\_page(self):

self.report.title\_page = "Отчет по лабораторной работе №1. Программирование"

def build\_objective(self):

self.report.objective = "Изучить основы объектно-ориентированного программирования"

def build\_task(self):

self.report.task = "Разработать программу для решения задачи"

def build\_theory(self):

self.report.theory = "Классы, объекты, наследование, полиморфизм"

def build\_experiment(self):

self.report.experiment = "Написание кода, отладка, тестирование"

def build\_results(self):

self.report.results = "Текст программы, результаты тестов"

def build\_analysis(self):

self.report.analysis = "Анализ сложности алгоритма"

def build\_conclusion(self):

self.report.conclusion = "Программирование позволяет создавать сложные системы"

class ReportGenerator:

def \_\_init\_\_(self, builder):

self.builder = builder

def generate\_report(self):

self.builder.build\_title\_page()

self.builder.build\_objective()

self.builder.build\_task()

self.builder.build\_theory()

self.builder.build\_experiment()

self.builder.build\_results()

self.builder.build\_analysis()

self.builder.build\_conclusion()

def get\_report(self):

return self.builder.get\_report()

*# Клиентский код для программы 2*

discipline = input("Выберите дисциплину (database, network, programming): ")if discipline == "database":

builder = DatabaseReportBuilder()elif discipline == "network":

builder = NetworkReportBuilder()elif discipline == "programming":

builder = ProgrammingReportBuilder()else:

print("Неверная дисциплина")

exit()

generator = ReportGenerator(builder)

generator.generate\_report()

report = generator.get\_report()

print("1. Полный отчет:")

report.display()

print("\n2. Отчет в формате HTML:")

html\_report = report.to\_html()print(html\_report)

*# Сохранение в HTML файл (необязательно)*with open("report.html", "w", encoding="utf-8") as f:

f.write(html\_report)

**7. Выводы о достоинствах и недостатках используемого шаблона проектирования:**

**Фабричный метод (Программа 1):**

* + **Достоинства:**
    - **Гибкость:** Позволяет добавлять новые типы документов без изменения существующего кода.
    - **Слабая связность:** Клиентский код не зависит от конкретных классов документов.
    - **Расширяемость:** Легко добавить новые фабрики для новых типов документов.
  + **Недостатки:**
    - **Усложнение структуры кода:** Требуется создание дополнительных классов (фабрик).
    - **Не всегда оправдано для простых задач:** Может быть избыточным, если количество типов документов невелико и не планируется расширение.

**Шаблон Строитель (Программа 2):**

* + **Достоинства:**
    - **Гибкость:** Позволяет создавать отчеты в разных форматах или с разным набором разделов, используя разные строители.
    - **Разделение логики:** Отделяет логику построения от представления.
    - **Контроль над процессом:** Клиент имеет контроль над процессом построения отчета.
  + **Недостатки:**
    - **Увеличение количества классов:** Требуется создание строителей, директора и продукта.
    - **Сложность для простых объектов:** Может быть избыточным, если отчет имеет простую структуру.

В заключение, выбор подходящего шаблона проектирования зависит от конкретной задачи и требований к гибкости, расширяемости и сложности кода.