Отчёт  
по лабораторной работе №1  
по дисциплине «МДК 01.01 Разработка программного модуля»  
Тема: «Порождающие 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 3.**

**Задание 1.**

Код программы:

from abc import ABC, abstractmethod

class Material(ABC):

@abstractmethod

def get\_name(self):

pass

class Brick(Material):

def get\_name(self):

return "Кирпич"

class Slab(Material):

def get\_name(self):

return "Железобетонная плита"

class Concrete(Material):

def get\_name(self):

return "Бетон"

class Supplier(ABC):

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

self.name = name

@abstractmethod

def get\_price(self, material):

pass

@abstractmethod

def get\_max\_daily\_supply(self, material):

pass

class Supplier1(Supplier):

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

super().\_\_init\_\_("Поставщик 1")

self.prices = {

"Кирпич": 100,

"Железобетонная плита": 5000,

"Бетон": 2500

}

self.max\_supply = {

"Кирпич": 1000,

"Железобетонная плита": 10,

"Бетон": 50

}

def get\_price(self, material\_name):

return self.prices.get(material\_name, float('inf'))

def get\_max\_daily\_supply(self, material\_name):

return self.max\_supply.get(material\_name, 0)

class Supplier2(Supplier):

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

super().\_\_init\_\_("Поставщик 2")

self.prices = {

"Кирпич": 110,

"Бетон": 2400

}

self.max\_supply = {

"Кирпич": 800,

"Бетон": 60

}

def get\_price(self, material\_name):

return self.prices.get(material\_name, float('inf'))

def get\_max\_daily\_supply(self, material\_name):

return self.max\_supply.get(material\_name, 0)

class Supplier3(Supplier):

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

super().\_\_init\_\_("Поставщик 3")

self.prices = {

"Железобетонная плита": 4800,

"Бетон": 2600

}

self.max\_supply = {

"Железобетонная плита": 15,

"Бетон": 45

}

def get\_price(self, material\_name):

return self.prices.get(material\_name, float('inf'))

def get\_max\_daily\_supply(self, material\_name):

return self.max\_supply.get(material\_name, 0)

class SupplierFactory(ABC):

@abstractmethod

def create\_supplier(self, supplier\_id):

pass

class ConcreteSupplierFactory(SupplierFactory):

def create\_supplier(self, supplier\_id):

if supplier\_id == 1:

return Supplier1()

elif supplier\_id == 2:

return Supplier2()

elif supplier\_id == 3:

return Supplier3()

else:

return None

def calculate\_optimal\_plan(daily\_needs, suppliers, materials):

optimal\_plan = {}

total\_cost = 0

for material in materials:

material\_name = material.get\_name()

needed\_quantity = daily\_needs.get(material\_name, 0)

remaining\_needed = needed\_quantity

material\_plan = []

if needed\_quantity == 0:

continue

sorted\_suppliers = sorted(suppliers, key=lambda s: s.get\_price(material\_name))

for supplier in sorted\_suppliers:

price = supplier.get\_price(material\_name)

max\_supply = supplier.get\_max\_daily\_supply(material\_name)

if price == float('inf') or max\_supply == 0:

continue

quantity\_to\_buy = min(remaining\_needed, max\_supply)

if quantity\_to\_buy > 0:

cost = quantity\_to\_buy \* price

material\_plan.append({"supplier": supplier.name, "quantity": quantity\_to\_buy, "cost": cost})

total\_cost += cost

remaining\_needed -= quantity\_to\_buy

if remaining\_needed == 0:

break

if remaining\_needed > 0:

print(f"Невозможно удовлетворить всю потребность в материале '{material\_name}'. Не хватает {remaining\_needed} единиц.")

optimal\_plan[material\_name] = material\_plan

return optimal\_plan, total\_cost

if \_\_name\_\_ == "\_\_main\_\_":

factory = ConcreteSupplierFactory()

suppliers = [factory.create\_supplier(1), factory.create\_supplier(2), factory.create\_supplier(3)]

materials = [Brick(), Slab(), Concrete()]

daily\_needs = {}

print("Введите суточную потребность в материалах:")

for material in materials:

try:

quantity = int(input(f"{material.get\_name()}: "))

daily\_needs[material.get\_name()] = quantity

except ValueError:

daily\_needs[material.get\_name()] = 0

optimal\_plan, total\_cost = calculate\_optimal\_plan(daily\_needs, suppliers, materials)

print("\nОптимальный план поставок:")

for material, plan in optimal\_plan.items():

if plan:

print(f"\n{material}:")

for item in plan:

print(f" - Поставщик: {item['supplier']}, Количество: {item['quantity']}, Стоимость: {item['cost']}")

print(f"\nОбщая стоимость поставок: {total\_cost}")

Результат (рис. 1):

Изображение выглядит как текст, снимок экрана, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 1

**Задание 2.**

Код программы:

from abc import ABC, abstractmethod

class Body(ABC):

    @abstractmethod

    def get\_description(self):

        pass

class Engine(ABC):

    @abstractmethod

    def get\_description(self):

        pass

class Gearbox(ABC):

    @abstractmethod

    def get\_description(self):

        pass

class Wheels(ABC):

    @abstractmethod

    def get\_description(self):

        pass

class SedanBody(Body):

    def get\_description(self):

        return "Кузов: Седан"

class CoupeBody(Body):

    def get\_description(self):

        return "Кузов: Купе"

class V8Engine(Engine):

    def get\_description(self):

        return "Двигатель: V8"

class V12Engine(Engine):

    def get\_description(self):

        return "Двигатель: V12"

class I6Engine(Engine):

    def get\_description(self):

        return "Двигатель: I6"

class AutomaticGearbox(Gearbox):

    def get\_description(self):

        return "Коробка передач: Автоматическая"

class ManualGearbox(Gearbox):

    def get\_description(self):

        return "Коробка передач: Механическая"

class RFTWheels(Wheels):

    def get\_description(self):

        return "Диски/шины: Run-flat"

class PerformanceWheels(Wheels):

    def get\_description(self):

        return "Диски/шины: Performance"

class CarFactory(ABC):

    @abstractmethod

    def create\_body(self, model):

        pass

    @abstractmethod

    def create\_engine(self, model):

        pass

    @abstractmethod

    def create\_gearbox(self, model):

        pass

    @abstractmethod

    def create\_wheels(self, model):

        pass

class BMWFactory(CarFactory):

    def create\_body(self, model):

        if model == "3 Series":

            return SedanBody()

        if model == "8 Series":

            return CoupeBody()

        if model == "M5":

            return SedanBody()

        return None

    def create\_engine(self, model):

        if model == "3 Series":

            return I6Engine()

        if model == "8 Series":

            return V8Engine()

        if model == "M5":

            return V8Engine()

        return None

    def create\_gearbox(self, model):

        if model == "3 Series":

            return AutomaticGearbox()

        if model == "8 Series":

            return AutomaticGearbox()

        if model == "M5":

            return AutomaticGearbox()

        return None

    def create\_wheels(self, model):

        if model == "3 Series":

            return RFTWheels()

        if model == "8 Series":

            return RFTWheels()

        if model == "M5":

            return PerformanceWheels()

        return None

class CarModel:

    def \_\_init\_\_(self, name, body, engine, gearbox, wheels):

        self.name = name

        self.body = body

        self.engine = engine

        self.gearbox = gearbox

        self.wheels = wheels

    def get\_description(self):

        return f"""

        <h2>Модель: {self.name}</h2>

        <ul>

            <li>{self.body.get\_description()}</li>

            <li>{self.engine.get\_description()}</li>

            <li>{self.gearbox.get\_description()}</li>

            <li>{self.wheels.get\_description()}</li>

        </ul>

        """

def assemble\_and\_describe\_models(factory):

    models = ["3 Series", "8 Series", "M5"]

    car\_models = []

    for model\_name in models:

        body = factory.create\_body(model\_name)

        engine = factory.create\_engine(model\_name)

        gearbox = factory.create\_gearbox(model\_name)

        wheels = factory.create\_wheels(model\_name)

        if all([body, engine, gearbox, wheels]):

            car\_models.append(CarModel(model\_name, body, engine, gearbox, wheels))

    html\_content = """

    <!DOCTYPE html>

    <html lang="ru">

    <head>

        <meta charset="UTF-8">

        <meta name="viewport" content="width=device-width, initial-scale=1.0">

        <title>Модели автомобилей</title>

        <style>

            body { font-family: sans-serif; line-height: 1.6; padding: 20px; }

            .container { max-width: 800px; margin: auto; background: #f4f4f4; padding: 20px; border-radius: 8px; }

            h1 { color: #333; text-align: center; }

            h2 { color: #555; }

            ul { list-style: none; padding: 0; }

            li { background: #e2e2e2; margin-bottom: 10px; padding: 10px; border-radius: 4px; }

        </style>

    </head>

    <body>

        <div class="container">

            <h1>Каталог автомобилей</h1>

    """

    for car in car\_models:

        html\_content += car.get\_description()

    html\_content += """

        </div>

    </body>

    </html>

    """

    with open("car\_models.html", "w", encoding="utf-8") as file:

        file.write(html\_content)

    print("HTML-файл 'car\_models.html' успешно сгенерирован.")

if \_\_name\_\_ == "\_\_main\_\_":

    bmw\_factory = BMWFactory()

    assemble\_and\_describe\_models(bmw\_factory)

Результат (рис. 2):

Изображение выглядит как текст, снимок экрана, программное обеспечение, Значок на компьютере

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 2

## Контрольные вопросы

1. **Назначение порождающих шаблонов проектирования. Перечень порождающих шаблонов.**   
   **Назначение:** управляют процессом создания объектов, обеспечивая гибкость и повторное использование кода.

**Перечень:** Абстрактная фабрика, Строитель, Фабричный метод, Прототип, Скелет

1. **Абстрактная фабрика**

**Назначение:** предоставляет интерфейс для создания семейств взаимосвязанных объектов без указания их конкретных классов. **Структура:** Интерфейс фабрики, конкретные фабрики, интерфейсы продуктов, конкретные продукты. **Преимущества:** Инкапсуляция создания объектов, обеспечение совместимости объектов. **Недостатки:** Усложнение кода при добавлении новых типов продуктов.

1. **Строитель**

**Назначение:** разделяет конструкцию сложного объекта от его представления.

**Структура:** Builder, ConcreteBuilder, Director, Product. **Преимущества:** Гибкий контроль над процессом создания, возможность создания разных представлений. **Недостатки:** Усложнение кода для простых объектов.

1. **Фабричный метод (Factory Method)**

**Назначение:** определяет интерфейс для создания объекта, но позволяет подклассам изменять тип создаваемых объектов. **Структура:** Creator (интерфейс), ConcreteCreator, Product (интерфейс), ConcreteProduct.

**Преимущества:** Гибкость, расширяемость, инкапсуляция создания. **Недостатки:** может привести к созданию большого числа подклассов.

1. **Отложенная инициализация**

**Назначение:** откладывает создание объекта до момента его первого использования.

**Структура:** Проверка на null, создание объекта при первом обращении.

**Преимущества:** Экономия ресурсов, ускорение запуска. **Недостатки:** Усложнение кода, возможные проблемы с многопоточностью.

1. **Пул одиночек**

**Назначение:** управляет картой ключ-объект, обеспечивая единственный экземпляр на ключ.

**Структура:** Статическое хранилище, метод доступа по ключу. **Преимущества:** Контроль над количеством экземпляров, экономия ресурсов.

**Недостатки:** Усложнение кода, возможные проблемы с многопоточностью.

1. **Объектный пул (Object Pool)**

**Назначение:** управляет набором инициализированных и готовых к использованию объектов.

**Структура:** Пул объектов, методы получения/возврата объектов. **Преимущества:** Экономия ресурсов, повышение производительности.

**Недостатки:** Усложнение кода, возможные утечки памяти.

1. **Прототип**

**Назначение:** создает новые объекты путем клонирования существующих.

**Структура:** Интерфейс прототипа, конкретные прототипы, клиент. **Преимущества:** Уменьшение количества классов, ускорение создания объектов.

**Недостатки:** Сложность клонирования сложных объектов.

1. **Инициализация при получении ресурса (RAII)**

**Назначение:** связывает управление ресурсами с временем жизни объектов.

**Структура:** Конструктор/деструктор класса управляет ресурсом. **Преимущества:** Автоматическое управление ресурсами, исключение утечек. **Недостатки:** Привязка к языкам с деструкторами (C++).

1. **Одиночка**

**Назначение:** гарантирует существование только одного экземпляра класса и предоставляет к нему глобальную точку доступа.

**Структура:** Приватный конструктор, статический метод доступа, статическое поле для хранения экземпляра.

**Преимущества:** Контроль над экземпляром, глобальная точка доступа. **Недостатки:** Усложнение тестирования, возможные проблемы с многопоточностью.