МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

“БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ”

ФАКУЛЬТЕТ ЭЛЕКТРОННО-ИНФОРМАЦИОННЫХ СИСТЕМ

Кафедра интеллектуальных информационных технологий

Отчет по лабораторной работе №3

Специальность ПО-13

Выполнил:

Д. В. Шибун

студент группы ПО-13

Проверил:

А. Д. Кулик

13.02.2026

Брест 2026

Цель работы:

Приобрести навыки применения паттернов проектирования

при решении практических задач с использованием языка Python

Общее задание

• Прочитать задания, взятые из каждой группы, соответствующей одному из

трех основных типов паттернов;

• Определить паттерн проектирования, который может использоваться при

реализации задания. Пояснить свой выбор;

• Реализовать фрагмент программной системы, используя выбранный паттерн.

Реализовать все необходимые дополнительные классы.

Первая группа заданий (порождающий паттерн)

2) Заводы по производству автомобилей. Реализовать возможность создавать автомобили различных типов на различных заводах.

Код программы:

```
from abc import ABC, abstractmethod

class Car(ABC):
    @abstractmethod
    def drive(self):
        pass

class BMWSedan(Car):
    def drive(self):
        return "BMW Sedan: комфортная езда"

class BMWSUV(Car):
    def drive(self):
        return "BMW SUV: мощный внедорожник"

class ToyotaSedan(Car):
    def drive(self):
        return "Toyota Sedan: экономичная езда"

class ToyotaSUV(Car):
    def drive(self):
        return "Toyota SUV: надёжный внедорожник"

class CarFactory(ABC):
    @abstractmethod
    def create_sedan(self) -> Car:
        pass

    @abstractmethod
    def create_suv(self) -> Car:
        pass
```

```
class BMWFactory(CarFactory):
    def create_sedan(self):
        return BMWSedan()

    def create_suv(self):
        return BMWSUV()
class ToyotaFactory(CarFactory):
    def create_sedan(self):
        return ToyotaSedan()

    def create_suv(self):
        return ToyotaSUV()

def client(factory: CarFactory):
    sedan = factory.create_sedan()
    suv = factory.create_suv()
    print(sedan.drive())
    print(suv.drive())

if __name__ == "__main__":
    print("=== BMW Factory ===")
    client(BMWFactory())

    print("\n=== Toyota Factory ===")
    client(ToyotaFactory())
```

Спецификация ввода

Введите название завода: <строка>

Введите тип автомобиля: <строка>

Пример

Введите название завода: BMW

Введите тип автомобиля: sedan

Спецификация вывода

Если завод существует и производит указанный тип автомобиля:

Автомобиль создан Модель: <строка> Описание: <строка>

Если завод не существует:

Ошибка: такого завода не существует

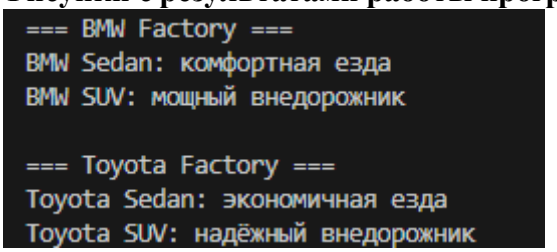
Если завод существует, но не производит указанный тип автомобиля:

Ошибка: данный завод не производит такой тип автомобиля

Пример

Введите название завода: BMW
Введите тип автомобиля: sedan
Автомобиль создан
Модель: BMW Sedan
Описание: комфортная езда

Рисунки с результатами работы программы



Вторая группа заданий (структурный паттерн)

2) Проект «Универсальная электронная карта». В проекте должна быть реализована универсальная электронная карта, в которой есть функции паспорта, страхового полиса, банковской карты и т. д.

Код программы:

```
from abc import ABC, abstractmethod

class Card(ABC):
    @abstractmethod
    def info(self) -> str:
        pass

class BaseCard(Card):
    def info(self) -> str:
        return "Универсальная электронная карта"

class CardDecorator(Card):
    def __init__(self, card: Card):
        self.card = card

    @abstractmethod
    def info(self) -> str:
        pass
class PassportDecorator(CardDecorator):
    def info(self) -> str:
        return self.card.info() + ", паспортные данные"

class InsuranceDecorator(CardDecorator):
    def info(self) -> str:
        return self.card.info() + ", страховой полис"

class BankCardDecorator(CardDecorator):
    def info(self) -> str:
        return self.card.info() + ", банковская карта"

class TransportDecorator(CardDecorator):
    def info(self) -> str:
        return self.card.info() + ", транспортная карта"

if __name__ == "__main__":
    card = BaseCard()
    card = PassportDecorator(card)
    card = InsuranceDecorator(card)
    card = BankCardDecorator(card)

    print(card.info())
```

Спецификация ввода

Введите список функций, которые нужно добавить к универсальной электронной карте. Каждая функция вводится отдельной строкой. Доступные функции:

- паспорт
- страховой полис
- банковская карта
- транспортная карта

Ввод завершается пустой строкой.

Пример

Введите функцию карты: паспорт

Введите функцию карты: страховой полис

Введите функцию карты: банковская карта

Введите функцию карты:

Спецификация вывода

Если введены корректные функции:

Универсальная электронная карта создана

Функции: <перечень функций>

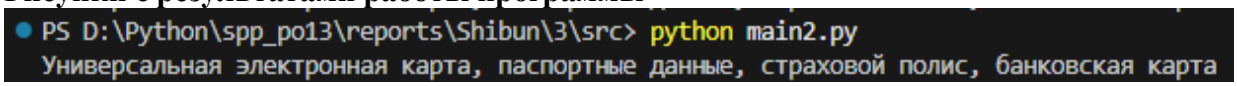
Если введена неизвестная функция:

Ошибка: функция "<строка>" не поддерживается

Пример

Универсальная электронная карта создана
Функции: паспортные данные, страховой полис, банковская карта

Рисунки с результатами работы программы



Третья группа заданий (поведенческий паттерн)

2) Проект «Принтеры». В проекте должны быть реализованы разные модели принтеров, которые выполняют разные виды печати.

Код программы:

```
from abc import ABC, abstractmethod

class PrintStrategy(ABC):
    @abstractmethod
    def print(self, text: str) -> str:
        pass
```

```
class ColorPrint(PrintStrategy):
    def print(self, text: str) -> str:
        return f"Цветная печать: {text}"

class BlackWhitePrint(PrintStrategy):
    def print(self, text: str) -> str:
        return f"Чёрно-белая печать: {text}"

class PhotoPrint(PrintStrategy):
    def print(self, text: str) -> str:
        return f"Фотопечать высокого качества: {text}"

class DraftPrint(PrintStrategy):
    def print(self, text: str) -> str:
        return f"Черновой режим: {text}"

class Printer:
    def __init__(self, model: str, strategy: PrintStrategy):
        self.model = model
        self.strategy = strategy

    def set_strategy(self, strategy: PrintStrategy):
        self.strategy = strategy

    def print(self, text: str):
        print(f"[{self.model}] {self.strategy.print(text)}")

if __name__ == "__main__":
    printer = Printer("HP LaserJet", BlackWhitePrint())
    printer.print("Документ 1")

    printer.set_strategy(ColorPrint())
    printer.print("Документ 2")

    printer.set_strategy(PhotoPrint())
    printer.print("Фото 3")
```

Спецификация ввода

Введите модель принтера: <строка>

Введите режим печати: <строка>

Доступные режимы печати:

- черно-белая
- цветная
- фото
- черновик

Пример

Введите модель принтера: HP LaserJet

Введите режим печати: цветная

Спецификация вывода

Если выбран корректный режим печати:

Печать выполнена

Принтер: <модель>

Режим: <режим>

Результат: <строка>

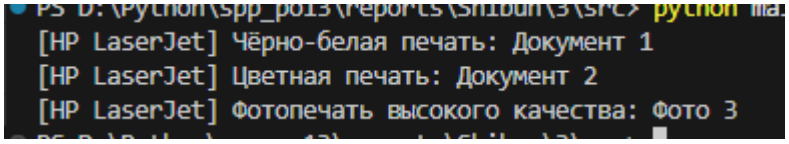
Если введён неизвестный режим:

Ошибка: режим печати "<строка>" не поддерживается

Пример

Печать выполнена
Принтер: HP LaserJet
Режим: фотопечать
Результат: Фотопечать высокого качества: <текст>

Рисунки с результатами работы программы



Вывод: В процессе выполнения работы были изучены и применены ключевые паттерны проектирования из трёх основных групп: порождающие, структурные и поведенческие. Реализация заданий позволила на практике увидеть, как паттерны помогают организовывать архитектуру программы, разделять ответственность между объектами и упрощать расширение функциональности.