МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
«БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»
ФАКУЛЬТЕТ ЭЛЕКТРОННО-ИНФОРМАЦИОННЫХ СИСТЕМ

Кафедра интеллектуальных информационных технологий

Отчёт по лабораторной работе №3

Специальность ПО11

Выполнил
П. А. Захарчук
студент группы ПО11

Проверил
А. А. Крощенко
ст. преп. кафедры ИИТ,
25.04.2025 г.

Брест 2025

Цель работы: приобрести навыки применения паттернов проектирования при решении практических задач с использованием языка Python.

Первая группа заданий (порождающий паттерн):

Кофе-автомат с возможностью создания различных кофейных напитков (предусмотреть 5 классов наименований)

Выполнение:

Код программы:

```
from abc import ABC, abstractmethod

# Абстрактный класс для напитков
class Coffee(ABC):
    @abstractmethod
    def get_name(self):
        pass

    @abstractmethod
    def get_ingredients(self):
        pass

    @abstractmethod
    def get_cost(self):
        pass

# Конкретные классы напитков
class Espresso(Coffee):
    def get_name(self):
        return "Эспрессо"

    def get_ingredients(self):
        return ["кофе", "вода"]

    def get_cost(self):
        return 2.5

class Americano(Coffee):
    def get_name(self):
        return "Американо"

    def get_ingredients(self):
        return ["кофе", "вода", "горячая вода"]

    def get_cost(self):
        return 3.0

class Cappuccino(Coffee):
    def get_name(self):
        return "Капучино"

    def get_ingredients(self):
        return ["кофе", "вода", "молоко", "пена"]

    def get_cost(self):
        return 4.0

class Latte(Coffee):
    def get_name(self):
        return "Латте"

    def get_ingredients(self):
```

```
        return ["кофе", "вода", "молоко"]

    def get_cost(self):
        return 4.5

class Mocha(Coffee):
    def get_name(self):
        return "Мокка"

    def get_ingredients(self):
        return ["кофе", "вода", "шоколад", "молоко"]

    def get_cost(self):
        return 5.0

# Абстрактный класс фабрики
class CoffeeFactory(ABC):
    @abstractmethod
    def create_coffee(self):
        pass

# Конкретные фабрики для каждого напитка
class EspressoFactory(CoffeeFactory):
    def create_coffee(self):
        return Espresso()

class AmericanoFactory(CoffeeFactory):
    def create_coffee(self):
        return Americano()

class CappuccinoFactory(CoffeeFactory):
    def create_coffee(self):
        return Cappuccino()

class LatteFactory(CoffeeFactory):
    def create_coffee(self):
        return Latte()

class MochaFactory(CoffeeFactory):
    def create_coffee(self):
        return Mocha()

# Класс кофе-автомата
class CoffeeMachine:
    def make_coffee(self, factory):
        coffee = factory.create_coffee()
        print(f"Готовим {coffee.get_name()}...")
        print(f"Ингредиенты: {', '.join(coffee.get_ingredients())}")
        print(f"Стоимость: ${coffee.get_cost()}")
        return coffee

# Словарь для соответствия выбора пользователя и фабрик
COFFEE_TYPES = {
    '1': ('Эспрессо', EspressoFactory()),
    '2': ('Американо', AmericanoFactory()),
    '3': ('Капучино', CappuccinoFactory()),
    '4': ('Латте', LatteFactory()),
    '5': ('Мокка', MochaFactory())
}

# Функция для отображения меню
def show_menu():
```

```
    print("\n=== Меню кофейни ===")
    for key, (name, _) in COFFEE_TYPES.items():
        coffee = COFFEE_TYPES[key][1].create_coffee()
        print(f"{key}. {name} - ${coffee.get_cost()}")
    print("0. Выход")

# Демонстрация работы
def main():
    machine = CoffeeMachine()

    while True:
        show_menu()
        choice = input("\nВыберите кофе (0-5): ").strip()

        if choice == '0':
            print("Спасибо за визит!")
            break

        if choice not in COFFEE_TYPES:
            print("Неверный выбор, попробуйте снова.")
            continue

        coffee_name, factory = COFFEE_TYPES[choice]
        print(f"\nВы выбрали: {coffee_name}")
        machine.make_coffee(factory)

if __name__ == "__main__":
    main()
```

Объяснение выбора паттерна

Подходящий паттерн: Фабричный метод.
Пояснение выбора: Фабричный метод позволяет определить общий интерфейс для создания объектов (напитков), но конкретные классы напитков создаются в подклассах. Это удобно, так как кофе-автомат может "знать", как создавать разные напитки, но сами напитки (их ингредиенты и логика) определяются отдельно. Паттерн упрощает добавление новых видов кофе без изменения основного кода автомата.

Спецификация ввода:
Выберите кофе (0-5): <Выбрать номер>

Пример:
Выберите кофе (0-5): 1

Спецификация вывода:
Вы выбрали: <Выбранный пользователем номер>
Готовим <Выбранный пользователем элемент>
Ингредиенты: <Ингредиенты для выбранного пользователем элемента>
Стоимость: <Сумма выбранных элементов>

Пример:
Вы выбрали: Эспрессо
Готовим Эспрессо...
Ингредиенты: кофе, вода
Стоимость: $2.5

Рисунки с результатами работы программы:

```
/home/twinkle/PycharmProjects/pythonProject/venv/bin/python /home/twinkle/PycharmProjects/pythonProject/lab3.py

=== Меню кофейни ===
1. Эспрессо - $2.5
2. Американо - $3.0
3. Капучино - $4.0
4. Латте - $4.5
5. Мокка - $5.0
0. Выход

Выберите кофе (0-5): 1

Вы выбрали: Эспрессо
Готовим Эспрессо...
Ингредиенты: кофе, вода
Стоимость: $2.5
```

Вторая группа заданий (структурный паттерн):

Проект «Часы». В проекте должен быть реализован класс, который дает возможность пользоваться часами со стрелками так же, как и цифровыми часами. В классе «Часы со стрелками» хранятся повороты стрелок.

Выполнение:

Код программы:

```python
from abc import ABC, abstractmethod

# Интерфейс для часов
class Clock(ABC):
    @abstractmethod
    def get_time(self):
        pass



# Класс цифровых часов
class DigitalClock(Clock):
    def __init__(self, hours, minutes):
        self.hours = hours
        self.minutes = minutes

    def get_time(self):
        return f"{self.hours:02d}:{self.minutes:02d}"



# Класс аналоговых часов (хранит углы стрелок)
class AnalogClock:
    def __init__(self, hour_angle, minute_angle):
        self.hour_angle = hour_angle  # угол часовой стрелки (градусы)
        self.minute_angle = minute_angle  # угол минутной стрелки (градусы)

    def get_angles(self):
        return self.hour_angle, self.minute_angle



# Адаптер для аналоговых часов
class AnalogToDigitalAdapter(Clock):
    def __init__(self, analog_clock):
        self.analog_clock = analog_clock

    def get_time(self):
        # Преобразуем углы в часы и минуты
        hour_angle, minute_angle = self.analog_clock.get_angles()
        # 360 градусов = 12 часов, 1 час = 30 градусов
        hours = int(hour_angle // 30) % 12
        # 360 градусов = 60 минут, 1 минута = 6 градусов
        minutes = int(minute_angle // 6)
```

```
        return f"{hours:02d}:{minutes:02d}"


# Демонстрация работы
def main():
    while True:
        print("\nВведите время для часов (или 'q' для выхода)")
        time_input = input("Формат ЧЧ:ММ (например, 14:45): ").strip()

        if time_input.lower() == 'q':
            print("Выход из программы.")
            break

        try:
            # Проверяем формат времени
            hours, minutes = map(int, time_input.split(':'))
            if not (0 <= hours <= 23 and 0 <= minutes <= 59):
                print("Ошибка: Часы должны быть от 0 до 23, минуты от 0 до 59.")
                continue

            # Создаем цифровые часы
            digital_clock = DigitalClock(hours, minutes)
            print("Цифровые часы:", digital_clock.get_time())

            # Преобразуем время в углы для аналоговых часов
            # Часовая стрелка: 30 градусов за час + 0.5 градуса за минуту
            hour_angle = (hours % 12) * 30 + minutes * 0.5
            # Минутная стрелка: 6 градусов за минуту
            minute_angle = minutes * 6
            analog_clock = AnalogClock(hour_angle, minute_angle)
            # Используем адаптер
            adapted_clock = AnalogToDigitalAdapter(analog_clock)
            print("Аналоговые часы (через адаптер):", adapted_clock.get_time())

        except ValueError:
            print("Ошибка: Неверный формат времени. Используйте ЧЧ:ММ (например, 14:45).")
            continue


if __name__ == "__main__":
    main()
```

Объяснение выбора паттерна

Подходящий паттерн: Адаптер.
Пояснение выбора: Аналоговые часы хранят время в виде углов поворота стрелок (например, часовая стрелка на 30 градусов = 1 час), а цифровые — в формате HH:MM. Адаптер позволяет представить аналоговые часы так, чтобы ими можно было пользоваться, как цифровыми (например, метод get_time() возвращает строку "12:30"). Паттерн конвертирует данные (углы стрелок) в нужный формат, не меняя внутреннюю логику аналоговых часов.

Спецификация ввода:
Формат ЧЧ:ММ (например, 14:45): <Ввод времени>

Пример:
Формат ЧЧ:ММ (например, 14:45): 15:55

Спецификация вывода:

Цифровые часы: <Введенное пользователем значение времени>
Аналоговые часы (через адаптер): <Преобразованное время>

Пример:
Цифровые часы: 15:55
Аналоговые часы (через адаптер): 03:55

Рисунки с результатами работы программы:

```
/home/twinkle/PycharmProjects/pythonProject/venv/bin/python /home/twinkle/PycharmProjects/pythonProject/lab3.py

Введите время для часов (или 'q' для выхода)
Формат ЧЧ:ММ (например, 14:45): 15:55
Цифровые часы: 15:55
Аналоговые часы (через адаптер): 03:55
```

Третья группа заданий (поведенческий паттерн)
Проект «Клавиатура настраиваемого калькулятора». Цифровые и арифметические кнопки имеют фиксированную функцию, а остальные могут менять своё назначение.
Выполнение:
Код программы:

Спецификация ввода:
Введите действие: <Номер действия>
Пример:

```python
from abc import ABC, abstractmethod
import math

# Интерфейс команды
class Command(ABC):
    @abstractmethod
    def execute(self, calculator):
        pass

# Класс калькулятора
class Calculator:
    def __init__(self):
        self.value = 0  # текущее значение

    def set_value(self, value):
        self.value = value
        print(f"Новое значение: {self.value}")

    def add(self, x):
        self.value += x
        print(f"Сложение: {self.value}")

    def multiply(self, x):
        self.value *= x
        print(f"Умножение: {self.value}")

    def sine(self):
        self.value = math.sin(self.value)
        print(f"Синус: {self.value}")

    def power(self, x):
        self.value = self.value ** x
```

```
        print(f"Возведение в степень: {self.value}")

# Конкретные команды
class NumberCommand(Command):
    def __init__(self, number):
        self.number = number

    def execute(self, calculator):
        calculator.set_value(self.number)

class AddCommand(Command):
    def __init__(self, number):
        self.number = number

    def execute(self, calculator):
        calculator.add(self.number)

class MultiplyCommand(Command):
    def __init__(self, number):
        self.number = number

    def execute(self, calculator):
        calculator.multiply(self.number)

class SineCommand(Command):
    def execute(self, calculator):
        calculator.sine()

class PowerCommand(Command):
    def __init__(self, exponent):
        self.exponent = exponent

    def execute(self, calculator):
        calculator.power(self.exponent)

# Класс кнопки
class Button:
    def __init__(self, name):
        self.name = name
        self.command = None  # команда, привязанная к кнопке

    def set_command(self, command):
        self.command = command
        print(f"Кнопке {self.name} назначена команда")

    def press(self, calculator):
        if self.command:
            print(f"Нажимаем кнопку {self.name}...")
            self.command.execute(calculator)
        else:
            print(f"Кнопка {self.name} не настроена")

# Класс клавиатуры
class Keyboard:
    def __init__(self):
        self.buttons = {}

    def add_button(self, name, command=None):
        button = Button(name)
        if command:
            button.set_command(command)
        self.buttons[name] = button
```

```
    def press_button(self, name, calculator):
        if name in self.buttons:
            self.buttons[name].press(calculator)
        else:
            print(f"Кнопка {name} не существует")

# Демонстрация работы
def main():
    calc = Calculator()
    keyboard = Keyboard()

    # Фиксированные кнопки (цифровые и арифметические)
    keyboard.add_button("1", NumberCommand(1))
    keyboard.add_button("5", NumberCommand(5))
    keyboard.add_button("Add_2", AddCommand(2))
    keyboard.add_button("Multiply_3", MultiplyCommand(3))

    # Настраиваемые кнопки
    keyboard.add_button("Func1")
    keyboard.add_button("Func2")

    # Меню для пользователя
    fixed_commands = {
        '1': ('Установить 1', '1'),
        '2': ('Установить 5', '5'),
        '3': ('Сложить 2', 'Add_2'),
        '4': ('Умножить на 3', 'Multiply_3')
    }

    configurable_commands = {
        '1': ('Синус', SineCommand()),
        '2': ('Возвести в степень 2', PowerCommand(2))
    }

    def show_menu():
        print("\n=== Меню калькулятора ===")
        print("Фиксированные команды:")
        for key, (desc, _) in fixed_commands.items():
            print(f"{key}. {desc}")
        print("\nНастраиваемые кнопки:")
        print("5. Назначить команду для Func1")
        print("6. Назначить команду для Func2")
        print("7. Нажать Func1")
        print("8. Нажать Func2")
        print("0. Выход")

    def show_configurable_menu():
        print("\n=== Выберите команду для кнопки ===")
        for key, (desc, _) in configurable_commands.items():
            print(f"{key}. {desc}")

    while True:
        show_menu()
        choice = input("\nВыберите действие (0-8): ").strip()

        if choice == '0':
            print("Выход из программы.")
            break

        if choice in fixed_commands:
            _, button_name = fixed_commands[choice]
```

```
            keyboard.press_button(button_name, calc)
            continue

        if choice in ['5', '6']:
            button_name = "Func1" if choice == '5' else "Func2"
            show_configurable_menu()
            config_choice = input("\nВыберите команду (1-2): ").strip()

            if config_choice in configurable_commands:
                _, command = configurable_commands[config_choice]
                keyboard.buttons[button_name].set_command(command)
            else:
                print("Неверный выбор команды.")
            continue

        if choice in ['7', '8']:
            button_name = "Func1" if choice == '7' else "Func2"
            keyboard.press_button(button_name, calc)
            continue

        print("Неверный выбор, попробуйте снова.")

if __name__ == "__main__":
    main()
```

Объяснение выбора паттерна

Подходящий паттерн: Команда.

Пояснение выбора: Паттерн Команда позволяет привязать к кнопкам разные действия (команды), которые можно менять динамически. Фиксированные кнопки (цифры, арифметика) имеют предопределенные команды, а настраиваемые кнопки могут получать новые команды во время работы. Это удобно для реализации гибкой системы, где пользователь может назначить кнопке любое действие.

Спецификация ввода:

Выберите действие (0-8): <Номер действия>

Пример:

=== Меню калькулятора ===
Фиксированные команды:
1. Установить 1
2. Установить 5
3. Сложить 2
4. Умножить на 3

Настраиваемые кнопки:
5. Назначить команду для Func1
6. Назначить команду для Func2
7. Нажать Func1
8. Нажать Func2
0. Выход

Выберите действие (0-8): 1
Нажимаем кнопку 1...
Новое значение: 1

Выберите действие (0-8): 3
Нажимаем кнопку Add_2...
Сложение: 3

Рисунки с результатами работы программы:

```
/home/twinkle/PycharmProjects/pythonProject/venv/bin/python /home/twinkle/PycharmProjects/pythonProject/lab3.py
Кнопке 1 назначена команда
Кнопке 5 назначена команда
Кнопке Add_2 назначена команда
Кнопке Multiply_3 назначена команда

=== Меню калькулятора ===
Фиксированные команды:
1. Установить 1
2. Установить 5
3. Сложить 2
4. Умножить на 3

Настраиваемые кнопки:
5. Назначить команду для Func1
6. Назначить команду для Func2
7. Нажать Func1
8. Нажать Func2
0. Выход

Выберите действие (0-8): 1
Нажимаем кнопку 1...
Новое значение: 1

=== Меню калькулятора ===
Фиксированные команды:
1. Установить 1
2. Установить 5
3. Сложить 2
4. Умножить на 3

Настраиваемые кнопки:
5. Назначить команду для Func1
6. Назначить команду для Func2
7. Нажать Func1
8. Нажать Func2
0. Выход
```

```
Настраиваемые кнопки:
5. Назначить команду для Func1
6. Назначить команду для Func2
7. Нажать Func1
8. Нажать Func2
0. Выход

Выберите действие (0-8): 7
Нажимаем кнопку Func1...
Синус: 0.1411200080598672

=== Меню калькулятора ===
Фиксированные команды:
1. Установить 1
2. Установить 5
3. Сложить 2
4. Умножить на 3

Настраиваемые кнопки:
5. Назначить команду для Func1
6. Назначить команду для Func2
7. Нажать Func1
8. Нажать Func2
0. Выход

Выберите действие (0-8): 0
Выход из программы.
```

Вывод: приобрел навыки применения паттернов проектирования при решении практических задач с использованием языка Python.