

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
Д. М. Андросюк
студент группы ПО11

Проверил
А. А. Крощенко
ст. преп. кафедры ИИТ,
15.03.2025 г.

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

Ход Работы

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.

Задание 1

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

-Паттерн “Фабричный метод”. Этот паттерн позволяет создавать объекты различных типов, не указывая конкретный класс объекта, который будет создан.

Код программы:

```
from abc import ABC, abstractmethod

class Coffee(ABC):
    @abstractmethod
    def prepare(self):
        pass

class Espresso(Coffee):
    def prepare(self):
        return "Espresso: крепкий кофе без молока!"

class Latte(Coffee):
    def prepare(self):
        return "Latte: эспрессо с молоком и молочной пенкой!"

class Cappuccino(Coffee):
    def prepare(self):
        return "Cappuccino: эспрессо с большим количеством молочной пенки!"

class Americano(Coffee):
    def prepare(self):
        return "Americano: эспрессо с добавлением горячей воды!"

class Mocha(Coffee):
    def prepare(self):
        return "Mocha: эспрессо с шоколадом и молоком!"

class CoffeeFactory(ABC):
    @abstractmethod
    def create_coffee(self) -> Coffee:
        pass
```

```

class EspressoFactory(CoffeeFactory):
    def create_coffee(self) -> Coffee:
        return Espresso()

class LatteFactory(CoffeeFactory):
    def create_coffee(self) -> Coffee:
        return Latte()

class CappuccinoFactory(CoffeeFactory):
    def create_coffee(self) -> Coffee:
        return Cappuccino()

class AmericanoFactory(CoffeeFactory):
    def create_coffee(self) -> Coffee:
        return Americano()

class MochaFactory(CoffeeFactory):
    def create_coffee(self) -> Coffee:
        return Mocha()

class CoffeeMachine:
    def __init__(self):
        self.factories = {
            "1": EspressoFactory(),
            "2": LatteFactory(),
            "3": CappuccinoFactory(),
            "4": AmericanoFactory(),
            "5": MochaFactory(),
        }

    def make_coffee(self, choice):
        factory = self.factories.get(choice)
        if factory:
            coffee = factory.create_coffee()
            print(coffee.prepare())
        else:
            print("Ошибка.")

    def display_menu():
        print("Выберите кофе:")
        print("1. Espresso")
        print("2. Latte")
        print("3. Cappuccino")
        print("4. Americano")
        print("5. Mocha")

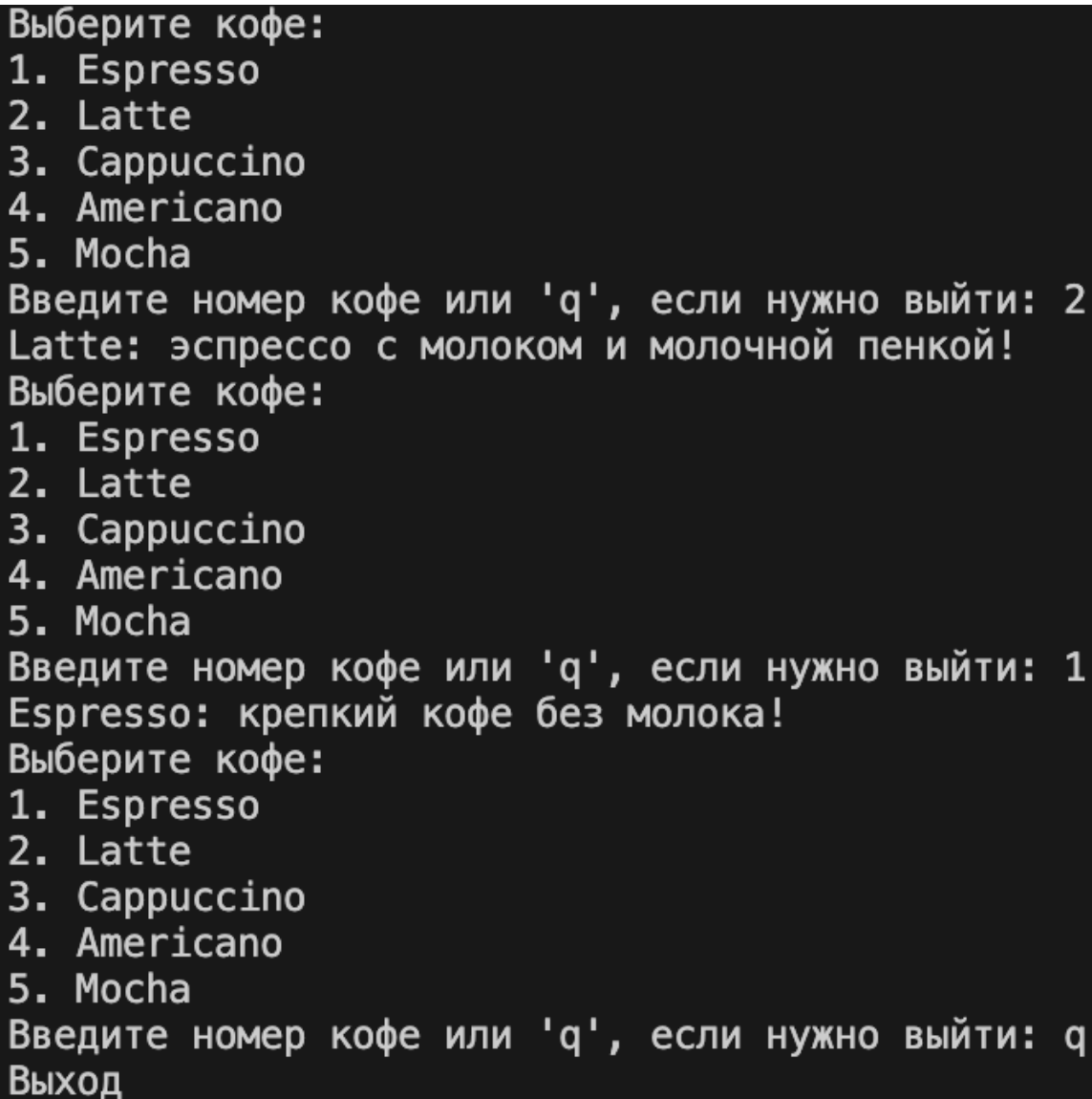
if __name__ == "__main__":
    coffee_machine = CoffeeMachine()

    while True:
        display_menu()
        choice = input("Введите номер кофе или 'q', если нужно выйти: ").strip()

```

```
if choice.lower() == 'q':  
    print("Выход")  
    break  
  
coffee_machine.make_coffee(choice)
```

Рисунки с результатами работы программы:



```
Выберите кофе:  
1. Espresso  
2. Latte  
3. Cappuccino  
4. Americano  
5. Mocha  
Введите номер кофе или 'q', если нужно выйти: 2  
Latte: эспрессо с молоком и молочной пенкой!  
Выберите кофе:  
1. Espresso  
2. Latte  
3. Cappuccino  
4. Americano  
5. Mocha  
Введите номер кофе или 'q', если нужно выйти: 1  
Espresso: крепкий кофе без молока!  
Выберите кофе:  
1. Espresso  
2. Latte  
3. Cappuccino  
4. Americano  
5. Mocha  
Введите номер кофе или 'q', если нужно выйти: q  
Выход
```

Задание 2

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.

-Паттерн «Адаптер». Этот паттерн позволяет объектам с несовместимыми интерфейсами работать вместе.

Код программы:

```
class DigitalClock:  
    def __init__(self, hours, minutes, seconds):  
        self.hours = hours
```

```
self.minutes = minutes
self.seconds = seconds
```

```
def display_time(self):
    return f"Цифровые часы: {self.hours:02d}:{self.minutes:02d}:{self.seconds:02d}"
```

```
class AnalogClock:
```

```
    def __init__(self, hour_angle, minute_angle, second_angle):
        self.hour_angle = hour_angle # Поворот часовой стрелки (0-360 градусов)
        self.minute_angle = minute_angle # Поворот минутной стрелки (0-360 градусов)
        self.second_angle = second_angle # Поворот секундной стрелки (0-360 градусов)
```

```
    def display_time(self):
        return f"Часы со стрелками: Часовая стрелка: {self.hour_angle}°, Минутная стрелка: {self.minute_angle}°, Секундная стрелка: {self.second_angle}°"
```

```
class DigitalToAnalogAdapter(AnalogClock):
```

```
    def __init__(self, digital_clock: DigitalClock):
        hours = digital_clock.hours % 12
        minutes = digital_clock.minutes
        seconds = digital_clock.seconds
```

```
        hour_angle = (hours * 30) + (minutes * 0.5) # 30° на час + 0.5° на минуту
        minute_angle = minutes * 6 # 6° на минуту
        second_angle = seconds * 6 # 6° на секунду
```

```
        super().__init__(hour_angle, minute_angle, second_angle)
```

```
def get_time_from_user():
```

```
    while True:
```

```
        time_input = input("Введите время в формате ЧЧ:ММ:СС или 'q', если нужно выйти: ").strip()
```

```
        if time_input.lower() == 'q':
            return None
```

```
        hours, minutes, seconds = map(int, time_input.split(':'))
        if 0 <= hours < 24 and 0 <= minutes < 60 and 0 <= seconds < 60:
            return hours, minutes, seconds
        else:
            print("Ошибка.")
```

```
if __name__ == "__main__":
```

```
    while True:
```

```
        time_data = get_time_from_user()
```

```
        if time_data is None:
            print("Выход")
```

```

break

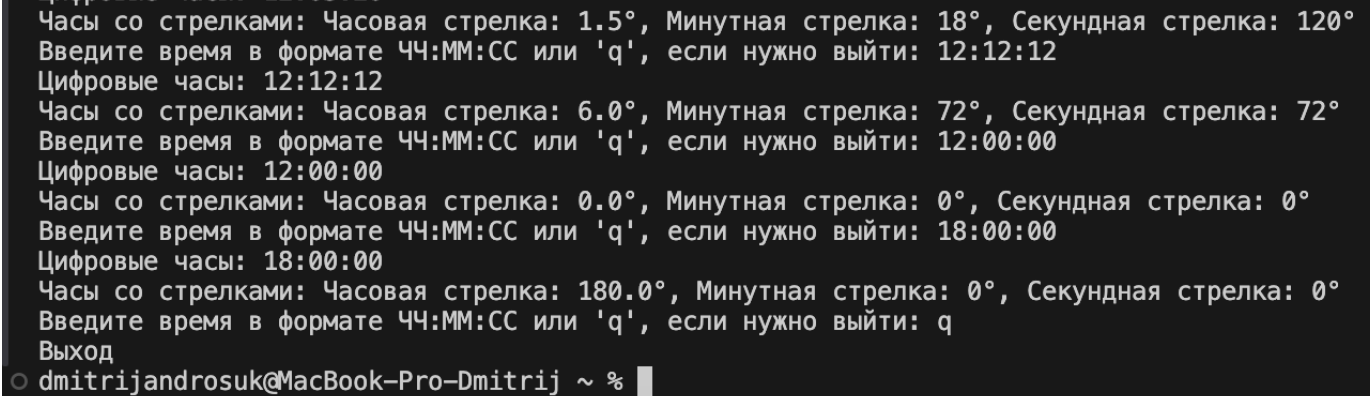
hours, minutes, seconds = time_data
digital_clock = DigitalClock(hours, minutes, seconds)

analog_clock = DigitalToAnalogAdapter(digital_clock)

print(digital_clock.display_time())
print(analog_clock.display_time())

```

Рисунки с результатами работы программы:



```

Часы со стрелками: Часовая стрелка: 1.5°, Минутная стрелка: 18°, Секундная стрелка: 120°
Введите время в формате ЧЧ:ММ:СС или 'q', если нужно выйти: 12:12:12
Цифровые часы: 12:12:12
Часы со стрелками: Часовая стрелка: 6.0°, Минутная стрелка: 72°, Секундная стрелка: 72°
Введите время в формате ЧЧ:ММ:СС или 'q', если нужно выйти: 12:00:00
Цифровые часы: 12:00:00
Часы со стрелками: Часовая стрелка: 0.0°, Минутная стрелка: 0°, Секундная стрелка: 0°
Введите время в формате ЧЧ:ММ:СС или 'q', если нужно выйти: 18:00:00
Цифровые часы: 18:00:00
Часы со стрелками: Часовая стрелка: 180.0°, Минутная стрелка: 0°, Секундная стрелка: 0°
Введите время в формате ЧЧ:ММ:СС или 'q', если нужно выйти: q
Выход
○ dmitrijandrosuk@MacBook-Pro-Dmitrij ~ %

```

Задание 3

Проект «Клавиатура настраиваемого калькулятора». Цифровые и арифметические кнопки имеют фиксированную функцию, а остальные могут менять своё назначение.

-Паттерн “Стратегия”. Этот паттерн позволяет динамически изменять поведение объектов, инкапсулируя алгоритмы в отдельные классы и делая их взаимозаменяемыми.

Код программы:

```

from abc import ABC, abstractmethod

class Button(ABC):
    @abstractmethod
    def press(self):
        pass

class FixedButton(Button):
    def __init__(self, label, action):
        self.label = label
        self.action = action

    def press(self):
        print(f"Нажата кнопка '{self.label}': {self.action}")
        return self.action

class ConfigurableButton(Button):
    def __init__(self, label):
        self.label = label
        self.strategy = None

```

```

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

def press(self):
    if self.strategy:
        result = self.strategy.execute()
        print(f'Нажатая кнопка '{self.label}': {result}')
        return result
    else:
        print(f'Кнопка '{self.label}' не настроена.")
        return None

class Strategy(ABC):
    @abstractmethod
    def execute(self):
        pass

class ClearStrategy(Strategy):
    def execute(self):
        return "Очистка экрана..."

class MemorySaveStrategy(Strategy):
    def execute(self):
        return "Сохранение в память..."

class MemoryRecallStrategy(Strategy):
    def execute(self):
        return "Извлечение из памяти..."

class SquareRootStrategy(Strategy):
    def execute(self):
        return "Вычисление квадратного корня..."

class PercentageStrategy(Strategy):
    def execute(self):
        return "Вычисление процента..."

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

    def add_button(self, name, button):
        self.buttons[name] = button

    def press_button(self, name):
        if name in self.buttons:
            return self.buttons[name].press()
        else:
            print(f'Кнопка '{name}' не найдена.")
            return None

    def list_buttons(self):
        print("Доступные кнопки:")
        for name in self.buttons:
            print(f'- {name}')

```

```

def display_menu():
    print("\nМеню:")
    print("1. Нажать кнопку")
    print("2. Назначить функцию настраиваемой кнопке")
    print("3. Список кнопок")
    print("4. Выйти")

# Основная логика программы
if __name__ == "__main__":
    keyboard = Keyboard()

    # Добавляем фиксированные кнопки
    keyboard.add_button("1", FixedButton("1", "Ввод цифры 1"))
    keyboard.add_button("2", FixedButton("2", "Ввод цифры 2"))
    keyboard.add_button("+", FixedButton("+", "Сложение"))
    keyboard.add_button("-", FixedButton("-", "Вычитание"))

    config_button_a = ConfigurableButton("A")
    config_button_b = ConfigurableButton("B")

    keyboard.add_button("A", config_button_a)
    keyboard.add_button("B", config_button_b)

    strategies = {
        "1": ClearStrategy(),
        "2": MemorySaveStrategy(),
        "3": MemoryRecallStrategy(),
        "4": SquareRootStrategy(),
        "5": PercentageStrategy(),
    }

    while True:
        display_menu()
        choice = input("Выберите действие: ").strip()

        if choice == "1":
            button_name = input("Введите название кнопки: ").strip()
            keyboard.press_button(button_name)

        elif choice == "2":
            button_name = input("Введите название настраиваемой кнопки (A или B): ").strip().upper()
            if button_name in ["A", "B"]:
                print("Доступные функции:")
                for key, strategy in strategies.items():
                    print(f"{key}. {strategy.execute()}")
                strategy_choice = input("Выберите функцию (1-5): ").strip()
                if strategy_choice in strategies:
                    if button_name == "A":
                        config_button_a.set_strategy(strategies[strategy_choice])
                    elif button_name == "B":
                        config_button_b.set_strategy(strategies[strategy_choice])
                    print(f"Функция назначена на кнопку '{button_name}'.")
                else:
                    print("Неверный выбор функции.")

```



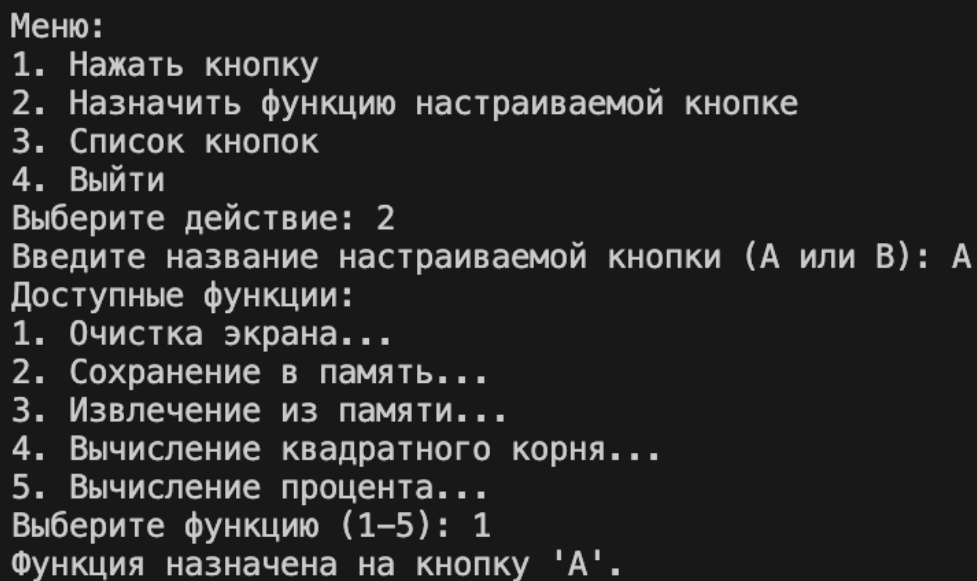
```
else:
    print("Неверное название кнопки.")

elif choice == "3":
    keyboard.list_buttons()

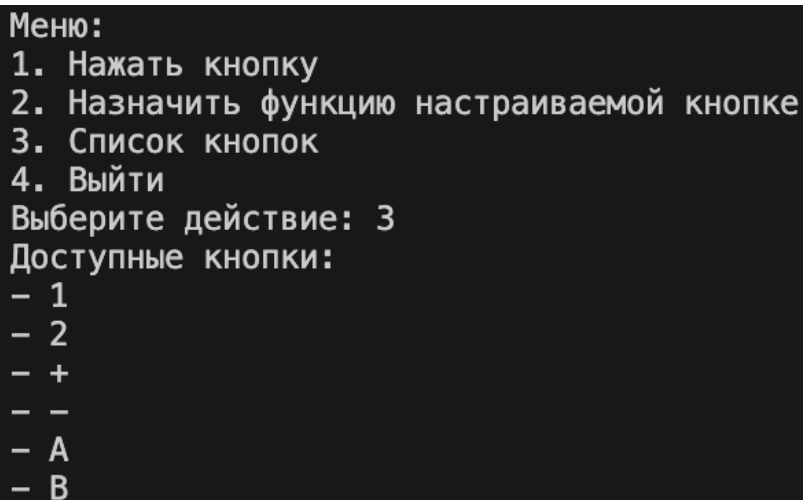
elif choice == "4":
    print("Выход")
    break

else:
    print("Ошибка.")
```

Рисунки с результатами работы программы:



```
Меню:
1. Нажать кнопку
2. Назначить функцию настраиваемой кнопке
3. Список кнопок
4. Выйти
Выберите действие: 2
Введите название настраиваемой кнопки (А или В): А
Доступные функции:
1. Очистка экрана...
2. Сохранение в память...
3. Извлечение из памяти...
4. Вычисление квадратного корня...
5. Вычисление процента...
Выберите функцию (1-5): 1
Функция назначена на кнопку 'А'.
```



```
Меню:
1. Нажать кнопку
2. Назначить функцию настраиваемой кнопке
3. Список кнопок
4. Выйти
Выберите действие: 3
Доступные кнопки:
- 1
- 2
- +
- -
- А
- В
```

Меню:

1. Нажать кнопку
2. Назначить функцию настраиваемой кнопке
3. Список кнопок
4. Выйти

Выберите действие: 1

Введите название кнопки: A

Нажата кнопка 'A': Очистка экрана...

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