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
С. С. Жватель
студент группы ПО11

Проверил
А. А. Крощенко
ст. преп. кафедры ИИТ,
12.04.2025 г.

Брест 2025

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.

Задание 1. Первая группа заданий (порождающий паттерн)

Проект «Бургер-закусочная». Реализовать возможность формирования заказа из определенных позиций (тип бургера (веганский, куриный и т.д.)), напиток (холодный – пепси, кока-кола и т.д.; горячий – кофе, чай и т.д.), тип упаковки – с собой, на месте. Должна формироваться итоговая стоимость заказа.

Выполнение:

Код программы:

```
"""Module for managing burger orders using the Builder pattern."""



class Burger:
    """Class representing a burger order with type, drink, and packaging."""

    def __init__(self):
        self.burger_type = None
        self.drink = None
        self.packaging = None

    def __str__(self):
        """Return string representation of the burger order."""
        return f"Burger: {self.burger_type}, Drink: {self.drink}, Packaging: {self.packaging}"

    def cost(self):
        """Calculate total cost of the burger order."""
        cost_map = {
            "vegan": 10,
            "chicken": 12,
            "pepsi": 3,
            "cola": 3,
            "coffee": 4,
            "tea": 4,
            "takeaway": 1,
            "eat_in": 0,
        }
        burger_cost = cost_map.get(self.burger_type, 0)
        drink_cost = cost_map.get(self.drink, 0)
        packaging_cost = cost_map.get(self.packaging, 0)
        return burger_cost + drink_cost + packaging_cost
```

```
class BurgerBuilder:
    """Builder class for constructing a Burger object step-by-step."""

    def __init__(self):
        self.burger = Burger()

    def set_burger_type(self, burger_type):
        """Set the burger type."""
        self.burger.burger_type = burger_type
        return self

    def set_drink(self, drink):
        """Set the drink type."""
        self.burger.drink = drink
        return self

    def set_packaging(self, packaging):
        """Set the packaging type."""
        self.burger.packaging = packaging
        return self

    def build(self):
        """Return the constructed Burger object."""
        return self.burger


def create_order():
    """Prompt user to create a burger order interactively."""
    builder = BurgerBuilder()

    print("Available burgers: vegan, chicken")
    while True:
        burger_type = input("Choose burger type: ").lower()
        if burger_type in ["vegan", "chicken"]:
            builder.set_burger_type(burger_type)
            break
        print("Invalid choice. Try again.")

    print("Available drinks: pepsi, cola, coffee, tea")
    while True:
        drink = input("Choose drink: ").lower()
        if drink in ["pepsi", "cola", "coffee", "tea"]:
            builder.set_drink(drink)
            break
        print("Invalid choice. Try again.")

    print("Available packaging: takeaway, eat_in")
    while True:
        packaging = input("Choose packaging: ").lower()
```

```
        if packaging in ["takeaway", "eat_in"]:
            builder.set_packaging(packaging)
            break
        print("Invalid choice. Try again.")

    return builder.build()


if __name__ == "__main__":
    order = create_order()
    print(order)
    print(f"Total cost: ${order.cost()}")
```

Спецификация ввода:

- Тип бургера (vegan, chicken).
- Напиток (pepsi, cola, coffee, tea).
- Упаковка (takeaway, eat_in).

Пример:

```
Available burgers: vegan, chicken
Choose burger type: vegan
Available drinks: pepsi, cola, coffee, tea
Choose drink: coffee
Available packaging: takeaway, eat_in
Choose packaging: takeaway
```

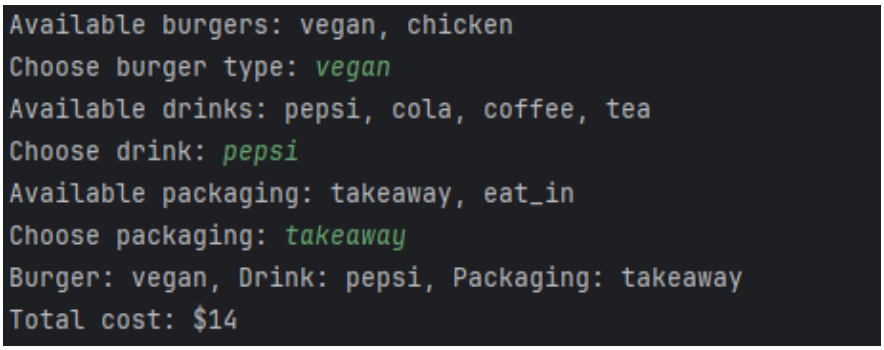
Спецификация вывода:

- Описание заказа: тип бургера, напиток, упаковка.
- Итоговая стоимость заказа.

Пример:

```
Burger: vegan, Drink: coffee, Packaging: takeaway
Total cost: $15
```

Рисунки с результатами работы программы:



Задание 2. 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:

```
"""Module for adapting analog clock to digital clock using the Adapter pattern."""
```

```
from abc import ABC, abstractmethod


class DigitalClock(ABC):
    """Abstract base class for digital clocks."""

    # pylint: disable=too-few-public-methods

    @abstractmethod
    def get_time(self):
        """Return the time in digital format (hh:mm)."""


class AnalogClock:
    """Class representing an analog clock with hour and minute hand angles."""

    # pylint: disable=too-few-public-methods

    def __init__(self, hour_angle, minute_angle):
        self.hour_angle = hour_angle
        self.minute_angle = minute_angle

    def get_angles(self):
        """Return the hour and minute hand angles."""
        return self.hour_angle, self.minute_angle


class AnalogToDigitalAdapter(DigitalClock):
    """Adapter to convert analog clock to digital clock interface."""

    # pylint: disable=too-few-public-methods

    def __init__(self, analog):
        self.analog_clock = analog

    def get_time(self):
        """Convert analog angles to digital time format (hh:mm)."""
        hour_angle, minute_angle = self.analog_clock.get_angles()
        hours = int((hour_angle % 360) / 30)
        minutes = int((minute_angle % 360) / 6)
        return f"{hours:02d}:{minutes:02d}"


def create_clock():
    """Prompt user to input angles for an analog clock."""
    while True:
        try:
            hour_angle = float(input("Enter hour hand angle (0-360): "))
            if 0 <= hour_angle <= 360:
```

```
            break
        print("Angle must be between 0 and 360.")
    except ValueError:
        print("Invalid input. Enter a number.")

while True:
    try:
        minute_angle = float(input("Enter minute hand angle (0-360): "))
        if 0 <= minute_angle <= 360:
            break
        print("Angle must be between 0 and 360.")
    except ValueError:
        print("Invalid input. Enter a number.")

return AnalogClock(hour_angle, minute_angle)


if __name__ == "__main__":
    clock = create_clock()
    adapter = AnalogToDigitalAdapter(clock)
    print(f"Time: {adapter.get_time()}")
```

Спецификация ввода:

- Угол поворота часовой стрелки (0–360 градусов).
- Угол поворота минутной стрелки (0–360 градусов).

Пример:

Enter hour hand angle (0-360): 90
Enter minute hand angle (0-360): 180

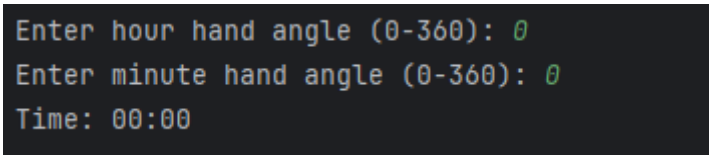
Спецификация вывода:

Время в цифровом формате (hh:mm).

Пример:

Time: 03:30

Рисунки с результатами работы программы:



Задание 3. Третья группа заданий (поведенческий паттерн)

Шифрование текстового файла. Реализовать класс-шифровщик текстового файла с поддержкой различных алгоритмов шифрования. Возможные варианты шифрования: удаление всех гласных букв из текста, изменение букв текста на буквы, получаемые фиксированным сдвигом из алфавита (например, шифром буквы а будет являться буква д для сдвига 4 и т.д.), применение операции исключающее или с заданным ключом.

Выполнение:

Код программы:

```
"""Module for encrypting text files using the Strategy pattern."""

import os
from abc import ABC, abstractmethod


class EncryptionStrategy(ABC):
    """Abstract base class for encryption strategies."""

    # pylint: disable=too-few-public-methods

    @abstractmethod
    def encrypt(self, text):
        """Encrypt the input text."""


class RemoveVowelsStrategy(EncryptionStrategy):
    """Strategy to encrypt text by removing vowels."""

    # pylint: disable=too-few-public-methods

    def encrypt(self, text):
        """Remove all vowels from the input text."""
        vowels = set("aeiouAEIOU")
        return "".join(char for char in text if char not in vowels)


class CaesarCipherStrategy(EncryptionStrategy):
    """Strategy to encrypt text using Caesar cipher with a shift."""

    # pylint: disable=too-few-public-methods

    def __init__(self, shift=4):
        self.shift = shift

    def encrypt(self, text):
        """Apply Caesar cipher with specified shift to the input text."""
        result = ""
        for char in text:
            if char.isalpha():
                ascii_offset = ord("A") if char.isupper() else ord("a")
                result += chr((ord(char) - ascii_offset + self.shift) % 26 + ascii_offset)
            else:
                result += char
        return result


class XORCipherStrategy(EncryptionStrategy):
    """Strategy to encrypt text using XOR with a key."""
```

```
    # pylint: disable=too-few-public-methods

    def __init__(self, key="secret"):
        self.key = key

    def encrypt(self, text):
        """Apply XOR encryption with the specified key to the input text."""
        key_bytes = self.key.encode()
        text_bytes = text.encode()
        result = bytearray()
        for i, byte in enumerate(text_bytes):
            result.append(byte ^ key_bytes[i % len(key_bytes)])
        return result.decode("latin1")


class Encryptor:
    """Class to manage encryption using a specified strategy."""

    # pylint: disable=too-few-public-methods

    def __init__(self, strategy: EncryptionStrategy):
        self.strategy = strategy

    def set_strategy(self, strategy: EncryptionStrategy):
        """Set the encryption strategy."""
        self.strategy = strategy

    def encrypt(self, text):
        """Encrypt the text using the current strategy."""
        return self.strategy.encrypt(text)


def run_encryption():
    """Prompt user to encrypt a text file using a chosen strategy."""
    while True:
        input_file = input("Enter input file name (e.g., input.txt): ")
        if os.path.isfile(input_file):
            break
        print("File does not exist. Try again.")

    try:
        with open(input_file, "r", encoding="utf-8") as f:
            text = f.read()
    except (FileNotFoundError, IOError) as e:
        print(f"Error reading file: {e}")
        return

    print("Available encryption methods:")
```

```
    print("1. Remove vowels")
    print("2. Caesar cipher")
    print("3. XOR cipher")

    while True:
        choice = input("Choose method (1-3): ")
        if choice in ["1", "2", "3"]:
            break
        print("Invalid choice. Try again.")

    encryptor = Encryptor(RemoveVowelsStrategy())

    if choice == "1":
        encryptor.set_strategy(RemoveVowelsStrategy())
    elif choice == "2":
        while True:
            try:
                shift = int(input("Enter shift for Caesar cipher (1-25): "))
                if 1 <= shift <= 25:
                    encryptor.set_strategy(CaesarCipherStrategy(shift))
                    break
                print("Shift must be between 1 and 25.")
            except ValueError:
                print("Invalid input. Enter a number.")
    elif choice == "3":
        key = input("Enter key for XOR cipher: ")
        encryptor.set_strategy(XORCipherStrategy(key if key else "secret"))

    encrypted_text = encryptor.encrypt(text)

    output_file = input("Enter output file name (e.g., output.txt): ")

    try:
        with open(output_file, "w", encoding="utf-8") as f:
            f.write(encrypted_text)
        print(f"Encrypted text written to {output_file}")
        print(f"Encrypted text: {encrypted_text}")
    except (FileNotFoundError, IOError) as e:
        print(f"Error writing file: {e}")


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

Спецификация ввода:

- Имя входного файла (например, input.txt).
- Метод шифрования (1 – удаление гласных, 2 – шифр Цезаря, 3 – XOR).
- Для шифра Цезаря: величина сдвига (1–25).
- Для XOR: ключ шифрования.
- Имя выходного файла (например, output.txt).

Пример:

Enter input file name (e.g., input.txt): input.txt
Available encryption methods:
1. Remove vowels
2. Caesar cipher
3. XOR cipher
Choose method (1-3): 2
Enter shift for Caesar cipher (1-25): 4
Enter output file name (e.g., output.txt): output.txt

Спецификация вывода:

- Подтверждение записи зашифрованного текста в файл.
- Зашифрованный текст.

Пример:

Encrypted text written to output.txt
Encrypted text: Lipps, Asvph!

Рисунки с результатами работы программы:

```
Enter input file name (e.g., input.txt): input.txt
Available encryption methods:
1. Remove vowels
2. Caesar cipher
3. XOR cipher
Choose method (1-3): 3
Enter key for XOR cipher: secret
Enter output file name (e.g., output.txt): output.txt
Encrypted text written to output.txt
Encrypted text: 9ZYY^'[ZVY2!
```

Вывод: закрепил знания паттернов Python при решении практических задач