Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Лабораторна робота №7

Технології розроблення програмного забезпечення

ШАБЛОН «MEDIATOR», «FACADE», «BRIDGE», «TEMPLATE METHOD»

Варіант 24

Виконала:
студентка групи ІА-14
Шеліхова А.О

Перевірив:
Мягкий М.Ю.

Київ 2023

Тема: ШАБЛОН «MEDIATOR», «FACADE», «BRIDGE», «TEMPLATE METHOD»

Завдання:

1. Ознайомитися з короткими теоретичними відомостями.

2. Реалізувати частину функціоналу робочої програми у вигляді класів та їхньої взаємодії для досягнення конкретних функціональних можливостей.

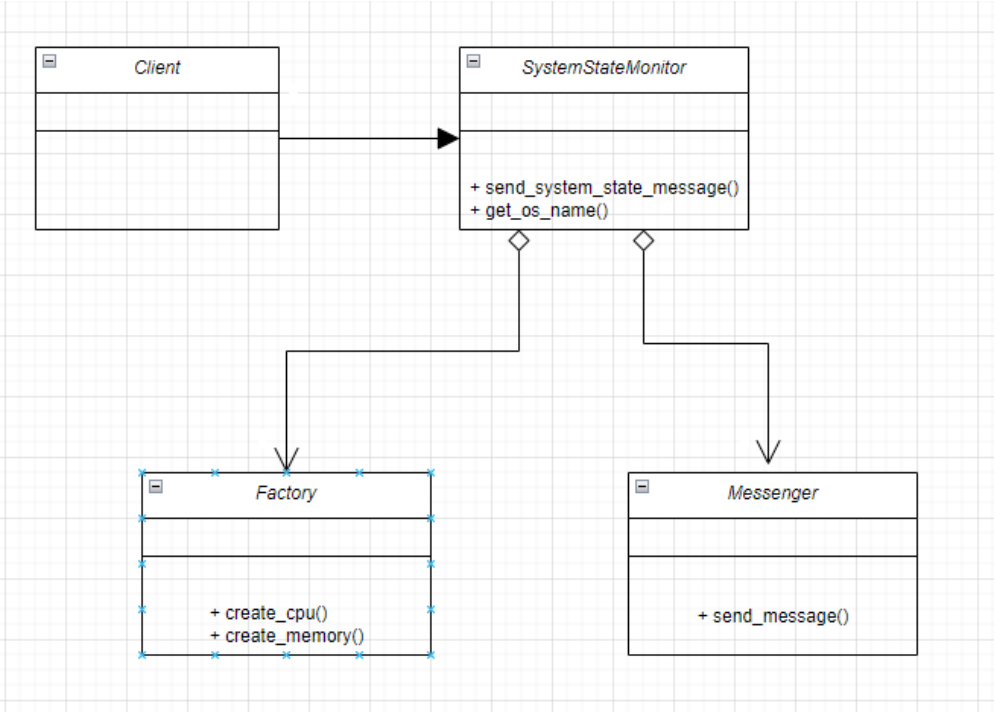
3. Застосування одного з розглянутих шаблонів при реалізації програми.

Виконання роботи:

У ході роботи було реалізовано програму для регулярної відправки повідомлень про стан системи (CPU та оперативної пам'яті) у вибраний мессенджер в залежності від операційної системи, на якій запускається програма.

Функціонал для отримання показників про стан системи та відправлення повідомлення у вибраний мессенджер було винесено в окремий метод окремого класу.

Шаблон “Facade”:



Код програми:

Код, який реалізує шаблон “Facade”:

```
from factories import FACTORIES
```

```
class SystemStateMonitor:

    def __init__(self, factory, messenger):

        self.__factory = factory
        self.__messenger = messenger

    def send_system_state_message(self, api_key, receiver):

        cpu, memory = self.__factory.create_cpu(), self.__factory.create_memory()

        message = f"System information ({self.get_os_name()}):\n\n" + \
            f"CPU Temperature: {cpu.get_temperature()} °C\n" + \
            f"CPU Usage: {cpu.get_load() * 100}%\n\n" + \
            f"Total Memory: {memory.get_total() / 1024 / 1024 / 1024:.3f} GB\n" + \
            f"Used Memory: {memory.get_used() / 1024 / 1024 / 1024:.3f} GB\n" + \
            f"Free Memory: {memory.get_free() / 1024 / 1024 / 1024:.3f} GB"

        return self.__messenger.send_message(api_key, receiver, message)


    def get_os_name(self):

        for _os in FACTORIES.values():
            if isinstance(self.__factory, _os['factory']):
                return _os['name']

        return None
```

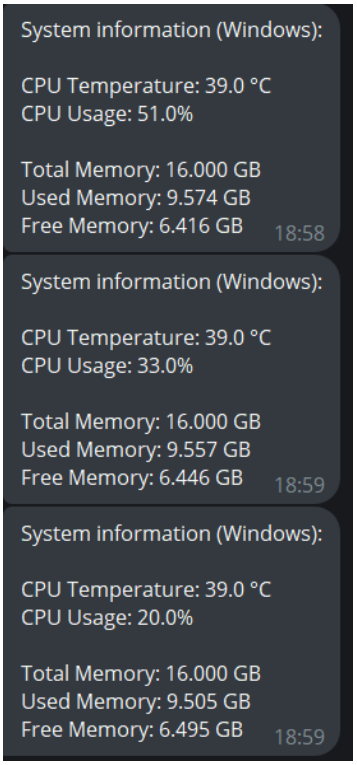
Головний код програми:

```
import os

from datetime import datetime as dt

from apscheduler.schedulers.background import BackgroundScheduler

from messengers import MESSENGERS
from factories import FACTORIES
from monitor import SystemStateMonitor
from config import API_KEYS, SEND_MESSAGE_EVERY


def execute(function):

    def executor():

        success = function()

        print(f"[{dt.now().strftime('%d-%m-%Y %H:%M:%S')}] {'Success' if success else 'Failed'}.")

    return executor


def main():

    success = False

    messengers = list(MESSENGERS.keys())

    while not success:

        try:

            print("\nChoose messenger:\n\t" + '\n\t'.join([str(i + 1) + ': ' + messengers[i] for i in
range(len(messengers))]))

            messenger_index = int(input("Messenger: ")) - 1

            try:
```

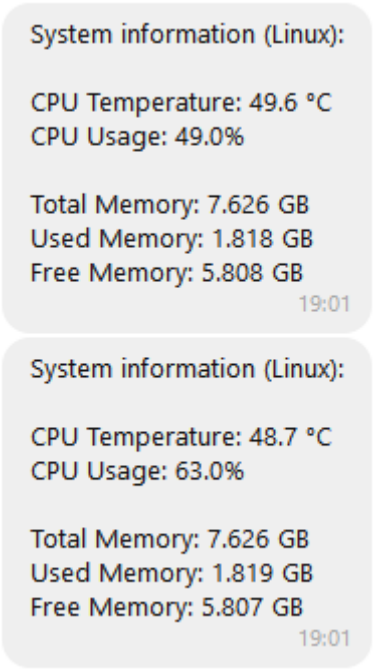
```
                messenger = messengers[messenger_index]
            except IndexError:
                print("\nInvalid messenger\n")
                continue

            receiver = input("Enter receiver: ")

            os_name = os.name

            factory = FACTORIES[os_name]['factory']()

            monitor = SystemStateMonitor(factory, MESSENGERS[messenger])

            print(f"\nOperating system: {monitor.get_os_name()}\n")

            scheduler = BackgroundScheduler()

            scheduler.add_job(
                func=execute(lambda: monitor.send_system_state_message(API_KEYS[messenger], receiver)),
                trigger="interval",
                seconds=SEND_MESSAGE_EVERY
            )

            scheduler.start()

            success = True

        except Exception as exception:
            print(f"\n{type(exception).__name__}: {exception}\n")

    print("\nSuccess!\n")

    while True:
        pass


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

Результати:

Повідомлення про стан системи Windows у Telegram:



Повідомлення про стан системи Linux у Viber:



Висновки:

При виконанні даної лабораторної роботи було вивчено шаблон проектування «Facade». У ході роботи було реалізовано програму для відправлення повідомлень про стан системи у вибраний мессенджер в залежності від операційної системи. Функціонал для отримання показників про стан системи та відправлення повідомлення у вибраний мессенджер було винесено в окремий метод окремого класу.