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 №6

Технології розроблення програмного забезпечення

ШАБЛОНИ «Abstract Factory», «Factory Method», «Memento», «Observer», «Decorator»

Варіант 24

Виконала:
студентка групи ІА-14
Шеліхова А.О

Перевірив:
Мягкий М.Ю.

Київ 2023

Тема: ШАБЛОНИ «Abstract Factory», «Factory Method», «Memento», «Observer», «Decorator»

Завдання:

1. Ознайомитися з короткими теоретичними відомостями.

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

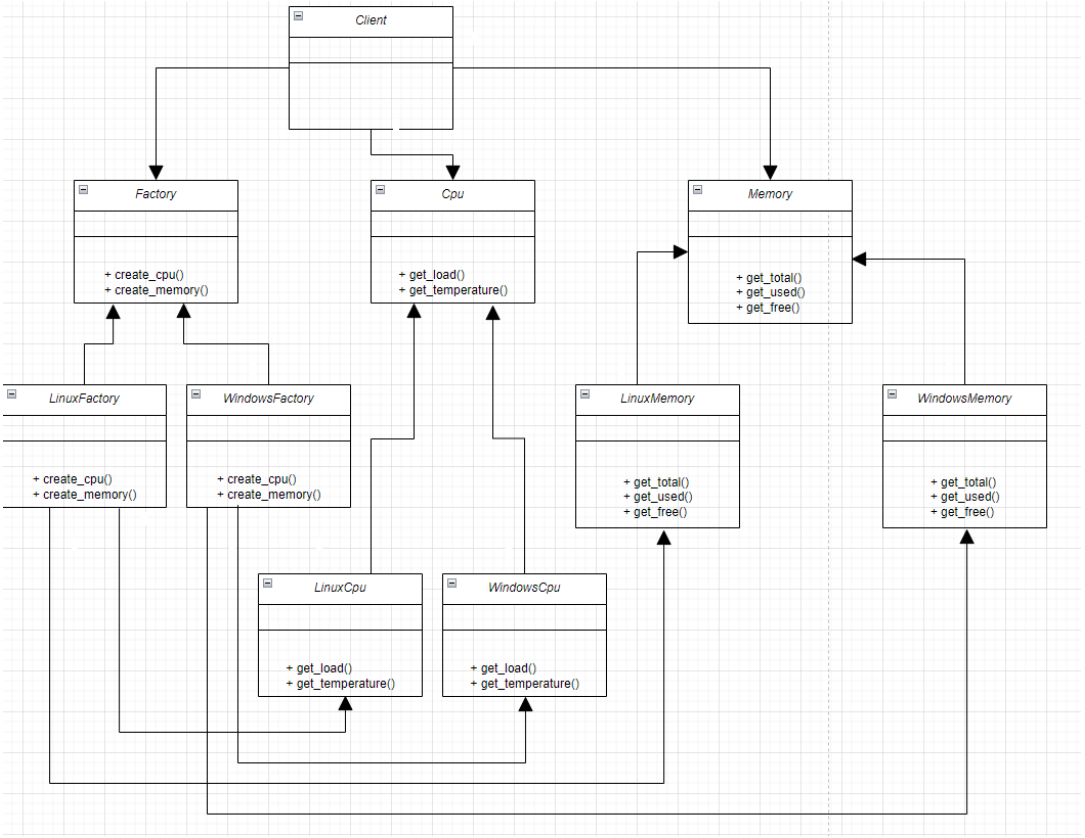
3. Застосування одного з розглянутих шаблонів при реалізації програми.

Виконання роботи:

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

Визначення операційної системи реалізовано за допомогою шаблону “Abstract Factory”.

Шаблон “Abstract Factory”:



Код програми:

Код, який реалізує шаблон “Abstract Factory”:

```
import re
import subprocess

from abc import ABC, abstractmethod


class Cpu(ABC):

    @abstractmethod
    def get_load(self):
        pass

    @abstractmethod
    def get_temperature(self):
        pass


class Memory(ABC):

    @abstractmethod
    def get_total(self):
        pass

    @abstractmethod
    def get_used(self):
        pass

    @abstractmethod
    def get_free(self):
        pass


class Factory(ABC):

    @abstractmethod
    def create_cpu(self):
        pass

    @abstractmethod
    def create_memory(self):
        pass


class LinuxCpu(Cpu):

    def get_load(self):

        command = "cat /proc/loadavg"

        output = subprocess.run(command, stdout=subprocess.PIPE, shell=True, text=True).stdout

        load = float(output.split()[0].strip())

        return load

    def get_temperature(self):

        command = "vcgencmd measure_temp"

        output = subprocess.run(command, stdout=subprocess.PIPE, shell=True, text=True).stdout

        temperature = float(re.search(r'\d+\.\d+', output).group())

        return temperature
```

```
class LinuxMemory(Memory):

    def get_total(self):

        command = "free -b"

        output = subprocess.run(command, stdout=subprocess.PIPE, shell=True, text=True).stdout

        memory = int(output.split('\n')[1].split()[1].strip())

        return memory

    def get_used(self):
        return self.get_total() - self.get_free()

    def get_free(self):

        command = "free -b"

        output = subprocess.run(command, stdout=subprocess.PIPE, shell=True, text=True).stdout

        memory = int(output.split('\n')[1].split()[3].strip())

        return memory


class WindowsCpu(Cpu):

    def get_load(self):

        command = "wmic cpu get loadpercentage"

        output = subprocess.run(command, stdout=subprocess.PIPE, shell=True, text=True).stdout

        load = float(output.split('\n\n')[1].strip()) / 100

        return load

    def get_temperature(self):

        command = "wmic /namespace:\\\\root\\cimv2 path
Win32_PerfFormattedData_Counters_ThermalZoneInformation get Temperature"

        output = subprocess.run(command, stdout=subprocess.PIPE, shell=True, text=True).stdout

        temperature = float(output.split('\n\n')[1].strip()) - 273

        return temperature


class WindowsMemory(Memory):

    def get_total(self):

        command = "wmic memorychip get capacity"

        output = subprocess.run(command, stdout=subprocess.PIPE, shell=True, text=True).stdout

        memory_units = [int(value.strip()) for value in output.split('\n\n')[1:] if value.strip()]

        memory = sum(memory_units)

        return memory

    def get_used(self):
        return self.get_total() - self.get_free()

    def get_free(self):

        command = "wmic OS get FreePhysicalMemory"

        output = subprocess.run(command, stdout=subprocess.PIPE, shell=True, text=True).stdout

        memory = int(output.split('\n\n')[1].strip()) * 1024
```

```
        return memory


class LinuxFactory(Factory):

    def create_cpu(self):
        return LinuxCpu()

    def create_memory(self):
        return LinuxMemory()


class WindowsFactory(Factory):

    def create_cpu(self):
        return WindowsCpu()

    def create_memory(self):
        return WindowsMemory()


FACTORIES = {

    'posix': {
        'name': "Linux",
        'factory': LinuxFactory
    },

    'nt': {
        'name': "Windows",
        'factory': WindowsFactory
    }

}
```

Головний код програми:

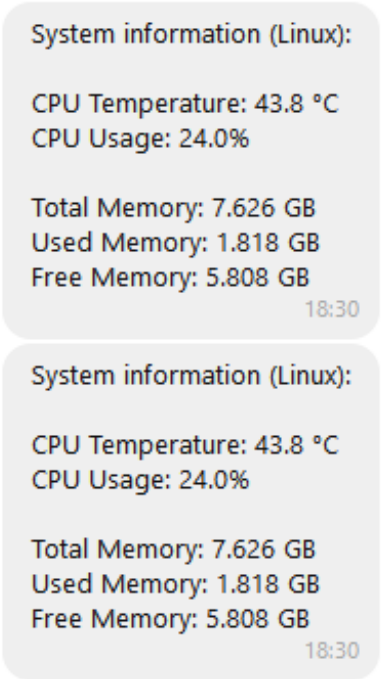
```
import os

from datetime import datetime as dt

from apscheduler.schedulers.background import BackgroundScheduler

from messengers import MESSENGERS
from factories import FACTORIES
from 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

            print(f"\nOperating system: {FACTORIES[os_name]['name']}\n")

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

            cpu, memory = factory.create_cpu(), factory.create_memory()

            message = f"System information ({FACTORIES[os_name]['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"

            scheduler = BackgroundScheduler()

            scheduler.add_job(
                func=execute(lambda: MESSENGERS[messenger].send_message(API_KEYS[messenger], receiver,
message)),
                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 «Abstract Factory». У ході роботи було реалізовано програму для відправлення повідомлень про стан системи у вибраний мессенджер в залежності від операційної системи. Визначення операційної системи реалізовано за допомогою шаблону “Abstract Factory”.