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 №8

Технології розроблення програмного забезпечення

ШАБЛОНИ «COMPOSITE», «FLYWEIGHT», «INTERPRETER», «VISITOR»

Варіант 24

Виконала:
студентка групи ІА-14
Шеліхова А.О

Перевірив:
Мягкий М.Ю.

Київ 2023

ШАБЛОНИ «COMPOSITE», «FLYWEIGHT», «INTERPRETER», «VISITOR»

Завдання:

1. Ознайомитися з короткими теоретичними відомостями.

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

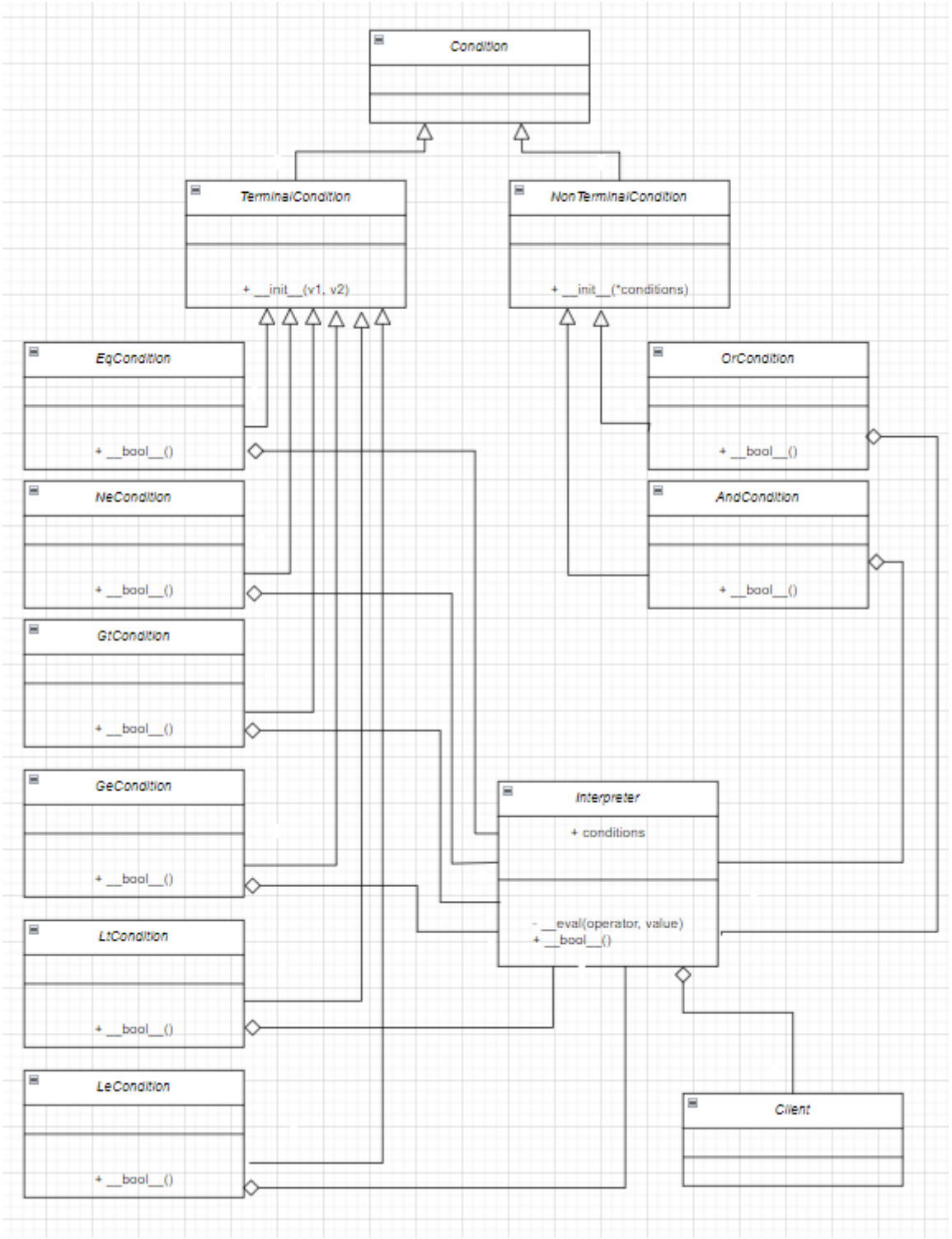
3. Застосування одного з розглянутих шаблонів при реалізації програми.

Виконання роботи:

У ході роботи було удосконалено API для створення, редагування та видалення завдань для регулярної відправки повідомлень про погоду, яке було створено у лабораторній роботі 5.

Було додано можливість вказувати умову, при якій відправляються повідомлення. Умова вказується у форматі JSON. Обробка умови реалізована за допомогою шаблону «Interpreter».

Шаблон “Interpreter”:



Примітка: Як правило в класах шаблону “Interpreter” використовується метод interpret з параметром context. В нашому випадку аналогом методу interpret(context) є магічний метод __bool__(), за допомогою якого ми реалізовуємо логіку приведення нашого об’єкта до логічного типу (True або False).

Код програми:

Код, який реалізує шаблон “Command”:

```
from abc import ABC


class Condition(ABC):
    pass


class TermainalCondition(Condition):

    def __init__(self, v1, v2):
        self.v1, self.v2 = v1, v2


class NonTerminalCondition(Condition):

    def __init__(self, *conditions):
        self.conditions = conditions


class EqCondition(TermainalCondition):

    def __bool__(self):
        return self.v1 == self.v2


class NeCondition(TermainalCondition):

    def __bool__(self):
        return self.v1 != self.v2


class GtCondition(TermainalCondition):

    def __bool__(self):
        return self.v1 > self.v2


class GeCondition(TermainalCondition):

    def __bool__(self):
        return self.v1 >= self.v2


class LtCondition(TermainalCondition):

    def __bool__(self):
        return self.v1 < self.v2


class LeCondition(TermainalCondition):

    def __bool__(self):
        return self.v1 <= self.v2


class OrCondition(NonTerminalCondition):

    def __bool__(self):
        return any(self.conditions)


class AndCondition(NonTerminalCondition):
```

```
    def __bool__(self):
        return all(self.conditions)


class Interpreter:

    conditions = {
        'eq': EqCondition,
        'ne': NeCondition,
        'gt': GtCondition,
        'ge': GeCondition,
        'lt': LtCondition,
        'le': LeCondition,
        'or': OrCondition,
        'and': AndCondition
    }

    def __init__(self, condition):
        self.__condition = condition

    def __eval(self, operator, value):

        if issubclass(Interpreter.conditions[operator], TermainalCondition):

            if not isinstance(value, list):
                raise TypeError(f"Invalid condition. Value for \"{operator}\" operator must be list.")

            if len(value) != 2:
                raise ValueError(f"Invalid condition. Value for \"{operator}\" operator must contain two
values.")

            return Interpreter.conditions[operator](*value)

        elif issubclass(Interpreter.conditions[operator], NonTerminalCondition):

            if not isinstance(value, dict):
                raise TypeError(f"Invalid condition. Value for \"{operator}\" operator must be
dictionary.")

            return Interpreter.conditions[operator](*[self.__eval(op, val) for op, val in value.items()])

    def __bool__(self):

        try:

            initial_operator = list(self.__condition.keys())[0]

            return bool(self.__eval(initial_operator, self.__condition[initial_operator]))

        except KeyError as exception:
            raise Exception(f"Invalid condition. Operator {exception} is not supported.")
```

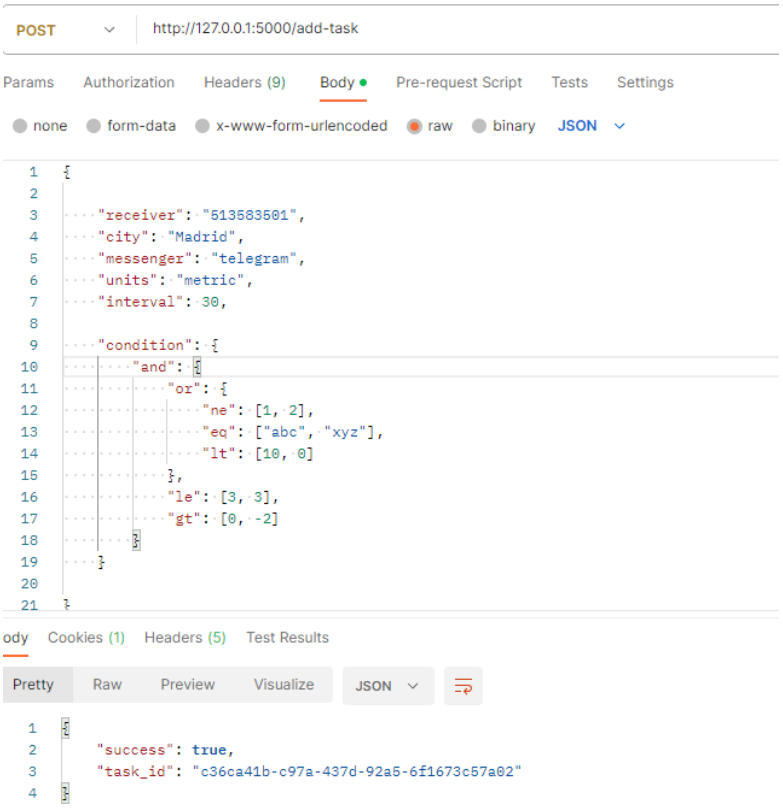
Головний код програми:

```
from flask import Flask, jsonify, request

from apscheduler.schedulers.background import BackgroundScheduler

from commands import AddTaskCommand, ModifyTaskCommand, RemoveTaskCommand, CommandExecutor
from conditions import Interpreter


app = Flask(__name__)


scheduler = BackgroundScheduler()

scheduler.start()

command_executor = CommandExecutor()


@app.route('/')
def index():
    return jsonify({
        'success': True,
        'message': "This is application for managing tasks"
    })


@app.route('/add-task', methods=['POST'])
def add_task():

    try:

        data = request.get_json()

        for filed in ['receiver', 'city', 'messenger', 'units', 'interval']:
            if filed not in data:
                raise Exception(f"Invalid request data. Field \"{filed}\" is required.")

        receiver = data['receiver']
        city = data['city']
        messenger = data['messenger']
        units = data['units']
        interval = data['interval']

        if not isinstance(interval, int):
            raise TypeError(f"Invalid request data. Field \"interval\" must be integer.")

        if interval <= 0:
            raise ValueError(f"Invalid request data. Field \"interval\" must be positive integer.")

        if 'condition' in data and not isinstance(data['condition'], dict):
            raise TypeError(f"Invalid request data. Field \"condition\" must be dictionary.")

        if 'condition' in data and not len(data['condition']) == 1:
            raise ValueError(f"Invalid request data. Field \"condition\" must contain only one root
operator.")

        if 'condition' not in data or Interpreter(data['condition']):

            command = AddTaskCommand(scheduler, receiver, city, messenger, units, interval)

            command_executor.set_command(command)

            result = command_executor.execute_command()

            return jsonify(result), 201

        return jsonify({'success': True, 'message': "Task not added"})

    except Exception as exception:
        return jsonify({
```

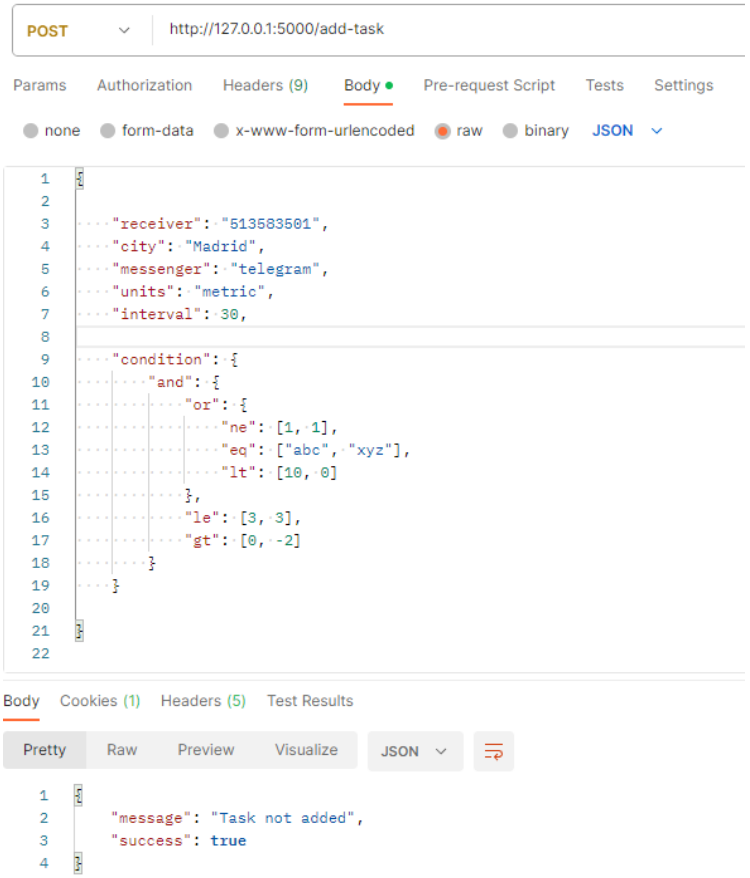
```
            'success': False,
            'error': {
                'type': type(exception).__name__,
                'message': str(exception)
            }
        })


@app.route('/modify-task/<task_id>', methods=['PUT'])
def modify_task(task_id):

    try:

        data = request.get_json()

        for filed in ['interval']:
            if filed not in data:
                raise Exception(f"Invalid request data. Field \"{filed}\" is required.")

        interval = data['interval']

        if not isinstance(interval, int):
            raise TypeError(f"Invalid request data. Field \"interval\" must be integer.")

        if interval <= 0:
            raise ValueError(f"Invalid request data. Field \"interval\" must be positive integer.")

        command = ModifyTaskCommand(scheduler, task_id, interval)

        command_executor.set_command(command)

        result = command_executor.execute_command()

        return jsonify(result)

    except Exception as exception:
        return jsonify({
            'success': False,
            'error': {
                'type': type(exception).__name__,
                'message': str(exception)
            }
        })


@app.route('/remove-task/<task_id>', methods=['DELETE'])
def remove_task(task_id):

    try:

        command = RemoveTaskCommand(scheduler, task_id)

        command_executor.set_command(command)

        result = command_executor.execute_command()

        return jsonify(result)

    except Exception as exception:
        return jsonify({
            'success': False,
            'error': {
                'type': type(exception).__name__,
                'message': str(exception)
            }
        })


if __name__ == '__main__':
    app.run()
```

Результати:

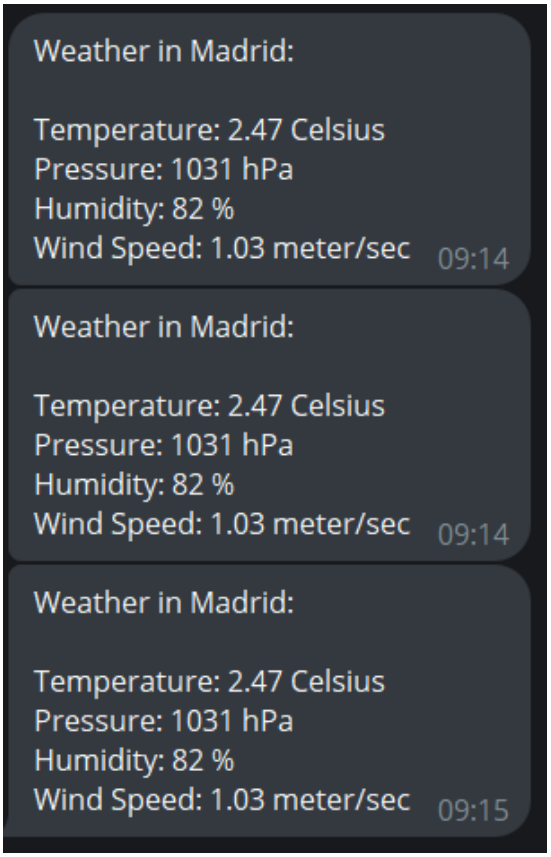
Нове завдання створюється, коли виконується умова:



Завдання не створюється, коли умова не виконується:



Погода в Мадриді у одиницях вимірювання metric (Telegram) кожні 30 секунд:



Висновки:

При виконанні даної лабораторної роботи було вивчено шаблон проектування «Interpreter». У ході роботи було удосконалено API для додавання, редагування та видалення завдань для відправки повідомлень з лабораторної роботи 5. Було додано можливість вказувати умову, при якій відправляються повідомлення. Умова вказується у форматі JSON. Обробка умови реалізована за допомогою шаблону «Interpreter».