Міністерство освіти і науки України

Донецький національний університет імені Василя Стуса Факультет інформаційних і прикладних технологій

Кафедра інформаційних технологій

ЗВІТ

з Індивідуальне творче завдання

з дисципліни «Основи програмування»

Виконав: студент гр. Б25_д/F3

Кручківський Ю.О.

Перевірив: доц. Бабаков Р. М.

Вінниця – 2025

Зміст

Вступ

Програмування сьогодні є невід'ємною частиною майже всіх сфер людської діяльності. Для створення ефективного програмного забезпечення фахівець у галузі ІТ має не лише володіти інструментами розробки, а й розуміти специфіку предметної області, для якої створюється продукт. Дана робота присвячена розробці прикладного програмного забезпечення для автоматизації професійної діяльності перекладача (Варіант 12).

Мета

Метою завдання є написання корисної програми, яка допомагає фахівцю у вирішенні щоденних професійних задач, таких як переклад тексту та ведення історії запитів.

Опис реалізації

Розроблений додаток «Програма-помічник» базується на сучасних принципах об'єктно-орієнтованого програмування та подієво-орієнтованій архітектурі (Event-Driven Design). Програма відповідає всім встановленим вимогам до ІТЗ:

- **Архітектура:** Використано модульну структуру, де ядро (core) взаємодіє з сервісами через шину подій (EventDispatcher), що забезпечує гнучкість та можливість розширення функціоналу.
- **Функціонал:** Реалізовано понад сім корисних функцій (команд), серед яких: переклад тексту, встановлення мов джерела та цілі, перегляд списку доступних мов, автоматичне визначення мови та керування історією перекладів.
- **Інтерфейс:** Програма має охайний консольний інтерфейс користувача з ієрархічним меню, побудований відповідно до логіки блок-схеми, де після виконання кожної команди користувач повертається до головного меню.
- **Додаткові можливості:** У програмі присутні обов'язкові пункти меню «Про програму» (з інформацією про розробника) та «Вихід». Також інтегровано систему «тригерів» для відстеження послідовності дій користувача.

Даний програмний продукт демонструє практичне застосування принципів абстракції, інкапсуляції та взаємодії компонентів системи через події для створення інструменту, що реально полегшує роботу перекладача.

Реалізовані сервіси та функції

Відповідно до індивідуального варіанта №12 (Перекладач), у програмі було спроєктовано та впроваджено спеціалізований сервіс TranslatorHelperService. Цей сервіс агрегує в собі логіку взаємодії з API перекладу та керує станом сесії користувача.

Згідно з технічними вимогами, у програмі реалізовано 8 основних функцій (команд), що перевищує мінімальний поріг у сім функцій:

1. **Переклад тексту (TranslateTextCommand):** Основна функція, що дозволяє вводити текстові дані, обробляти їх через зовнішній API та виводити результат перекладу на екран.
2. **Встановлення мови джерела (SetSourceLanguageCommand):** Дозволяє користувачу обрати мову, з якої буде здійснюватися переклад. Реалізовано перевірку на належність введеного коду мови до припустимого діапазону.
3. **Встановлення мови перекладу (SetTargetLanguageCommand):** Налаштування цільової мови для коректної обробки даних.
4. **Показати поточні мови (ShowLanguagesCommand):** Довідкова функція, що виводить активну мовну пару для перевірки налаштувань перед роботою.
5. **Показати доступні мови (ListLanguagesCommand):** Виводить повний перелік підтримуваних мов, що полегшує користувачу вибір параметрів.
6. **Визначення мови тексту (DetectLanguageCommand):** Інтелектуальна функція, яка аналізує введений текст та повертає код мови, якою він написаний.
7. **Історія перекладів (ShowHistoryCommand):** Функція, що використовує цикли для перебору збережених записів та виведення їх у вигляді структурованого списку.
8. **Очищення історії (ClearHistoryCommand):** Команда для обнулення накопичених даних про попередні переклади в межах поточної сесії.

Додатково реалізовано обов'язкові системні пункти:

1. **Про програму:** Команда для виводу інформації про програму.
2. **Про розробника:** Команда для виводу інформації про розробника.

Блок схема

В ході реалізції програми було досягнуто висновку, що в даному випадку блок-схема алгоритму не має сенсу оскільки основний сенс було вкладено в архітектуру додатку. Тому в звіт було вкладено UML діаграму що описує зв’язки та структуру додатку.

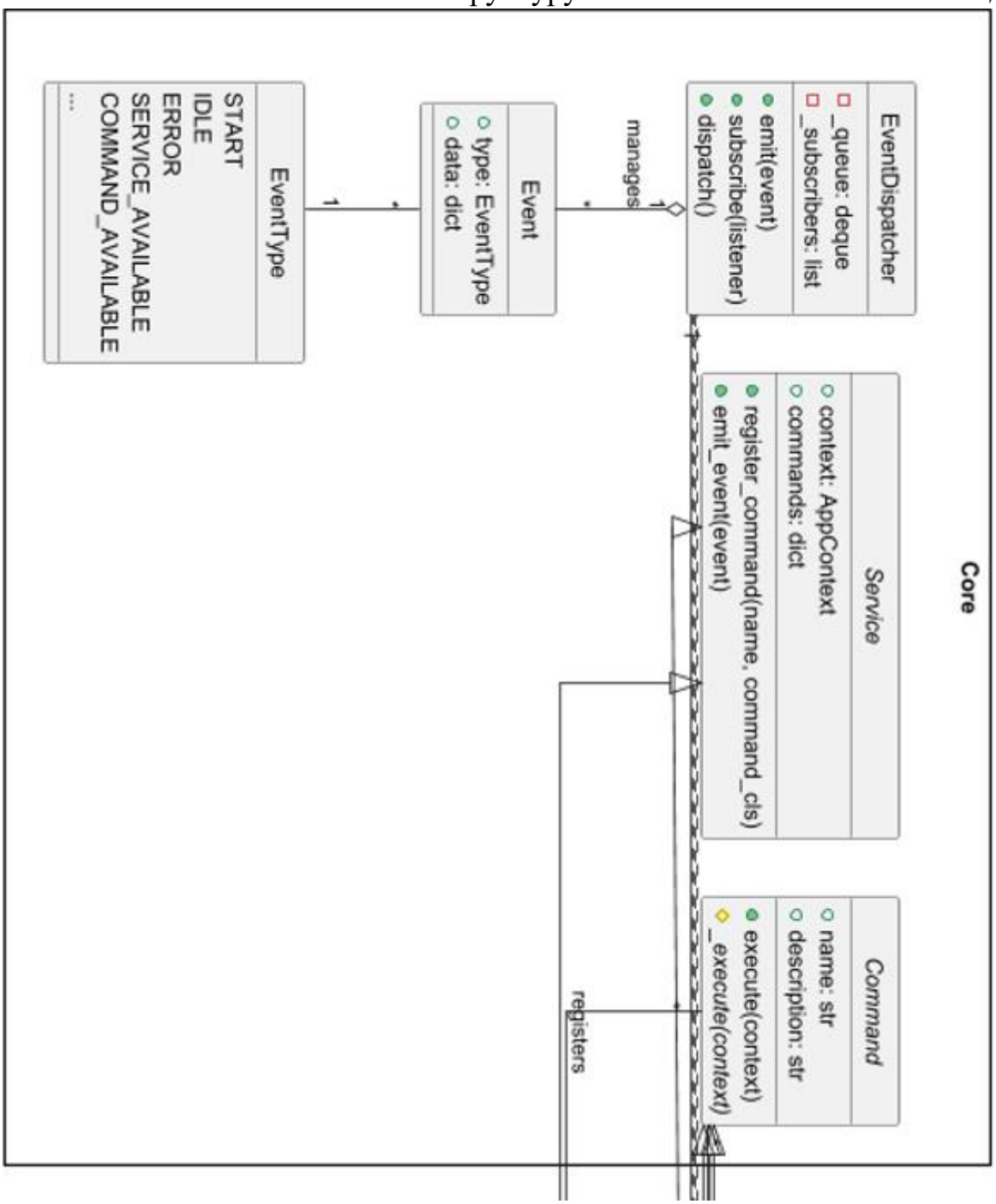

Рисунок 1 – UML схема (ч1)

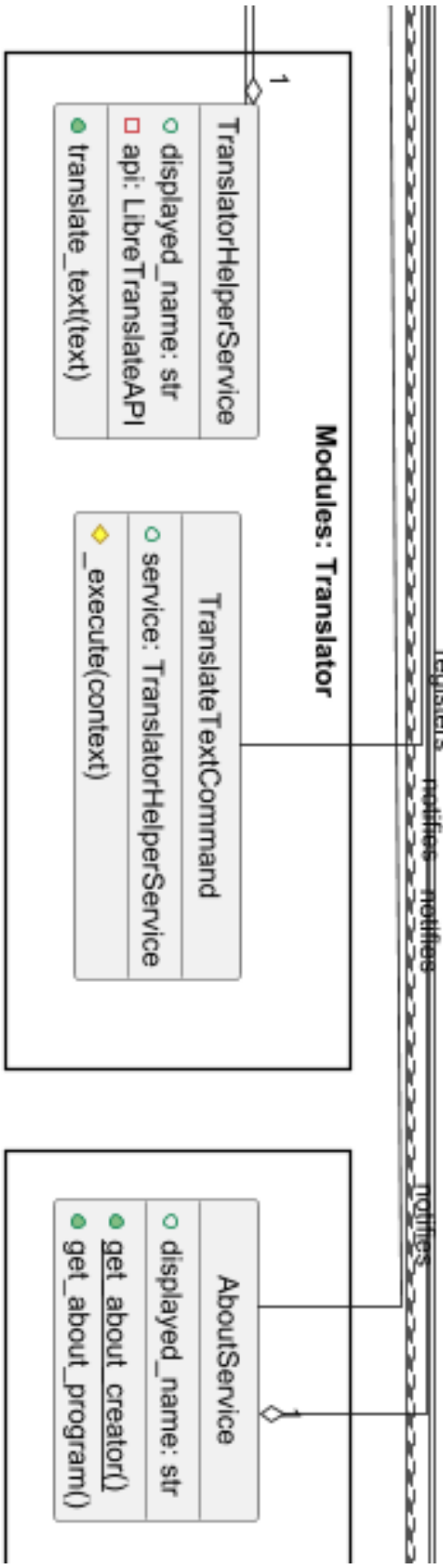

Рисунок 2 – UML схема (ч2)

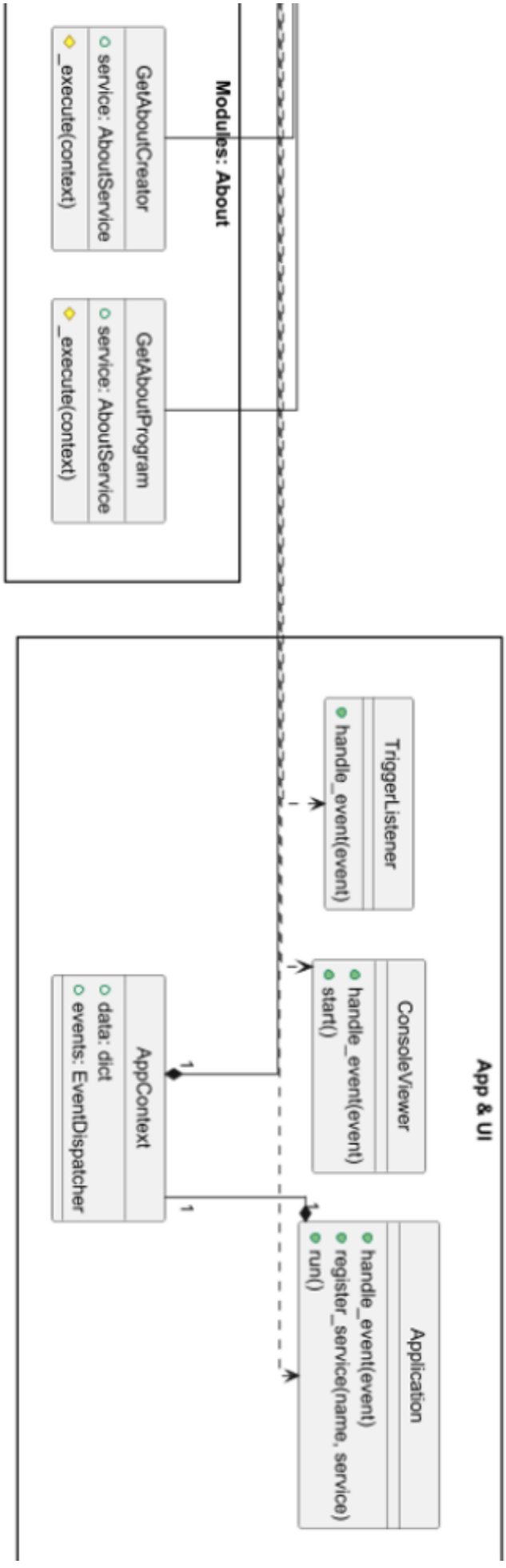

Рисунок 3 – UML схема (ч3)

Структура проєкту

```
|   main.py
+---app
|       application.py
|       context.py
|
+---core
|       command.py
|       events.py
|       service.py
|       __init.py__.py
|
+---modules
|   +---about
|   |       commands.py
|   |       service.py
|   |       __init__.py
|   |
|   \---translator_helper
|            commands.py
|            service.py
|            __init__.py
|
+---UI
|   \---console
|            ConsoleViewer.py
|
\---utils
         TriggerListener.py
```

Використані технології та платформи

Для реалізації функціоналу перекладу та забезпечення стабільної роботи додатка було використано сучасний стек технологій, що дозволяє відокремити логіку програми від сервісів обробки даних.

1. LibreTranslate та Docker Compose Оскільки програма потребує потужного рушія для перекладу, було використано LibreTranslate - систему машинного перекладу з відкритим вихідним кодом.
   - Контейнеризація: Для розгортання локального сервера перекладу використано Docker Compose. Це дозволяє запустити повний стек LibreTranslate однією командою, гарантуючи ідентичність середовища розробки та виконання.
   - Автономність: Завдяки Docker додаток не залежить від сторонніх платних API, а працює з локальним вузлом на **`http://localhost:5000`**, що забезпечує приватність даних та високу швидкість відгуку.
2. Бібліотека libretranslatepy Для взаємодії між кодом на Python та сервером перекладу використано клієнтську бібліотеку libretranslatepy.
   - Інтеграція: Бібліотека дозволяє зручно надсилати запити на переклад, визначення мови та отримання списку підтримуваних мов через об'єкт LibreTranslateAPI.

- Обробка помилок: У сервісі TranslatorHelperService реалізовано перевірку зв'язку з API під час ініціалізації, що запобігає збоям програми у разі відсутності запущеного контейнера.

3. Мова програмування та бібліотеки
   - Python 3.x: Основна мова розробки, обрана за її гнучкість та широкі можливості обробки тексту.
   - Collections (deque): Використовується для ефективного збереження черги подій та послідовності дій користувача для системи тригерів.
   - Winsound: Бібліотека для відтворення звукових сигналів.

Лістинг

Лістинг app.applications.py

```
from core.service import Service
from core.events import Event, EventType


class Application:
    def __init__(self, context):
        self.context = context
        self.services = {}
        self.running = False

    def handle_event(self, event):
        if event.type == EventType.EXIT:
            self.running = False

    def register_service(self, name: str, service: Service):
        self.context.events.emit(Event(EventType.SERVICE_AVAILABLE,
service_name=name, service=service))
        self.services[name] = service

    def run_command(self, command):
        return command.execute(self.context)

    def run(self):
        self.running = True

        self.context.events.emit(Event(EventType.START))

        while self.running:
            if not self.context.events.has_events():
                self.context.events.emit(Event(EventType.IDLE))

            self.context.events.dispatch()
```

Лістинг app.context.py

```
class AppContext:
    def __init__(self):
        self.data = {"app_name": "Програма помічник"}
        self.events = None
```

Лістинг core.command.py

```
from abc import ABC, abstractmethod
from core.events import Event, EventType

class Command(ABC):
```

```
    name: str = "unknown"
    description: str = "unknown"

    def execute(self, context):
        try:
            result = self._execute(context)

context.events.emit(Event(EventType.COMMAND_EXECUTED,command=self.name))

            return result

        except Exception as e:

context.events.emit(Event(EventType.ERROR,error_message=str(e)))

    @abstractmethod
    def _execute(self, context):
        pass
```

Лістинг core.events.py

```
from enum import Enum, auto
from collections import deque

class EventType(Enum):
    START = auto()
    IDLE = auto()
    STEP = auto()
    FINISH = auto()
    UPDATE = auto()
    ERROR = auto()
    EXIT = auto()
    SERVICE_AVAILABLE = auto()
    SERVICE_SELECTED = auto()
    COMMAND_AVAILABLE = auto()
    SHOW_SERVICES_MENU = auto()
    COMMAND_SELECTED = auto()
    COMMAND_EXECUTED = auto()
    EXECUTE_COMMAND = auto()
    EASTER_EGG = auto()


class Event:
    def __init__(self, type_: EventType, **data):
        self.type = type_
        self.data = data

    def __repr__(self):
        return f"Event: {self.type.name}: {self.data}"


class EventDispatcher:
    def __init__(self):
        self._queue = deque()
        self._subscribers = []

    def has_events(self):
        return bool(self._queue)

    def emit(self, event):
        self._queue.append(event)
```

```
    def next_event(self):
        if self._queue:
            event = self._queue.popleft()
            return event
        return None

    def subscribe(self, listener):
        self._subscribers.append(listener)

    def dispatch(self):
        event = self.next_event()
        if not event:
            return
        for listener in self._subscribers:
            listener.handle_event(event)
```

Лістинг core.service.py

```
from abc import ABC
from app.context import AppContext
from core.events import Event, EventType

class Service(ABC):
    def __init__(self, context: AppContext):
        self.context = context
        self.commands = {}

    def register_command(self, name: str, command_cls):
        self.commands[name] = command_cls
        self.context.events.emit(
            Event(EventType.COMMAND_AVAILABLE,
service_name=self.__class__.displayed_name,                    name=name,
command_cls=command_cls)
        )

    def emit_event(self, event):
        if hasattr(self.context, "events") and self.context.events:
            self.context.events.emit(event)
```

Лістинг modules.about.commands.py

```
from core.command import Command


class GetAboutCreator(Command):
    name = "Про розробника"
    description = "Про розробника"
    expected_params = []

    def __init__(self, service):
        self.service = service

    def _execute(self, context):
        return self.service.get_about_creator()


class GetAboutProgram(Command):
    name = "Про програму"
    description = "Про програму"
    expected_params = []
```

```
    def __init__(self, service):
        self.service = service

    def _execute(self, context):
        return self.service.get_about_program()
```

Лістинг modules.about.service.py

```
from core.service import Service
from modules.about.commands import *


class AboutService(Service):
    displayed_name = "Про програму"

    def __init__(self, context):
        super().__init__(context)

        self.register_command(GetAboutCreator.name, GetAboutCreator)
        self.register_command(GetAboutProgram.name, GetAboutProgram)

    @staticmethod
    def get_about_creator():
        about_creator_text = (

"╔═══════════════════════════════════════════════════════════════════╗\n"
            "║ Розробив студент групи Б25_F3(A)                          ║\n"
            "║ Кручківський Юрій Олександрович                           ║\n"

"╚═══════════════════════════════════════════════════════════════════╝"
        )
        return about_creator_text

    def get_about_program(self):
        about_program_text = (

"╔═══════════════════════════════════════════════════════════════════╗\n"
            f"║              ПРОГРАМА <{self.context.data.get('app_name',
'Помічник')}>                    ║\n"

"╠═══════════════════════════════════════════════════════════════════╣\n"
            "║ Це модульний асистент, побудований на базі                ║\n"
            "║ подієво-орієнтованої архітектури.                         ║\n"
            "║                                                           ║\n"
            "║ Основні можливості:                                       ║\n"
            "║ • Динамічна реєстрація сервісів та команд.                ║\n"
            "║ • Система глобальних подій та підписок.                   ║\n"
            "║ • Гнучке розширення функціоналу без зміни ядра.           ║\n"
            "║ • Відстеження ланцюжків подій.                            ║\n"

"╚═══════════════════════════════════════════════════════════════════╝"
        )
        return about_program_text
```

Лістинг modules.translator_helper.commands.py

```
from core.command import Command


class TranslateTextCommand(Command):
    name = "Перекласти текст"
```

```
    description = "Перекласти текст"
    expected_params = ["text"]

    def __init__(self, service, text: str):
        self.service = service
        self.text = text

    def _execute(self, context):
        return self.service.translate_text(self.text)


class SetSourceLanguageCommand(Command):
    name = "Встановити мову джерела"
    description = "Встановити мову джерела"
    expected_params = ["lang"]

    def __init__(self, service, lang: str):
        self.service = service
        self.lang = lang

    def _execute(self, context):
        return self.service.set_source_language(self.lang)


class SetTargetLanguageCommand(Command):
    name = "Встановити мову перекладу"
    description = "Встановити мову перекладу"
    expected_params = ["lang"]

    def __init__(self, service, lang: str):
        self.service = service
        self.lang = lang

    def _execute(self, context):
        return self.service.set_target_language(self.lang)


class ShowLanguagesCommand(Command):
    name = "Показати поточні мови"
    description = "Показати поточні мови"
    expected_params = []

    def __init__(self, service):
        self.service = service

    def _execute(self, context):
        return self.service.show_languages()


class DetectLanguageCommand(Command):
    name = "Визначити мову тексту"
    description = "Визначити мову тексту"
    expected_params = ["text"]

    def __init__(self, service, text: str):
        self.service = service
        self.text = text

    def _execute(self, context):
        return self.service.detect_language(self.text)
```

```
class ShowHistoryCommand(Command):
    name = "Показати історію перекладів"
    description = "Показати історію перекладів"
    expected_params = []

    def __init__(self, service):
        self.service = service

    def _execute(self, context):
        return self.service.list_history()


class ClearHistoryCommand(Command):
    name = "Очистити історію перекладів"
    description = "Очистити історію перекладів"
    expected_params = []

    def __init__(self, service):
        self.service = service

    def _execute(self, context):
        return self.service.clear_history()

class ListLanguagesCommand(Command):
    name = "Показати доступні мови"
    description = "Показати доступні мови"
    expected_params = []

    def __init__(self, service):
        self.service = service

    def _execute(self, context):
        return self.service.get_supported_languages()
```

Лістинг modules.translator_helper.service.py

```
from libretranslatepy import LibreTranslateAPI
from core.service import Service
from modules.translator_helper.commands import *

class TranslatorHelperService(Service):
    displayed_name = "Перекладач"

    def __init__(self, context):
        super().__init__(context)
        self.SUPPORTED_LANGS = {"en", "uk", "de", "fr", "pl", "es"}

        self.api = LibreTranslateAPI("http://localhost:5000")

        try:
            self._call_api()
        except:
            raise BaseException("Немає LibreTranslate")

        self.source_lang = "en"
        self.target_lang = "uk"
        self.history = []
```

```
        self.register_command(TranslateTextCommand.name,
TranslateTextCommand)
        self.register_command(SetSourceLanguageCommand.name,
SetSourceLanguageCommand)
        self.register_command(SetTargetLanguageCommand.name,
SetTargetLanguageCommand)
        self.register_command(ShowLanguagesCommand.name,
ShowLanguagesCommand)
        self.register_command(ListLanguagesCommand.name,
ListLanguagesCommand)
        self.register_command(DetectLanguageCommand.name,
DetectLanguageCommand)
        self.register_command(ShowHistoryCommand.name,
ShowHistoryCommand)
        self.register_command(ClearHistoryCommand.name,
ClearHistoryCommand)


    def _call_api(self):
        self.api.languages()

    def translate_text(self, text: str):
        translated = self.api.translate(
            text,
            self.source_lang,
            self.target_lang
        )

        self.history.append({
            "source": self.source_lang,
            "target": self.target_lang,
            "text": text,
            "result": translated
        })

        return translated



    def set_source_language(self, lang: str):
        if lang not in self.SUPPORTED_LANGS and lang != "auto":
            raise ValueError(f"Непідтримувана мова: {lang}")
        self.source_lang = lang

        return f"Мова джерела встановлена: {lang}"

    def set_target_language(self, lang: str):
        self.target_lang = lang
        return f"Мова перекладу встановлена: {lang}"

    def show_languages(self):
        return {
            "source_language": self.source_lang,
            "target_language": self.target_lang
        }

    def detect_language(self, text: str):
        result = self.api.detect(text)
        return f'"language": {result[0]["language"]}'
```

```
    def list_history(self):
        if not self.history:
            return "Історія перекладів порожня"

        lines = []
        for i, item in enumerate(self.history, 1):
            lines.append(
                f"{i}. [{item['source']}→{item['target']}] "
                f"{item['text']} → {item['result']}"
            )

        return "\n".join(lines)

    def get_supported_languages(self):
        try:
            langs = self.api.languages()
            return "\n".join([
                f"{lang['code']}:\t{lang['name']}"
                for lang in langs
            ])
        except Exception as e:
            raise RuntimeError(f"Не вдалося отримати список мов: {e}")

    def clear_history(self):
        self.history.clear()
        return "Історія перекладів очищена"
```

Лістинг utils.TriggerListener.py

```
from collections import deque
from core.events import EventType, Event


class TriggerListener:
    def __init__(self, context):
        self.context = context
        self.correct_order = [
            "Визначити мову тексту",
            "Перекласти текст",
            "Визначити мову тексту"
        ]
        self.sequence = deque(maxlen=len(self.correct_order))

    def handle_event(self, event):
        if event.type == EventType.COMMAND_EXECUTED:
            cmd = event.data["command"]

            self.sequence.append(cmd)
            if list(self.sequence) == self.correct_order:
                self.trigger()

    def trigger(self):
        self.context.events.emit(
            Event(
                EventType.EASTER_EGG,
                message="Великодка активована!"
            )
        )
```

Лістинг UI.console.ConsoleViewer.py

```
from core.events import Event, EventType
```

```
import winsound

class ConsoleViewer:
    def __init__(self, context):
        self.context = context
        self.commands = {}
        self.services = {"Вийти": ""}

        self.exit_menu = {
            "вийти": {"EventType": EventType.EXIT},
            "повернутись в головне меню": {"EventType":
EventType.SHOW_SERVICES_MENU},
        }

        self.running = True

    def handle_event(self, event):
        if event.type == EventType.EASTER_EGG:
            self.print_banner(str(event.data["message"]))
            winsound.PlaySound("sound/easter_egg_cs.wav",
winsound.SND_FILENAME)

        if event.type == EventType.IDLE:
            self.show_exit_menu()
            self.pick_exit_menu_command()

        if event.type == EventType.SERVICE_AVAILABLE:
            service_name = event.data["service_name"]
            self.services[service_name] = event.data["service"]

        if event.type == EventType.COMMAND_AVAILABLE:
            command_name = event.data['name']
            self.commands[command_name] = {
                "command_cls": event.data["command_cls"],
                "service": event.data["service_name"]
            }

        if event.type == EventType.START:
            self.print_banner(self.context.data["app_name"])
            self.context.events.emit(Event(EventType.SHOW_SERVICES_MENU))

        if event.type == EventType.SHOW_SERVICES_MENU:
            self.show_service_menu()
            self.pick_service()

        if event.type == EventType.SERVICE_SELECTED:
            self.show_commands(event.data["service_name"])
            self.pick_command(event.data["service_name"])

        if event.type == EventType.EXECUTE_COMMAND:
            command_cls                                                =
self.commands[event.data["command_name"]]["command_cls"]
            service_cls = self.services[event.data["service_name"]]
            print(self.execute_command(command_cls, service_cls))

        if event.type == EventType.EXIT:
            self.running = False

        if event.type == EventType.ERROR:
            self.print_banner(str(event.data["error_message"]))
```

```
    @staticmethod
    def print_banner(title: str):
        width = len(title) + 6
        print("╔" + "═" * (width - 2) + "╗")
        print(f"║  {title}  ║")
        print("╚" + "═" * (width - 2) + "╝")


    def show_service_menu(self):
        print("\n== Доступні сервіси ==")
        for i, service in enumerate(self.services.keys(), 1):
            print(f"{i}. {service}")


    def show_exit_menu(self):
        print("\n== Оберіть пункт ==")
        for i, service in enumerate(self.exit_menu.keys(), 1):
            print(f"{i}. {service}")


    def pick_exit_menu_command(self):
        while True:
            try:
                choice = int(input("\nОберіть пункт (номер): ")) - 1
                event_type = list(self.exit_menu.values())[choice]
                self.context.events.emit(Event(event_type["EventType"]))
                break
            except (ValueError, IndexError):
                print("Невірний вибір. Спробуйте ще раз.")

    def pick_service(self):
        while True:
            try:
                choice = int(input("\nОберіть сервіс (номер): ")) - 1
                if choice == 0:
                    self.context.events.emit(Event(EventType.EXIT))
                    break
                service_name = list(self.services.keys())[choice]

self.context.events.emit(Event(EventType.SERVICE_SELECTED,
service_name=service_name))
                break
            except (ValueError, IndexError):
                print("Невірний вибір. Спробуйте ще раз.")

    def show_commands(self, service_name):
        print(f"\n== Команди сервісу '{service_name}' ==")
        commands_for_service = [k for k, v in self.commands.items() if
v["service"] == service_name]
        for i, cmd_name in enumerate(commands_for_service, 1):
            print(f"{i}. {cmd_name}")
        print(f"{len(commands_for_service) + 1}. Повернутися до меню
сервісів")  # опція Back

    def pick_command(self, service_name):
        commands_for_service = [k for k, v in self.commands.items() if
v["service"] == service_name]
        while True:
```

```
            try:
                choice = int(input("\nОберіть команду (номер): "))
                if choice == len(commands_for_service) + 1:

self.context.events.emit(Event(EventType.SHOW_SERVICES_MENU))
                else:
                    cmd_name = commands_for_service[choice - 1]
                    self.context.events.emit(
                        Event(EventType.EXECUTE_COMMAND,
command_name=cmd_name, service_name=service_name)
                    )
                break
            except (ValueError, IndexError):
                print("Невірний вибір. Спробуйте ще раз.")

    def execute_command(self, command_cls, service):
        if hasattr(command_cls, "description"):
            print(f"\nКоманда: {command_cls.description}")

        params = []
        if hasattr(command_cls, "expected_params"):
            for param in command_cls.expected_params:
                while True:
                    try:
                        value = input(f"Введіть {param}: ")
                        params.append(value)
                        break
                    except ValueError:
                        print("Невірне значення, спробуйте ще раз.")

        cmd_instance = command_cls(service, *params)
        result = cmd_instance.execute(self.context)
        return result
```

Лістинг main.py

```
from app.application import Application
from app.context import AppContext

from core.events import EventDispatcher

from modules.translator_helper.service import TranslatorHelperService
from modules.about.service import AboutService

from utils.TriggerListener import TriggerListener
from UI.console.ConsoleViewer import ConsoleViewer

context = AppContext()
context.events = EventDispatcher()

app = Application(context)
viewer = ConsoleViewer(context)
trigger = TriggerListener(context)

context.events.subscribe(app)
context.events.subscribe(viewer)
context.events.subscribe(trigger)

about_service = AboutService(context)
t_h_service = TranslatorHelperService(context)
app.register_service(t_h_service.displayed_name, t_h_service)
```

```
app.register_service(about_service.displayed_name, about_service)

app.run()
```

Приклад використання

Переклад перекладу тексту

```
╔═══════════════════════╗
║  Програма помічник    ║
╚═══════════════════════╝

== Доступні сервіси ==
1. Вийти
2. Перекладач
3. Про програму

Оберіть сервіс (номер): 2

== Команди сервісу 'Перекладач' ==
1. Перекласти текст
2. Встановити мову джерела
3. Встановити мову перекладу
4. Показати поточні мови
5. Показати доступні мови
6. Визначити мову тексту
7. Показати історію перекладів
8. Очистити історію перекладів
9. Повернутися до меню сервісів

Оберіть команду (номер): 2

Команда: Встановити мову джерела
Введіть lang: en
Мова джерела встановлена: en

== Оберіть пункт ==
1. вийти
2. повернутись в головне меню

Оберіть пункт (номер): 2

== Доступні сервіси ==
1. Вийти
2. Перекладач
3. Про програму

Оберіть сервіс (номер): 2

== Команди сервісу 'Перекладач' ==
1. Перекласти текст
2. Встановити мову джерела
3. Встановити мову перекладу
4. Показати поточні мови
5. Показати доступні мови
6. Визначити мову тексту
7. Показати історію перекладів
8. Очистити історію перекладів
9. Повернутися до меню сервісів
```

```
Оберіть команду (номер): 3

Команда: Встановити мову перекладу
Введіть lang: uk
Мова перекладу встановлена: uk

== Оберіть пункт ==
1. вийти
2. повернутись в головне меню

Оберіть пункт (номер): 2

== Доступні сервіси ==
1. Вийти
2. Перекладач
3. Про програму

Оберіть сервіс (номер): 2

== Команди сервісу 'Перекладач' ==
1. Перекласти текст
2. Встановити мову джерела
3. Встановити мову перекладу
4. Показати поточні мови
5. Показати доступні мови
6. Визначити мову тексту
7. Показати історію перекладів
8. Очистити історію перекладів
9. Повернутися до меню сервісів

Оберіть команду (номер): 1

Команда: Перекласти текст
Введіть text: conputer science
наука

== Оберіть пункт ==
1. вийти
2. повернутись в головне меню

Оберіть пункт (номер): 2

== Доступні сервіси ==
1. Вийти
2. Перекладач
3. Про програму

Оберіть сервіс (номер): 2

== Команди сервісу 'Перекладач' ==
1. Перекласти текст
2. Встановити мову джерела
3. Встановити мову перекладу
4. Показати поточні мови
5. Показати доступні мови
6. Визначити мову тексту
7. Показати історію перекладів
8. Очистити історію перекладів
9. Повернутися до меню сервісів
```

```
Оберіть команду (номер): 1

Команда: Перекласти текст
Введіть text: computer science
комп'ютерна наука

== Оберіть пункт ==
1. вийти
2. повернутись в головне меню

Оберіть пункт (номер): 1

Process finished with exit code 0
```

Приклад визначення мови

```
╔═══════════════════╗
║ Програма помічник ║
╚═══════════════════╝

== Доступні сервіси ==
1. Вийти
2. Перекладач
3. Про програму

Оберіть сервіс (номер): 2

== Команди сервісу 'Перекладач' ==
1. Перекласти текст
2. Встановити мову джерела
3. Встановити мову перекладу
4. Показати поточні мови
5. Показати доступні мови
6. Визначити мову тексту
7. Показати історію перекладів
8. Очистити історію перекладів
9. Повернутися до меню сервісів

Оберіть команду (номер): 6

Команда: Визначити мову тексту
Введіть text: computer scince
"language": en

== Оберіть пункт ==
1. вийти
2. повернутись в головне меню

Оберіть пункт (номер): 1

Process finished with exit code 0
```

Приклад виконання команди “Про програму”

```
╔═══════════════════╗
║ Програма помічник ║
╚═══════════════════╝

== Доступні сервіси ==
1. Вийти
2. Перекладач
3. Про програму
```

```
Оберіть сервіс (номер): 3

== Команди сервісу 'Про програму' ==
1. Про розробника
2. Про програму
3. Повернутися до меню сервісів

Оберіть команду (номер): 2

Команда: Про програму
╔═══════════════════════════════════════════════════════════╗
║            ПРОГРАМА <Програма помічник>                   ║
╠═══════════════════════════════════════════════════════════╣
║ Це модульний асистент, побудований на базі                ║
║ подієво-орієнтованої архітектури.                         ║
║                                                           ║
║ Основні можливості:                                       ║
║ • Динамічна реєстрація сервісів та команд.                ║
║ • Система глобальних подій та підписок.                   ║
║ • Гнучке розширення функціоналу без зміни ядра.           ║
║ • Відстеження ланцюжків подій.                            ║
╚═══════════════════════════════════════════════════════════╝

== Оберіть пункт ==
1. вийти
2. повернутись в головне меню

Оберіть пункт (номер): 1

Process finished with exit code 0
```

Висновок

У результаті виконання індивідуального творчого завдання було успішно розроблено модульний програмний комплекс «Помічник перекладача», адаптований під потреби фахівця згідно з варіантом №12. В основу проєкту закладено принципи об’єктно-орієнтованого програмування та сучасну подієво-орієнтовану архітектуру (Event-Driven Design). Такий підхід дозволив створити систему з низькою пов'язаністю компонентів, де ядро програми, графічний інтерфейс та функціональні модулі взаємодіють через централізовану шину подій. Програма повністю відповідає вимогам щодо наявності ієрархічного меню, коректного виходу та повернення до головного екрана після завершення кожної операції.

Завершений проєкт продемонстрував важливість поєднання технічних навичок програмування з глибоким розумінням особливостей предметної області. Окрім виконання базових вимог, у програму було додано систему тригерів для відстеження активності користувача та обов'язковий інформаційний блок «Про програму» з даними про розробника. Таким чином, створений продукт є не лише навчальною роботою, а й прикладом гнучкого, розширюваного програмного забезпечення, готового до впровадження нових сервісів у майбутньому.