МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
КАФЕДРА САПР

Практическая работа №1

по дисциплине

«Объектно-ориентированное программирование»

Студент гр. 4354 _____ Нем А.В.

Преподаватель _____ Кулагин М.В.

Санкт-Петербург

2025

1. Задание

Напишите программу, которая с консоли считывает поисковый запрос пользователя, и выводит результат поиска по Википедии. После выбора нужной статьи программа должна открывать ее в браузере. Программа должна реагировать корректно на любой пользовательский ввод.

Задача разбивается на 5 этапов:

1. Считать введенные пользователем данные
2. Сделать запрос к серверу
3. Распарсить ответ
4. Вывести результат поиска
5. Открыть нужную страницу в браузере

2. Спецификация программы

Описание реализованных классов

Программа состоит из трёх основных классов:

- WikipediaSearcher — выполняет HTTP-запрос к API Википедии и получает сырые данные в формате JSON.
- ArticleProcessor — обрабатывает результаты поиска: извлекает заголовки и ID статей, выводит их в консоль, запрашивает выбор пользователя.
- WikipediaBrowser — координирует взаимодействие между компонентами: принимает ввод пользователя, вызывает поиск и обработку, открывает выбранную статью в браузере.

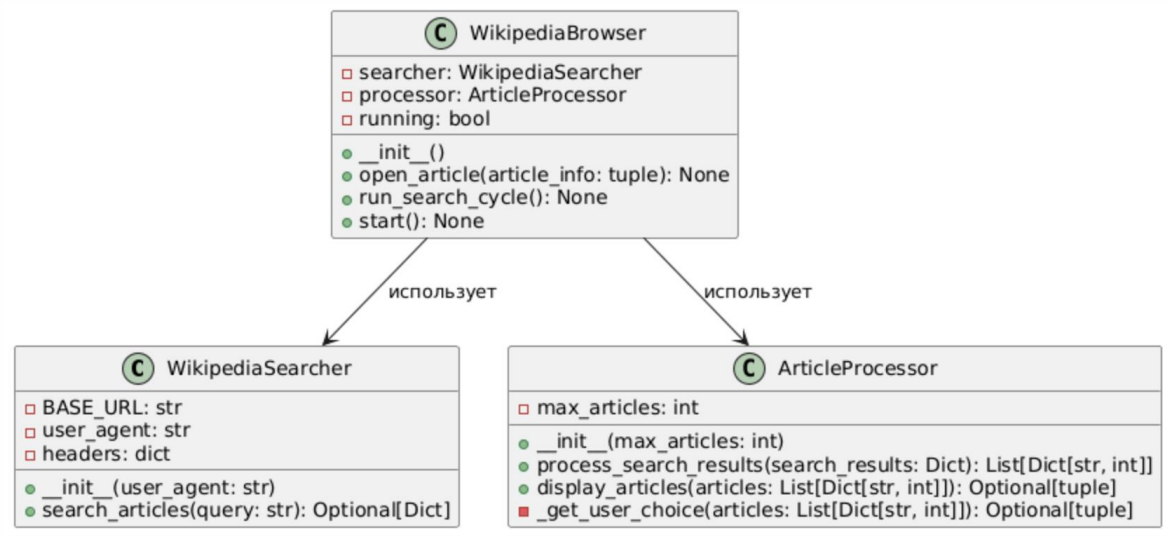

Рисунок 1. Диаграмма классов

3. Описание интерфейса пользователя программы

Взаимодействие с пользователем происходит через последовательный диалог, где система выводит запросы, а пользователь вводит команды с клавиатуры. Интерфейс работает в двух основных состояниях:

1. Состояние: Ввод поискового запроса.

Программа ожидает ввода текстового запроса для поиска информации. Для выхода из программы пользователь может ввести символ "0".

2. Состояние: Выбор из списка результатов

После выполнения поиска программа выводит нумерованный список найденных статей. Пользователь может выбрать номер статьи для просмотра подробной информации или ввести "0" для возврата к поиску.

3. Состояние: Отображение содержимого статьи

Обеспечивает просмотр содержимого выбранной статьи с последующим возвратом к списку результатов, началу нового поиска или завершению программы, реализуя тем самым полный цикл навигации по информационной системе.

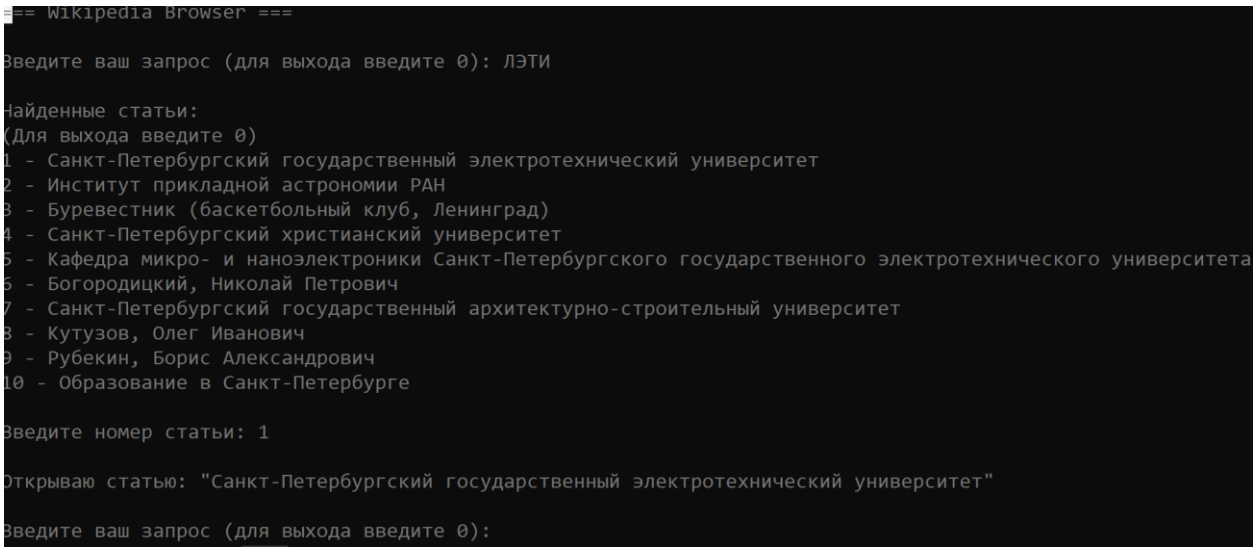

Рисунок 2. Пример работы программы

4. Текст программы

```
import requests
import webbrowser
import json
from requests.exceptions import RequestException
from typing import List, Dict, Optional

class WikipediaSearcher:

    BASE_URL = "https://ru.wikipedia.org/w/api.php"

    def __init__(self, user_agent: str = "MyApp/1.0"):

        self.user_agent = user_agent
        self.headers = {'User-Agent': user_agent}

    def search_articles(self, query: str) -> Optional[Dict]:

        try:
            params = {
                'action': 'query',
                'list': 'search',
                'utf8': '',
                'format': 'json',
                'srsearch': f'"{query}"'
            }

            response = requests.get(
```

```
                self.BASE_URL,
                headers=self.headers,
                params=params
            )
            response.raise_for_status()

            return response.json()

        except requests.RequestException as e:
            print(f'Ошибка при выполнении запроса: {e}')
            return None

        except json.JSONDecodeError as e:
            print(f'Ошибка при разборе JSON: {e}')
            return None

class ArticleProcessor:

    def __init__(self, max_articles: int = 10):
        self.max_articles = max_articles

    def process_search_results(self, search_results: Dict) ->
List[Dict[str, int]]:

        if not search_results:
            return []

        articles = search_results.get('query', {}).get('search',
[])
        if not articles:
            return []

        processed_articles = []
        for article in articles[:self.max_articles]:
            processed_articles.append({
                article['title']: article['pageid']
            })

        return processed_articles

    def display_articles(self, articles: List[Dict[str, int]]) ->
Optional[str]:

        if not articles:
            print("Ваш запрос не дал результатов")
            return None

        print('\nНайденные статьи:\n(Для выхода введите 0)')

        for index, article in enumerate(articles, 1):
            title = list(article.keys())[0]
```

```
            print(f'{index} - {title}')

        return self._get_user_choice(articles)

    def _get_user_choice(self, articles: List[Dict[str, int]]) ->
Optional[str]:

        while True:
            choice = input('\nВведите номер статьи: ').strip()

            if choice == '0':
                return None

            if not choice.isdigit():
                print('Пожалуйста, введите число')
                continue

            choice_num = int(choice)

            if not (1 <= choice_num <= len(articles)):
                print(f'Пожалуйста, введите число от 1 до
{len(articles)}')
                    continue

                selected_article = articles[choice_num - 1]
                article_title = list(selected_article.keys())[0]
                page_id = selected_article[article_title]

                return article_title, page_id

    class WikipediaBrowser:

        def __init__(self):
            self.searcher = WikipediaSearcher()
            self.processor = ArticleProcessor()
            self.running = False

        def open_article(self, article_info: tuple) -> None:

            if not article_info:
                return

            title, page_id = article_info
            url =
f'https://ru.wikipedia.org/w/index.php?curid={page_id}'
            print(f'\nОткрываю статью: "{title}"')
            webbrowser.open(url)

        def run_search_cycle(self) -> None:

            while True:
```

```
                user_request = input("\nВведите ваш запрос (для выхода введите 0): ").strip()

                if user_request == '0':
                    print("До свидания!")
                    break

                if not user_request:
                    print("Запрос не может быть пустым")
                    continue

                # Поиск статей
                search_results = self.searcher.search_articles(user_request)

                if search_results is None:
                    print("Произошла ошибка при поиске")
                    continue

                # Обработка результатов
                articles = self.processor.process_search_results(search_results)

                # Отображение и выбор статьи
                article_info = self.processor.display_articles(articles)

                # Открытие выбранной статьи
                if article_info:
                    self.open_article(article_info)

        def start(self) -> None:
            print("=== Wikipedia Browser ===")
            self.running = True
            self.run_search_cycle()

    def main():
        app = WikipediaBrowser()
        app.start()

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

5. Выводы

В ходе выполнения практической работы была разработана консольная программа, позволяющая выполнять поиск статей в русскоязычной Википедии и открывать их в веб-браузере. Программа реализована с применением принципов объектно-ориентированного программирования: каждый класс отвечает за свою зону ответственности, обеспечена чёткая модульность и слабая связанность компонентов.

Архитектура приложения легко расширяема: возможно добавление поддержки других языков, кэширования результатов, сохранения истории поиска и т.д.

Исходный код размещён в репозитории на GitHub: https://github.com/1iper/OOP-1lab-Wikipedia-API-search-

Проект собран и запущен локально без использования систем сборки (Gradle/Maven), так как это Python-приложение. Зависимости (requests) устанавливаются через pip.