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

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

Виконала
студентка групи ІА – 13:
Майданюк Анастасія

Перевірив:
Мягкий М.Ю.

Завдання:

1. Ознайомитися з короткими теоретичними відомостями.

2. Реалізувати частину функціоналу робочої програми у вигляді класів та їх взаємодій для

досягнення конкретних функціональних можливостей.

3. Застосування одного з розглянутих шаблонів при реалізації програми.

Варіант:

..11 Web crawler (proxy, chain of responsibility, memento, template method, composite, p2p)

Веб-сканер повинен вміти розпізнавати структуру сторінок сайту, переходити за посиланнями, збирати необхідну інформацію про зазначений термін, видаляти не семантичні одиниці (рекламу, об'єкти javascript і т.д.), зберігати знайдені дані у вигляді структурованого набору html файлів вести статистику відвіданих сайтів і метадані.

Хід роботи

Патерн "Шаблонний метод" є стратегією проектування в області програмування, яка встановлює основний скелет алгоритму, залишаючи деталізацію окремих етапів на розсуд підкласів. Цей патерн дозволяє підкласам модифікувати окремі кроки алгоритму без необхідності зміни його основної структури.

Клас ScraperTemplate служить як абстрактний шаблон, який викладає основні контури алгоритму парсингу. Цей клас включає декілька абстрактних методів, таких як get_job_details, log_page_info, log_website_info та scrape_jobs, які мають бути специфічно реалізовані у спадкоємцях цього класу.

JobScraper, у свою чергу, є конкретним нащадком ScraperTemplate, який наслідує його загальну структуру. JobScraper вносить свої конкретні реалізації абстрактних методів та вводить додаткові функції, які є унікальними для процесу парсингу вакансій.

```python
class ScraperTemplate(ABC):
    def create_memento(self):
        return Memento(copy.deepcopy(self.__dict__))

    def set_memento(self, memento):
        self.__dict__.update(memento.state)

    @abstractmethod
    def get_job_details(self, job_url, writer):
        pass

    @abstractmethod
    def log_page_info(self):
        pass

    @abstractmethod
    def log_website_info(self):
        pass

    @abstractmethod
    def scrape_jobs(self, output_file):
        pass
```

```python
class JobScraper(ScraperTemplate):
    def __init__(self, base_url, keyword, num_pages=1):
        self.base_url = base_url
        self.keyword = keyword
        self.num_pages = num_pages
        self.proxy_requests = ProxyRequests()
        self.writers = [HTMLWriter]

    def create_memento(self):
        return Memento(copy.deepcopy(self.__dict__))

    def set_memento(self, memento):
        self.__dict__.update(memento.state)

    def job_writer(self, composite_writer, salary, date_and_view_info):
        composite_writer.write(f"<p class='salary'><strong>Salary:</strong> {salary}</p>")
        composite_writer.write("<div class='info-container'>")
        composite_writer.write("<p class='info-label'><strong>Additional information:</strong></p>")
        composite_writer.write(f"<p><strong>Date:</strong> {date_and_view_info['date']}</p>")
        composite_writer.write(f"<p><strong>Views:</strong> {date_and_view_info['views']}</p>")
        composite_writer.write(f"<p><strong>Responses:</strong> {date_and_view_info['responses']}</p>")
        composite_writer.write("</div>")

    def get_job_details(self, job_url, composite_writer):
        response = self.proxy_requests.get(job_url)
        if response.status_code == 200:
            soup = BeautifulSoup(response.text, 'html.parser')
            salary_handler = SalaryHandler()
            date_and_view_handler = DateAndViewHandler()
            salary = salary_handler.handle_request(soup)
            date_and_view_info = date_and_view_handler.handle_request(soup)
            self.job_writer(composite_writer, salary, date_and_view_info)
        else:
            print(f"Request error. Status code: {response.status_code}")
```

```
    def get_jobs_from_page(self, page_number, composite_writer):
        try:
            url = f"{self.base_url}?primary_keyword={self.keyword}&page={page_number}"
            response = self.proxy_requests.get(url)

            if response.status_code == 200:
                soup = BeautifulSoup(response.text, 'html.parser')
                job_links = soup.find_all('a', class_='h3 job-list-item__link')

                for link in job_links:
                    job_title = link.text.strip()
                    job_url = urljoin(self.base_url, link['href'])
                    composite_writer.write_job_container(job_title, job_url,
                                                         lambda: self.get_job_details(job_url, composite_writer))
            else:
                print(f"Request error. Status code: {response.status_code}")

        except Exception as e:
            print(f"An error occurred: {e}")

    def log_page_info(self):
        timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
        log_message = f"Keyword: {self.keyword}, Timestamp: {timestamp}"
        logging.info(log_message)

```

```
    def log_website_info(self):
        try:
            response = self.proxy_requests.get(self.base_url)
            if response.status_code == 200:
                soup = BeautifulSoup(response.text, 'html.parser')
                title_tag = soup.find('title')
                website_title = title_tag.text.strip() if title_tag else 'N/A'

                meta_description_tag = soup.find('meta', attrs={'name': 'description'})
                website_description = meta_description_tag.get('content').strip() if meta_description_tag else 'N/A'

                log_message = f"Website Title: {website_title}, Description: {website_description}"
                logging.info(log_message)
            else:
                logging.error(f"Failed to retrieve website metadata. Status code: {response.status_code}")
        except Exception as e:
            logging.error(f"An error occurred while retrieving website metadata: {e}")

```

```
    def scrape_jobs(self, output_file):
        self.log_page_info()
        self.log_website_info()
        mementos = []
        results_directory = 'results'
        os.makedirs(results_directory, exist_ok=True)
        logs_directory = 'logs'
        os.makedirs(logs_directory, exist_ok=True)

        html_output_path = os.path.join(results_directory, output_file)
        log_file_path = os.path.join(logs_directory, '../../scraping_log.txt')

        with open(html_output_path, 'w', encoding='utf-8') as file:
            composite_writer = CompositeWriter([HTMLWriter(file)])
            composite_writer.write_header()

            for page in range(1, self.num_pages + 1):
                memento = self.create_memento()
                mementos.append(memento)

                composite_writer.write_page_header(page)
                self.get_jobs_from_page(page, composite_writer)

            composite_writer.write_footer()

        logging.basicConfig(filename=log_file_path, level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s')

        for memento in mementos:
            self.set_memento(memento)
```

Висновок:

У ході виконання лабораторної роботи на реалізації паттерну "Template Method", який я використала для реалізації класу "ScraperTemplate". Цей шаблон дозволив мне створити абстрактний скелет алгоритму парсингу, залишаючи деталізацію конкретних кроків на розсуд підкласів. Я реалізувала декілька абстрактних методів, таких як "get_job_details", "log_page_info", "log_website_info" та "scrape_jobs", які потім були специфічно адаптовані у класі "JobScraper".