ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 2

Тема: Паттерн проектирования «Декоратор» (Decorator Pattern)

Студент: Тоц Леонид Александрович
Группа: ИВТ-2

---

1. Цель работы

Изучить структурный паттерн проектирования «Декоратор». Научиться применять его для динамического добавления ответственности объектам. Закрепить навыки работы с абстрактными классами (ABC), аннотациями типов (PEP-484), документированием кода (PEP-257) и написанием модульных тестов (unittest).

2. Задание

1. Реализовать базовый компонент для получения курсов валют в формате JSON через API Центрального Банка РФ.
2. Реализовать паттерн «Декоратор» с использованием интерфейса на основе `abc.ABC` и `@abstractmethod`.
3. Создать конкретные декораторы для преобразования данных в форматы **YAML** (библиотека `PyYAML`) и **CSV** (библиотека `csv`).
4. Реализовать метод сохранения данных в файл для каждого класса.
5. Соблюсти принципы **DAST**:
   - **D**ocumentation (PEP-257)
   - **A**nnotations (PEP-484)
   - **S**pecification (PEP-8)
   - **T**ests (по 2 теста на каждый класс: базовый и декораторы).

3. Краткие теоретические сведения

Паттерн «Декоратор» — это структурный паттерн проектирования, который позволяет динамически добавлять объектам новую функциональность, оборачивая их в полезные «обёртки». В отличие от наследования, декорирование является гибкой альтернативой, позволяющей комбинировать поведения во время выполнения.

Основные участники:

- **Компонент (Component):** Общий интерфейс для объектов, которые можно оборачивать.
- **Конкретный компонент (ConcreteComponent):** Базовый объект, содержащий основную логику.
- **Декоратор (Decorator):** Базовый класс обёртки, хранящий ссылку на компонент.
- **Конкретные декораторы (ConcreteDecorators):** Классы, добавляющие специфическое поведение (в данном случае — конвертацию формата).

4. Ход работы и описание реализации

4.1. Структура проекта

Проект состоит из двух основных файлов:

1. `main.py` — содержит реализацию классов и бизнес-логику.
2. `test_lab2.py` — содержит модульные тесты.

4.2. Иерархия классов

В качестве базового интерфейса использован абстрактный класс `CurrencyProvider`:

```python
class CurrencyProvider(abc.ABC):
    @abc.abstractmethod
    def get_data(self) -> Any: ...

    @abc.abstractmethod
    def save_to_file(self, filename: str) -> None: ...
```

Классы:

1. `CBRProvider` **(ConcreteComponent):**
   - Осуществляет HTTP-запрос к `https://www.cbr-xml-daily.ru/daily_json.js`.
   - Возвращает данные в виде словаря Python (JSON).
   - Сохраняет данные в файл `.json`.
2. `CurrencyDecorator` **(Decorator):**
   - Наследует `CurrencyProvider`.
   - Принимает в конструктор объект `CurrencyProvider`.
   - Делегирует вызовы методов обёрнутому объекту.
3. `YamlDecorator` **(ConcreteDecorator):**
   - Преобразует полученный словарь в строку формата YAML с помощью библиотеки `pyyaml`.

- Сохраняет результат в файл `.yaml`.

4. `CsvDecorator` **(ConcreteDecorator):**
   - Преобразует вложенную структуру JSON в плоский список.
   - Формирует CSV-строку с заголовками (CharCode, Name, Value, Nominal).
   - Сохраняет результат в файл `.csv`.

4.3. Соблюдение стандартов (DAST)

- **Annotations:** Все методы имеют аннотации типов (например, `-> str`, `-> Dict[str, Any]`).
- **Documentation:** Каждый класс и метод снабжен docstring, описывающим назначение, аргументы и возвращаемые значения.
- **Specification:** Код отформатирован согласно PEP-8 (отступы, именование переменных, длина строк).
- **Tests:** Написаны тесты с использованием `unittest` и `unittest.mock`. Мокирование используется для изоляции тестов от реального API, что обеспечивает стабильность проверки логики декораторов.

5. Тестирование

Было разработано 6 тестовых случаев:

№	Класс	Название теста	Описание
1	`CBRProvider`	`test_get_data_returns_dict`	Проверка, что метод возвращает словарь.
2	`CBRProvider`	`test_save_to_file_creates_json`	Проверка создания файла и валидности JSON.
3	`YamlDecorator`	`test_get_data_returns_yaml_string`	Проверка формата возвращаемой строки.
4	`YamlDecorator`	`test_save_to_file_writes_yaml`	Проверка записи в файл с расширением .yaml.
5	`CsvDecorator`	`test_get_data_returns_csv_string`	Проверка наличия заголовков и разделителей CSV.
6	`CsvDecorator`	`test_save_to_file_writes_csv`	Проверка валидности структуры CSV файла.

Результат запуска тестов:

```
......
----------------------------------------------------------------------
Ran 6 tests in 0.005s

OK
```

6. Результаты работы

Программа была запущена в среде выполнения. В результате работы были сгенерированы три файла с актуальными курсами валют.

Пример вывода в консоль:

```
Получение данных через CBRProvider...
Тип данных: dict
Ключи верхнего уровня: ['Date', 'Previous', 'Valute', ...]
----------------------------------------
Файл rates.json сохранен.

Получение данных через YamlDecorator...
Тип данных: str
Предпросмотр (100 символов): Date: '2023-10-27T00:00:00'
Previous: '2023-10-26T00:00:00'
Valute:
  AUD:
    ID: R01010
...
Файл rates.yaml сохранен.

Получение данных через CsvDecorator...
Тип данных: str
Предпросмотр (100 символов): CharCode,Name,Value,Nominal
USD,Доллар США,99.5305,1
EUR,Евро,105.8733,1
...
Файл rates.csv сохранен.
```

Пример содержимого файла `rates.csv`:

```
CharCode,Name,Value,Nominal
USD,Доллар США,99.5305,1
EUR,Евро,105.8733,1
```

```
GBP,Фунт стерлингов Соединенного королевства,121.2275,1
...
```

7. Вывод

В ходе выполнения лабораторной работы был успешно реализован паттерн «Декоратор».

1. **Гибкость:** Паттерн позволил добавить новые форматы вывода (YAML, CSV) без изменения кода базового класса `CBRProvider`.
2. **Интерфейсы:** Использование `abc.ABC` гарантировало, что все декораторы поддерживают единый интерфейс (`get_data` , `save_to_file`), что позволяет клиентскому коду работать с ними полиморфно.
3. **Качество кода:** Соблюдение принципов DAST обеспечило читаемость, поддерживаемость и надежность кода. Модульные тесты подтвердили корректность работы каждого компонента в изоляции.

Паттерн «Декоратор» является эффективным инструментом для расширения функциональности классов, когда использование наследования приводит к чрезмерному усложнению иерархии.

Приложение: Исходный код программы (`main.py` , `test_lab2.py`) прилагается к отчёту в электронном виде.