Конструирование программного обеспечения

Контрольная работа №1. Паттерны проектирования.

Дедлайн сдачи: 17 марта, 23:59.
Формат сдачи КР предлагается семинаристом. По умолчанию, КР выполняется на языках который является основным на семинарских занятиях, но по согласованию с семинаристом можно выполнить на другом ЯП.

Предисловие

Многие компании развивают собственные финтех-сервисы, а порой открывают свои Банки… НИУ ВШЭ не намеренны оставаться в стороне!
ВШЭ-банк в своем будущем приложении сделает основной упор на продвинутом учете финансов (УФ). Данный модуль будет представлять инструменты, предназначенные для облегчения управления личными финансовыми средствами пользователей банка. Он позволяет отслеживать доходы и расходы, что будет способствовать улучшению финансового планирования и достижению личных финансовых целей.

Перед вами поставлена задача *разработать классы* доменной модели ключевого модуля будущего приложения – *модуль «Учет финансов». Доменная модель классов должна быть реализована* с соблюдением принципов SOLID, ключевыми идеями GRASP: High Cohesion и Low Coupling, а также рядом паттернов GoF: порождающих; структурных и поведенческих.

В ходе общения с заказчиком были выявлены:

Основные требования к функциональности модуля «Учет финансов»

1. Работа с доменной моделью: создание, редактирование и удаление счетов, категорий, операций (доходов/расходов).

Опциональная функциональность модуля «Учет финансов». В каждом пункте можно пропустить/модифицировать любые подпункты, а также добавить свои – поможет вам уместнее применять паттерны.

1. **Аналитика**:
    a. Подсчет разницы доходов и расходов за выбранный период.
    b. Группировка доходов и расходов по категориям.
    c. Любая другая аналитика.
2. **Импорт и экспорт данных**:
    a. Экспорт всех данных в файлы CSV, YAML, JSON.
    b. Импорт данных из этих форматов.
3. **Управление данными**:
    a. Возможность пересчета баланса при несоответствиях.
4. **Статистика**:
    a. Измерение времени работы отдельных пользовательских сценариев.

На встрече с доменными экспертами вы определили три основных класса: BankAccount, Category и Operation:

1. **BankAccount**:
    - id: уникальный идентификатор счета.
    - name: название счета (например, "Основной счет").
    - balance: текущий баланс счета.
2. **Category**:
    - id: уникальный идентификатор категории.
    - type: тип категории (доход или расход).
    - name: название категории (например, "Кафе", "Зарплата").

    *Примеры категорий: "Кафе" или "Здоровье" – для расходов. "Зарплата" или "Кэшбэк" – для доходов
3. **Operation**:
    - id: уникальный идентификатор операции.
    - type: тип операции (доход или расход).
    - bank_account_id: ссылка на счет, к которому относится операция.
    - amount: сумма операции.
    - date: дата операции.
    - description: описание операции (необязательное поле).
    - category_id: ссылка на категорию, к которой относится операция.

На встрече с командой вы продумали и пришли к выводу, что будут уместны следующие паттерны (эти идеи можно проигнорировать и реализовать свои):

1. **Фасад** – поможет вам объединить по смыслу несколько методов. Например, можно упаковать в отдельный фасад всю работу с Category (создание, изменение, получение, удаление). Также отдельные фасады для BankAccount, Operation. Всю работу с аналитикой тоже можно вынести в отдельный фасад.
2. **Команда + декоратор** - вы сможете каждый пользовательский сценарий представить в виде паттерна Команда. Это позволит вам проще применить над каждой из команд паттерн Декоратор, для измерения времени работы пользовательского сценария. Паттерн декоратор в свою очередь позволит вам написать измерение времени работы пользовательского сценария один раз, а дальше просто оборачивать в этот декоратор сами сценарии - Команды.
3. **Шаблонный метод** – пригодится для импорта данных из файлов различных форматов. В сценарии загрузки из файла отличается только часть парсинга данных - из json, yaml, csv. Остальная часть будет одинаковой, когда данные из файла уже прочитаны.
4. **Посетитель** - подойдет при выгрузке данных в файл.
5. **Фабрика** – подойдет для того, чтобы гарантировать, что все доменные объекты создаются в одном и том же месте кода. Это в свою очередь поможет вам поддержать валидацию этих объектов - например запретить создание Operation с отрицательным amount. Если создание объектов в вашем коде через new происходит в нескольких местах, то в каждом из этих мест вам нужно валидировать ваш свежесозданный объект - происходит дублирование кода и со

временем вы забудете внести очередные изменения в валидацию в одном из этих мест, тем самым допустив создание невалидных доменных объектов.

6. **Прокси** - если вы решите использовать базу данных для персистентности данных после перезапуска приложения, вы можете использовать паттерн Прокси для реализации in-memory кэша. При инициализации приложения вы выгружаете все данные в кэш и дальше при чтении данных читаете их оттуда. При записи данных вы пишете их в кэш и бд.

ТРЕБУЕТСЯ

1. Разработать классы доменной модели модуля «Учет финансов». Доменная модель классов должна быть реализована с соблюдением принципов SOLID, ключевыми идеями GRASP: High Cohesion и Low Coupling, а также рядом паттернов GoF: порождающих; структурных и поведенческих.
2. Создать консольное приложение, в котором продемонстрировать функциональность разработанной доменной модели через взаимодействие с пользователем.
3. Написать отчет, в котором отразить:
    a. Общую идею вашего решения (какой функционал реализовали, особенно если вносили изменения в функциональные требования).
    b. Опишите какие принципы из SOLID и GRASP вы реализовали, скажите в каких классах (модулях).
    c. Опишите какие паттерны GoF вы реализовали, обоснуйте их важность, скажите в каких классах (модулях) они реализованы.

Отчет пишется в свободной форме (например, redme-файл к проекту в формате md). Задача отчета помочь проверяющему увидеть всё, что вы реализовали.

4. Добавить инструкция по запуску вашего приложения.

Критерии оценки

- **+ 2 балла** за полную реализацию основных требований к функциональности
- **+ 0.5 балла** (max 3 балла по данному критерию) за каждый реализованный (из предложенных нами или выбранных вами) паттерн.
- **+ 2 балла** за соблюдение принципов SOLID и GRASP
- **+ 1 балл** если покрыто тестами более 65% кода
- **+ 1 балл** за использование DI-контейнера
- **+/- 1 балл** за очную защиту проекта семинаристу*

***Работы, которые получат максимальные 9 баллов необходимо будет защитить своим семинаристом. Формат защиты определяется семинаристом и может проходить в формате: интервьюирования; защита принятых решений на проекте; коллоквиума и иных форматах. На защите можно получить +/- 1 балл.**

Штрафы

1. до – 2 баллов за наличие ошибки во время выполнения кода;
2. до – 5 баллов, если программа не собирается;
3. – 1 балл за каждый день просрочки дедлайна
4. – 1 балл за грубое игнорирование кодстайла.