**Отчет по большому домашнему заданию №1 по курсу «Конструирование программного обеспечения»**

**Каспари Давид Эрвинович.**

В данном отчете я отражу общую идею решения, какие паттерны и принципы были использованы при написании программы, а также инструкцию к запуску.

1. Инструкция к запуску.

К каждому файлу программы (.cs) даны пояснительные комментарии, для запуска программы необходимо открыть проект в Visual Studio или любом другом подходящем IDE, в Program.cs находится код, тестирующий работу функций. Можно его использовать для теста программы. В проекте устанавливались пакеты для работы с DI-контейнером и импортом файлов (пакеты DependencyInjection и NewtonsoftJson). Они должны быть установлены для корректного запуска программы.

1. Общая идея решения.

Мною были реализованы Основные требования к

функциональности модуля «Учет финансов» (Работа с доменной моделью: создание, редактирование и удаление счетов, категорий, операций (доходов/расходов)). Для этого необходимо было реализовать три основных класса: BankAccount, Category, Operation. Все эти требования были реализованы с использованием паттернов, указанных в задании. В следующих пунктах будет расписано, где и как были применены эти паттерны.

Между тем, был реализован ряд опциональных функций, необходимых для наглядности использования паттернов. Перечислим эти функции (так как были реализованы не все, что есть в задании):

1. Измерение времени работы отдельных пользовательских сценариев.
2. Экспорт данных в файлы Json, Yaml.
3. Пересчёт баланса при несоответствиях.
4. Подсчет разницы доходов и расходов за выбранный период.
5. Группировка доходов и расходов по категориям.

Далее будет объяснение, какие паттерны были реализованы при написании всех этих функций.

1. Принципы из SOLID и GRASP.

S (Single Responsibility) – как можно заметить, в программе достаточно большое разбиение на классы, это показывает, что каждый компонент отвечает за свою задачу (BankAccountFactory за создание, BankAccountFacade за фасад, например).

O (Open/Closed) – в нашей программе есть интерфейсы, которые позволяют действовать по этому принципу.

L (Liskov) – Объекты в программе должны быть заменяемы экземплярами их подтипов без изменения правильности работы программы. У нас есть, например, наследование BankAccountProxy от IDataProxy<BankAccount>, и там контракт выполняется.

I (Interface segregation) – интерфейсы, к примеру, IVisitor или ICommand содержат только необходимые для пользователя методы.

D (Dependency inversion) – классы в программе не зависят от конкретных реализаций, зависят от абстракций.

Про GRASP:

High Cohesion – в нашей программе, как видно, большое количество классов, и каждый отвечает за свое. Думаю, понятно, что данный принцип выполняется.

Low Coupling – изменения в каком-то конкретном классе слабо влияют на другие классы. Это потому что мы наследуемся от абстракций.

1. Реализованные паттерны GoF.
2. Фасад

Объясним подробнее, как был реализован этот паттерн. Тут, как и предлагается в задании, вся работа с созданием, удалением и т.п. у объектов Operation, BankAccount, Category вынесена в отдельный класс. Сами классы и комментарии с пояснениями можно найти в файлах BankAccountFacade.cs, OperationFacade.cs, CategoryFacade.cs.

1. Команда
2. Декоратор

Мы создали определённые команды, которые реализуют определённое поведение пользователя. Это команды CreateOperationOnCommand и CreateAndUpdateBankAccountOnCommand.

Помимо этого, мы создали декоратор, который измеряет время работы этих команд.

реализующие конкретные форматы.

1. Посетитель

Используется для экспорта данных из файлов Json и Yaml. Реализовано интерфейсом IVisitor и классами JsonFileExporter и YamlFileExporter.

1. Фабрика

Мы вынесли всю работу по созданию объекта в отдельные классы-фабрики, чтобы не дублировать код. Подробнее можно рассмотреть в файлах CategoryFactory, BankAccountFactory, OperationFactory.

1. Прокси

Используется для кэширования данных, чтоб меньше обращаться к базе данных. Реализовано с помощью классов, к примеру, OperationDatabase и OperationProxy. Аналогично с другими объектами.

Теперь пройдемся по критериям оценивания:

В программе реализованы основные требования к функциональности, реализовано 6 паттернов, соблюдаются принципы SOLID и GRASP, а также используется DI-контейнер.

Если есть вопросы к работе, то вот telegram: @davidkspr