Бершицкий Дмитрий, БПИ-234. Описание проекта к БДЗ-1.

Структура проекта

Для соблюдения принципов High Cohesion и Low Coupling решение будет разбито на несколько слоев:

A) Доменный слой – содержит основные сущности.
B) Сервисный слой – содержит бизнес-логику.
C) Инфраструктурный слой – для работы с базой данных, импортам/экспортом.
D) Пользовательский слой – для взаимодействия с пользователем через консоль.

Паттерны GoF

Порождающие:

A) Фабрика – для создания объектов и введения ограничений (например, не допускается Operation с отрицательным amount). Реализовано в DomainFactory.

Структурные:

A) Фасад – для упрощения работы с большими сервисами. Также фасады отлавливают исключения, если они возникли внутри сервисов. Реализовано в BankAccountFacade, CategoryFacade, OperationFacade, AnalyticsFacade.
B) Прокси – для кэширования запросов к базе данных. Реализовано в RepositoryProxy.

Поведенческие:

A) Команда – пользовательские запросы передаются в виде команд. Используется в классах, реализующих ICommand.
B) Декоратор – “обертка” под команду, которая вычисляет время ее выполнения. Реализовано в CommandTimerDecorator.
C) Посетитель – для экспорта данных в Json. Обходит классы доменной модели и применяет к ним форматирующие алгоритмы. Реализовано в Visitor.

Принципы SOLID и GRASP

SOLID:

S) Принцип единственной ответственности. Каждый класс или модуль отвечает за свою конкретную задачу:
    a. BankAccount отвечает только за хранение данных счета
    b. Operation представляет только операцию (доход или расход) и её атрибуты (сумма, дата, категория)
O) Принцип открытости/закрытости. Модули открыты для расширения, но закрыты для модификации:
    a. Поддержка новых типов данных (BankAccount, Category, …) возможна без модификации Repository (он работает с любым представителем IEntity).
L) Принцип подстановки Лисков. Подклассы должны быть взаимозаменяемыми с базовыми классами:
    a. В проекте принцип Лисков негде применить, так как не создаются классы-наследники от не абстрактных классов. В любом случае, принцип не нарушается.
I) Принцип разделения интерфейсов. Клиент не должен зависеть от методов, которые он не использует:

    a. Интерфейсы для различных сервисов разделены по функциональности. Например, интерфейс для работы со счетами содержит только методы, относящиеся к управлению счетом.

D) Принцип инверсии зависимостей. Модули верхнего уровня не должны зависеть от модулей нижнего уровня, а оба уровня должны зависеть от абстракций:
    a. Сервисные классы и фасады взаимодействуют с доменными объектами и репозиториями через интерфейсы (например, IOperationService или IBankAccountRepository), что позволяет легко подменять реализации при изменениях в инфраструктуре или при тестировании.

GRASP:

A) High Cohesion (Высокая сцепленность). Каждый модуль должен иметь четко определённую и узкую область ответственности:
    a. Доменные классы (BankAccount, Category, Operation) содержат только данные и базовую бизнес-логику, связанную непосредственно с их сущностью.
    b. Сервисные и фасадные классы сфокусированы на выполнении конкретных задач (например, аналитика или управление операциями) без смешения с другими обязанностями.
B) Low Coupling (Низкое связывание). Компоненты системы должны иметь минимальные зависимости друг от друга, что упрощает внесение изменений и повторное использование кода:
    a. Взаимодействие между модулями происходит через абстракции и интерфейсы, а не через прямые ссылки на конкретные классы.
    b. Паттерн Фабрика централизует создание доменных объектов, что помогает избежать дублирования валидационной логики и снижает зависимость от конкретных реализаций.

Запуск программы

Точка входа – файл Program.cs.