РАЗРАБОТКА АРХИТЕКТУРЫ ПРИЛОЖЕНИЯ «ЛИЧНЫЙ БУХГАЛТЕР»

Самниев Д.В., студент
Гилёв А.Ю., старший преподаватель
Бирский Филиал УУНиТ, г. Бирск, Россия

Аннотация. В статье рассматривается проектирование структуры учебного приложения в соответствии с современными практиками. Выделены классы и их обязанности. Приведены основные классы приложения и диаграмма классов.

Ключевые слова: проектирование программного обеспечения, объектно-ориентированное проектирование, шаблоны проектирования.

Введение

Нами рассматривается задача разработки архитектуры простого учебного приложения с использованием современных методов разработки сложного программного обеспечения. Данная задача рассматривается в контексте изучения предмета «Проектирование информационных систем».

Приложение «Личный бухгалтер» призвано систематизировать расходы и доходы пользователя и информировать его о влиянии трат на текущее состояние кошелька.

Архитектура приложения «Личный бухгалтер»

Основные классы приложения: - **Класс BudgetManager** – основной класс, управляющий всеми транзакциями (доходами и расходами). - **Классы Transaction, CreditDeposit, Savings** и другие вспомогательные классы, которые хранят информацию о транзакциях, кредитах и накоплениях. - **Интерфейс**

IBudgetManager для определения методов управления бюджетом. - **Класс Repository**, который отвечает за сохранение и загрузку данных о бюджете.

Класс BudgetManager

Этот класс представляет собой основное хранилище и управление финансов пользователя. Он содержит следующие свойства и методы:

- Свойства:
  - *incomes*: список всех доходов,
  - *expenses*: список всех расходов,
  - *creditDeposits*: список всех кредитов и вкладов,
  - *savingsList*: список всех накоплений.
- Методы:
  - *AddIncome()*: добавляет новый доход,
  - *AddExpense()*: добавляет новый расход,
  - *AddCreditDeposit()*: добавляет кредит или вклад,
  - *AddSavings()*: добавляет накопления,
  - *GetBalance()*: возвращает текущий баланс (доходы минус расходы),
  - *GetHistory()*: возвращает историю всех транзакций,
  - *GetCreditDeposits()*: возвращает список кредитов и вкладов,
  - *GetSavings()*: возвращает список накоплений.

Класс Transaction

Этот класс описывает отдельную транзакцию и включает следующие элементы:

- Свойства:
  - *Amount*: сумма транзакции,
  - *SourceOrCategory*: источник дохода или категория расхода,
  - *Type*: тип транзакции (доход или расход),
- Конструктор: принимает параметры для инициализации объекта Transaction.

Класс CreditDeposit

Этот класс описывает кредиты и вклады. Его элементы включают:

- Свойства:
  - *Amount*: сумма кредита или вклада,
  - *Percent*: процентная ставка,
  - *Type*: тип (кредит или вклад),
  - *SourceOrCategory*: источник или категория,
- Конструктор: инициализирует объект CreditDeposit.

Класс Savings

Этот класс описывает накопления пользователя и включает:

- Свойства:
  - *TargetAmount*: целевая сумма накоплений,
  - *MonthlySavings*: сумма, которую нужно откладывать ежемесячно,
  - *Months*: количество месяцев до достижения цели,
  - *Description*: описание накоплений,
- Конструктор: инициализирует объект Savings.

Интерфейс IBudgetManager

Интерфейс определяет методы, которые должен реализовать класс управления бюджетом:

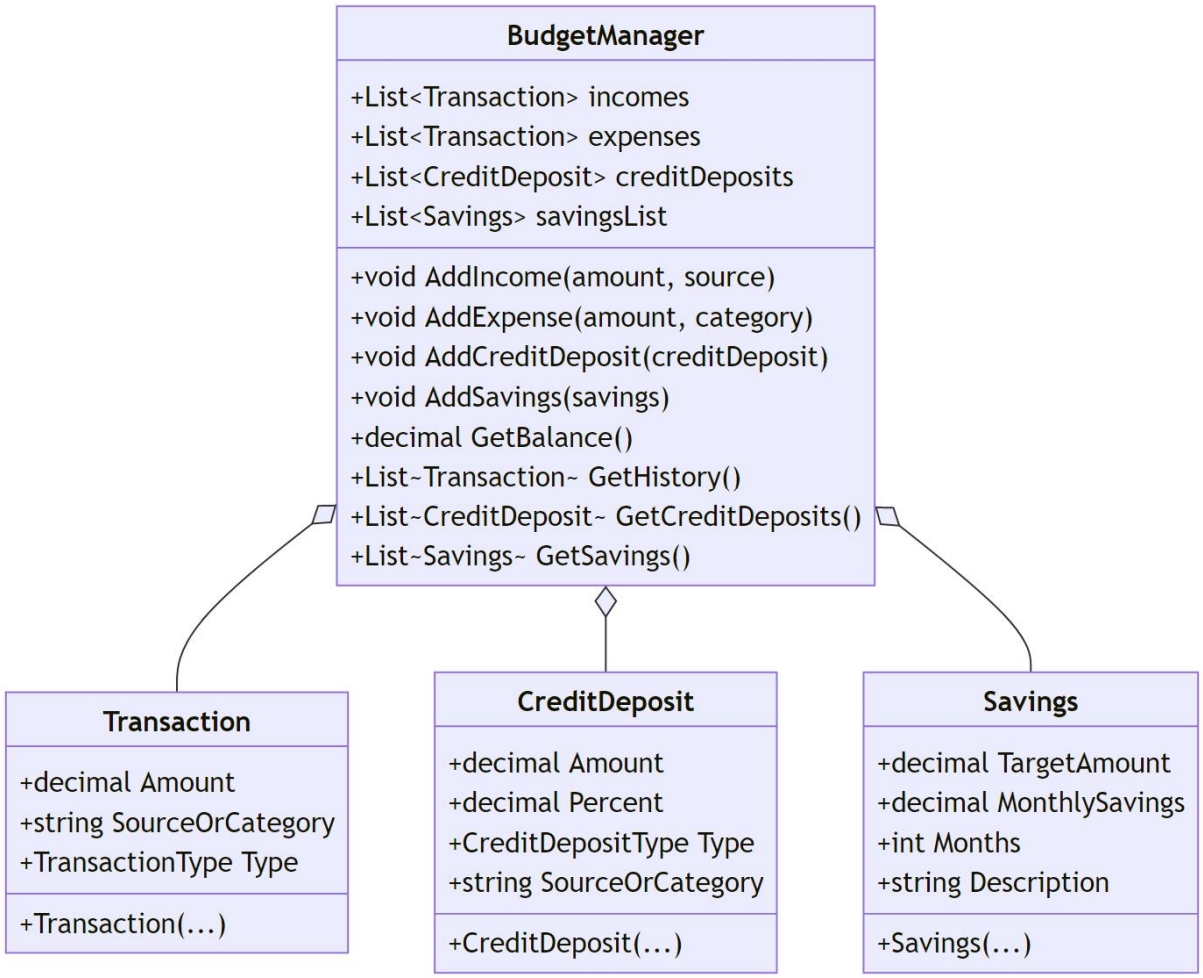
- *AddIncome()*: добавить доход,
- *AddExpense()*: добавить расход,
- *AddCreditDeposit()*: добавить кредит,
- *AddSavings()*: добавить сбережения,
- *GetBalance()*: получить баланс,
- *GetHistory()*: получить историю операций,
- *GetCreditDeposits()*: получить информацию о кредитах,
- *GetSavings()*: получить информацию о сбережениях

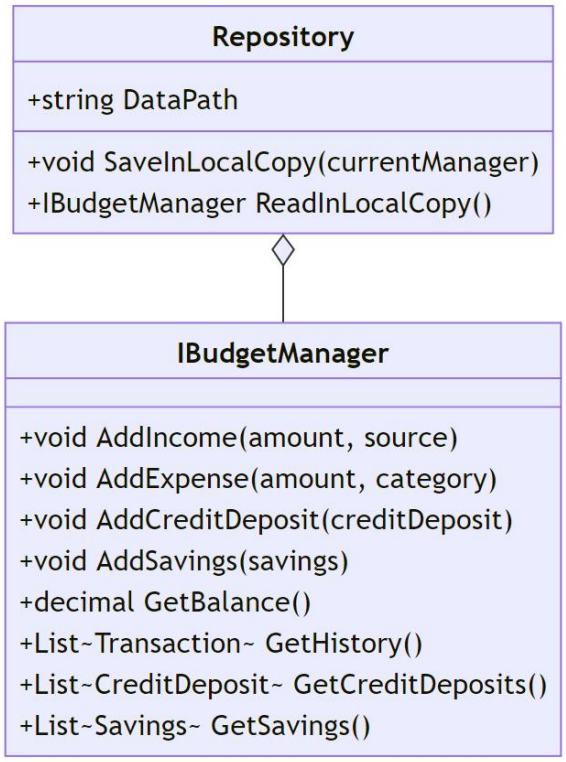
Класс Repository

Этот компонент предназначен для хранения и загрузки данных о бюджете:

- Свойство *DataPath*: путь к файлу для сохранения данных,
- Метод *SaveInLocalCopy()*: сохраняет состояние менеджера бюджета в текстовый файл,
- Метод *ReadInLocalCopy()*: загружает состояние менеджера бюджета из текстового файла.

Диаграмма классов





Пример использования приведённых классов

```
IBudgetManager manager = new BudgetManager();

// Добавление дохода
manager.AddIncome(5000, "Зарплата");

// Добавление расхода
manager.AddExpense(1500, "Питание");

// Получение текущего баланса
Console.WriteLine($"Текущий баланс: {manager.GetBalance()}");
// Вывод: 3500

// Получение истории транзакций
var history = manager.GetHistory();
Console.WriteLine("История:");
foreach (var transaction in history)
{
    Console.WriteLine(
        $"{transaction.Type}: {transaction.SourceOrCategory} -
{transaction.Amount}");
}
```

В этом примере кода демонстрируется процесс создания экземпляра менеджера бюджета, добавления доходов и расходов, а также вывода текущего баланса и истории транзакций.

Выводы

Архитектура приложения «Личный бухгалтер» четко разделяет обязанности между компонентами: класс BudgetManager описывает управление бюджетом, интерфейс IBudgetManager определяет методы для взаимодействия, а классы Transaction, CreditDeposit и Savings содержат данные. Такая структура, гипотетически должна упростить разработку в дальнейшем, если бы это приложение, гипотетически, было гораздо сложнее.

Литература

1. Арораа Г., Чилберто Д. Паттерны проектирования для C# и платформы .NET Core. — СПб.: Питер, 2021. — 352 с.

2. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Д. Приемы объектно-ориентированного проектирования. паттерны проектирования. — СПб.: Питер, 2018. — 368 с.

3. Мартин Р., Мартин М. Принципы, паттерны и методики гибкой разработки на языке C#. – СПб.: Символ-Плюс, 2011. – 768 с.

4. Тепляков С. Паттерны проектирования на платформе .NET. — СПб.: Питер, 2015. — 320 с.