САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ
ФАКУЛЬТЕТ ИНФОКОММУНИКАЦИОННЫХ ТЕХНОЛОГИЙ

Отчет по лабораторной работе №1

по курсу «ООП»

Тема: “Вендинговый автомат”

Выполнил:
Гашимов И.Ф.
К2139
Проверил:
Слюсаренко. С.В

Санкт-Петербург
2025 г.

Проект реализации консольной вендиноговой машины:
Модель продуктов, хранилище монет и продуктов, логика покупок/возвартов и дачи сдачи, сервисный слов для взаимодействия.

Классы: Product, ProductStorage, CoinStorage ICoinStorage, VendingMachine IVendingMachine, VendingMachineService, ConsoleApp.

1. Product - модель продукта

```
/// <summary>
/// Модель продукта: хранит имя, цену и количество.
/// </summary>
Ссылок: 43
public class Product
{
    /// <summary>Имя продукта.</summary>
    Ссылок: 5
    public string Name { get; set; }

    /// <summary>Цена продукта.</summary>
    Ссылок: 7
    public decimal Price { get; set; }

    /// <summary>Количество продукта.</summary>
    Ссылок: 10
    public int Count { get; set; }

    /// <summary>
    /// Создаёт продукт.
    /// </summary>
    /// <param name="name">Имя продукта.</param>
    /// <param name="price">Цена продукта.</param>
    /// <param name="quantity">Количество продукта.</param>
    Ссылок: 34
    public Product(string name, decimal price, int quantity)
    {
        Name = name.Trim();
        Price = price;
        Count = quantity;
    }
}
```

Хранит имя, цену и количество.

ProductStorage - хранение продуктов

```
/// <summary>
/// Хранилище продуктов: добавление, удаление,поиск по имени и формат списка продуктов.
/// </summary>
Ссылок: 14
public class ProductStorage
{
    private List<Product> products = new();
```

AddProduct. **Назначение:** добавляет новый продукт или, если имя совпадает (по GetProductByName), увеличивает количество существующего, без изменения цены.

```
/// <summary>
/// Добавляет продукт в хранилище. Если уже есть — увеличивает количество.
/// Цена не учитывается, если добавляется продукт с существующим названием.
/// </summary>
/// <param name="product">Продукт для добавления.</param>
/// <returns>Всегда true.</returns>
Ссылок: 29
public bool AddProduct(Product product)
{
    Product? curr_product = GetProductByName(product.Name);
    if (curr_product == null)
    {
        products.Add(product);
    }
    else
    {
        curr_product.Count += product.Count;
    }

    return true;
}
```

RemoveProduct. **Назначение:** уменьшает Count указанного экземпляра продукта на count, возвращает true при успешном уменьшении.

```
/// <summary>
/// Удаляет указанное количество продуктов из хранилища.
/// </summary>
/// <param name="product">Ссылка на экземпляр продукта.</param>
/// <param name="count">Количество для удаления.</param>
/// <returns>True при успешном удалении, false при ошибке.</returns>
Ссылок: 4
public bool RemoveProduct(Product? product, int count)
{
    if (product != null && products.Contains(product))
    {
        if (product.Count >= count)
        {
            product.Count -= count;
            return true;
        }
    }
    return false;
}
```

CountProducts. **Назначение:** возвращает количество разных наименований.

```
/// <summary>
/// Возвращает количество разных наименований продуктов.
/// </summary>
/// <returns>Количество наименований.</returns>
Ссылок: 6
public int CountProducts()
{
    return products.Count;
}
```

HasProductByName. **Назначение:** проверяет наличие продукта по имени.

```
/// <summary>
/// Проверяет наличие продукта по имени, регистронезависимо.
/// </summary>
/// <param name="name">Имя продукта.</param>
/// <returns>True если продукт найден.</returns>
Ссылок: 11
public bool HasProductByName(string name)
{
    return products.Any(product => product.Name.Equals(name.Trim(), StringComparison.OrdinalIgnoreCase));
}
```

GetProductByName. **Назначение:** возвращает экземпляр продукта по имени или null.

```
/// <summary>
/// Получает продукт по имени, регистронезависимо.
/// </summary>
/// <param name="name">Имя продукта для поиска.</param>
/// <returns>Экземпляр продукта или null, если не найден.</returns>
Ссылок: 8
public Product? GetProductByName(string name)
{
    return products.FirstOrDefault(product => product.Name.Equals(name.Trim(), StringComparison.OrdinalIgnoreCase));
}
```

GetProducts. **Назначение:** формирует строковое представление всех продуктов.

```
/// <summary>
/// Строковое представление всех продуктов.
/// </summary>
/// <returns>Список продуктов в виде строки.</returns>
Ссылок: 1
public string GetProducts()
{
    string info = string.Empty;
    for (int i = 0; i < products.Count; i++)
    {
        info += $"Продукт: {products[i].Name} | Количество: {products[i].Count} | Цена: {products[i].Price}\n";
    }
    return info;
}
```

ICoinStorage (интерфейс)

```
/// <summary>
/// Хранилище монет: номинал -> количество.
/// </summary>
Ссылок: 4
public interface ICoinStorage
{
    /// <summary>
    /// Добавляет монеты заданного номинала.
    /// </summary>
    /// <param name="nominal">Номинал монеты.</param>
    /// <param name="count">Количество для добавления.</param>
    Ссылок: 29
    void AddCoins(decimal coin, int count);

    /// <summary>
    /// Удаляет указанное количество монет данного номинала, если возможно.
    /// </summary>
    /// <param name="nominal">Номинал монеты.</param>
    /// <param name="count">Количество для удаления.</param>
    /// <returns>True при успешном удалении, false при ошибке.</returns>
    Ссылок: 11
    bool RemoveCoins(decimal coin, int count);
```

```
/// <summary>
/// Возвращает количество монет указанного номинала.
/// </summary>
/// <param name="nominal">Номинал монеты.</param>
/// <returns>Количество (0 если номинал отсутствует).</returns>
Ссылок: 9
int GetCoinCount(decimal coin);

/// <summary>
/// Возвращает список доступных номиналов.
/// </summary>
/// <returns>Список номиналов.</returns>
Ссылок: 7
List<decimal> GetCoinNominals();

/// <summary>
/// Строковое представление состояния кошелька.
/// </summary>
/// <returns>Текст с номиналами и количествами.</returns>
Ссылок: 5
string GetStorage();

/// <summary>
/// Сумма всех монет в хранилище.
/// </summary>
/// <returns>Сумма всех монет в хранилище.</returns>
Ссылок: 12
decimal GetSumAllCoins();
```

CoinStorage (реализация ICoinStorage)

```
/// <summary>
/// Хранилище монет: номинал -> количество.
/// </summary>
Ссылок: 16
public class CoinStorage : ICoinStorage
{
    /// <summary>Словарь номиналов и их количества.</summary>
    Ссылок: 16
    public Dictionary<decimal, int> Coins { get; set; } = new();
```

AddCoins. **Назначение:** добавляет count монет номиналом nominal (если ключа нет — создаёт; иначе прибавляет).

```
Ссылок: 29
public void AddCoins(decimal nominal, int count)
{
    if (!Coins.TryAdd(nominal, count))
    {
        Coins[nominal] += count;
    }
}
```

GetCoinCount. **Назначение:** возвращает количество монет указанного номинала.

```
Ссылок: 9
public int GetCoinCount(decimal nominal)
{
    if (Coins.TryGetValue(nominal, out int value))
    {
        return value;
    }
    else
    {
        return 0;
    }
}
```

RemoveCoins. **Назначение:** пытается убрать count монет данного номинала; возвращает true если достаточно монет и удаление выполнено.

```
Ссылок: 11
public bool RemoveCoins(decimal nominal, int count)
{
    if (Coins.TryGetValue(nominal, out _))
    {
        if (Coins[nominal] >= count)
        {
            Coins[nominal] -= count;
            return true;
        }
    }
    return false;
}
```

GetCoinNominals. **Назначение :** вернуть список доступных номиналов.

```
Ссылок: 7
public List<decimal> GetCoinNominals()
{
return [.. Coins.Keys];
}
```

GetStorage. **Назначение:** формирует строковое представление состояния кошелька (номинал - количество).

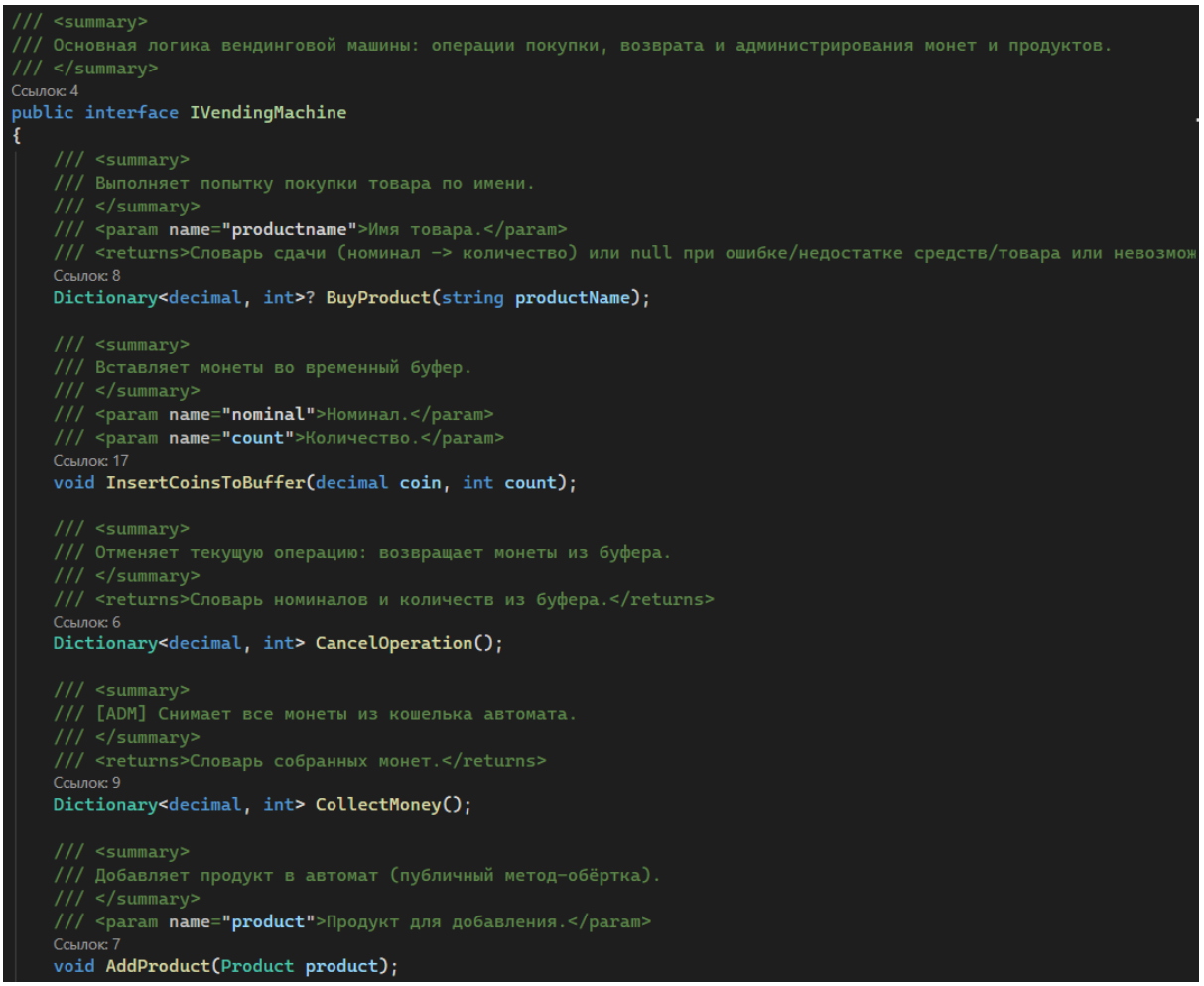
```
Ссылок: 5
public string GetStorage()
{
    string storage = "CoinStorage:\n";

    foreach (var kvp in Coins)
    {
    storage += $"Номинал: {kvp.Key} | Количество: {kvp.Value}\n";
    }
    return storage;
}
```

GetSumAllCoins. **Назначение:** суммирует общую сумму денег в хранилище (сумма nominal * count по всем номиналам).

```
Ссылок: 12
public decimal GetSumAllCoins()
{
    decimal summ_coins = 0;
    foreach(var kvp in Coins)
    {
        summ_coins += kvp.Key* kvp.Value;
    }
    return summ_coins;
}
```

IVendingMachine (интерфейс)



```
/// <summary>
/// Основная логика вендинговой машины: операции покупки, возврата и администрирования монет и продуктов.
/// </summary>
Ссылок: 4
public interface IVendingMachine
{
    /// <summary>
    /// Выполняет попытку покупки товара по имени.
    /// </summary>
    /// <param name="productname">Имя товара.</param>
    /// <returns>Словарь сдачи (номинал -> количество) или null при ошибке/недостатке средств/товара или невозмож
    Ссылок: 8
    Dictionary<decimal, int>? BuyProduct(string productName);

    /// <summary>
    /// Вставляет монеты во временный буфер.
    /// </summary>
    /// <param name="nominal">Номинал.</param>
    /// <param name="count">Количество.</param>
    Ссылок: 17
    void InsertCoinsToBuffer(decimal coin, int count);

    /// <summary>
    /// Отменяет текущую операцию: возвращает монеты из буфера.
    /// </summary>
    /// <returns>Словарь номиналов и количеств из буфера.</returns>
    Ссылок: 6
    Dictionary<decimal, int> CancelOperation();

    /// <summary>
    /// [ADM] Снимает все монеты из кошелька автомата.
    /// </summary>
    /// <returns>Словарь собранных монет.</returns>
    Ссылок: 9
    Dictionary<decimal, int> CollectMoney();

    /// <summary>
    /// Добавляет продукт в автомат (публичный метод-обёртка).
    /// </summary>
    /// <param name="product">Продукт для добавления.</param>
    Ссылок: 7
    void AddProduct(Product product);
```

```
/// <summary>
/// [ADM] Добавляет монеты в кошелёк автомата.
/// </summary>
/// <param name="nominal">Номинал.</param>
/// <param name="count">Количество.</param>
Ссылок: 16
void AddCoinsToVM(decimal nominal, int count);

/// <summary>
/// Строковое представление списка продуктов автомата (обертка).
/// </summary>
/// <returns>Список продуктов.</returns>
Ссылок: 3
string GetProducts();
```

VendingMachine

```
/// <summary>
/// Основная логика вендинговой машины: операции покупки, возврата и администрирования монет и продуктов.
/// </summary>
Ссылок: 19
public class VendingMachine : IVendingMachine
{
    Ссылок: 5
    private CoinStorage VendingWallet { get; set; } = new();
    Ссылок: 5
    private ProductStorage VendingProducts { get; set; } = new();
    Ссылок: 9
    private CoinStorage VendingBuffer { get; set; } = new();
```

AddProduct. **Назначение:** публичная обёртка для добавления продукта в VendingProducts.

```
Ссылок: 7
public void AddProduct(Product product)
{
    VendingProducts.AddProduct(product);
}
```

InsertCoinsToBuffer. **Назначение:** добавляет монеты в буфер, вставленные пользователем.

```
Ссылок: 17
public void InsertCoinsToBuffer(decimal nominal, int count)
{
    VendingBuffer.AddCoins(nominal, count);
}
```

BuyProduct. **Назначение:** основная логика покупки: проверяет товар и внесённую сумму (из буфера), формирует сдачу через GetChange, снимает монеты из буфера/кошелька, переносит остаток буфера в кошелёк автомата, уменьшает Count продукта и возвращает словарь сдачи (номинал→количество) или null при ошибке.

```
Ссылок: 8
public Dictionary<decimal, int>? BuyProduct(string productname)
{
    // Проверка на существование товара по заданному имени
    if (!VendingProducts.HasProductByName(productname)) return null;

    Product? product = VendingProducts.GetProductByName(productname);

    // Очень странная проверка, написана чтобы visual studio не жаловалась
    if (product == null) return null;

    decimal paidSumm = VendingBuffer.GetSumAllCoins();

    // Проверка на наличие товара и достаточно ли внесенных средств для оплаты
    if (product.Price > paidSumm || product.Count == 0) return null;

    decimal change = paidSumm - product.Price;

    Dictionary<decimal, int> dict_change = GetChange(change);

    if (change > 0)
    {

        // Если не удалось сформировать сдачу, сдача обязательно должна быть т.к. change > 0
        if (dict_change.Count == 0) return null;

        foreach (var coin in dict_change)
        {
            // формирование количества монет, которая будет снята с буффера
            int changeFromBuffer = Math.Min(VendingBuffer.GetCoinCount(coin.Key), coin.Value);


            VendingBuffer.RemoveCoins(coin.Key, changeFromBuffer);

            // остаток сдачи снимаем с кошелька автомата
            VendingWallet.RemoveCoins(coin.Key, coin.Value - changeFromBuffer);
        }
    }

    // Перенос всех монет с буффера в кошелек автомата
    TransferAllCoinsFromBufferToVendingWallet();

    // Уменьшаем количество продукта
    VendingProducts.RemoveProduct(product, 1);

    return dict_change;
}
```

AddCoinsToVM. **Назначение:** добавляет монеты в внутренний кошелёк автомата.

```
Ссылок: 16
public void AddCoinsToVM(decimal nominal, int count)
{
    VendingWallet.AddCoins(nominal, count);
}
```

CancelOperation. **Назначение:** возвращает копию содержимого буфера и очищает буфер (реализует возврат вставленных пользователем монет).

```
Ссылок: 6
public Dictionary<decimal, int> CancelOperation()
{
    Dictionary<decimal, int> bufferCoins = new Dictionary<decimal, int>(VendingBuffer.Coins);

    // Монеты возвращаем пользователю в другом классе сервис
    VendingBuffer.Coins.Clear();

    return bufferCoins;
}
```

CollectMoney. **Назначение:** возвращает текущие монеты из внутреннего кошелька автомата и очищает его.

```
Ссылок: 9
public Dictionary<decimal, int> CollectMoney()
{
    Dictionary<decimal, int> collectedCoins = new(VendingWallet.Coins);

    // Через сервис начисляем собранные монету пользователю под админкой
    VendingWallet.Coins.Clear();

    return collectedCoins;
}
```

GetProducts. **Назначение:** обёртка для получения строкового списка продуктов.

```
Ссылок: 3
public string GetProducts()
{
    return VendingProducts.GetProducts();
}
```

TransferAllCoinsFromBufferToVendingWallet. **Назначение:** переносит все монеты из буфера в кошелёк автомата.

```
/// <summary>
/// Перенос всех монет из буфера в кошелек автомата.
/// </summary>
Ссылок: 3
internal void TransferAllCoinsFromBufferToVendingWallet()
{
    // Копия нужна чтобы итерироваться по всем элеменетам и не столкунться с ошибкой в foreach
    var copy_buffer = new Dictionary<decimal, int>(VendingBuffer.Coins);
    foreach(var kvp in copy_buffer)
    {
        AddCoinsToVM(kvp.Key, kvp.Value);
        VendingBuffer.RemoveCoins(kvp.Key, kvp.Value);
    }
}
```

GetChange. **Назначение:** пытается сформировать сдачу change (>=0) жадным алгоритмом, комбинируя монеты из буфера и кошелька автомата; возвращает словарь требуемой сдачи или пустой словарь, если собрать нельзя.

```
/// <summary>
/// Попытка составить сдачу для заданной суммы из кошелька и буффера взятых вместе.
/// </summary>
/// <param name="change">Необходимая сумма сдачи (>= 0).</param>
/// <returns>Словарь (номинал -> количество) требуемой сдачи или пустой словарь если собрать нельзя.</returns>
Ссылок: 5
internal Dictionary<decimal, int> GetChange(decimal change)
{
    // Словарь всех монет из буффера и кошелька. При таком подходе сдача с большим шансом сформируется т.к. в диа
    Dictionary<decimal, int> temp_wallet = new();

    foreach (var kvp in VendingBuffer.Coins)
    {
        temp_wallet[kvp.Key] = kvp.Value;
    }
    foreach (var kvp in VendingWallet.Coins)
    {
        if (temp_wallet.ContainsKey(kvp.Key)) temp_wallet[kvp.Key] += kvp.Value;
        else temp_wallet[kvp.Key] = kvp.Value;
    }

    Dictionary<decimal, int> dict_change = new();


    decimal decimal_change = change;

    // Словарь для жадного алгоритма, ключ в сортировке значение номинала
    List<decimal> sorted_wallet = [.. temp_wallet.Keys.OrderByDescending(v => v)];

    foreach (var coin in sorted_wallet)
    {
        // Защита от ошибок
        if (decimal_change <= 0)
        {
            break;
        }
```

```
        if (temp_wallet[coin] > 0 && coin <= decimal_change)
        {
            // число максимального возможного количества монет данного номинала
            int count_curr_coin = temp_wallet[coin] > (int)(decimal_change / coin) ? (int)(decimal_change / coin)

            if (count_curr_coin > 0)
            {
                if (!dict_change.ContainsKey(coin)) dict_change[coin] = count_curr_coin;
                else dict_change[coin] += count_curr_coin;

                decimal_change -= count_curr_coin * coin;

                temp_wallet[coin] -= count_curr_coin;
            }
        }
    }
    // При неудачной формировке сдачи. Возвращаем пустой словарь.
    if (decimal_change > 0)
    {
        return new Dictionary<decimal, int>();
    }
    return dict_change;
}
```

VendingMachineService

```
/// <summary>
/// Сервис для взаимодействия пользователя и автомата (кошелёк пользователя + автомат).
/// </summary>
Ссылок: 3
public class VendingMachineService
{
    private readonly IVendingMachine _vendingMachine;
    private readonly ICoinStorage _userWallet;
    private Dictionary<decimal, int> _change;

    /// <summary>
    /// Создаёт сервис для указанного автомата и кошелька пользователя.
    /// </summary>
    /// <param name="vendingMachine">Экземпляр автомата.</param>
    /// <param name="userWallet">Кошелёк пользователя.</param>
    Ссылок: 1
    public VendingMachineService(IVendingMachine vendingMachine, ICoinStorage userWallet)
    {
        _vendingMachine = vendingMachine;
        _userWallet = userWallet;
        _change = new Dictionary<decimal, int>();
    }
```

InfoBar. **Назначение:** возвращает текст меню операций для консоли.

```
/// <summary>
/// Возвращает текст меню операций.
/// </summary>
/// <returns>Строка меню.</returns>
Ссылок: 1
public string InfoBar()
{
    string info = "Введите номер операции:\n" +
        "1.Посмотреть список товаров\n" +
        "2.Вставить монеты\n" +
        "3.Выбрать товар и получить\n" +
        "4.Получить сдачу.\n" +
        "5.Отменить операцию.\n" +
        "6.[ADMIN] Пополнить ассортимент\n" +
        "7.[ADMIN] Сбор собранных средств\n";

    return info;
}
```

InsertCoins. **Назначение:** переносит монеты из кошелька пользователя в буфер автомата; проверяет наличие номинала и достаточного количества в кошельке пользователя.

```
/// <summary>
/// Переносит монеты из кошелька пользователя в буфер автомата.
/// </summary>
/// <param name="nominal">Номинал монеты.</param>
/// <param name="count">Количество монет.</param>
/// <returns>True при успешной вставке, false если номинал отсутствует или недостаточн
Ссылок: 1
public bool InsertCoins(decimal nominal, int count)
{
    if (!_userWallet.GetCoinNominals().Contains(nominal)) return false;

    if (_userWallet.GetCoinCount(nominal) < count) return false;

    _userWallet.RemoveCoins(nominal, count);
    _vendingMachine.InsertCoinsToBuffer(nominal, count);

    return true;
}
```

BuyProduct. **Назначение:** вызывает BuyProduct автомата; если возвращена сдача (словарь), копирует её во внутреннюю _change.

```
/// <summary>
/// Попытка купить товар по имени. Сформированную сдачу складывает во внутренний буфер
/// </summary>
/// <param name="productName">Имя товара.</param>
/// <returns>True при успешной покупке.</returns>
Ссылок: 1
public bool BuyProduct(string productName)
{
    Dictionary<decimal, int>? newChange = _vendingMachine.BuyProduct(productName);

    if (newChange == null) return false;

    foreach (var kvp in newChange)
    {

        if(_change.ContainsKey(kvp.Key))
        {
            _change[kvp.Key] += kvp.Value;
        }
        else
        {
            _change.Add(kvp.Key, kvp.Value);
        }
    }

    return true;
}
```

ReturnChange. **Назначение:** возвращает накопленную сдачу пользователю (переносит монеты в кошелёк пользователя и очищает внутренний буфер).

```
/// <summary>
/// Возвращает накопленную сдачу пользователю (перенос в кошелек пользователя).
/// </summary>
/// <returns>True если была и успешно возвращена сдача.</returns>
Ссылок: 1
public bool ReturnChange()
{
    if (_change == null || _change.Count == 0) return false;

    foreach (var kvp in _change)
    {
        _userWallet.AddCoins(kvp.Key, kvp.Value);
    }

    _change.Clear();

    return true;
}
```

CancelOperation. **Назначение:** отменяет операцию: получает содержимое буфера из автомата и возвращает монеты пользователю.

```
/// <summary>
/// Отмена операции, пользователь получает монеты из буффера автомата.
/// </summary>
/// <returns>True при успешном возврате, False при отсутствии монет в буфере.</returns
Ссылок: 1
public bool CancelOperation()
{
    Dictionary<decimal, int> refund = _vendingMachine.CancelOperation();

    if (refund.Count == 0) return false;

    foreach (var kvp in refund)
    {
        _userWallet.AddCoins(kvp.Key, kvp.Value);
    }

    return true;
}
```

AddProducts. **Назначение:** добавляет продукт в автомат (создаёт Product и передаёт в автомат).

```
/// <summary>
/// [ADM] Добавляет продукты в автомат.
/// </summary>
/// <param name="productName">Имя товара.</param>
/// <param name="price">Цена товара (учитывается при создании нового имени).</param>
/// <param name="count">Количество для добавления.</param>
Ссылок: 6
public void AddProducts(string productName,decimal price, int count)
{
    _vendingMachine.AddProduct(new Product(productName, price, count));
}
```

AddCoins. **Назначение:** пополняет кошелёк автомата монетами (админ-операция).

```
/// <summary>
/// [ADM] Пополняет внутренний кошелёк автомата монетами, не из кошелька.
/// </summary>
/// <param name="nominal">Номинал.</param>
/// <param name="count">Количество.</param>
Ссылок: 5
public void AddCoins(decimal nominal, int count)
{
    _vendingMachine.AddCoinsToVM(nominal, count);
}
```

CollectMoney. **Назначение:** собирает деньги из автомата и добавляет их в кошелёк пользователя (админ-операция).

```
/// <summary>
/// [ADM] Собирает деньги из автомата и добавляет их в кошелёк пользователя.
/// </summary>
Ссылок: 1
public void CollectMoney()
{
    Dictionary<decimal, int> collectedMoney = _vendingMachine.CollectMoney();

    foreach (var kvp in collectedMoney)
    {
        _userWallet.AddCoins(kvp.Key, kvp.Value);
    }
}
```

ConsoleApp

```
/// <summary>
/// Консольное приложение: input и output система и работа с сервисом.
/// </summary>
Ссылок: 1
public static class ConsoleApp
{
    private static IVendingMachine _vendingMachine = new VendingMachine();
    private static ICoinStorage _userWallet = new CoinStorage();
    private static VendingMachineService _VMS = new (_vendingMachine, _userWallet);
    private static readonly int adminPass = 123132;
```

Run. **Назначение:** основной цикл приложения: инициализация данных и чтение команд из консоли.

```
/// <summary>
/// Запуск консольного приложения
/// </summary>
Ссылок: 1
public static void Run()
{
    Init_Data();
    while (true)
    {
        Console.WriteLine("\n\n"+_VMS.InfoBar());
        string? input = Console.ReadLine();
        ConsoleInput(input);
    }
}
```

ShowProducts. **Назначение:** выводит список товаров в консоль.

```
/// <summary>
/// Показывает список продуктов.
/// </summary>
Ссылок: 1
private static void ShowProducts()
{
    Console.WriteLine("Список продуктов:\n");
    Console.WriteLine(_vendingMachine.GetProducts());
}
```

Init_Data. **Назначение:** заполняет автомат и кошелёк пользователя начальными данными.

```
/// <summary>
/// Инициализация данных для приложения.
/// </summary>
Ссылок: 1
private static void Init_Data()
{

    // Добавление продуктов в автомат
    _VMS.AddProducts("Egg", 1m, 5);
    _VMS.AddProducts("Sprunk", 0.7m, 4);
    _VMS.AddProducts("Bread", 2.5m, 3);
    _VMS.AddProducts("Meat", 4m, 2);
    _VMS.AddProducts("Orange", 2m, 1);

    // Добавление денег в автомат
    _VMS.AddCoins(0.2m, 100);
    _VMS.AddCoins(0.5m, 100);
    _VMS.AddCoins(1m, 50);
    _VMS.AddCoins(2m, 40);
    _VMS.AddCoins(5m, 30);

    //Добавление денег в кошелек пользователя
    _userWallet.AddCoins(0.2m, 5);
    _userWallet.AddCoins(0.5m, 4);
    _userWallet.AddCoins(1m, 2);
    _userWallet.AddCoins(2m, 1);
    _userWallet.AddCoins(5m, 1);
}
```

ConsoleInput. **Назначение:** маршрутизирует ввод пользователя на конкретные методы.

```
/// <summary>
/// Выбор команд через консоль.
/// </summary>
/// <param name="input">Ввод пользователя</param>
Ссылок: 1
private static void ConsoleInput(string? input)
{
    switch (input)
    {
        case "1": ShowProducts(); break;
        case "2": InsertCoins(); break;
        case "3": BuyProduct(); break;
        case "4": ReturnChange(); break;
        case "5": CancelOperation(); break;
        case "6": AddProduct(); break;
        case "7": CollectMoney(); break;
        default: Console.WriteLine("Неверный выбор"); break;
    }
}
```

InsertCoins. **Назначение:** запрашивает номинал и количество, валидирует ввод и вызывает сервис для вставки монет в буфер.

```
/// <summary>
/// Вставка монет пользователем.
/// </summary>
Ссылок: 1
private static void InsertCoins()
{
    Console.WriteLine("Внесение монет\n" + "Кошелек:\n" + $"{_userWallet.GetStorage()}");
    Console.WriteLine("Введите номинал монеты (например 0.2, 0.5, 1, 2, 5): ");

    string? inputNominal = Console.ReadLine();

    if (!decimal.TryParse(inputNominal, CultureInfo.InvariantCulture, out decimal nominal)
        || !_userWallet.GetCoinNominals().Contains(nominal))
    {
        Console.WriteLine("[Ошибка] Неверный формат номинала");
        return;
    }

    Console.WriteLine($"Введите количество монет номинала {inputNominal}, которые хотите внести:");
    string? inputCount = Console.ReadLine();
    if (!int.TryParse(inputCount, out int count) || count <= 0)
    {
        Console.WriteLine("[Ошибка] Неверный формат количества монет");
        return;
    }
    if (!_VMS.InsertCoins(nominal, count))
    {
        Console.WriteLine("[Ошибка] Введенный номинал не найден либо у вас отсутсвует данное количет
        return;
    }
    Console.WriteLine($"Вы успешно пополнили автомат на {count} монет номинала {nominal}");
}
```

BuyProduct. **Назначение:** Вызов покупки: ввод названия, вызов VendingMachineService.BuyProduct, вывод результата.

```
/// <summary>
/// Покупка товара
/// </summary>
Ссылок: 1
public static void BuyProduct()
{
    Console.WriteLine("Покупка товара\n" + "Список товаров:\n" + $"{_vendingMachine.GetProducts()}");
    Console.WriteLine("Введите название товара:");

    string? inputName = Console.ReadLine();

    if (inputName != null && _VMS.BuyProduct(inputName))
    {
        Console.WriteLine($"Вы успешно купили товар {inputName}");
        Console.WriteLine("Сдача сформирована, для получения введите 4");
    }
    else
    {
        Console.WriteLine("[Ошибка] Вы не смогли купить товар");
    }
}
```

ReturnChange. **Назначение:** Вызов получения накопленной сдачи у сервиса.

```
/// <summary>
/// Получение сдачи пользователем.
/// </summary>
Ссылок: 1
public static void ReturnChange()
{
    Console.WriteLine("Получение сдачи");

    if(_VMS.ReturnChange())
    {
        Console.WriteLine("Вы успешно получили сдачу");
        Console.WriteLine("Текущее состояние кошелька:\n" + _userWallet.GetStorage());
    }
    else
    {
        Console.WriteLine("[Критическая ошибка] Что-то пошло не так");
    }
}
```

CancelOperation. **Назначение:** отменяет текущую операцию и возвращает монеты пользователю через сервис.

```
/// <summary>
/// Отмена операции, получаем вставленные монеты (из буффера).
/// </summary>
Ссылок: 1
public static void CancelOperation()
{
    Console.WriteLine("Отмена операции");

    if( _VMS.CancelOperation())
    {
        Console.WriteLine("Операция успешно отменена. Деньги возвращены.");
        Console.WriteLine("Текущее состояние кошелька:\n" + _userWallet.GetStorage());
    }
    else
    {
        Console.WriteLine("[Ошибка] Отсутсвует операция для отмены");
    }
}
```

AddProduct. **Назначение:** Добавление продукта через админку и вызов сервиса.

```
/// <summary>
/// [ADM] Добавление товара
/// </summary>
Ссылок: 1
public static void AddProduct()
{
    Console.WriteLine("Добавление товара");
    Console.WriteLine("Введите админ-пароль:");

    string? inputAdminPass = Console.ReadLine();

    if (inputAdminPass == null || !int.TryParse(inputAdminPass, out int AdminPass) || AdminPass != adminPass)
    {
        Console.WriteLine("Вы ввели неверный админ-пароль");
        return;
    }

    Console.Write("Введите название товара:");
    string? inputName = Console.ReadLine();
    if (string.IsNullOrEmpty(inputName)) { Console.WriteLine("[Ошибка] Название не может быть пустым"); return;

    Console.Write("Введите цену товара(цена будет учитываться только в случае нового товара):");
    string? inputPrice = Console.ReadLine();
    if (!decimal.TryParse(inputPrice, CultureInfo.InvariantCulture, out decimal price))
    {
        Console.WriteLine("[Ошибка] Неверный формат цены");
        return;
    }

    Console.WriteLine("Введите количество товара:");
    string? inputCount = Console.ReadLine();
    if (!int.TryParse(inputCount, out int count))
    {
        Console.WriteLine("[Ошибка] Неверный формат количества товара");
        return;
    }

    _VMS.AddProducts(inputName, price, count);
    Console.WriteLine($"Товар {inputName} в количестве {count} успешно добавлен в автомат");
}
```

CollectMoney. **Назначение:** Сбор денег через админку, вызывает VendingMachineService.CollectMoney.

```
/// <summary>
/// [ADM] Сбор денег из автомата в кошелек.
/// </summary>
Ссылок: 1
public static void CollectMoney()
{
    Console.WriteLine("Введите админ-пароль:");

    string? inputAdminPass = Console.ReadLine();

    if (inputAdminPass == null || !int.TryParse(inputAdminPass, out int AdminPass) || AdminPass != adminPass)
    {
        Console.WriteLine("Вы ввели неверный админ-пароль");
        return;
    }

    Console.WriteLine("Сбор денег");
    _VMS.CollectMoney();
    Console.WriteLine("Вы успешно собрали деньги с автомата и перенесли в кошелек");

    Console.WriteLine("Текущее состояние кошелька:\n" + _userWallet.GetStorage());
}
```

Program. **Назначение:** точка входа — запускает консольное приложение.

```
Ссылок: 0
class Program
{
    Ссылок: 0
    static void Main()
    {
        ConsoleApp.Run();
    }
}
```

Так-же были написаны тесты, которые успешно проходят.

```
[xUnit.net 00:00:00.00] xUnit.net VSTest Adapter v2.5.3.1+6b60a9e56a (64-bit .NET 8.0.20)
[xUnit.net 00:00:00.75]   Discovering: MyFirstProject.Tests
[xUnit.net 00:00:00.78]   Discovered:  MyFirstProject.Tests
[xUnit.net 00:00:00.79]   Starting:    MyFirstProject.Tests
[xUnit.net 00:00:01.21]   Finished:    MyFirstProject.Tests
  MyFirstProject.Tests (тест) успешно выполнено (3,8 с)

Сводка теста: всего: 41; сбой: 0; успешно: 41; пропущено: 0; длительность: 3,8 с
Сборка успешно выполнено через 8,6 с
```

Итоги:

В результате мы получили консольную программу: реализация содержит модель продукта, хранилище продуктов, хранилище монет с операциями добавления/удаления/подсчёта, бизнес-логику автомата (покупка, формирование сдачи, буфер), сервисный слой для взаимодействия с пользователем и консольный интерфейс с админ-операциями.