Министерство науки и высшего образования Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

(Университет ИТМО)

Факультет **Инфокоммуникационных технологий**

Образовательная программа **Мобильные и сетевые технологии**

Направление подготовки(специальность) **09.03.03 Прикладная информатика**

ЛАБОРАТОРНАЯ РАБОТА №1

Тема: Моделирование торгового автомата

Выполнил Нгуен Динь Нам; К3240.

Проверил

Дата _____

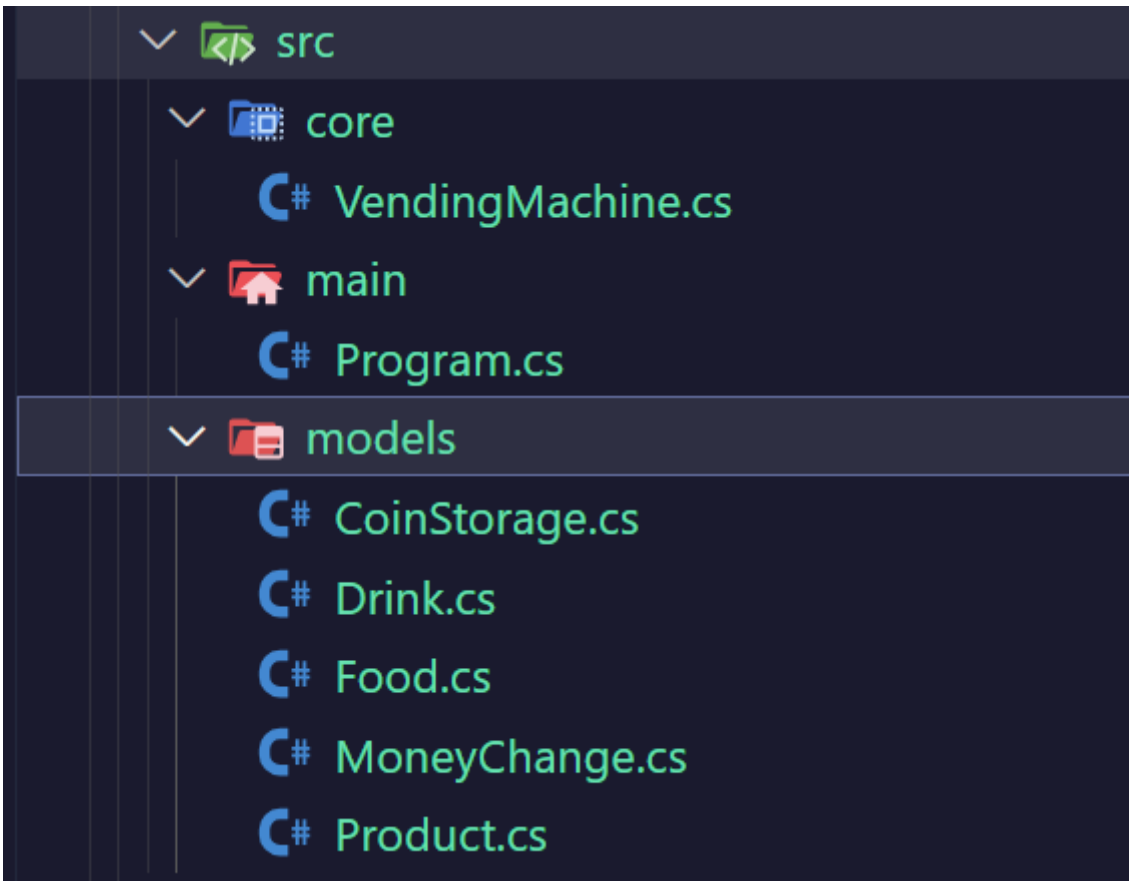
Санкт-Петербург 2025

Цель работы

Цель практической работы — применить принципы объектно-ориентированного программирования для разработки системы торгового автомата. Система должна обеспечивать проверку корректности данных (номинал внесённых средств, идентификатор товара, остатки), поддерживать обработку операций (внесение денег, покупка, возврат средств, выдача сдачи) и режим администратора (пополнение товара, инкассация).

Ход работы

Экран 1.



- В этой лабораторной работе я организовал проект на три небольшие папки:

  + **Main – Program.cs**: инициализация приложения и VendingMachine, вывод меню, обработка выбора, проверка ввода, печать результата/ошибок.

+ **Core – VendingMachine**: основная бизнес-логика: внесение средств, покупка по Id, возврат, расчёт и запрос сдачи, проверка остатка/достаточности средств, админ-действия.

+ **Models – Product.cs, Drink.cs, Food.cs, CoinStorage.cs, MoneyChange.cs**: модели предметной области и вспомогательные структуры данных.

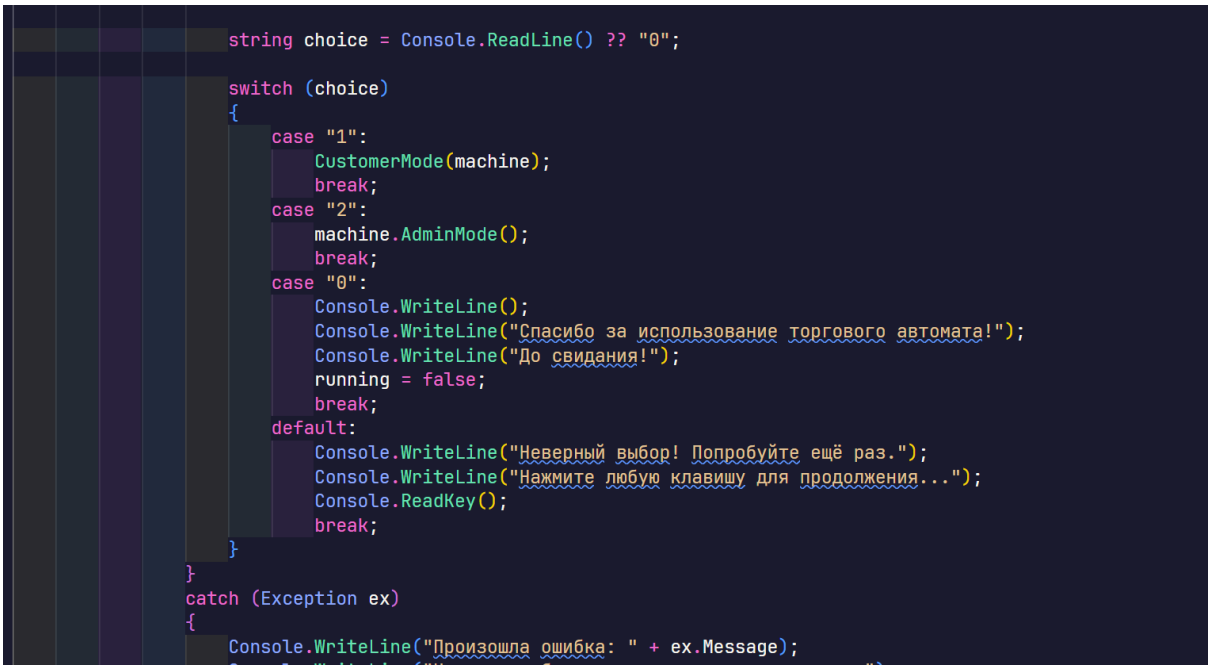
1. Main – Program.cs

Экран 2.

```
using System;
using Lab1.Core;

namespace Lab1
{
    0 references
    class Program
    {
        0 references
        static void Main(string[] args)
        {
            Console.OutputEncoding = System.Text.Encoding.UTF8;
            Console.InputEncoding = System.Text.Encoding.UTF8;

            VendingMachine machine = new VendingMachine();
            bool running = true;

            while (running)
            {
                try
                {
                    Console.Clear();
                    Console.WriteLine("╔═══════════════════════════════╗");
                    Console.WriteLine("║      ТОРГОВЫЙ АВТОМАТ ITMO    ║");
                    Console.WriteLine("╠═══════════════════════════════╣");
                    Console.WriteLine("║ 1. Режим покупателя           ║");
                    Console.WriteLine("║ 2. Режим администратора       ║");
```

- В этом файле я запускаю консольное приложение, связываю модули **VendingMachine**, **Product**, **Drink**, **Food**, **CoinStorage**, **MoneyChange**.
- Показываю меню, даю пользователю видеть ассортимент и внесённую сумму, выбирать действия и получать понятные сообщения.

Экран 3.



```
            string choice = Console.ReadLine() ?? "0";

            switch (choice)
            {
                case "1":
                    CustomerMode(machine);
                    break;
                case "2":
                    machine.AdminMode();
                    break;
                case "0":
                    Console.WriteLine();
                    Console.WriteLine("Спасибо за использование торгового автомата!");
                    Console.WriteLine("До свидания!");
                    running = false;
                    break;
                default:
                    Console.WriteLine("Неверный выбор! Попробуйте ещё раз.");
                    Console.WriteLine("Нажмите любую клавишу для продолжения...");
                    Console.ReadKey();
                    break;
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine("Произошла ошибка: " + ex.Message);
```

- Я поддерживаю депозит, покупку по идентификатору, возврат, вход в режим администратора с паролем и выход из программы.
- Обработка входных данных с помощью “TryParse”, фильтрация недопустимых значений, запрос повторного ввода в случае некорректности.
- Я организую работу через главный цикл меню, переключаюсь между режимом пользователя и админа, а операции передаю в VendingMachine, чтобы интерфейс был простым.

Экран 4.

```
while (customerRunning)
{
    try
    {
        Console.Clear();

        machine.ShowProducts();
        machine.ProcessCustomerInteraction();

        Console.WriteLine();
        Console.WriteLine("═══════════════════════════════════════════════════");
        Console.WriteLine("1. Продолжить покупки");
        Console.WriteLine("0. Вернуться в главное меню");
        Console.WriteLine("═══════════════════════════════════════════════════");
        Console.Write("Выберите действие: ");

        string continueChoice = Console.ReadLine() ?? "0";

        switch (continueChoice)
        {
            case "1":
                Console.Clear();
                break;
            case "0":
                customerRunning = false;
                break;
            default:
                Console.WriteLine("Неверный выбор! Попробуйте ещё раз.");
                Console.WriteLine("Нажмите любую клавишу для продолжения...");
                Console.ReadKey();
                Console.Clear();
```

2. Models – Product.cs

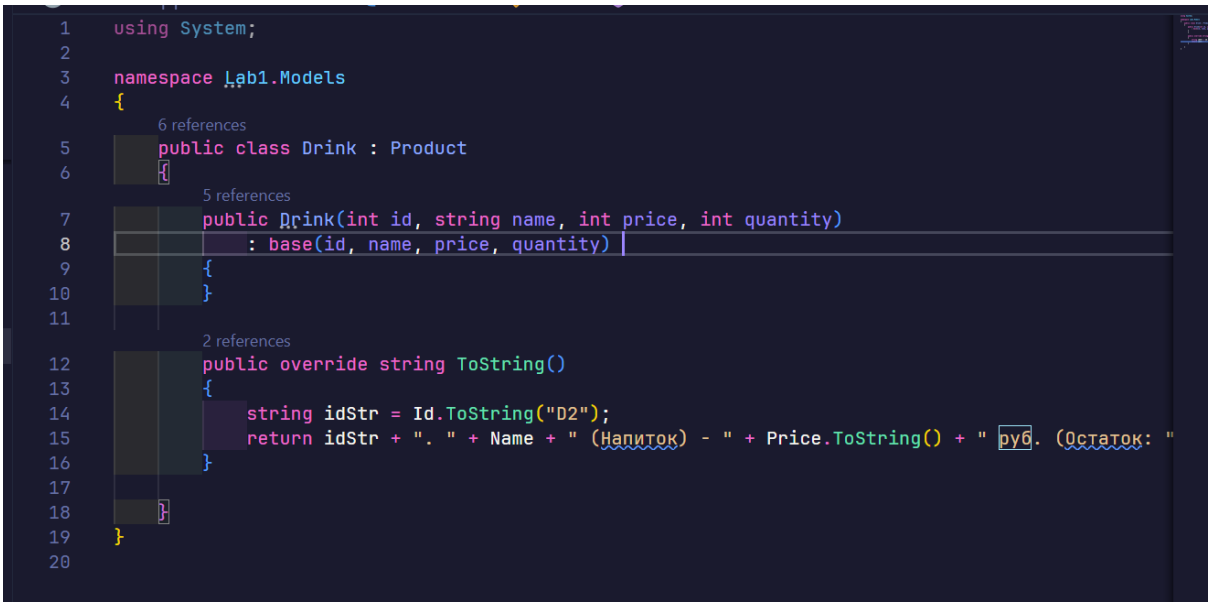
Экран 5.

```
13 references
public class Product
{
    8 references
    public int Id { get; set; }
    7 references
    public string Name { get; set; }
    13 references
    public int Price { get; set; }
    9 references
    public int Quantity { get; set; }

    2 references
    public Product(int id, string name, int price, int quantity)
    {
        if (id <= 0)
            throw new ArgumentException("Номер товара должен быть положительным", nameof(id));
        if (string.IsNullOrWhiteSpace(name))
            throw new ArgumentException("Название товара не может быть пустым", nameof(name));
        if (price < 0)
            throw new ArgumentException("Цена не может быть отрицательной", nameof(price));
        if (quantity < 0)
            throw new ArgumentException("Количество не может быть отрицательным", nameof(quantity));

        Id = id;
        Name = name;
        Price = price;
        Quantity = quantity;
    }
```

- **Product** используется как базовый класс, а Drink и Food наследуются от него для повторного использования общего кода.

- Я задаю единый шаблон данных для всех товаров с полями “Id”, “Name”, “Price”, “Quantity”, проверяю корректность при создании и обновлении.
- Благодаря вынесению общего функционала в **Product** я уменьшаю дублирование и упрощаю расширение новыми типами без влияния на логику в **Core.**
- Я проверяю входные данные при создании и обновлении и бросаю исключения при нарушении инвариантов — пустое или пробельное имя, отрицательные цена или количество, а также любые действия, которые приводят к отрицательному остатку.

Экран 6.



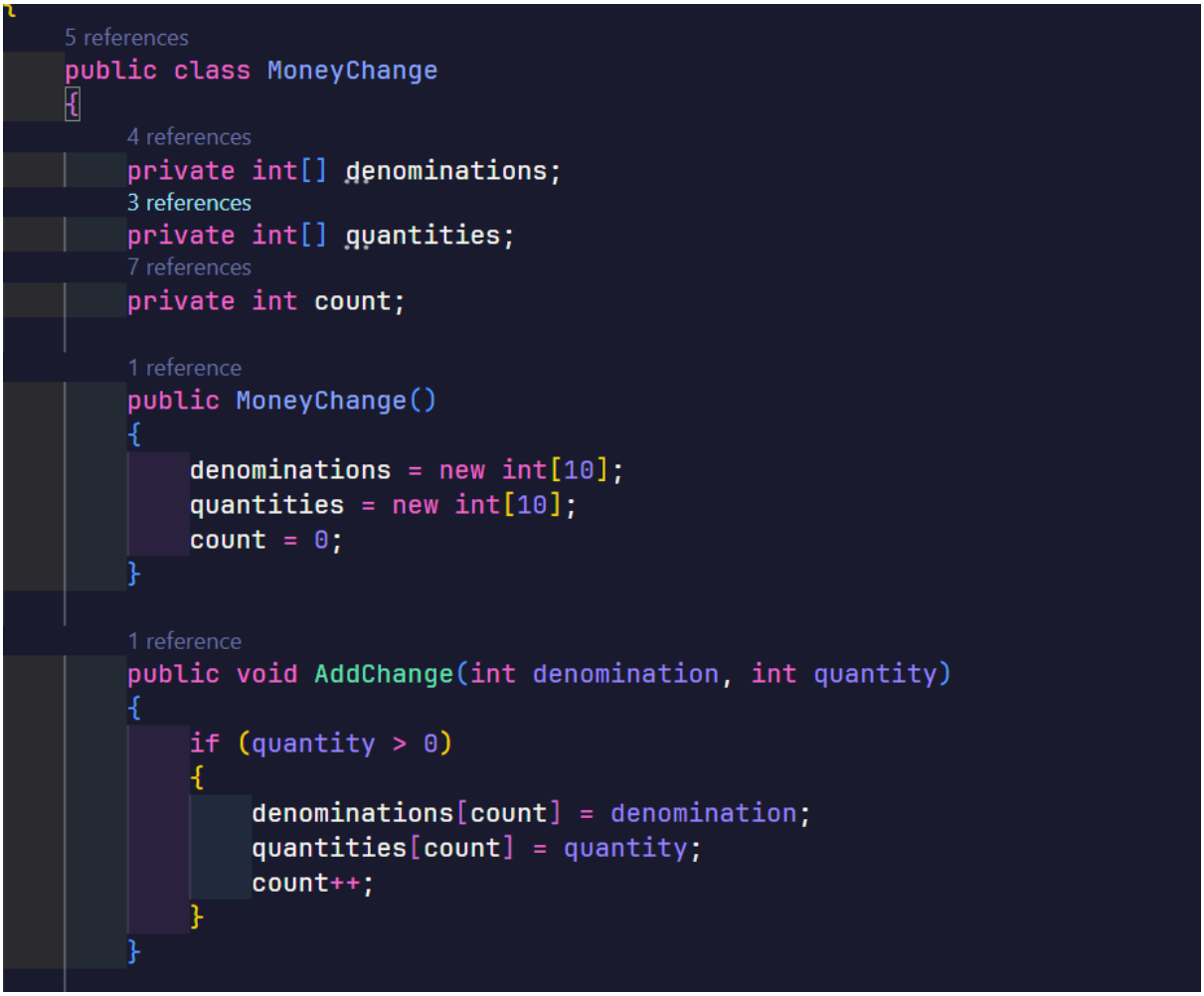
Экран 7.

```
using System;

namespace Lab1.Models
{
    6 references
    public class Food : Product
    {
        5 references
        public Food(int id, string name, int price, int quantity)
            : base(id, name, price, quantity)
        {
        }

        2 references
        public override string ToString()
        {
            string idStr = Id.ToString("D2");
            return idStr + ". " + Name + " (Закуска) - " + Price.ToString() + " руб. (Остаток: "
        }

    }
}

```

- Классы **Drink** и **Food** наследуются от Product, представляют два типа товаров — напитки и еду, повторно используют общий шаблон данных и проверки базового класса, сохраняют единообразный вывод; при необходимости могут переопределить метод отображения, добавив соответствующую метку (Напиток, Еда).

3. Models – MoneyChange.cs

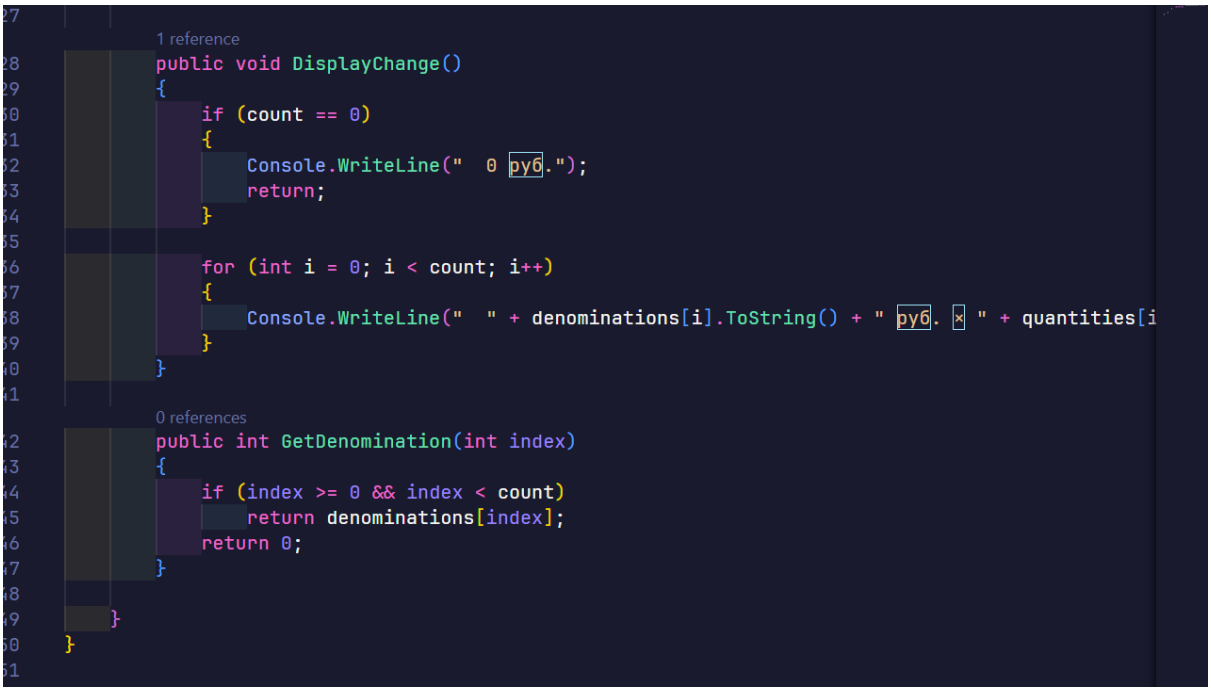
Экран 8.



```
5 references
public class MoneyChange
{
    4 references
    private int[] denominations;
    3 references
    private int[] quantities;
    7 references
    private int count;

    1 reference
    public MoneyChange()
    {
        denominations = new int[10];
        quantities = new int[10];
        count = 0;
    }

    1 reference
    public void AddChange(int denomination, int quantity)
    {
        if (quantity > 0)
        {
            denominations[count] = denomination;
            quantities[count] = quantity;
            count++;
        }
    }
```

- В **MoneyChange.cs** я создал класс, который объединяет номиналы купюр и соответствующие суммы сдачи.
- **Конструктор MoneyChange():** Я создаю пустую структуру для пар номинал–количество, обнуляю счётчик и использую массив фиксированного размера из 10 элементов.
- **Метод AddChange**(): Я добавляю новый элемент, когда quantity > 0, записываю номинал и количество в текущую позицию и увеличиваю счётчик. Проверку переполнения свыше 10 элементов я пока не выполняю.

Экран 9.



```
        1 reference
        public void DisplayChange()
        {
            if (count == 0)
            {
                Console.WriteLine("  0 руб.");
                return;
            }

            for (int i = 0; i < count; i++)
            {
                Console.WriteLine("  " + denominations[i].ToString() + " руб. × " + quantities[i
            }
        }

        0 references
        public int GetDenomination(int index)
        {
            if (index >= 0 && index < count)
                return denominations[index];
            return 0;
        }

    }
}
```

- **Метод DisplayChange** (): Я печатаю 0 руб. если данных нет. Когда элементы есть, я вывожу каждую строку от первого до последнего добавленного.
- **Метод GetDenomination():** Я возвращаю номинал по корректному индексу при условии “0 ≤ index < текущее_число_элементов”, в противном случае возвращаю 0.

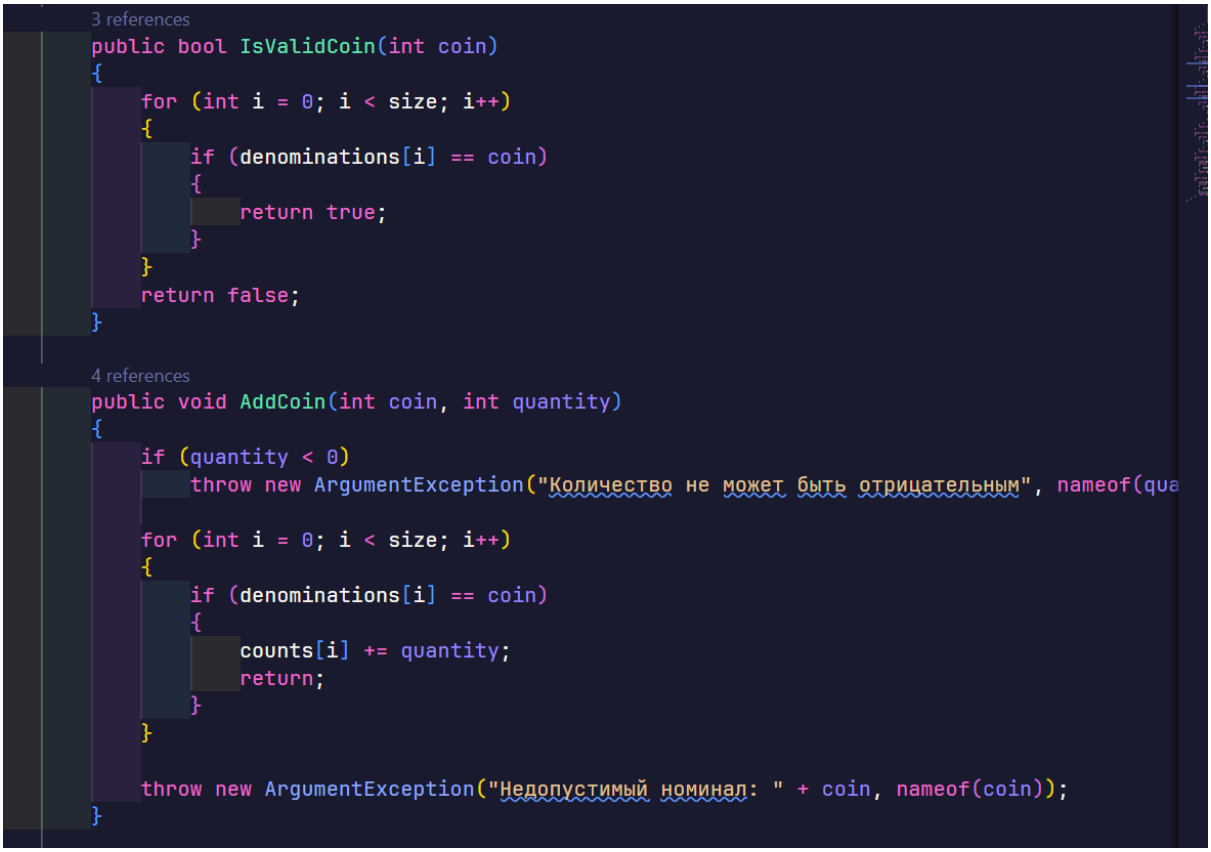
4. Models – CoinStorage.cs

Экран 10.

```
6 references
public class CoinStorage
{
    12 references
    private int[] denominations;
    22 references
    private int[] counts;
    14 references
    private int size;

    1 reference
    public CoinStorage()
    {
        denominations = new int[] { 1, 2, 5, 10, 20, 50, 100 };
        size = denominations.Length;
        counts = new int[size];

        for (int i = 0; i < size; i++)
        {
            int d = denominations[i];
            if (d == 1) counts[i] = 50;
            else if (d == 2) counts[i] = 40;
            else if (d == 5) counts[i] = 30;
            else if (d == 10) counts[i] = 25;
            else if (d == 20) counts[i] = 20;
            else if (d == 50) counts[i] = 15;
            else if (d == 100) counts[i] = 10;
            else counts[i] = 0;
        }
    }
```

- **CoinStorage** управляет запасом монет в торговом автомате по фиксированным номиналам. Здесь я инициализирую хранилище либо значениями по умолчанию.

Экран 11.



```
3 references
public bool IsValidCoin(int coin)
{
    for (int i = 0; i < size; i++)
    {
        if (denominations[i] == coin)
        {
            return true;
        }
    }
    return false;
}

4 references
public void AddCoin(int coin, int quantity)
{
    if (quantity < 0)
        throw new ArgumentException("Количество не может быть отрицательным", nameof(qua

    for (int i = 0; i < size; i++)
    {
        if (denominations[i] == coin)
        {
            counts[i] += quantity;
            return;
        }
    }

    throw new ArgumentException("Недопустимый номинал: " + coin, nameof(coin));
}
```

- Прежде всего, **IsValidCoin()** проверяет, входит ли номинал в допустимый список. Далее **AddCoin()** пополняет монеты и отклоняет отрицательные количества; одновременно **RemoveCoin()** выполняет выдачу при достаточном остатке и сообщает об успехе или неудаче.

Экран 12.

```
4 references
public int GetCoinCount(int coin)
{
    for (int i = 0; i < size; i++)
    {
        if (denominations[i] == coin)
        {
            return counts[i];
        }
    }
    return 0;
}

1 reference
public int[] GetAllDenominations()
{
    int[] result = new int[size];
    for (int i = 0; i < size; i++)
    {
        result[i] = denominations[i];
    }
    return result;
}

3 references
public int[] GetSortedDenoms()
{
    int[] sorted = new int[size];
    for (int i = 0; i < size; i++)
    {
        sorted[i] = denominations[i];
    }
    Array.Sort(sorted, (a, b) => b.CompareTo(a));
```

- Кроме того, **GetCoinCount()** позволяет быстро узнать текущий остаток по конкретному номиналу. Более того, **GetAllDenominations()** возвращает все номиналы в исходном порядке; наконец, **GetSortedDenoms()** отдаёт список номиналов в убывающем порядке для удобства расчёта сдачи.

5. Core – VendingMachine.cs

Экран 13.

```
namespace Lab1.Core
{
    // Main vending machine class
    4 references
    public class VendingMachine
    {
        14 references
        private List<Product> products;
        21 references
        private int insertedMoney;
        4 references
        private int collectedMoney;
        15 references
        private CoinStorage cashbox;

        1 reference
        public VendingMachine()
        {
            products = new List<Product>();
            insertedMoney = 0;
            collectedMoney = 0;
            cashbox = new CoinStorage();
            InitializeProducts();
        }

        1 reference
        private void InitializeProducts()
        {   // Drinks items
            products.Add(new Drink(1, "Кока-Кола", 120, 10));
            products.Add(new Drink(2, "Пепси", 120, 8));
```

- Здесь сосредоточены основные операции автомата: показ ассортимента, обработка оплаты, завершение покупки и выдача сдачи.
- Создаются коллекция товаров, счётчики внесённых и собранных денег, монетник “CoinStorage”; далее **InitializeProducts**() заполняет стартовый ассортимент.
- ShowProducts печатает список по “Id” и текущую сумму; **ProcessCustomerInteraction**() ведёт сессию: без денег — приглашение внести; с деньгами — выбор “Id”, проверка остатка и достаточности, при нехватке — подсказки, при готовности — покупка.

Экран 14.

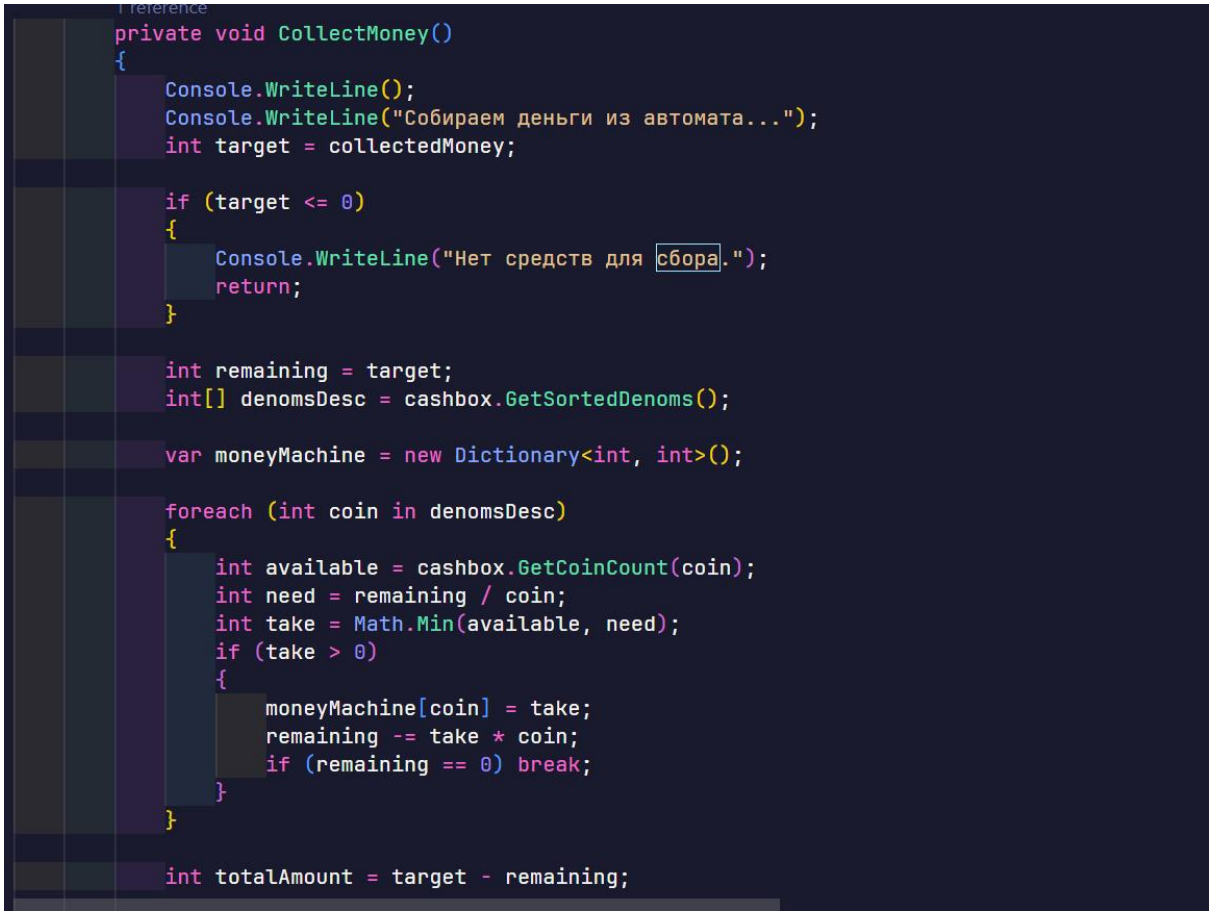
```
private bool CanMakeChange(int amount)
{
    // Create temporary cashbox to simulate change calculation
    CoinStorage tempCashbox = new CoinStorage(true);
    int[] denominations = cashbox.GetAllDenominations();

    for (int i = 0; i < denominations.Length; i++)
    {
        int count = cashbox.GetCoinCount(denominations[i]);
        tempCashbox.AddCoin(denominations[i], count);
    }

    int remaining = amount;
    int[] sortedDenoms = tempCashbox.GetSortedDenoms();

    for (int i = 0; i < sortedDenoms.Length; i++)
    {
        int coin = sortedDenoms[i];
        int coinsNeeded = remaining / coin;
        int coinsAvailable = tempCashbox.GetCoinCount(coin);
        int coinsToUse = Math.Min(coinsNeeded, coinsAvailable);

        if (coinsToUse > 0)
        {
            remaining -= coinsToUse * coin;
            tempCashbox.RemoveCoin(coin, coinsToUse);
        }

        if (remaining <= 0) break;
    }

    return remaining <= 0;
```

```
1 reference
private MoneyChange CalculateChange(int amount)
{
    MoneyChange result = new MoneyChange();
    int remaining = amount;
    int[] sortedDenoms = cashbox.GetSortedDenoms();

    for (int i = 0; i < sortedDenoms.Length; i++)
    {
        int coin = sortedDenoms[i];
        int coinsNeeded = remaining / coin;
        int coinsAvailable = cashbox.GetCoinCount(coin);
        int coinsToDispense = Math.Min(coinsNeeded, coinsAvailable);

        if (coinsToDispense > 0)
        {
            remaining -= coinsToDispense * coin;
            cashbox.RemoveCoin(coin, coinsToDispense);
            result.AddChange(coin, coinsToDispense);
        }

        if (remaining <= 0) break;
    }

    return result;
}
```

- После подтверждения списывается цена и уменьшается остаток, сумма уходит в «собрано»; если нужна сдача — **CanMakeChange**() проверяет возможность, **CalculateChange**() рассчитывает номиналы, DispenseChange выводит результат. Если сдачу выдать нельзя, выполняется **Refund**().

Экран 15.

```
1 reference
private void RestockProducts()
{
    Console.WriteLine();
    Console.WriteLine("ПОПОЛНЕНИЕ ТОВАРОВ");
    ShowProducts();

    Console.Write("Введите номер товара для пополнения (ID): ");
    string input = Console.ReadLine() ?? "0";
    int.TryParse(input, out int productId);

    if (productId < 1 || !ContainsId(productId))
    {
        Console.WriteLine("Неверный номер товара!");
        return;
    }

    Product product = GetById(productId);
    Console.WriteLine("Текущий остаток " + product.Name + ": " + product.Quantity + " шт
    Console.Write("Добавить количество: ");

    string quantityInput = Console.ReadLine() ?? "0";
    int.TryParse(quantityInput, out int quantity);

    if (quantity > 0)
    {
        product.Quantity += quantity;
        Console.WriteLine("Добавлено " + quantity + " шт. Новый остаток: " + product.Qua
    }
    else
    {
        Console.WriteLine("Неверное количество!");
```



```
1 reference
private void CollectMoney()
{
    Console.WriteLine();
    Console.WriteLine("Собираем деньги из автомата...");
    int target = collectedMoney;

    if (target <= 0)
    {
        Console.WriteLine("Нет средств для сбора.");
        return;
    }

    int remaining = target;
    int[] denomsDesc = cashbox.GetSortedDenoms();

    var moneyMachine = new Dictionary<int, int>();

    foreach (int coin in denomsDesc)
    {
        int available = cashbox.GetCoinCount(coin);
        int need = remaining / coin;
        int take = Math.Min(available, need);
        if (take > 0)
        {
            moneyMachine[coin] = take;
            remaining -= take * coin;
            if (remaining == 0) break;
        }
    }

    int totalAmount = target - remaining;
```

- **AdminMode()** запрашивает пароль и предоставляет минимум функций: **RestockProducts()** для пополнения по “Id”, **CollectMoney**() для инкассации доступными номиналами. Сообщения короткие и информативные. В **CollectMoney**() я изымаю собранные средства жадно по доступным номиналам из CoinStorage. Если изымать нечего — сразу сообщаю; иначе рассчитываю и списываю монеты, уменьшаю collectedMoney на фактически снятую сумму и печатаю итог.
- Дополнительно ставлю “Thread.Sleep”, чтобы смоделировать задержку подсчёта и сделать поведение ближе к реальному автомату.
- Проверка алгоритма с использованием написанных функций.

Экран 16.

```
public class VendingMachine_Tests
{
    // AddMoney - valid denomination
    [Fact]
    0 references
    public void AddMoney_ValidCoin_PrintsResultOk()
    {
        // Arrange
        var vm = new VendingMachine();
        var prevIn = Console.In;
        var prevOut = Console.Out;
        var s = new StringWriter();
        Console.SetIn(new StringReader("50\n"));
        Console.SetOut(s);

        // Act
        vm.AddMoney();

        // Assert
        Console.SetIn(prevIn);
        Console.SetOut(prevOut);
        var output = s.ToString();
        Assert.Contains("OK", output);
        Assert.Contains("Внесено:", output);
    }
```

```
    // AddMoney - invalid denomination
    [Fact]
    0 references
    public void AddMoney_InvalidCoin_ResultWarning()
    {
        // Arrange
        var vm = new VendingMachine();
        var prevIn = Console.In;
        var prevOut = Console.Out;
        var s = new StringWriter();
        Console.SetIn(new StringReader("3\n"));
        Console.SetOut(s);

        // Act
        vm.AddMoney();

        // Assert
        Console.SetIn(prevIn);
        Console.SetOut(prevOut);
        Assert.Contains("Недопустимый номинал", s.ToString());
    }
```

Вывод

Применение объектно-ориентированного программирования (ООП) в этой работе позволило мне решать ключевые задачи автомата: проверять данные, управлять каталогом и рассчитывать/выдавать сдачу. Я абстрагировал сущности в Product, CoinStorage, MoneyChange, VendingMachine; инкапсулировал правила и проверки внутри классов для согласованности данных; использовал наследование для устранения дублирования и полиморфизм для единой обработки и отображения товаров как Product.