АВТОНОМНАЯ НЕКОММЕРЧЕСКАЯ ОРГАНИЗАЦИЯ

«ПРОФЕССИОНАЛЬНАЯ ОБРАЗОВАТЕЛЬНАЯ ОРГАНИЗАЦИЯ»

«ОТКРЫТЫЙ ТАВРИЧЕСКИЙ КОЛЛЕДЖ»

ИНДИВИДУАЛЬНЫЙ ПРОЕКТ № 1

ПО междисциплинарному курсу

**МДК.01.04 Системное программирование**

ПО СПЕЦИАЛЬНОСТИ

09.02.07 Информационные системы и программирование

Квалификация: программист

Выполнили студенты   
28-В группы

Силкин А.А, Ибриш Р.И.

Руководитель:

преподаватель Яковенко Л.В.

Симферополь, 2025 г.

**Задача 1**

**1. Постановка задачи.**

Необходимо создать класс банковского счета с возможностью перевода средств. Перевод средств будет выполняться асинхронно, и после завершения операции перевода нужно будет проверить, был ли перевод успешным, и обновить баланс счета.

**2.Алгоритм решения.**

Шаг 1. Ввод пользователем с клавиатуры суммы средств, хранящейся на первом аккаунте.

Шагу 2. Ввод пользователем с клавиатуры суммы средств, хранящейся на втором аккаунте.

Шаг 3. Если баланс на первом аккаунте больше 0 и баланс на втором аккаунте больше 0, то вывод на экран «Баланс аккаунта 1», «Баланс аккаунта 2», переход на шаг 4. Иначе вывод на экран сообщения «Баланс аккаунта не может быть ниже 0», переход на шаг 11.

Шаг 4. Инициализация суммы перевода в размере 300 рублей.

Шаг 5. Вывод на экран сообщения «Начало перевода...».

Шаг 6. Если сумма перевода меньше чем, баланс первого аккаунта, то из суммы средств, хранящихся на первом аккаунте вычитается сумма перевода и добавляется на второй аккаунт, переход на шаг 7. Иначе вывод на экран «Недостаточно средств для перевода», переход к шагу 11.

Шаг 7. Вывод на экран «Перевод завершен успешно».

Шаг 8. Вывод на экран «Баланс аккаунта 1».

Шаг 9. Вывод на экран «Баланс аккаунта 2».

Шаг 10. Вывод на экран «Ожидаем завершения перевода», переход к шагу 11.

Шаг 11. Завершение программы.

**3. Листинг кода.**

using System;

using System.Diagnostics.Eventing.Reader;

using System.Threading.Tasks;

class BankAccount

{

public decimal Balance { get; private set; }

public BankAccount(decimal initialBalance)

{

Balance = initialBalance;

}

// Метод для перевода средств

public Task TransferAsync(decimal amount, BankAccount destination)

{

return Task.Run(() =>

{

// Имитируем процесс перевода с задержкой

Console.WriteLine("Начало перевода...");

Task.Delay(2000).Wait(); // Задержка 2 секунды (симуляция процесса)

if (amount <= Balance)

{

Balance -= amount;

destination.Balance += amount;

Console.WriteLine($"Перевод на сумму {amount} завершен успешно.");

}

else

{

Console.WriteLine("Недостаточно средств для перевода.");

}

});

}

}

class Program

{

static void Main()

{

Console.WriteLine("Введите сумму средств, храняющуюся на первом аккаунте");

decimal accsum1 = Convert.ToInt32(Console.ReadLine());

Console.WriteLine("Введите сумму средств, храняющуюся на втором аккаунте");

decimal accsum2 = Convert.ToInt32(Console.ReadLine());

BankAccount account1 = new BankAccount(accsum1);

BankAccount account2 = new BankAccount(accsum2);

if (account1.Balance > 0) && (account2.Balance > 0){

Console.WriteLine($"Баланс аккаунта 1: {account1.Balance}");

Console.WriteLine($"Баланс аккаунта 2: {account2.Balance}");

decimal amountToTransfer = 300;

// Асинхронный перевод с продолжением

account1.TransferAsync(amountToTransfer, account2)

.ContinueWith(task =>

{

// После завершения перевода

Console.WriteLine("\nПеревод завершен.");

Console.WriteLine($"Баланс аккаунта 1: {account1.Balance}");

Console.WriteLine($"Баланс аккаунта 2: {account2.Balance}");

});

Console.WriteLine("Ожидаем завершения перевода...");

Console.ReadLine(); // Ожидаем завершения работы программы

}

else

{

Console.WriteLine("Баланс аккаунта не может быть ниже 0");

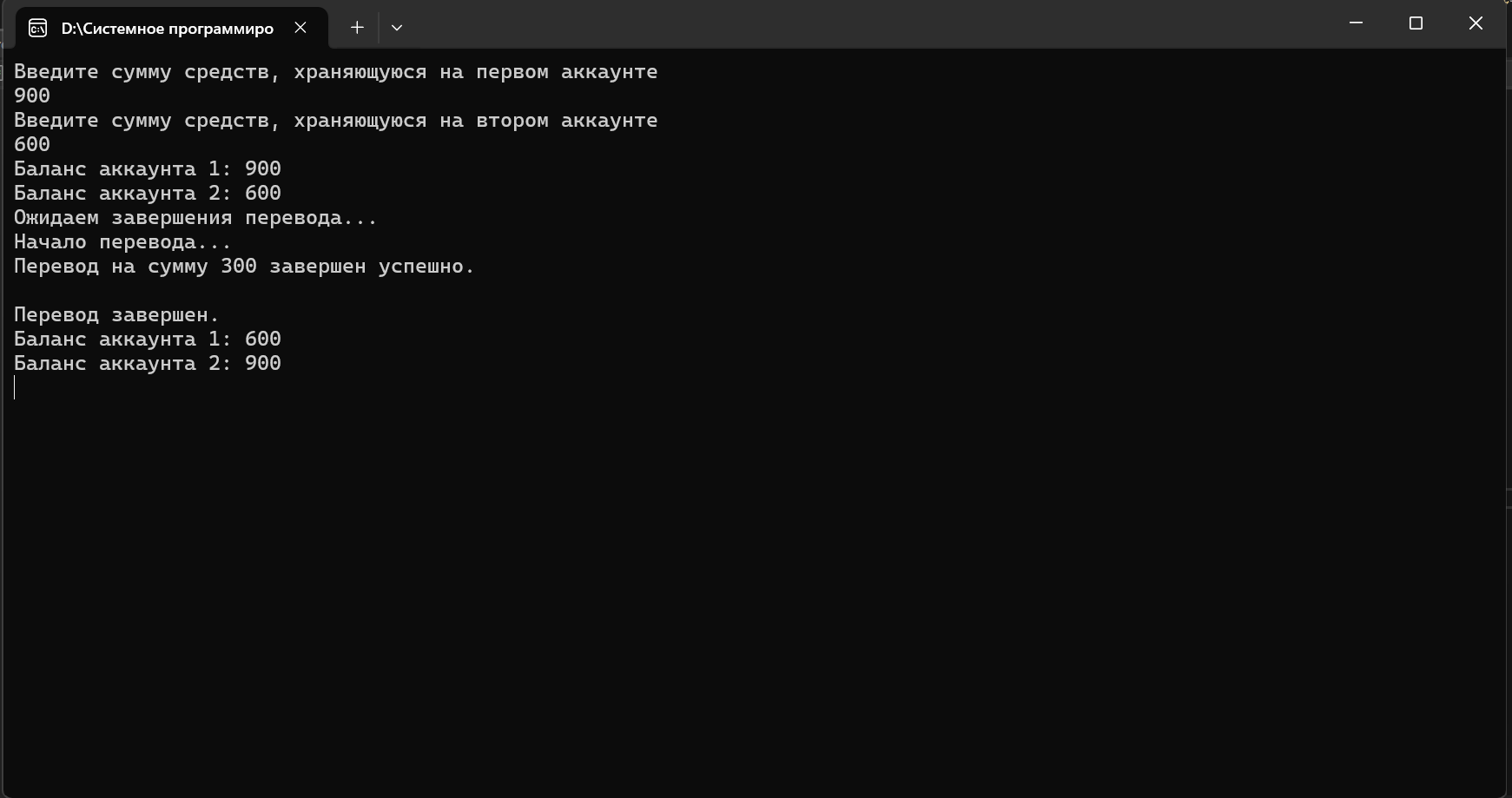
}

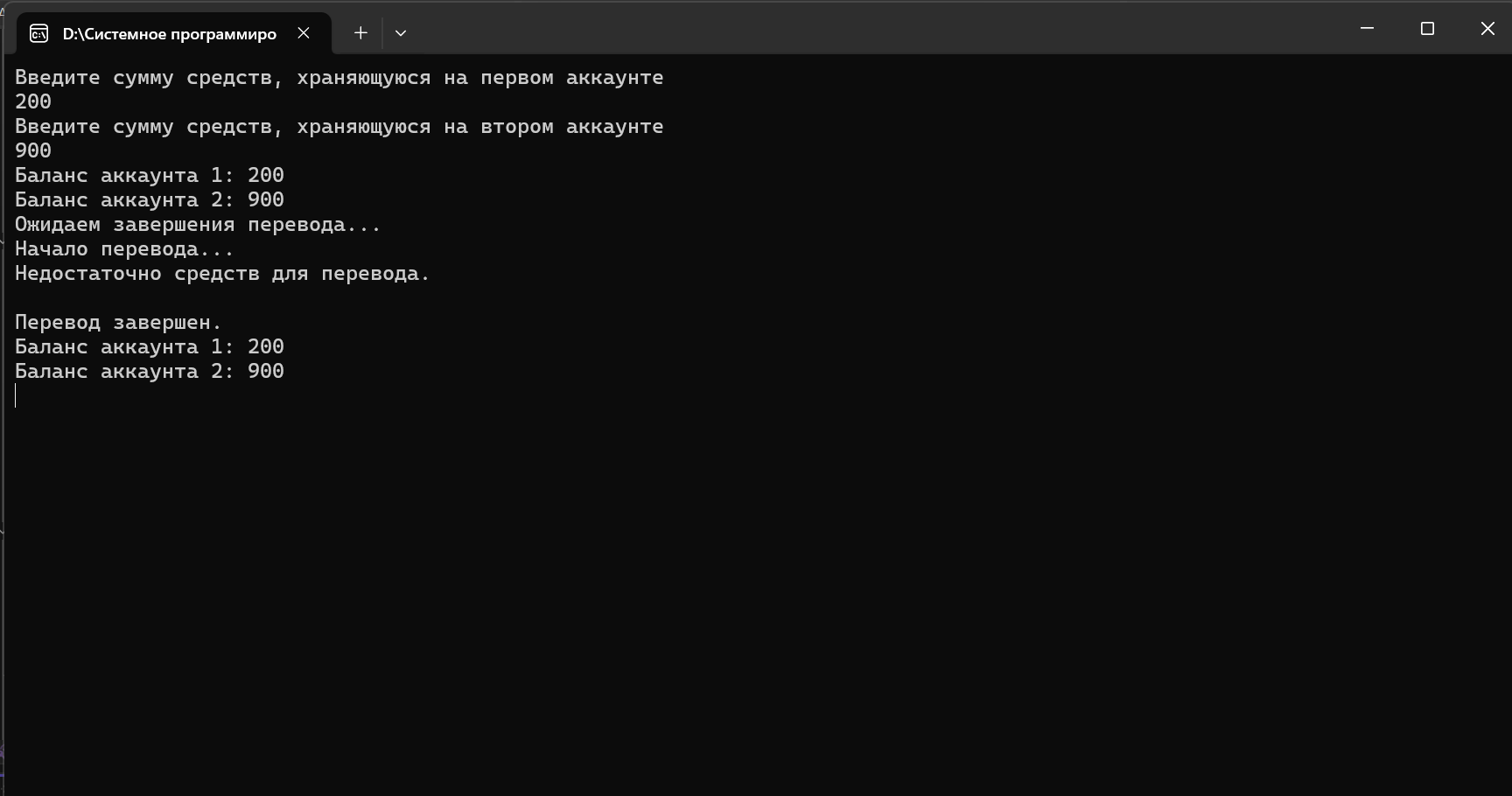
}

}

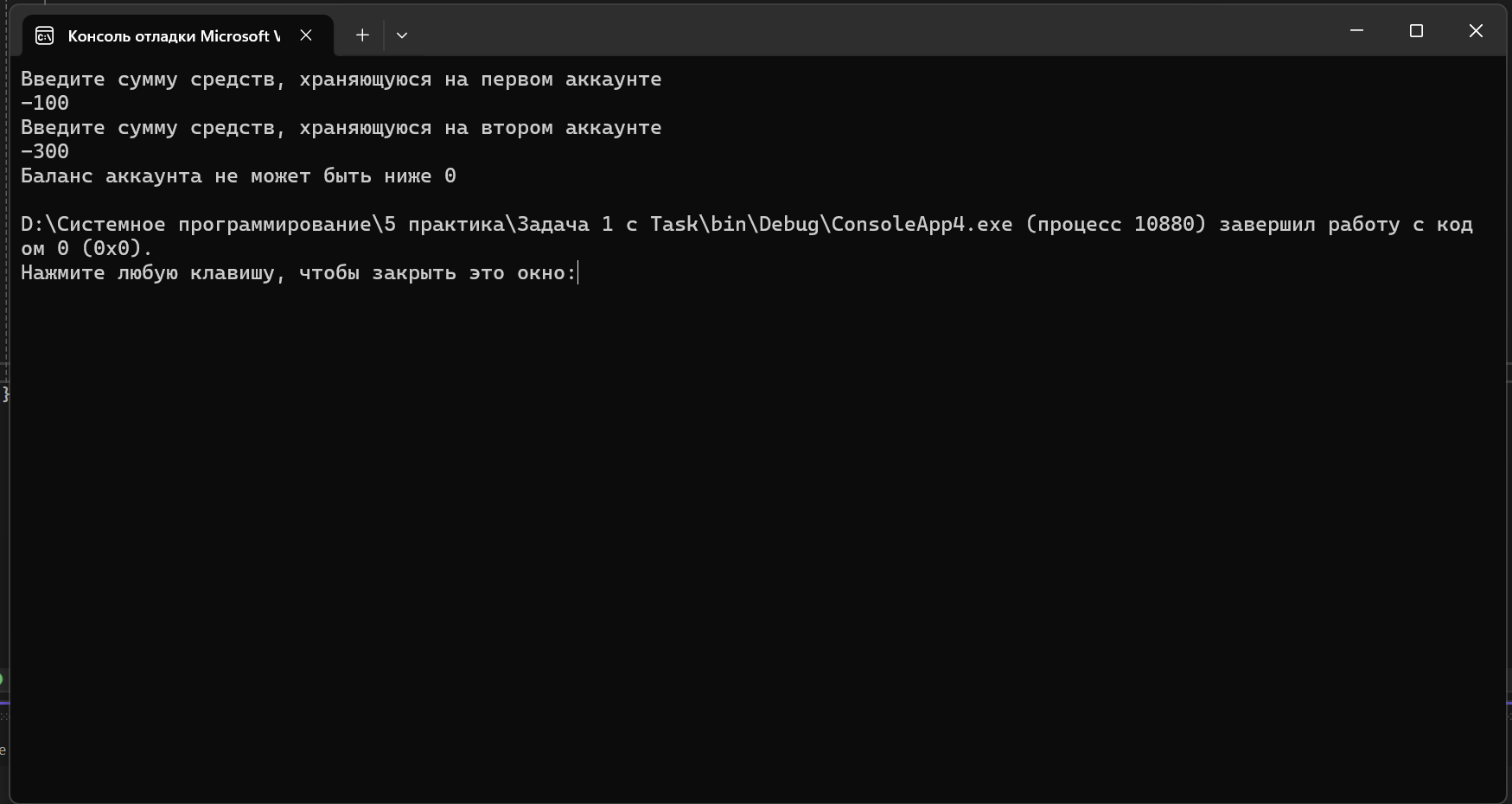
**4. Результаты тестирования работоспособности программы.**

При успешной операции перевода:

****

При недостатке средств на первом счёте:****

При балансе каждого из аккаунтов ниже 0:



**5. Вывод о работоспособности программы.**

Разработанная программа автоматизирует процесс учета денежных средств на двух аккаунтах пользователя и предназначена для применения в терминалах банка. Организует корректный перевод средств с одного счёта на другой. Асинхронность реализуется за счет предварительной проверки: пока транзакция перевода денежных средств не проведена, вывод сообщений пользователю и просмотр баланса не доступны.

По результатам ручного тестирования можно сделать вывод о работоспособности и функциональности программы.

**Задача 2.**

**1. Постановка задачи.**

В магазине реализована система, в которой несколько пользователей могут одновременно делать покупки. Каждый покупатель должен пройти через процесс оплаты, который включает несколько этапов:

1. Выбор товаров.
2. Оформление счета на оплату.
3. Проведение оплаты.

Необходимо организовать обработку оплаты для нескольких покупателей одновременно с использованием параллельных вычислений. Для этого будет использоваться класс Parallel для одновременной обработки нескольких операций оплаты.

**2. Описание алгоритма.**

Шаг 1. Создание списка покупателей, переход к шагу 2.

Шаг 2. В цикле организуется параллельное выполнение с учетом выполнения следующего условия: пока не обработаны все покупатели, то переход на шаг 3, иначе переход на шаг 9.

Шаг 3. Вывод на экран «Покупатель выбирает товары...», переход к шагу 4.

Шаг 4. Вывод на экран «Покупатель выбрал товары...», переход к шагу 5.

Шаг 5. Вывод на экран «Покупатель оформляет счёт...», переход к шагу 6.

Шаг 6. Вывод на экран «Покупатель оформил счёт...», переход к шагу 7.

Шаг 7. Вывод на экран «Покупатель обрабатывает оплату...», переход к шагу 8.

Шаг 8. Вывод на экран «Покупатель завершил оплату...», переход на шаг 2.

Шаг 9. Вывод на экран «Все операции завершены», переход на шаг 10.

Шаг 10. Завершение выполнения.

**3. Листинг кода.**

using System;

using System.Threading;

using System.Threading.Tasks;

class Program

{

static void Main(string[] args)

{

// Список покупателей

var shoppers = new Shopper[]

{

new Shopper("Покупатель 1"),

new Shopper("Покупатель 2"),

new Shopper("Покупатель 3"),

new Shopper("Покупатель 4")

};

// Параллельная обработка всех операций

Parallel.ForEach(shoppers, shopper =>

{

// Параллельно выполняем процесс оплаты

shopper.SelectProducts();

shopper.GenerateInvoice();

shopper.ProcessPayment();

});

Console.WriteLine("Все операции завершены.");

}

}

class Shopper

{

public string Name { get; }

private Random \_random = new Random();

public Shopper(string name)

{

Name = name;

}

public void SelectProducts()

{

Console.WriteLine($"{Name} выбирает товары...");

// Симуляция задержки

Thread.Sleep(\_random.Next(1000, 3000));

Console.WriteLine($"{Name} выбрал товары.");

}

public void GenerateInvoice()

{

Console.WriteLine($"{Name} оформляет счет...");

// Симуляция задержки

Thread.Sleep(\_random.Next(500, 1500));

Console.WriteLine($"{Name} оформил счет.");

}

public void ProcessPayment()

{

Console.WriteLine($"{Name} обрабатывает оплату...");

// Симуляция задержки

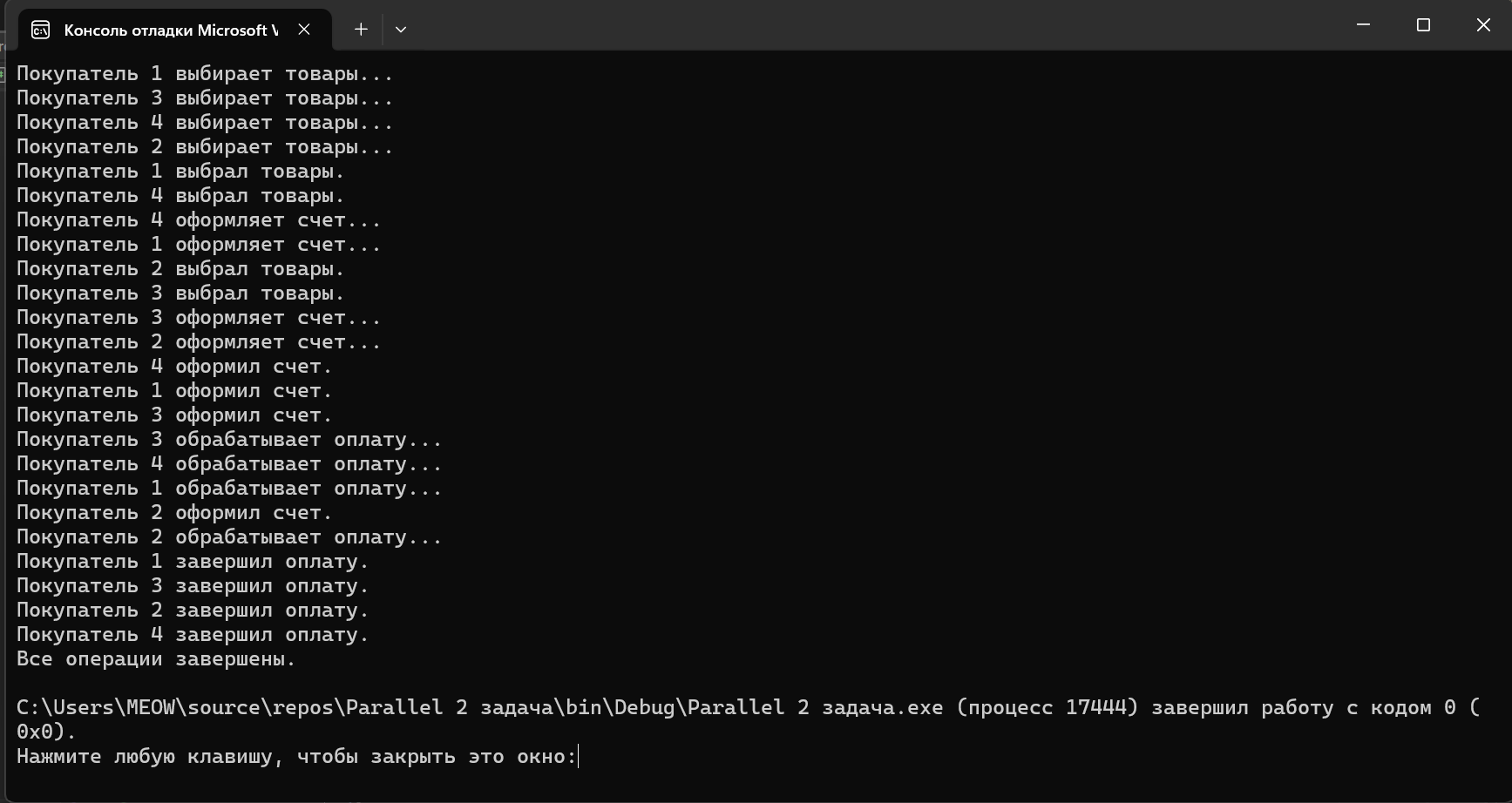
Thread.Sleep(\_random.Next(2000, 4000));

Console.WriteLine($"{Name} завершил оплату.");

}

}

**4. Результаты тестирования работоспособности программы.**

****

**5. Вывод о работоспособности приложения.**

Разработанная программа автоматизирует процесс обработки покупателей магазина и предназначена для использования в магазинах. Параллельность реализуется за счёт реализации механизма вывода информации об операциях каждого покупателя не друг за другом, а параллельно. Эмуляция режима выбора товара пользователем организуется за счёт задержки в пределам 1000-3000 мс.

По результатам ручного тестирования можно сделать вывод о работоспособности и функциональности программы.