НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

КАФЕДРА АВТОМАТИЗОВАНИХ СИСТЕМ ОБРОБКИ ІНФОРМАЦІЇ ТА УПРАВЛІННЯ

**КУРСОВА РОБОТА**

з                       Програмування -2. Структури даних та алгоритми

(назва дисципліни)

на тему:     Гаманець. Керування власним бюджетом та фінансами

Студента 1 курсу ІС-93 групи

напряму підготовки   126

спеціальності Інформаційні

системи та технології

Пінчука А.Ю.

(прізвище та ініціали)

Керівник ст. викладач\_\_Проскура С.Л.

(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Національна оцінка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кількість балів: \_\_\_\_\_

Оцінка ECTS:\_\_\_\_\_

|  |  |  |  |
| --- | --- | --- | --- |
| Члени комісії |  |  |  |
|  | (підпис) |  | (вчене звання, науковий ступінь, прізвище та ініціали) |

Київ - 2020 рік

Національний технічний університет України “КПІ”

(назва вищого навчального закладу)

Кафедра автоматизованих систем обробки інформації і управління

Дисципліна Програмування -2. Структури даних та алгоритми

Спеціальність 126 "Інформаційні системи та технології"

Курс 1 Група ІС-93 Семестр 2

**ЗАВДАННЯ**

на курсову роботу студента

Пінчука Артура Юрійовича

(прізвище, ім’я, по батькові)

1. Тема роботи Гаманець. Керування власним бюджетом та фінансами. 

2. Строк здачі студентом закінченої роботи 04 червня 2020

3. Вихідні дані до роботи Програма

4. Зміст розрахунково-пояснювальної записки (перелік питань, які підлягають розробці)

Вступ, аналіз предметної області, опис програмного забезпечення, тестування, висновки, перелік посилань, додаток







5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

Таблиці, скріншоти





6. Дата видачі завдання 01 квітня 2020 

**КАЛЕНДАРНИЙ ПЛАН**

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Назва етапів курсової роботи | Термін виконання етапів роботи | Підписи керівника, студента |
| 1. | Отримання теми курсової роботи | 01 квітня 2020 |  |
| 2. | Підготовка ТЗ | 02 квітня 2020 |  |
| 3. | Аналіз предметної області | 02 квітня 2020 |  |
| 4. | Проектування архітектури програмної системи | 14 квітня 2020 |  |
| 6. | Узгодження  сценарію роботи програми з керівником | 15 квітня 2020 |  |
| 5. | Розробка сценарію роботи програми | 20-22 квітня 2020 |  |
| 6. | Узгодження з керівником інтерфейсу користувача | 23 квітня 2020 |  |
| 7. | Розробка програмного забезпечення | 01-14 травня 2020 |  |
| 8. | Узгодження з керівником плану тестування | 15 травня 2020 |  |
| 9. | Тестування програми | 15 травня 2020 |  |
| 10. | Підготовка пояснювальної записки | 17 травня 2020 |  |
| 11. | Здача курсової роботи  на перевірку | 26 травня 2020 |  |
| 12. | Захист курсової роботи | 04 червня 2020 |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Студент

(підпис)

Керівник 

(підпис)                                                                                                            (прізвище, ім’я, по батькові)

"04" червня\_2020 р.

**ЗМІСТ**

[ВСТУП 5](https://docs.google.com/document/d/17rKX4LbkxtO-HLDNAyzbhsFlRqu5MXM0/edit#heading=h.30j0zll)

[1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ](https://docs.google.com/document/d/17rKX4LbkxtO-HLDNAyzbhsFlRqu5MXM0/edit#heading=h.1fob9te) 7

[1.1. Постановка задачі](https://docs.google.com/document/d/17rKX4LbkxtO-HLDNAyzbhsFlRqu5MXM0/edit#heading=h.3znysh7) 7

[1.2. Ціль та причини роботи](https://docs.google.com/document/d/17rKX4LbkxtO-HLDNAyzbhsFlRqu5MXM0/edit#heading=h.2et92p0) 7

[2. ОПИС ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ](https://docs.google.com/document/d/17rKX4LbkxtO-HLDNAyzbhsFlRqu5MXM0/edit#heading=h.tyjcwt) 8

[2.1. Загальний опис проекту.](https://docs.google.com/document/d/17rKX4LbkxtO-HLDNAyzbhsFlRqu5MXM0/edit#heading=h.3dy6vkm) 8

[2.2. Опис структури проекту.](https://docs.google.com/document/d/17rKX4LbkxtO-HLDNAyzbhsFlRqu5MXM0/edit#heading=h.17dp8vu) 8

[3. ТЕСТУВАННЯ 1](https://docs.google.com/document/d/17rKX4LbkxtO-HLDNAyzbhsFlRqu5MXM0/edit#heading=h.2m6kmyk)3

[3.1. План тестування 1](https://docs.google.com/document/d/17rKX4LbkxtO-HLDNAyzbhsFlRqu5MXM0/edit#heading=h.11bux6d)3

[3.1.1. Введення даних та команд для перевірки коректності роботі програми. 1](https://docs.google.com/document/d/17rKX4LbkxtO-HLDNAyzbhsFlRqu5MXM0/edit#heading=h.3lbifu6)3

[3.1.2. Введення некоректних даних для отримування очікуваних результатів.](https://docs.google.com/document/d/17rKX4LbkxtO-HLDNAyzbhsFlRqu5MXM0/edit#heading=h.20gsq1z) 13

[3.2. Проведення тестування 1](https://docs.google.com/document/d/17rKX4LbkxtO-HLDNAyzbhsFlRqu5MXM0/edit#heading=h.4kgg8ps)3

[3.2.1. Введення даних та команд для перевірки коректності роботи програми. 15](https://docs.google.com/document/d/17rKX4LbkxtO-HLDNAyzbhsFlRqu5MXM0/edit#heading=h.2zlqixl)

[3.2.2. Введення некоректних даних для отримування очікуваних результатів. 19](https://docs.google.com/document/d/17rKX4LbkxtO-HLDNAyzbhsFlRqu5MXM0/edit#heading=h.1er0t5e)

[ВИСНОВКИ 2](https://docs.google.com/document/d/17rKX4LbkxtO-HLDNAyzbhsFlRqu5MXM0/edit#heading=h.3yqobt7)2

[ПЕРЕЛІК ПОСИЛАНЬ 2](https://docs.google.com/document/d/17rKX4LbkxtO-HLDNAyzbhsFlRqu5MXM0/edit#heading=h.3d0wewm)3

**ВСТУП**

Об’єктно орієнтоване програмування, або ООП – одна з парадигм програмування, що дозволяє розглядати програму як множину так званих об’єктів, що взаємодіють між собою. В основі об’єктно - орієнтованого програмування лежать два основних поняття: клас та об’єкт.

Об’єкт є базовим поняттям ООП. Це конкретна реалізація, екземпляр класу, що складається з трьох частин: стан (змінна стану), методи (операції, функції) та ім’я об’єкта.

Клас є групою даних та методів чи функцій для роботи з цими даними. Клас утворюють об’єкти з однаковими наборами змінних стану та методів. Класи можуть бути абстрактними.

Основними властивостями, або принципами об’єктно – орієнтованого програмування вважають:

* Інкапсуляцію. Здатність об’єктів скривати деякі способи обробки даних, тобто методи, та самі дані від навколишнього цифрового середовища. Інкапсуляція потрібна для того, аби запобігти використанню користувачами інтерфейсу тих частин реалізації, які, швидше за все, будуть змінюватись.
* Поліморфізм. Здатність об’єкта використовувати методи похідного класу. Тобто такого, який не існує на момент створення базового класу. Поліморфізм означає залежність поведінки від класу, в якому ця поведінка викликається, тобто, два або більше класів можуть реагувати по-різному на однакові повідомлення.
* Успадкування. Здатність дозволяти добувати нові визначення класів на основі вже існуючих класів. Клас може мати «підкласи», спеціалізовані, розширені версії надкласу. Можуть навіть утворюватись цілі дерева успадкування.
* Абстракція. Надання об’єкту таких характеристик, що будуть відрізняти його від інших об’єктів, чітко визначаючи його концептуальні кордони.

1. **АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ**

1.1. Постановка задачі

Власний бюджет складається з декількох рахунків, які поповнюються за заданими статтями прибутку. Гроші цих рахунків можуть бути переведені з одного на інший, можуть витрачатись за заданими статтями витрат. Підсумовуючи витрати та прибутки, можливо отримати інформацію, скільки було витрачено/отримано загалом/за певною статтею по заданому рахунку. Користувач може створювати рахунки, користуватися всіма функціями, які було перелічено вище.

1.2. Ціль та задача роботи

Метою курсової роботи є опрацювання отриманих протягом семестру навичок з основ об’єктно-орієнтованого програмування. Ціллю є досягнення гнучкості коду, опрацювання отриманих навиків та розроблення програми для обліку товарів на складі, яка б в відповідала постановці задачі та видавала очікуваний результат.

1. **ОПИС ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ**

## 2.1. Загальний опис проекту.

Проект містить у собі 3 класи: Account (рахунок/обліковий запис), Menu (меню), Transfers (перекази).

Клас Account – базовий клас для створення рахунків. Містить конструктор та реалізацію методів, потрібних для використання усіх функцій програми.

Клас Menu – клас для обробки головного меню програми. Містить реалізацію текстового інтерфейсу та виклик методів, створених у класі Account.

Клас Transfers - клас, похідний від класу Menu. Створений для обробки переказів з одного рахунку на інший, або поповнення рахунку / витрат коштів.

## 2.2. Опис структури проекту.

Клас Account є базовим класом, що реалізує ініціалізацію рахунків. Клас містить приватні поля:

* account\_name типу string – унікальну назву кожного рахунку;
* account\_value типу decimal. – унікальний баланс кожного рахунку.

Клас містить публічні методи:

Таблиця 2.1 – Клас Account

|  |  |  |  |
| --- | --- | --- | --- |
| № | Назва методу | Призначення методу | Опис вхідних параметрів |
| 1 | Account | Конструктор класу Account з двома параметрами. | name value |
| 2 | Output | Повертає назву рахунку та його поточний баланс. | - |
| 3 | Type\_output | Повертає звіт по витратам для заданого рахунку. | - |
| 4 | Оператор + | Перевантаження оператору задля реалізації отримання / витрат коштів | obj  value |
| 5 | Adder | Головний метод для поповнення рахунку. | value |
| 6 | Transfer | Переказує кошти з одного рахунку на інший | obj1  obj2  value |
| 7 | Replenish | Функція для поповнення рахунку за заданими статтями | obj1  value  type |
| 8 | Spend | Функція для витрат коштів з рахунку за заданими статтями | obj1  value  type |

Клас Menu є класом, що реалізує головне меню.

Клас містить приватні поля:

* counter типу int – базовий лічильник для реалізації списку об’єктів;
* accounts типу array – список створених об’єктів класу Account;

Клас містить публічні методи:

Таблиця 2.2 – Клас Menu

|  |  |  |  |
| --- | --- | --- | --- |
| № | Назва методу | Призначення методу | Опис вхідних параметрів |
| 1 | MainMenu | Викликає в консоль головне меню.  Використовується для виклику інших методів. | - |
| 2 | NewAcc | Викликає меню для створення нових рахунків. Створює нові рахунки. | - |
| 3 | Accounts | Повертає список всіх створених рахунків та їхні баланси. | - |
| 4 | TransferMenu | Викликає меню для здійснення переказів, поповнення рахунку чи витрат коштів. | - |
| 5 | Summary | Повертає звіт витрат / надходжень для заданого рахунку. | - |

Клас Transfers є похідним класом класу Menu. Він реалізує методи, потрібні для здійснення переказів, поповнення рахунку чи витрат коштів.

Клас містить приватні поля:

* Transfer типу delegate – делегат, потрібний для реалізації функцій витрат коштів та поповнення рахунку;
* to типу Transfer - витрати коштів;
* back типу Transfer - поповнення рахунку;

Клас містить публічні методи:

Таблиця 2.3 – Клас Transfer

|  |  |  |  |
| --- | --- | --- | --- |
| № | Назва методу | Призначення методу | Опис вхідних параметрів |
| 1 | SpendOrReplenish | Викликає меню з вибором потрібного типу переказу (витрата чи поповнення). Виконує бажану дію. | - |
| 2 | OneToAnother | Викликає меню, що відповідає за переказ коштів з одного рахунку на інший. | - |

1. **ТЕСТУВАННЯ**

3.1. План тестування

3.1.1. Введення даних та команд для перевірки коректності роботі програми.

Користувач вводить дані у відповідному для коректного користування порядку. Усі введені дані відповідають заданим стандартам та командам. Користувач має гарантовано отримати очікуваний результат, повідомлення про успішне виконання тих чи інших операцій.

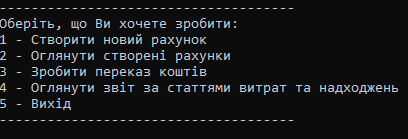
3.1.2. Введення некоректних даних для отримування очікуваних результатів.

Користувач вводить дані або у невідповідному для коректного виконання порядку, або намагається отримати неіснуючі товари чи замовлення. Користувач має отримати відповідні повідомлення про помилки у роботі із товарами чи замовленнями.

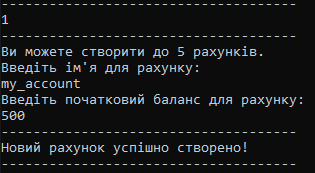
3.2. Проведення тестування

3.2.1. Введення даних та команд для перевірки коректності роботи програми.

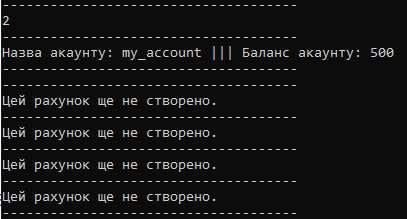
Користувач має можливість ознайомитись з головним меню програми та переліком її функцій (Рис. 3.1.).

Рис. 3.1.

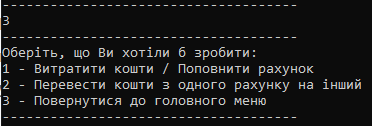
Після ознайомлення, користувач має можливість створити новий рахунок. Усього рахунків може бути створено 5. Для створення необхідно обрати функцію №1, ввести бажане ім’я для рахунку та його початковий баланс. Після створення рахунку користувач отримує повідомлення про успішне створення рахунку (Рис. 3.2.).

Рис. 3.2.

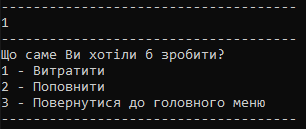
Користувач має змогу перевірити інформацію про створені рахунки, а саме їх назву та поточний баланс. Для цього потрібно обрати пункт меню №2. Якщо хоча б один з п’яти рахунків ще не було створено, користувач отримає повідомлення про це. (Рис. 3.3.).

Рис. 3.3.

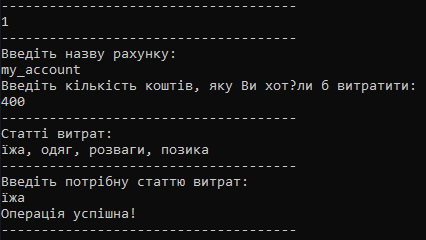
Також можна проводити різні операції з коштами. Для цього потрібно обрати пункт меню №3. (Рис. 3.4.).

Рис. 3.4.

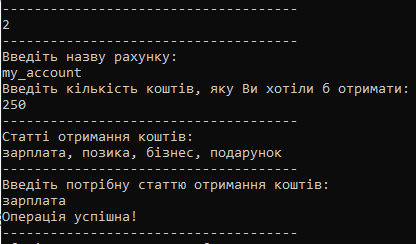
Для витрат/поповнення рахунку необхідно обрати пункт підменю переказів №1. (Рис. 3.5.), а далі обрати бажану дію: витрата за пунктом №1 чи поповнення за пунктом №2.

Рис. 3.5.

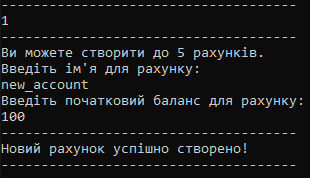
Для витрачання коштів потрібно ввести назву рахунку, з якого буде знято кошти, кількість грошей для витрат та бажану статтю витрат з заданих чотирьох. Після виконання переказу програма повідомить користувача про успішність операції (Рис. 3.6.).

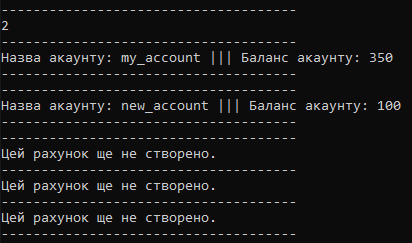
Рис. 3.6.

Для поповнення рахунку необхідно ввести назву рахунку, баланс якого користувач бажає поповнити, кількість коштів для поповнення та бажану статтю прибутку з заданих чотирьох. Після виконання переказу програма повідомить користувача про успішність операції (Рис. 3.7.).

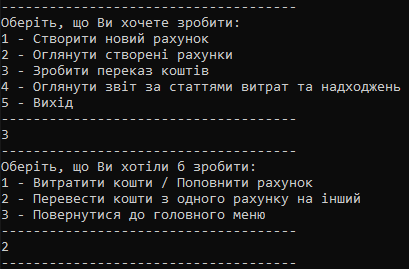
Рис. 3.7.

Для тестування функції переказу коштів з одного рахунку на інший, створимо ще один рахунок (Рис. 3.8.) та переглянемо поточний баланс створених рахунків (Рис. 3.9.)

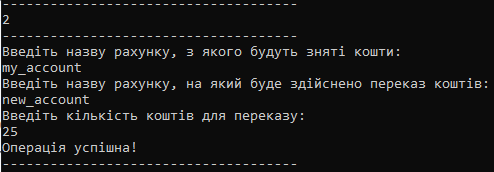
Рис. 3.8.

Рис. 3.9.

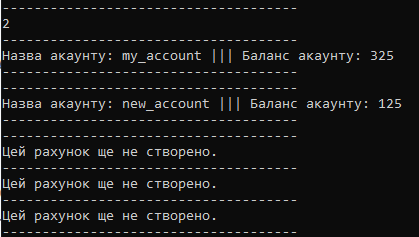
Щоб виконати переказ коштів з одного рахунку на інший необхідно обрати пункт меню №3, а у підменю обрати пункт №2 (Рис. 3.10.).

Рис. 3.10.

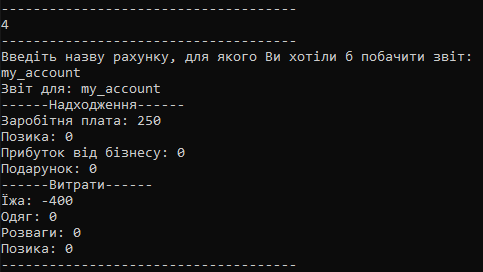
Далі користувачу необхідно ввести назву рахунку, з якого знімаються кошти. Потім назву рахунку, на який кошти нараховуються, а також бажану суму переказу. Після виконання операції, користувач отримує повідомлення про її успішність (Рис. 3.11.)

Рис. 3.11.

Перевіримо виконання операції звіривши поточні рахунки за рахунками на Рис. 3.9. (Рис. 3.12.). Операція працює коректно.

Рис. 3.12.

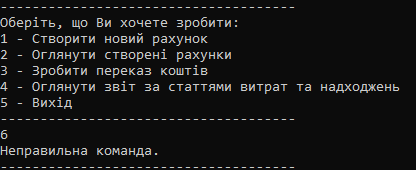
Останньою функцією, доступною користувачеві, є перегляд звіт прибутків/витрат за всіма заданими статтями. Для рахунку my\_account для прикладу було витрачено 400 одиниць коштів на їжу та отримано 250 одиниць коштів як зарплату. Щоб переглянути звіт необхідно обрати пункт меню №4 та ввести назву рахунку, для якого потрібно вивести звіт витрат/прибутків (Рис. 3.13.).

Рис. 3.13.

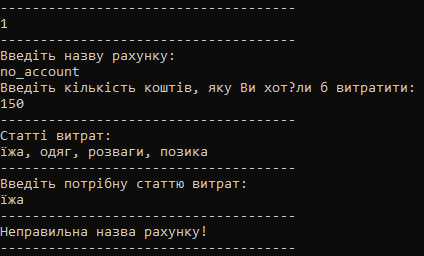
Інформація на Рис. 3.13. є достовірною, отже програма працює коректно.

3.2.2. Введення некоректних для отримання очікуваних результатів.

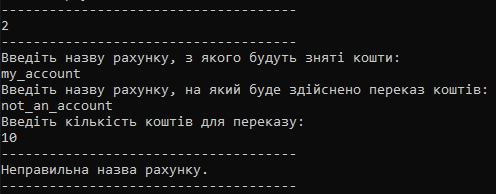
У разі введення неіснуючого пункту меню, користувач отримує повідомлення про те, що введена команда є неправильною (Рис. 3.14.)

Рис. 3. 14.

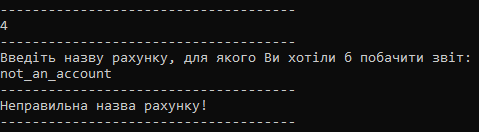
У разі спроби провести операцію з неіснуючим рахунком, користувач отримує повідомлення про некоректність введення назви рахунку. (Рис. 3.15.)

Рис. 3. 15.

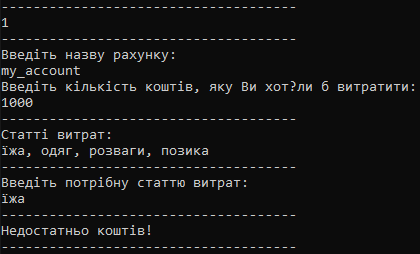
Таке ж повідомлення користувач отримує при спробі переказати кошти з існуючого рахунку на неіснуючий та навпаки (Рис. 3.16.).

Рис. 3.16.

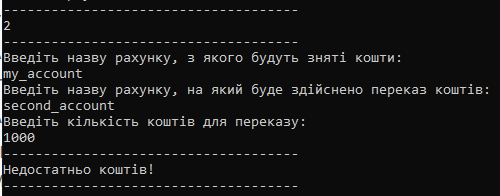
Це ж повідомлення побачить користувач, якщо спробує переглянути звіт за рахунком, якого не існує (Рис. 3.17.).

Рис. 3.17.

При спробі витратити більше коштів, ніж наразі є на рахунку, користувач отримує повідомлення про те, що на рахунку недостатньо коштів (Рис. 3.18.).

Рис. 3.18.

Таке ж повідомлення користувач отримує при спробі переказати кошти з одного рахунка на інший при недостатній кількості коштів на рахунку, з якого вони будуть зняті (Рис. 3.19.).



**ВИСНОВКИ**

Під час виконання курсової роботи було розроблено програму, що реалізує роботу гаманцем та веденням власного бюджету. Було опрацьовано базові навички об’єктно орієнтованого програмування, отримані протягом семестру. Розроблена програма працює коректно та видає очікуваний на початку розробки результат, що було показано за допомогою тестування.

Таким чином було досягнуто мети курсової роботи та розроблено робочу програму ведення власного бюджету

**ПЕРЕЛІК ПОСИЛАНЬ**

1. OOP in C#

https://metanit.com/sharp/

1. OOP basics

https://www.cl.cam.ac.uk/teaching/0910/OOProg/OOP.pdf

**ДОДАТОК А**







Program.cs

using System;

namespace Wallet

{

class Program

{

static void Main(string[] args)

{

Menu.MainMenu();

}

}

}

Wallet.dll

using System;

using System.Collections.Generic;

using System.Text;

namespace Wallet

{

public class Account

{

internal string account\_name;

decimal account\_value;

const string replenish1 = "зарплата";

decimal r1 = 0;

const string replenish2 = "позика";

decimal r2 = 0;

const string replenish3 = "бiзнес";

decimal r3 = 0;

const string replenish4 = "подарунок";

decimal r4 = 0;

const string spend1 = "їжа";

decimal s1 = 0;

const string spend2 = "одяг";

decimal s2 = 0;

const string spend3 = "розваги";

decimal s3 = 0;

const string spend4 = "позика";

decimal s4 = 0;

public Account(string name, decimal value)

{

this.account\_name = name;

this.account\_value = value;

}

public void Output()

{

Console.WriteLine("Назва акаунту: " + account\_name + " ||| Баланс акаунту: " + account\_value);

return;

}

public void Type\_output()

{

Console.WriteLine("Звiт для: " + account\_name);

Console.WriteLine("------Надходження------");

Console.WriteLine("Заробiтня плата: " + r1);

Console.WriteLine("Позика: " + r2);

Console.WriteLine("Прибуток вiд бiзнесу: " + r3);

Console.WriteLine("Подарунок: " + r4);

Console.WriteLine("------Витрати------");

Console.WriteLine("Їжа: " + s1);

Console.WriteLine("Одяг: " + s2);

Console.WriteLine("Розваги: " + s3);

Console.WriteLine("Позика: " + s4);

}

public static decimal operator +(Account obj, decimal value)

{

return obj.account\_value + value;

}

public void Adder(int value)

{

account\_value += value;

return;

}

static public void Transfer(Account obj1, Account obj2, decimal value)

{

if (value > obj1.account\_value)

{

Console.WriteLine("-------------------------------------");

Console.WriteLine("Недостатньо коштiв!");

Menu.TransferMenu();

}

else

{

obj2.account\_value += value;

obj1.account\_value -= value;

}

return;

}

static public void Replenish(Account obj1, decimal value, string type)

{

switch (type.ToLower())

{

case replenish1:

obj1.r1 += value;

obj1.account\_value += value;

break;

case replenish2:

obj1.r2 += value;

obj1.account\_value += value;

break;

case replenish3:

obj1.r3 += value;

obj1.account\_value += value;

break;

case replenish4:

obj1.r4 += value;

obj1.account\_value += value;

break;

default:

Console.WriteLine("Неправильна стаття надходження коштів.");

break;

}

return;

}

static public void Spend(Account obj1, decimal value, string type)

{

if (value > obj1.account\_value)

{

Console.WriteLine("-------------------------------------");

Console.WriteLine("Недостатньо коштiв!");

Transfers.SpendOrReplenish();

}

else

{

switch (type.ToLower())

{

case spend1:

obj1.s1 -= value;

obj1.account\_value -= value;

break;

case spend2:

obj1.s2 -= value;

obj1.account\_value -= value;

break;

case spend3:

obj1.s3 -= value;

obj1.account\_value -= value;

break;

case spend4:

obj1.s4 -= value;

obj1.account\_value -= value;

break;

default:

Console.WriteLine("Неправильна стаття витрат!");

break;

}

}

return;

}

}

public class Menu

{

internal static Account[] accounts = new Account[5];

static int counter = 0;

static public void MainMenu()

{

Console.WriteLine("-------------------------------------");

Console.WriteLine("Оберiть, що Ви хочете зробити: ");

Console.WriteLine("1 - Створити новий рахунок");

Console.WriteLine("2 - Оглянути створенi рахунки");

Console.WriteLine("3 - Зробити переказ коштiв");

Console.WriteLine("4 - Оглянути звiт за статтями витрат та надходжень");

Console.WriteLine("5 - Вихiд");

Console.WriteLine("-------------------------------------");

string input;

input = Console.ReadLine();

switch (input)

{

case "1":

NewAcc();

break;

case "2":

Accounts();

break;

case "3":

TransferMenu();

break;

case "4":

Summary();

break;

case "5":

return;

default:

Console.WriteLine("Неправильна команда.");

MainMenu();

break;

}

}

static public void NewAcc()

{

Console.WriteLine("-------------------------------------");

Console.WriteLine("Ви можете створити до 5 рахункiв.");

Console.WriteLine("Введiть iм'я для рахунку: ");

string name = Console.ReadLine();

Console.WriteLine("Введiть початковий баланс для рахунку: ");

string value = Console.ReadLine();

decimal dec = Convert.ToDecimal(value);

Account acc = new Account(name, dec);

accounts[counter] = acc;

counter += 1;

Console.WriteLine("-------------------------------------");

Console.WriteLine("Новий рахунок успiшно створено!");

Console.WriteLine("-------------------------------------");

MainMenu();

}

static public void Accounts()

{

for (int i = 0; i < accounts.Length; i++)

{

try

{

Console.WriteLine("-------------------------------------");

accounts[i].Output();

Console.WriteLine("-------------------------------------");

}

catch

{

Console.WriteLine("Цей рахунок ще не створено.");

}

}

MainMenu();

}

static public void TransferMenu()

{

Console.WriteLine("-------------------------------------");

Console.WriteLine("Оберiть, що Ви хотiли б зробити: ");

Console.WriteLine("1 - Витратити кошти / Поповнити рахунок");

Console.WriteLine("2 - Перевести кошти з одного рахунку на iнший");

Console.WriteLine("3 - Повернутися до головного меню");

Console.WriteLine("-------------------------------------");

string input;

input = Console.ReadLine();

switch (input)

{

case "1":

Transfers.SpendOrReplenish();

break;

case "2":

Transfers.OneToAnother();

break;

case "3":

MainMenu();

break;

}

}

static public void Summary()

{

Console.WriteLine("-------------------------------------");

Console.WriteLine("Введiть назву рахунку, для якого Ви хотiли б побачити звiт: ");

string name;

name = Console.ReadLine();

for (int i = 0; i < accounts.Length; i++)

{

try

{

if (name == accounts[i].account\_name)

{

accounts[i].Type\_output();

}

}

catch

{

Console.WriteLine("-------------------------------------");

Console.WriteLine("Неправильна назва рахунку!");

}

finally

{

MainMenu();

}

}

}

}

public class Transfers : Menu

{

delegate void Transfer(Account obj1, decimal value, string type);

static Transfer to = Account.Spend;

static Transfer back = Account.Replenish;

public static void SpendOrReplenish()

{

Console.WriteLine("-------------------------------------");

Console.WriteLine("Що саме Ви хотiли б зробити?");

Console.WriteLine("1 - Витратити");

Console.WriteLine("2 - Поповнити");

Console.WriteLine("3 - Повернутися до головного меню");

Console.WriteLine("-------------------------------------");

string choice;

choice = Console.ReadLine();

switch (choice)

{

case "1":

Console.WriteLine("-------------------------------------");

Console.WriteLine("Введiть назву рахунку: ");

string name;

name = Console.ReadLine();

Console.WriteLine("Введiть кiлькiсть коштiв, яку Ви хотіли б витратити: ");

decimal value;

value = Convert.ToDecimal(Console.ReadLine());

Console.WriteLine("-------------------------------------");

Console.WriteLine("Статтi витрат: ");

Console.WriteLine("їжа, одяг, розваги, позика");

Console.WriteLine("-------------------------------------");

Console.WriteLine("Введiть потрiбну статтю витрат: ");

string type;

type = Console.ReadLine();

try

{

if (name == accounts[0].account\_name)

{

to(accounts[0], value, type);

Console.WriteLine("Операцiя успiшна!");

MainMenu();

}

else if (name == accounts[1].account\_name)

{

to(accounts[1], value, type);

Console.WriteLine("Операцiя успiшна!");

MainMenu();

}

else if (name == accounts[2].account\_name)

{

to(accounts[2], value, type);

Console.WriteLine("Операцiя успiшна!");

MainMenu();

}

else if (name == accounts[3].account\_name)

{

to(accounts[3], value, type);

Console.WriteLine("Операцiя успiшна!");

MainMenu();

}

else if (name == accounts[4].account\_name)

{

to(accounts[4], value, type);

Console.WriteLine("Операцiя успiшна!");

MainMenu();

}

}

catch

{

Console.WriteLine("-------------------------------------");

Console.WriteLine("Неправильна назва рахунку!");

}

finally

{

SpendOrReplenish();

}

break;

case "2":

Console.WriteLine("-------------------------------------");

Console.WriteLine("Введiть назву рахунку: ");

string repname;

repname = Console.ReadLine();

Console.WriteLine("Введiть кiлькiсть коштiв, яку Ви хотiли б отримати: ");

decimal repvalue;

repvalue = Convert.ToDecimal(Console.ReadLine());

Console.WriteLine("-------------------------------------");

Console.WriteLine("Статтi отримання коштiв: ");

Console.WriteLine("зарплата, позика, бiзнес, подарунок");

Console.WriteLine("-------------------------------------");

Console.WriteLine("Введiть потрiбну статтю отримання коштiв: ");

string reptype;

reptype = Console.ReadLine();

try

{

if (repname == accounts[0].account\_name)

{

back(accounts[0], repvalue, reptype);

Console.WriteLine("Операцiя успiшна!");

MainMenu();

}

else if (repname == accounts[1].account\_name)

{

back(accounts[1], repvalue, reptype);

Console.WriteLine("Операцiя успiшна!");

MainMenu();

}

else if (repname == accounts[2].account\_name)

{

back(accounts[2], repvalue, reptype);

Console.WriteLine("Операцiя успiшна!");

MainMenu();

}

else if (repname == accounts[3].account\_name)

{

back(accounts[3], repvalue, reptype);

Console.WriteLine("Операцiя успiшна!");

MainMenu();

}

else if (repname == accounts[4].account\_name)

{

back(accounts[4], repvalue, reptype);

Console.WriteLine("Операцiя успiшна!");

MainMenu();

}

}

catch

{

Console.WriteLine("-------------------------------------");

Console.WriteLine("Неправильна назва рахунку!");

}

finally

{

SpendOrReplenish();

}

break;

case "3":

MainMenu();

break;

default:

Console.WriteLine("Неправильний вибiр.");

SpendOrReplenish();

break;

}

}

public static void OneToAnother()

{

Console.WriteLine("-------------------------------------");

Console.WriteLine("Введiть назву рахунку, з якого будуть знятi кошти: ");

string name1;

name1 = Console.ReadLine();

Console.WriteLine("Введiть назву рахунку, на який буде здiйснено переказ коштiв: ");

string name2;

name2 = Console.ReadLine();

Console.WriteLine("Введiть кiлькiсть коштiв для переказу: ");

decimal value;

value = Convert.ToDecimal(Console.ReadLine());

for (int i = 0; i < accounts.Length; i++)

{

for (int j = 0; j < accounts.Length; j++)

{

while (i != j)

{

try

{

if (name1 == accounts[i].account\_name && name2 == accounts[j].account\_name)

{

Account.Transfer(accounts[i], accounts[j], value);

Console.WriteLine("Операцiя успiшна!");

MainMenu();

}

else

{

Console.WriteLine("Неправильна назва рахунку.");

MainMenu();

}

}

catch

{

Console.WriteLine("-------------------------------------");

Console.WriteLine("Неправильна назва рахунку.");

}

finally

{

MainMenu();

}

}

}

}

}

}

}