**[KURS C#] Manager banku**



**UWAGA!**

**Nowa wersja kursu dostępna** jest na moim portalu z kursami. Nadal jest on **BEZPŁATNY**! A jednocześnie posiada więcej lekcji, wygodną nawigację i możliwość zapamiętania, w którym miejscu skończyłeś ostatnio!. Znajdziesz go [**POD TYM LINKIEM**](https://kurs-szarpania.pl/nauka/product/szkola-szarpania-intro)!

Mamy już zbudowane zaplecze naszego banku. Przyszła pora na ostatni element czyli **przestrzeń dla użytkowników**. Założenie jest takie żeby użytkownikom aplikacji bankowej, a więc na początku Jackowi, dać możliwość wykonywania operacji we w miarę przystępny sposób. Program w pierwszej wersji posiada co prawda tylko kilka podstawowych funkcji, ale bez jakiejś formy interakcji nawet ich nie można wykorzystać do rozpoczęcia pracy.

**Od menu do menu**

Plan jest taki aby aplikacja dawała możliwość poruszania się po niej za pomocą **menu testowego**. Tak więc będziemy mieli **menu główne**, z którego będzie można przejść do konkretnych opcji.

**Opcje** jakie pojawią się w tym menu to:

* 1 – Lista kont klienta
* 2 – Dodaj konto rozliczeniowe
* 3 – Dodaj konto oszczędnościowe
* 4 – Wpłać pieniądze na konto
* 5 – Wypłać pieniądze z konta
* 6 – Lista klientów
* 7 – Wszystkie konta
* 8 – Zakończ miesiąc
* 0 – Zakończ

Nie bez powodu dodałem na tej liście **liczby**. Użytkownik będzie wybierał poszczególne opcje poprzez wpisanie liczby przypisanej do konkretnej akcji i zatwierdzenie klawiszem ***Enter***.

Po wybraniu akcji albo dostanie na ekranie jakiś rezultat w postaci np. listy, albo będzie musiał wpisać jakieś dane i dopiero wtedy wyświetlą mu się jakieś informacje albo zostanie wprowadzona zmiana.

**Budujemy panel sterowania**

Na samym początku zaznaczę, że sam **manager banku** – jak go nazwiemy – nie jest zaprojektowany zgodnie ze sztuką. W tym przypadku służy on przede wszystkim do możliwości obsługiwania naszego programu i jest trochę z konieczności.

Nie chciałem na początku wprowadzać zbyt dużej złożoności do tego elementu. Dlatego mimo, że starałem się uniknąć części duplikacji kodu to takie się pojawią. Również zakres obowiązków klasy jest zbyt duży.

Jeżeli jesteś ambitny i chcesz już teraz zaznać samodzielnej pracy to zachęcam Cię w tym miejscu do samodzielnego rozwiązania tego problemu poprzez znalezienie odpowiednich **wzorców projektowych**, które pozwolą lepiej rozplanować tę część aplikacji. Podpowiem tylko, że przyda się tutaj dodanie kilku innych **klas** i **interfejsów**.

Zacznijmy od dodania do projektu pliku z klasą **BankManager**.

Następnie dodajemy w nim **prywatne zmienne** dla naszego **managera kont** i **drukarki**, które od razu tworzymy w **konstruktorze**:

using System;

namespace Bank

{

class BankManager

{

private AccountsManager \_accountsManager;

private IPrinter \_printer;

public BankManager()

{

\_accountsManager = new AccountsManager();

\_printer = new Printer();

}

}

}

Teraz czas na wyświetlenie na ekranie **listy dostępnych komend**. W tym celu dodajemy w klasie **managera banku** prywatną metodę PrintMainMenu():

private void PrintMainMenu()

{

Console.Clear();

Console.WriteLine("Wybierz akcję:");

Console.WriteLine("1 - Lista kont klienta");

Console.WriteLine("2 - Dodaj konto rozliczeniowe");

Console.WriteLine("3 - Dodaj konto oszczędnościowe");

Console.WriteLine("4 - Wpłać pieniądze na konto");

Console.WriteLine("5 - Wypłać pieniądze z konta");

Console.WriteLine("6 - Lista klientów");

Console.WriteLine("7 - Wszystkie konta");

Console.WriteLine("8 - Zakończ miesiąc");

Console.WriteLine("0 - Zakończ");

}

Wyświetlany w osobnych linijkach kolejne **akcje** wraz z przypisanymi do nich **numerami**, które użytkownik będzie mógł **wybrać**.

Wszystko jest na razie znajome. Może poza metodą Clear(). Służy ona do tego żeby wyczyścić konsolę z tekstu, który był na niej wcześniej wyświetlony. Dzięki temu zawsze będziemy mieli na ekranie tylko to co aktualnie jest istotne, bez informacji, które wyświetlaliśmy w jakiejś innej sytuacji.

**Pętla władzy**

No dobra, mamy metodę do wyświetlania menu. Jest ona prywatna, co sugeruje, że gdzieś ją użyjemy wewnątrz naszej klasy. Dokładnie tak będzie.

**Wyjściem na świat** i miejscem, które pozwoli skorzystać z naszego **managera banku**, niejako otworzyć drzwi wejściowe go holu, będzie publiczna metoda Run(). Dodajmy ją więc w klasie **BankManager** i wyświetlmy w niej **menu główne** korzystając z dodanej niedawno funkcji:

public void Run()

{

PrintMainMenu();

}

Super. Teraz tej metody Run() możemy użyć w programie i już mieć **menu**…. i nic więcej. Gdzie jest miejsce na **interakcję użytkownika**? Czas to rozwiązać.

**Wybór użytkowników**

Najpierw dodajmy w managerze **prywatną metodę** do odczytywanie **numeru akcji** jaką użytkownik chce wybrać. Spokojnie, za moment dowiesz się po co Ci to:

private int SelectedAction()

{

Console.Write("Akcja: ");

string action = Console.ReadLine();

if (string.IsNullOrEmpty(action))

{

return -1;

}

return int.Parse(action);

}

Po pierwsze wypisujemy na ekranie zachętę do wpisania czegoś, w tym wypadku **numeru akcji**. Tutaj, w przeciwieństwie do większości wcześniejszych sytuacji,  wypisujemy tekst tak, że następna rzecz po nim nie będzie w nowej linii. Tym się różni metoda Write() od WriteLine() – nie dodaje znaku końca linii. Dzięki temu to co użytkownik wpisze pojawi się w tej samej linii co nasz tekst. Będzie to wyglądać bardziej estetycznie.

Kolejna rzecz to **odczytanie** po wciśnięciu klawisza ***Enter***tego co użytkownik wpisał i zapisanie tego do zmiennej. Korzystamy tutaj z kolejnej metody z klasy **Console**. Zwraca ona wartość typu **string** zawierającą to co użytkownik wpisał w aktualnej linii.

Kolejna linijka to sprawdzenie czy tekst wpisany przez użytkownika **nie jest pusty**. Klasa **string** posiada bardzo fajną **funkcję pomocniczą**, która zwróci true kiedy podana zmienna będzie pusta. Dzięki temu możemy łatwo to sprawdzić bez pisania porównania samodzielnie. Jeżeli wartość jest pusta to znak, że użytkownik nie wpisał nic tylko od razu wcisnął ***Enter***. W takim wypadku nie mamy czego zamieniać na wybraną liczbę więc zwracamy wartość, która nie istnieje w naszym menu. W tym wypadku **-1**.

I teraz jeżeli użytkownik jednak coś wpisał to zamieniamy ten **tekst** na **wartość liczbową**. Służy do tego funkcja dostępna w typie **int** (i w każdym innym typie liczbowym), która przyjmuje jako parametr tekst i zwraca liczbę, która była za jego pomocą zapisana już jako wartość typu liczbowego. Jest to bardzo uproszczona czynność. Normalnie byśmy sprawdzili chociażby czy tekst zawiera tylko cyfry. W obecnej formie wpisanie czegoś innego spowoduje, że program rzuci błąd.

Ok, mamy teraz dodaną funkcję do **pobierania wyboru użytkownika**, która zwraca wybrany przez niego **numer** w postaci **wartości liczbowej**. Ale co teraz?

**Do skutku**

Teraz chcemy żeby program pozwalał **wykonywać wielokrotnie** różne operacje. Przynajmniej do czasu kiedy użytkownik nie wpisze wartości **0** oznaczającej **koniec**. Skoro coś ma się dziać wielokrotnie to może zastosować **pętlę**? Dokładnie tak! W tym wypadku chcemy żeby **menu** wyświetliło się **przynajmniej raz** zanim użytkownik cokolwiek wybierze. Dlatego dobrym pomysłem będzie skorzystanie z **pętli do…while**:

public void Run()

{

int action;

do

{

PrintMainMenu();

action = SelectedAction();

}

while (action != 0);

}

Najpierw przed pętlą dodaliśmy sobie zmienną, którą wykorzystamy do przechowywania wyboru użytkownika aplikacji bankowej.

Następnie korzystamy ze **słowa kluczowego** do, którym oznaczamy **blok kodu**, który ma się wykonać w ramach **pętli**. Sam kod wstawiamy pomiędzy **nawiasy klamrowe**. To co będzie się wykonywać za każdym razem to wyświetlenie menu i odczytanie wyboru użytkownika.

Na końcu, poza blokiem do , mamy **warunek**, który sprawdza czy pętla powinna się nadal wykonywać. Jeżeli program przeszedł przez cały **blok** do to teraz **sprawdza warunek** podany obok słowa kluczowego while. Jeżeli jest prawdziwy to wraca do słowa do i wykonuje ponownie podany **blok kodu**. W naszym przypadku tym warunkiem jest sprawdzenie czy wybrana przez użytkownika akcja jest **różna od 0**, które to, według naszego menu oznacza **koniec programu**.

W tym miejscu możemy już zatrudnić do pracy naszego **managera banku**. Przejdźmy więc do funkcji Main() w klasie **Program**. Wyrzućmy to co mieliśmy tam do tej pory. Utwórzmy **obiekt managera** i uruchommy go:

namespace Bank

{

class Program

{

static void Main(string[] args)

{

BankManager bankManager = new BankManager();

bankManager.Run();

}

}

}

Włączając teraz naszą aplikację zobaczymy na ekranie **menu**. Możemy też wpisać **dowolną liczbę**. Jeżeli nie jest to **0** to program ponownie wyświetli menu i ponownie pozwoli nam wpisać liczbę. Po wpisaniu **0** aplikacja wyłączy się.

Przy okazji widać, że teraz zadaniem funkcji Main() jest tylko uruchomienie innych części **systemu**. Jest to dobra praktyka, aby punkt wejścia, w postaci tej funkcji, służył faktycznie tylko jako wejście do **pozostałych części naszej aplikacji**.

W porządku, mamy menu na ekranie, możemy wpisać liczbę i dla zera aplikacja się wyłączy. Ale co z pozostałymi akcjami?! Jakoś musimy je uruchamiać po wpisaniu przez użytkownika **poprawnej liczby**.

**Na rozkaz!**

Aby rozwiązać problem wyboru konkretnej akcji i uruchomienia związanej z nią **funkcjonalności** zastosujemy **dedykowane metody** dla każdej z opcji i wybór na podstawie wpisanej wartości.

Ta część jest właśnie fragmentem, który można lepiej rozwiązać jeżeli ktoś ma trochę doświadczenia. Jednak aby wszystko było widoczne na pierwszy rzut oka dla osób mniej zaznajomionych z kodem w **C#** czy w ogóle **kodem obiektowym** to wybrałem takie rozwiązanie.

**Do wyboru…**

Na początek sprawa przydzielania wybranej funkcji do **konkretnej liczby**. W tym celu zastosujemy w metodzie Run() i znajdującej się w niej pętli do…while przełącznik. Chodzi o instrukcję switch  dostępną w języku **C#**. Jeszce o niej nie mówiliśmy, a to kolejna z **instrukcji warunkowych**, jaka jest dla nas dostępna. Najpierw zobaczmy jak wygląda jej zastosowanie. Dopóki nie napiszemy docelowych funkcji to skorzystamy z wypisywania na ekranie tekstu mówiącego o tym jaka **opcja została wybrana**:

public void Run()

{

int action;

do

{

PrintMainMenu();

action = SelectedAction();

switch (action)

{

case 1:

Console.Clear();

Console.WriteLine("Wybrano listę kont klienta");

Console.ReadKey();

break;

case 2:

Console.Clear();

Console.WriteLine("Wybrano otwarcie konta rozliczeniowego");

Console.ReadKey();

break;

case 3:

Console.Clear();

Console.WriteLine("Wybrano otwarcie konta oszczędnościowego");

Console.ReadKey();

break;

case 4:

Console.Clear();

Console.WriteLine("Wybrano wpłatę pieniędzy na konto");

Console.ReadKey();

break;

case 5:

Console.Clear();

Console.WriteLine("Wybrano wypłatę pieniędzy z konta");

Console.ReadKey();

break;

case 6:

Console.Clear();

Console.WriteLine("Wybrano listę klientów");

Console.ReadKey();

break;

case 7:

Console.Clear();

Console.WriteLine("Wybrano listę wszystkich kont");

Console.ReadKey();

break;

case 8:

Console.Clear();

Console.WriteLine("Wybrano zamknięcie miesiąca");

Console.ReadKey();

break;

}

}

while (action != 0);

}

W tym momencie dla każdej **dostępnej akcji** robimy trzy rzeczy: **czyścimy konsolę**, **wyświetlany informację** o akcji i czekamy aż użytkownik **wciśnie jakiś klawisz**, żeby miał szansę zobaczyć dostatecznie długo to co zostało wyświetlone.

Kiedy uruchomimy program i zaczniemy wpisywać wybrane numery to dostaniemy w konsoli komunikat mówiący o tym jaka akcja została wybrana.

Widzimy tutaj strukturę **instrukcji switch**, która, jak nazwa wskazuje jest swego rodzaju **przełącznikiem**. Przekazujemy jej wartość, na podstawie której będzie szukała pasującego przypadku (**case**). U nas ta wartość znajduje się w zmiennej action. Instrukcja **switch** przechodzi przez kolejne przypadki zapisane jako case wartosc: i sprawdza czy to co dostała na początku pasuje do wartości jaka jest związana z danym przypadkiem. Jeżeli tak to wykonuje kod znajdujący się za dwukropkiem, aż do instrukcji break, która oznacza **koniec danego bloku**. Także oznaczanie bloku kodu różni się w tym wypadku od tego znanego z pozostałych części programu, gdzie bloki oznaczaliśmy **nawiasami klamrowymi**.

Po wykonaniu pasującego przypadku **switch** kończy swoje działanie nie sprawdzając kolejnych wartości. Jeżeli żadna wartość nie będzie pasować, czyli np. u nas będzie to wartość **0** to **switch** po prostu nie wykona niczego i się zakończy. Moglibyśmy jednak obsłużyć taki przypadek dodając blok default:

switch (action)

{

case 1:

Console.Clear();

Console.WriteLine("Wybrano listę kont klienta");

Console.ReadKey();

break;

default:

Console.Clear();

Console.WriteLine("Nieznane polecenie");

Console.ReadKey();

break;

}

Umieszczony na końcu zadziała w momencie kiedy **żadna wartość** w dodanych do **switch’a** przypadkach nie pasowała do wartości jaką przekazaliśmy. Możemy wtedy np. wyświetlić pokazany powyżej komunikat o nieznanym poleceniu.

Nie pozostało nic innego jak dodanie faktycznych funkcji do naszego menu, zamiast wyświetlanie tylko informacji o nich.

**Lista kont klienta**

Na początek opcja kryjąca się pod numerem **1** w menu czyli wyświetlenie **listy wszystkich konta danego klienta**.

To co musimy zrobić to:

* Pobrać dane o kliencie
* Wyciągnąć listę wszystkich jego kont na podstawie tych danych
* Wydrukować dane każdego konta

Prywatna funkcja, która to będzie robiła wygląda w następujący sposób i nazywa się u mnie ListOfAccounts():

private void ListOfAccounts()

{

Console.Clear();

CustomerData data = ReadCustomerData();

Console.WriteLine();

Console.WriteLine("Konta klienta {0} {1} {2}", data.FirstName, data.LastName, data.Pesel);

foreach(Account account in \_accountsManager.GetAllAccountsFor(data.FirstName, data.LastName, data.Pesel))

{

\_printer.Print(account);

}

Console.ReadKey();

}

Najpierw mamy znane nam już **czyszczenie konsoli** (będziemy go używać w każdej funkcji żeby całość ładnie się prezentowała). Potem jest **zebranie danych** o kliencie. Wyświetlenie tych danych żeby było wiadomo co robimy. A potem już tylko wyciągnięcie wszystkich kont danego klienta za pomocą przygotowanej do tego funkcji w **managerze kont** i od razu użycie ich w **pętli foreach**, która pozwoli nam wydrukować za pomocą przygotowanej kiedyś drukarki dane każdego konta.

Ale chwila moment! Przy pobieraniu danych przeleciałem tak szybko jakby wszystko było oczywiste, a przecież tam się coś dzieje! Masz rację. Samo pobieranie informacji o kliencie przeniosłem do **innej funkcji** bo będzie potrzebne w kilku miejscach. Dodatkowo utworzyłem na nie klasę, bo nie mogłem z metody zwrócić trzech wartości jednocześnie.

W tym samym pliku, czyli ***BankManager.cs***, za klasą **BankManager** dodałem drugą klasę **CustomerData**:

class CustomerData

{

public string FirstName { get; }

public string LastName { get; }

public long Pesel { get; }

public CustomerData(string firstName, string lastName, string pesel)

{

FirstName = firstName;

LastName = lastName;

Pesel = long.Parse(pesel);

}

}

Zrobiłem to w ten sposób bo tej klasy potrzebujemy tak naprawdę tylko w tym **managerze banku** dlatego nie było potrzeby wydzielania jej do **osobnego pliku**. Warto zauważyć, że zamianę tekstu na liczbę jaką jest **PESEL** zrobiłem w **konstruktorze**. Dzięki temu taki szczegół mamy zamknięty i możemy po prostu przekazać jako argument bezpośrednio to co zwróci funkcja odczytująca tekst wpisany przez użytkownika.

Mając taki **kontener** na zebrane od użytkownika **dane klienta** brakuje jeszcze funkcji do pobierania tych danych. Dodajmy więc taką w klasie **BankManager**:

private CustomerData ReadCustomerData()

{

string firstName;

string lastName;

string pesel;

Console.WriteLine("Podaj dane klienta:");

Console.Write("Imię: ");

firstName = Console.ReadLine();

Console.Write("Nazwisko: ");

lastName = Console.ReadLine();

Console.Write("PESEL: ");

pesel = Console.ReadLine();

return new CustomerData(firstName, lastName, pesel);

}

Tutaj już nie dzieje się nic dla nas zaskakującego. Dodanie zmiennych na dane. Wyświetlenie jakichś zachęt w konsoli. Pobranie tego co użytkownik wpisał i zapisanie w odpowiedniej zmiennej. Na koniec zwrócenie nowego obiektu ze wszystkimi danymi klienta jakie dostaliśmy od użytkownika aplikacji. I właśnie tej funkcji używamy w poprzedniej metodzie do wyświetlania listy kont klienta. Zwraca nam ona dane, które potem możemy wykorzystać w różnych celach.

**Nowy klient = nowe konto**

Teraz pora na akcję dodawania **nowego konta rozliczeniowego**. To co musimy tutaj zrobić to:

* Pobrać dane klienta
* Utworzyć nowe konto w managerze kont
* Wyświetlić dane nowego konta

Zadanie prostsze niż poprzedni punkt o tyle, że mamy już gotową funkcję do pobierania danych klienta. Dlatego właśnie została ona wydzielona jako osobny kawałek kodu, który możemy teraz użyć kolejny raz:

private void AddBillingAccount()

{

Console.Clear();

CustomerData data = ReadCustomerData();

Account billingAccount = \_accountsManager.CreateBillingAccount(data.FirstName, data.LastName, data.Pesel);

Console.WriteLine("Utworzono konto rozliczeniowe:");

\_printer.Print(billingAccount);

Console.ReadKey();

}

Wyczyszczenie konsoli.  Pobranie danych klienta. Utworzenie konta w systemie za pomocą odpowiedniej funkcji, którą dodaliśmy poprzednio w **managerze kont**. Funkcja ta zwraca obiekt nowo utworzonego konta dlatego wykorzystamy to i od razu wyświetlamy informacje o tym koncie użytkownikowi.

**Ćwiczenie 1**

Jako ćwiczenie dodaj analogiczną metodę do **tworzenia kont oszczędnościowych**. Ma robić to samo tylko dla drugiego **typu konta**. Dla zachowania spójności w dalszych etapach przyjmijmy, że nazwa tej metody to będzie AddSavingsAccount().

**Ćwiczenie 2**

W dalszym etapie budowania **managera banku** będziemy przypisywali poszczególne funkcje do konkretnych liczb w menu. Dla metody do zakładania **konta oszczędnościowego** znajdź odpowiednią pozycję w instrukcji **switch** i dodaj jej wywołanie samodzielnie zamieniając znajdujący się tam kod.

**Odkładam na emeryturę**

Przyszła pora na pierwszą operację zmieniającą coś na jednym z kont. Dodajmy funkcję umożliwiającą **wpłacenie określonej kwoty** na konto rozliczeniowe albo oszczędnościowe. Kroki jakie musimy zrobić to:

* Pobieramy numer konta, na które wpłacamy pieniądze
* Pobieramy kwotę jaka ma być wpłacona
* Korzystając z funkcji managera kont przekazujemy podaną kwotę na konto
* Pobieramy obiekt tego konta i wyświetlamy informacje o nim, aby być pewnym, że zmieniło się saldo na nim

Najpierw funkcja, potem wyjaśnienia. Jednak również tutaj nie ma zbyt dużej dużego zaskoczenia jeżeli uważnie przechodzisz przez cały kurs:

private void AddMoney()

{

string accountNo;

decimal value;

Console.WriteLine("Wpłata pieniędzy");

Console.Write("Numer konta: ");

accountNo = Console.ReadLine();

Console.Write("Kwota: ");

value = decimal.Parse(Console.ReadLine());

\_accountsManager.AddMoney(accountNo, value);

Account account = \_accountsManager.GetAccount(accountNo);

\_printer.Print(account);

Console.ReadKey();

}

Dodajemy zmienne, która pozwolą nam przechować dane podane przez użytkownika. Potem pobieramy od niego numer konta na jakie ma być wykonana wpłata. Pobieramy kwotę i chcąc ją włożyć do zmiennej zamieniamy tekst na liczbę typu **decimal** bo takiego typu liczby wykorzystujemy w operacjach na pieniądzach. Następnie używamy funkcji z **managera kont**, która pozwala dodać pieniądze do konta. I na koniec zostaje nam tylko pobranie obiektu tego konta za pomocą jego numeru i przygotowanej w tym celu funkcji i wyświetlenie jego danych przy użyciu doskonale nam znanej drukarki.

**Ćwiczenie 3**

Dodaj w klasie **BankManager** analogiczną metodę tyle, że do **wypłacania pieniędzy z konta**. Zasada działania będzie taka sama – pobranie **numeru konta** i **kwoty**, a następnie zmiana salda na koncie. Niech nazywa się TakeMoney(). Wykorzystaj odpowiednią funkcję z **managera kont**, która umożliwia **pobieranie pieniędzy** z podanego konta.

**Ćwiczenie 4**

W dalszym etapie budowania **managera banku** będziemy przypisywali poszczególne funkcje do konkretnych **liczb w menu**. Dla metody do **wypłacania pieniędzy** z konta znajdź odpowiednią pozycję w instrukcji **switch** i dodaj jej wywołanie samodzielnie zamieniając znajdujący się tam kod.

**Gdzie są wszyscy?**

Następna z opcji jakie mamy do wyboru to wyświetlanie **listy wszystkich klientów**. Na szczęście nasz **manager kont** posiada odpowiednią metodę do tego dlatego mamy proste zadanie.

* Pobierzemy już sformatowaną listę klientów
* Wyświetlimy dane każdego z nich w osobnej linii

Sprawa nie jest skomplikowana:

private void ListOfCustomers()

{

Console.Clear();

Console.WriteLine("Lista klientów:");

foreach(string customer in \_accountsManager.ListOfCustomers())

{

Console.WriteLine(customer);

}

Console.ReadKey();

}

Mamy już taką wprawę w pisaniu tego typu kodu, że temat nie wymaga więcej wyjaśnień. Po prostu korzystamy z odpowiedniej metody zwracającej listę obiektów typu **string**.

**Przegląd zasobów**

Przedostatnia funkcja jakiej potrzebujemy to wyświetlanie **listy wszystkich kont założonych w naszym systemie**. Sprawa jest jeszcze prostsza niż w przypadku listy kont konkretnego klienta bo w tym wypadku nie musimy nawet zbierać informacji o tym kliencie. Wystarczy, że skorzystamy z metody zwracającej wszystkie konta z **managera kont** i wyświetlimy w konsoli dane każdego z nich.

private void ListOfAllAccounts()

{

Console.Clear();

Console.WriteLine("Wszystkie konta:");

foreach(Account account in \_accountsManager.GetAllAccounts())

{

\_printer.Print(account);

}

Console.ReadKey();

}

**Podsumowanie miesiąca**

Ostatnia metoda, którą dopisujemy jest jednocześnie najprostszą. **Zamykanie miesiąca** to nic innego jak wykonanie jednej funkcji z **managera kont** i poinformowanie o tym fakcie użytkownika.

private void CloseMonth()

{

Console.Clear();

\_accountsManager.CloseMonth();

Console.WriteLine("Miesiąc zamknięty");

Console.ReadKey();

}

Dobrze, że tego typu operacje zamknęliśmy **w osobnych funkcjach**, które teraz możemy po prostu używać nie przejmując się co się dzieje pod spodem.

**Kolejno odlicz!**

Zostało nam przypisanie dodanych przed chwilą funkcji do **konkretnych numerów w naszym menu**. Będzie to moment spięcia całości w jeden działający twór. Można to porównać do włączenia zasilania i przekręcenia klucza w zamku w drzwiach do naszego oddziału banku, który w pocie czoła budowaliśmy do tej pory.

Wróćmy więc do **instrukcji switch**, którą wcześniej omawialiśmy:

public void Run()

{

int action;

do

{

PrintMainMenu();

action = SelectedAction();

switch (action)

{

case 1:

Console.Clear();

Console.WriteLine("Wybrano listę kont klienta");

Console.ReadKey();

break;

case 2:

Console.Clear();

Console.WriteLine("Wybrano otwarcie konta rozliczeniowego");

Console.ReadKey();

break;

case 3:

Console.Clear();

Console.WriteLine("Wybrano otwarcie konta oszczędnościowego");

Console.ReadKey();

break;

case 4:

Console.Clear();

Console.WriteLine("Wybrano wpłatę pieniędzy na konto");

Console.ReadKey();

break;

case 5:

Console.Clear();

Console.WriteLine("Wybrano wypłatę pieniędzy z konta");

Console.ReadKey();

break;

case 6:

Console.Clear();

Console.WriteLine("Wybrano listę klientów");

Console.ReadKey();

break;

case 7:

Console.Clear();

Console.WriteLine("Wybrano listę wszystkich kont");

Console.ReadKey();

break;

case 8:

Console.Clear();

Console.WriteLine("Wybrano zamknięcie miesiąca");

Console.ReadKey();

break;

}

}

while (action != 0);

}

Sprawdźmy teraz jaka akcja przypada na jaki numer i po prostu podmieńmy kod, który się tam znajduje na wywołanie konkretnej funkcji. Cała funkcja Run() powinna prezentować się tak jak pokazałem poniżej.

**Uwaga: niektóre przypadki nie zostaną przeze mnie tutaj podmienione z uwagi na ćwiczenia, które znajdowały się w tekście powyżej**.

public void Run()

{

int action;

do

{

PrintMainMenu();

action = SelectedAction();

switch (action)

{

case 1:

ListOfAccounts();

break;

case 2:

AddBillingAccount();

break;

case 3:

Console.Clear();

Console.WriteLine("Wybrano otwarcie konta oszczędnościowego");

Console.ReadKey();

break;

case 4:

AddMoney();

break;

case 5:

Console.Clear();

Console.WriteLine("Wybrano wypłatę pieniędzy z konta");

Console.ReadKey();

break;

case 6:

ListOfCustomers();

break;

case 7:

ListOfAllAccounts();

break;

case 8:

CloseMonth();

break;

default:

Console.Write("Nieznane polecenie");

break;

}

}

while (action != 0);

}

Jeżeli w tym momencie uruchomimy program i zaczniemy wpisywać wybrane liczby to przekonamy się, że mamy **w pełni działający program bankowy**! Oznacza to, że zlecenie dla kolegi Jacka zostało wykonane i możemy mu je przekazać.

Tym zajmiemy się w ostatniej części gdzie podsumujemy naszą pracę. W końcu każdy poważny projekt powinien zakończyć się poważnym podsumowaniem i zweryfikowaniem początkowych założeń i oczekiwań klienta. Wtedy też podsumujemy czego się w trakcie całego kursu nauczyliśmy i jakie kolejne kroki można podjąć w celu rozwijania swojej wiedzy.

A na razie świętujmy sukces i na wszelki wypadek sprawdźmy czy na pewno wszystko działa bo testów nigdy za wiele ;)