

Materiały do przedmiotu:

Podstawy programowania

Laboratorium nr 1

Wprowadzenie do programowania. Instalacja środowiska, konfiguracja IDE i tworzenie pierwszych programów konsolowych. Zmienne i typy danych

Autor:

mgr Krystyna Kiersztyn



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Spis treści

Wprowadzenie, tematyka zajęć	3
Cel laboratorium	3
Instrukcja laboratoryjna	3
Ćwiczenia samodzielne	30
Pytania pomocnicze	33
Podsumowanie	34
Literatura	34



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Projekt współfinansowany ze środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Wprowadzenie, tematyka zajęć

Programowanie polega na tworzeniu zestawu *instrukcji*, które komputer wykonuje w celu rozwiązania określonego problemu. Podstawą każdego programu jest *algorytm*, czyli skończony, uporządkowany zestaw kroków prowadzących do rozwiązania danego zadania. Do zapisu algorytmów wykorzystuje się *języki programowania*.

Język C# jest językiem wysokiego poziomu, opracowanym przez firmę Microsoft, działającym w ramach platformy .NET. Umożliwia on tworzenie aplikacji różnego typu – od prostych programów konsolowych po złożone aplikacje okienkowe i webowe. C# jest językiem *kompilowanym*. Oznacza to, że przed uruchomieniem kod źródłowy jest tłumaczony przez kompilator na kod pośredni, który następnie jest uruchamiany przez środowisko wykonawcze. Kod w C# jest uporządkowany w *klasy* i *metody*. Instrukcje kończą się średnikiem, a nawiasy klamrowe { } wyznaczają bloki kodu.

Zmienne to nazwy reprezentujące obszary pamięci, w których przechowywane są dane. Każda zmienna posiada określony *typ*, który definiuje rodzaj wartości, jakie mogą być do niej przypisane. Do wykonywania operacji na zmiennych używane są *operatory* arytmetyczne, porównania, logiczne, przypisania, bitowe, inkrementacji i dekrementacji.

Cel laboratorium

Po zrealizowaniu laboratorium student:

- potrafi zainstalować i skonfigurować środowisko programistyczne,
- zna strukturę prostego programu konsolowego w C#,
- potrafi zadeklarować i wykorzystać zmienne różnych typów,
- potrafi stosować podstawowe operatory arytmetyczne, logiczne i porównania.

Instrukcja laboratoryjna

I. Instalacja i konfiguracja środowiska programistycznego

W celu pisania i uruchamiania programów, potrzebne jest odpowiednie środowisko programistyczne, czyli zestaw narzędzi, które wspierają tworzenie, testowanie i uruchamianie kodu. W przypadku języka C# często używa się środowiska Visual Studio, w połączeniu z platformą .NET.



Fundusze Europejskie
dla Rozwoju Społecznego

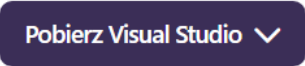


Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską

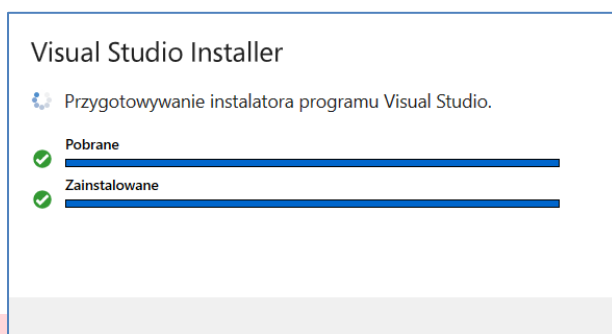


1. Pobranie instalatora Visual Studio.

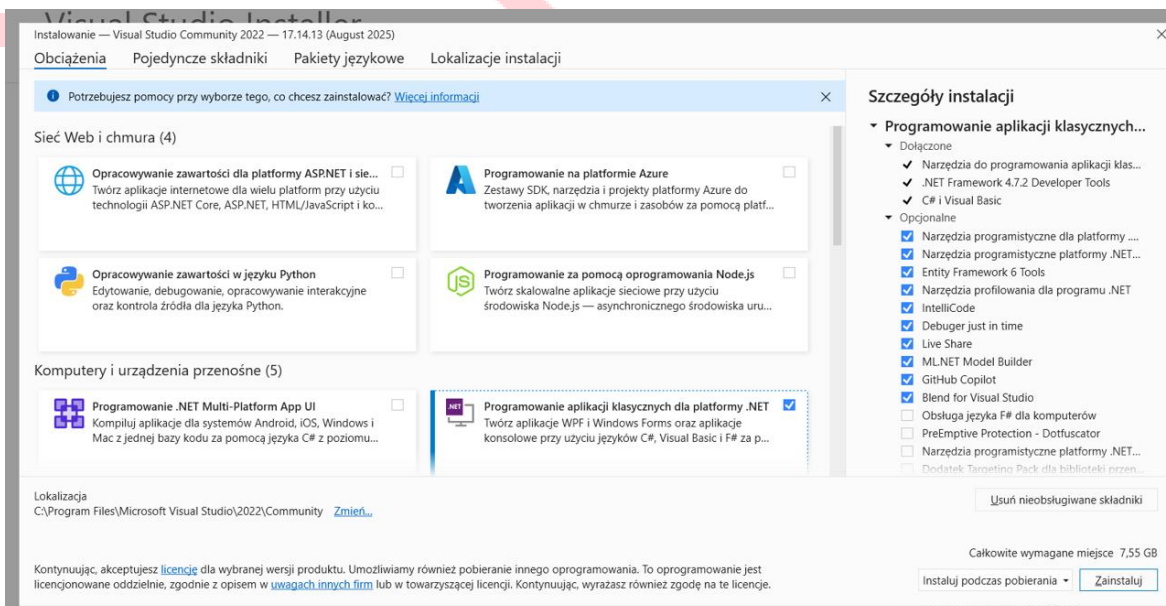
- Przejdź na stronę <https://visualstudio.microsoft.com/pl/vs/>.
- Kliknij przycisk 
- Wybierz wersję *Community* i pobierz plik instalatora *VisualStudioSetup.exe*.

2. Uruchomienie instalatora i wybór składników.

- Otwórz pobrany plik instalatora *VisualStudioSetup.exe*.



- Po krótkim przygotowaniu instalator wyświetli listę zestawów narzędzi dostosowanych do konkretnych typów projektów. Zaznacz opcję **Programowanie aplikacji klasycznych dla platformy .NET**. Pakiet ten umożliwia tworzenie aplikacji konsolowych i okienkowych w języku C#. Wybierz preferowaną lokalizację instalacji w zakładce **Lokalizacje instalacji**. Następnie kliknij przycisk **Zainstaluj**.



Fundusze Europejskie
dla Rozwoju Społecznego



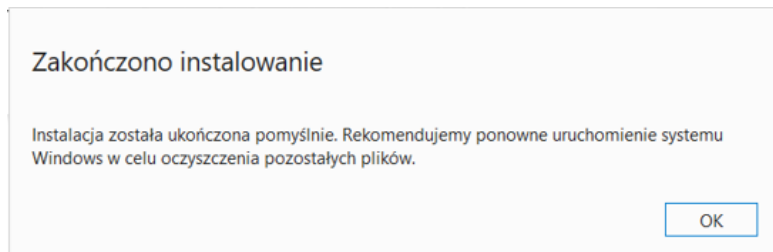
Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



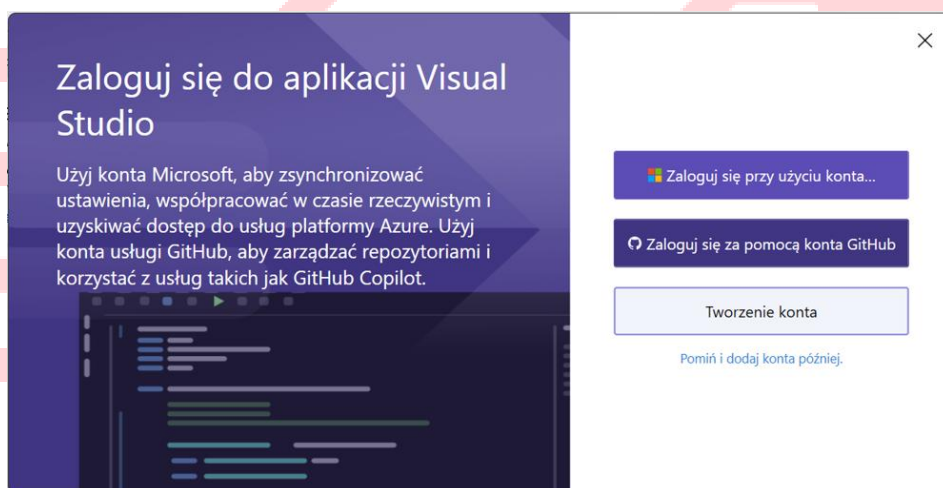
Projekt współfinansowany ze środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

- Po zakończeniu procesu instalacji zalecane jest ponowne uruchomienie systemu.

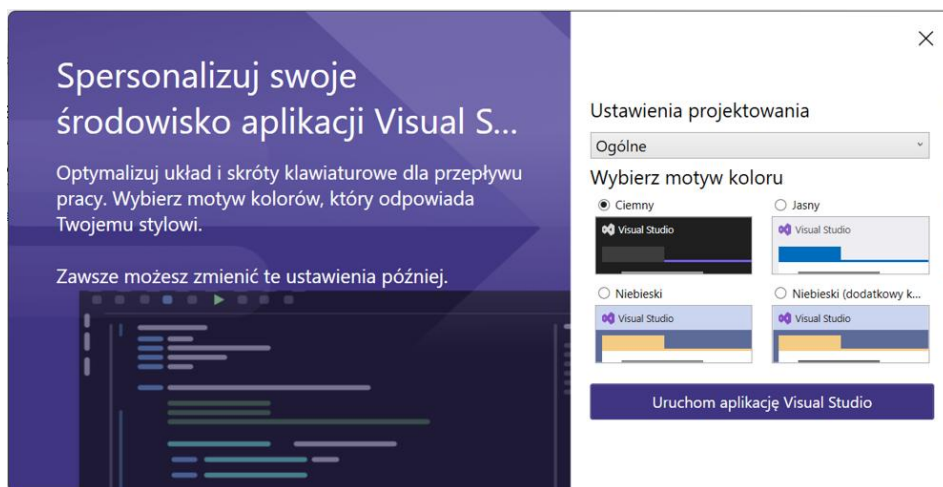


3. Pierwsze uruchomienie Visual Studio.

- Uruchom Visual Studio. Przy pierwszym uruchomieniu może pojawić się prośba o zalogowanie się do konta Microsoft. Możesz ten krok pominąć lub zalogować się np. za pomocą konta studenckiego (w tym celu wybierz opcję „Zaloguj się przy użyciu konta Microsoft”).

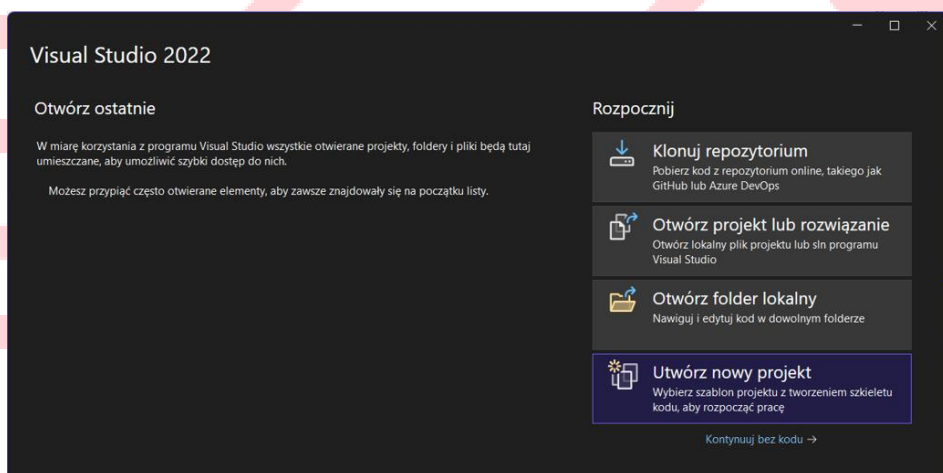


- Wybierz ustawienia projektowania (**Ogólne** lub **Visual C#**) oraz preferowany motyw kolorystyczny interfejsu. Następnie kliknij przycisk **Uruchom aplikację Visual Studio**.



II. Tworzenie pierwszego programu konsolowego w języku C#

1. Po uruchomieniu Visual Studio kliknij przycisk **Utwórz nowy projekt**.



2. Wybierz szablon **Aplikacja konsoli** i kliknij przycisk **Dalej**.



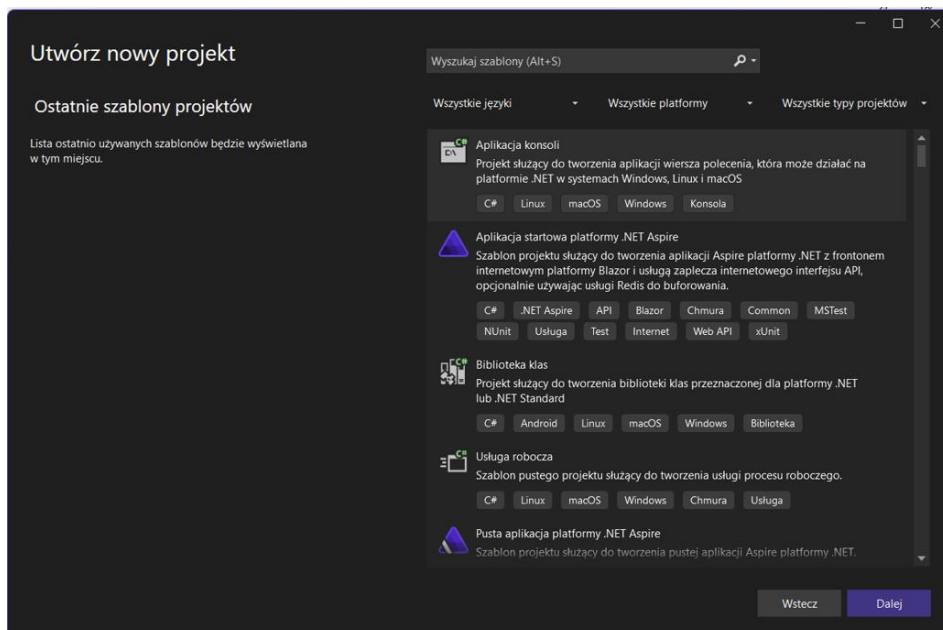
Fundusze Europejskie
dla Rozwoju Społecznego



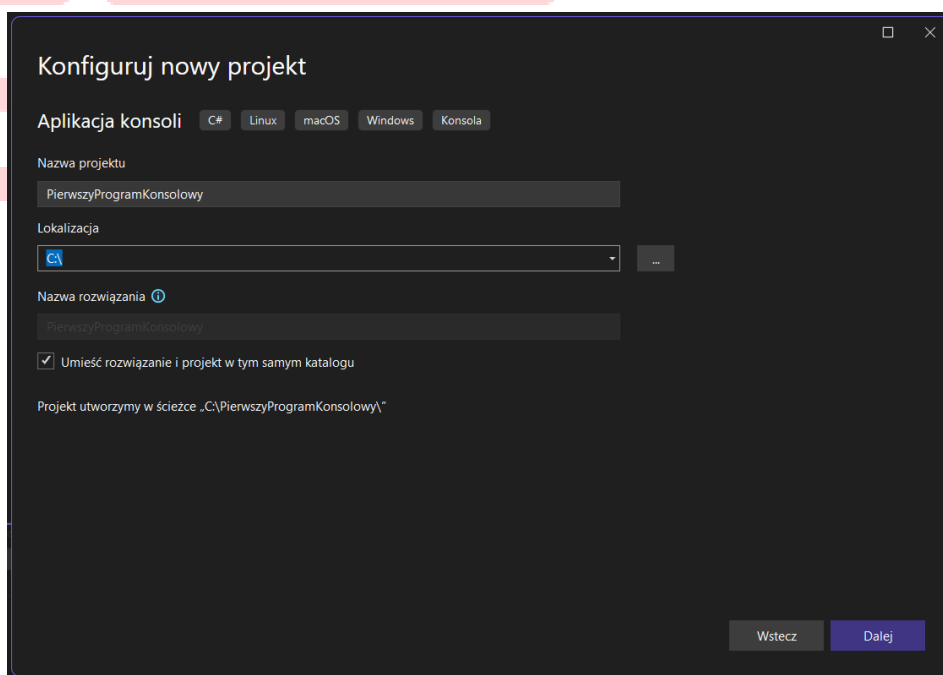
Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską





3. Nadaj projektowi nazwę *PierwszyProgramKonsolowy*. Dodatkowo można zaznaczyć opcję **Umieść rozwiązanie i projekt w tym samym katalogu** oraz ustawić odpowiednią lokalizację projektu. Następnie kliknij przycisk **Dalej**.



4. W następnym kroku wybierz dostępną platformę np. **.NET 8.0**.



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

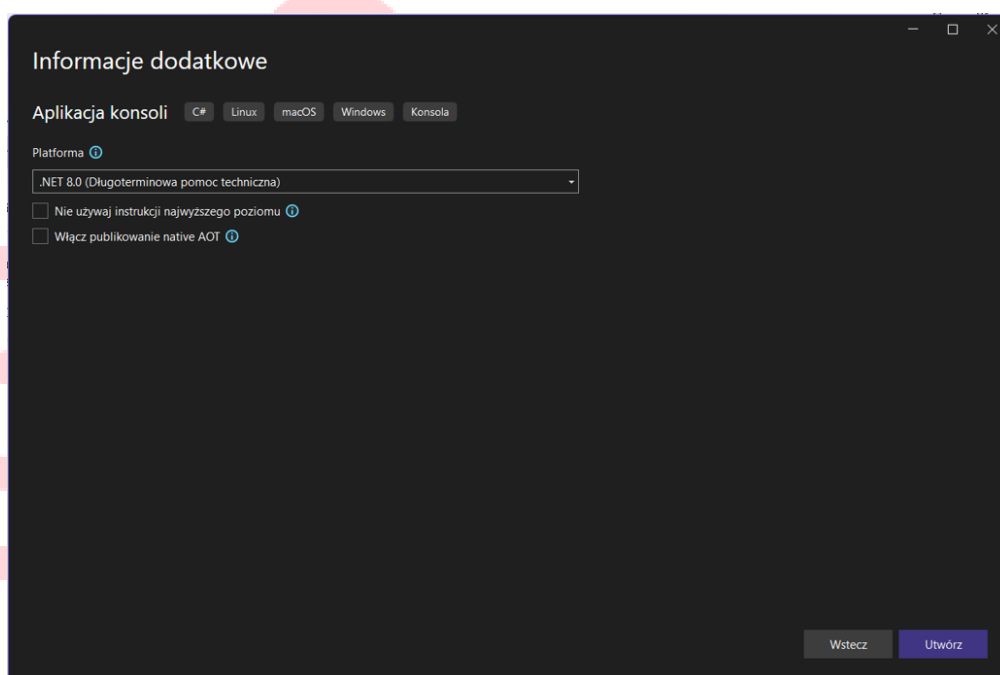
Dofinansowane przez
Unię Europejską



5. W tym miejscu warto zwrócić uwagę na pole wyboru **Nie używaj instrukcji najwyższego poziomu**. To opcja decydująca o tym, czy program zostanie wygenerowany z wykorzystaniem instrukcji najwyższego poziomu, czy z tradycyjną strukturą opartą na klasie i metodzie `Main()`. Rozważmy tu dwie możliwości:

a) **Nie zaznaczona opcja „Nie używaj instrukcji najwyższego poziomu”.**

- Po wybraniu platformy kliknij **Utwórz**. Visual Studio wygeneruje program z tzw. *instrukcjami najwyższego poziomu* (top-level statements).



- Po utworzeniu projektu, Visual Studio otworzy plik *Program.cs*. Znajduje się w nim domyślny kod:



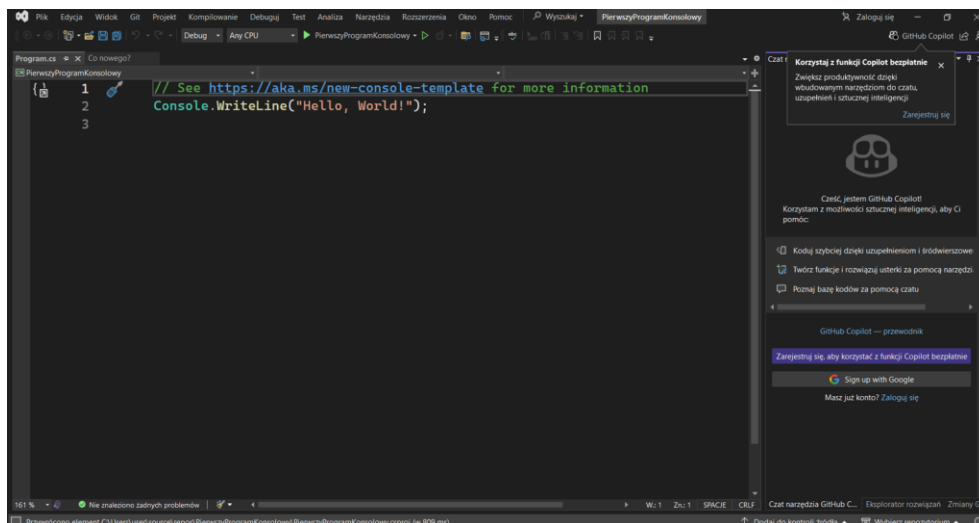
Fundusze Europejskie
dla Rozwoju Społecznego



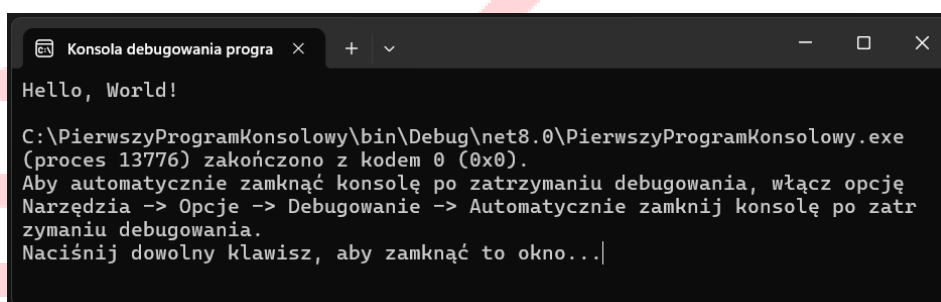
Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską





- Aby uruchomić program kliknij przycisk  **PierwszyProgramKonsolowy** lub wciśnij **Ctrl + F5** (uruchomienie bez debugowania). W nowym oknie konsoli powinien wyświetlić się napis „Hello, World!”.



Komentarze jednowierszowe języka C# zaczynają się od // i trwają do końca bieżącej wiersza. Dostępne są także komentarze wielowierszowe, które rozpoczyna się od /*, a kończy na */. Instrukcja Console.WriteLine z przestrzeni nazw System odpowiada za wyświetlanie danych w konsoli.

W klasycznej postaci programu konieczne jest zdefiniowanie odpowiedniej klasy oraz metody Main() (pokazano w pkt. b)). Kompilator sam tworzy te elementy w tle, gdy używasz instrukcji najwyższego poziomu.



Fundusze Europejskie
dla Rozwoju Społecznego



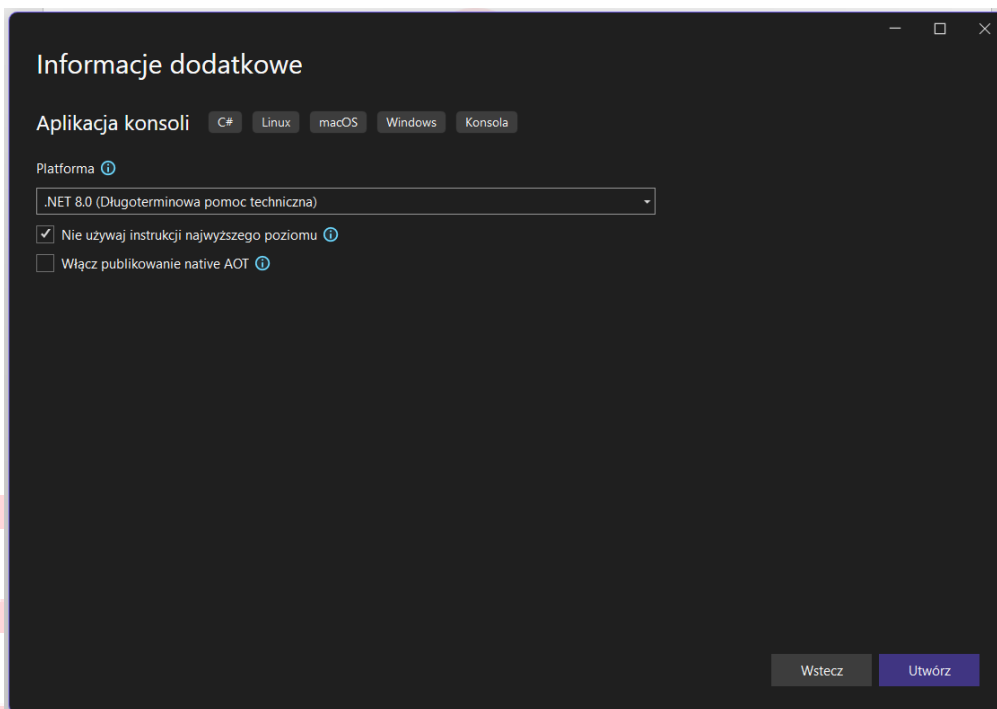
Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską

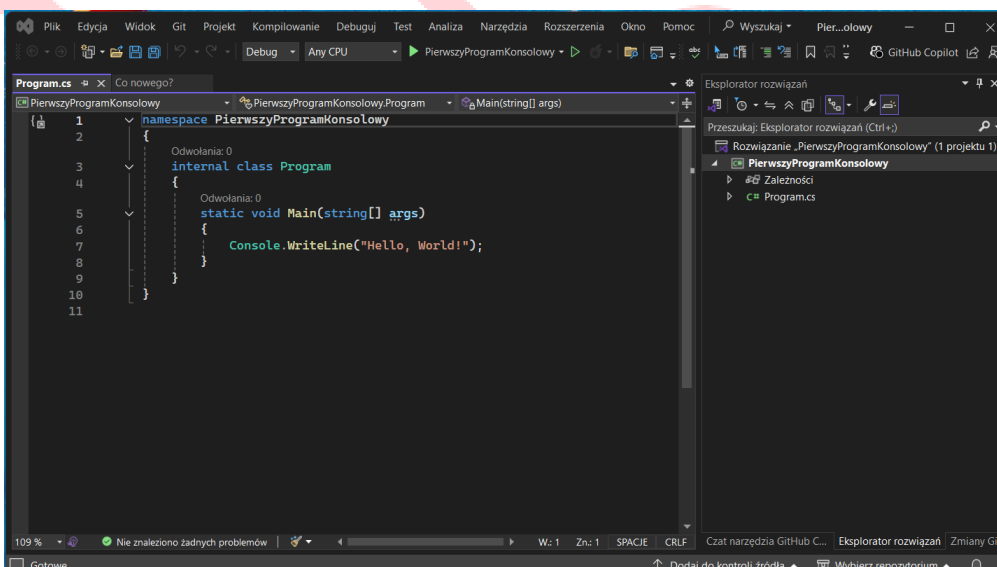


b) Zaznaczona opcja „Nie używaj instrukcji najwyższego poziomu”.

- Po wybraniu platformy kliknij **Utwórz**. Visual Studio wygeneruje klasyczny kod programu, z pełną strukturą.



- Podobnie jak wcześniej, po utworzeniu projektu, Visual Studio otworzy plik *Program.cs*. Znajduje się w nim domyślny kod:



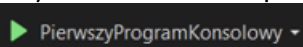
Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



- Program uruchamiamy w ten sam sposób jak wcześniej, poprzez kliknięcie przycisku  lub wciśnięcie **Ctrl + F5**.

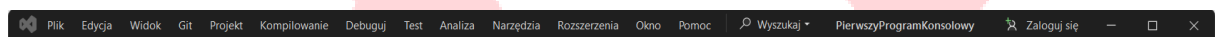
Powyższy program zaczyna się od deklaracji przestrzeni nazw `PierwszyProgramKonsolowy`, która porządkuje kod i pozwala unikać konfliktów nazw. W jej obrębie znajduje się klasa `Program`. Klasa ta zawiera jedną metodę `Main`, oznaczoną modyfikatorem `static`. To właśnie `Main` stanowi punkt wejścia programu w C#. Metoda przyjmuje tablicę argumentów `string[] args`, co pozwala przekazywać dane do programu podczas jego uruchamiania. Wewnątrz `Main` znajduje się wywołanie `Console.WriteLine("Hello, World!");`, które korzysta z klasy `Console` z przestrzeni nazw `System` i wypisuje tekst "Hello, World!" w oknie konsoli.

III. Opis interfejsu środowiska programistycznego Visual Studio

Na potrzeby niniejszego skryptu korzystamy z polskiej wersji interfejsu Visual Studio. W praktyce jednak w branży IT standardem jest używanie angielskiej wersji, ponieważ większość dokumentacji odnosi się do angielskich nazw, a w zespołach programistycznych dominuje angielska terminologia.

Po uruchomieniu programu Visual Studio wyświetlane jest główne okno środowiska programistycznego. Wygląd może się różnić w zależności od wersji i ustawionego układu. Składa się ono z kilku podstawowych części:

1. Na górze okna znajduje się **pasek menu** z następującymi opcjami: *Plik, Edycja, Widok, Projekt, Kompilowanie, Debuguj, Test, Analiza, Narzędzia, Rozszerzenia, Okno, Pomoc, Wyszukaj*, nazwa aktualnie otwartego rozwiązania, *Zaloguj się* oraz przyciski systemowe (minimalizacja, maksymalizacja, zamknięcie okna).



2. Pod menu znajduje się standardowy **pasek narzędzi**, który umożliwia szybki dostęp do najczęściej używanych poleceń. Zawiera m.in.: przyciski nawigacji między oknami, *Nowy projekt, Otwórz plik, Zapisz, Zapisz wszystko, Cofnij, Wykonaj ponownie*, listę rozwijaną *Konfiguracje rozwiązania* (*Debug, Release*), *Platformy rozwiązania* (*Any CPU, x86, x64*), *Rozpocznij debugowanie, Uruchom bez debugowania, Przeładowywanie na gorąco, Znajdź w plikach* oraz *Uruchom okno*.

Standardowy pasek narzędzi można dostosować, np. ukryć wybrane przyciski lub dodać inne korzystając z menu *Widok – Paski narzędzi*.



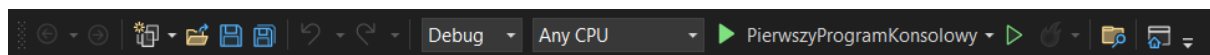
Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską





3. Po prawej stronie zwykle znajduje się **Eksplorator rozwiązań**, w którym wyświetlana jest struktura rozwiązania, czyli projekty, pliki źródłowe, zasoby i referencje.
4. Po lewej stronie umieszczony jest **Przybornik**, który zawiera gotowe elementy do aplikacji okienkowych.
5. Pod Eksploratorem rozwiązań znajduje się okno **Właściwości**, które pokazuje szczegółowe ustawienia wybranego elementu (np. przycisku w aplikacji okienkowej, pliku lub projektu).
6. Centralną część okna zajmuje **edytor kodu**. To w nim programista pisze kod źródłowy. Edytor oferuje kolorowanie składni, podpowiedzi IntelliSense oraz możliwość pracy na wielu otwartych plikach jednocześnie.
7. Na dole okna znajdują się zakładki takie jak:
 - *Dane wyjściowe* – komunikaty kompilatora i narzędzi,
 - *Lista błędów* – zestawienie błędów i ostrzeżeń w projekcie,
 - *Terminal* – wbudowany wiersz poleceń.
8. W trybie debugowania pojawiają się dodatkowe okna: *Lokalne*, *Stos wywołań*, *Okno bezpośrednie*.

IV. Publikacja aplikacji na różne platformy

W .NET jedną aplikację konsolową można przygotować tak, aby działała na różnych systemach operacyjnych. Przy publikacji należy zwrócić uwagę na dwa kluczowe elementy:

- **Tryb wdrożenia:**
 - *zależne od platformy* – generowane pliki są mniejsze, ale wymagają zainstalowanego środowiska .NET na komputerze użytkownika,
 - *samodzielny* – aplikacja zawiera dołączony runtime, dzięki czemu działa bez instalacji .NET, ale zajmuje więcej miejsca.
- **Docelowe środowisko uruchomieniowe**, np. win-x64, linux-x64, linux-arm64, osx-x64, osx-arm64.

Przykładowy przebieg procesu publikacji aplikacji w środowisku Visual Studio został przedstawiony w kolejnych krokach.



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

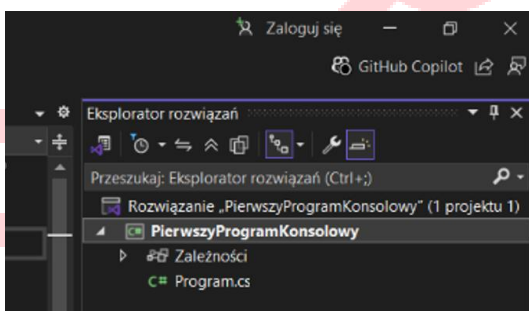
Dofinansowane przez
Unię Europejską



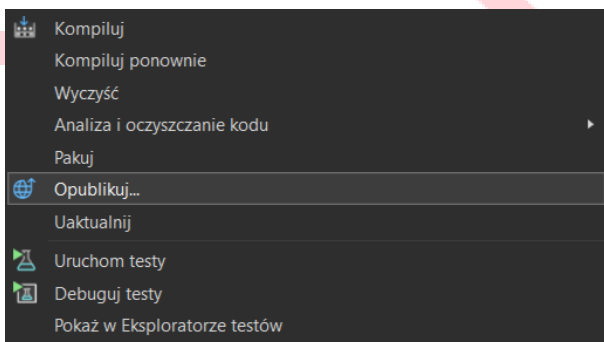
1. Otwórz projekt *PierwszyProgramKonsolowy*. W pliku *Program.cs*, w metodzie *Main()*, po instrukcji wyświetlającej tekst „*Hello, World!*” dodaj polecenie `Console.ReadKey();`. Dzięki temu okno konsoli nie zamknie się automatycznie, a pozostanie otwarte do momentu wciśnięcia dowolnego klawisza przez użytkownika.
2. Upewnij się, że w pasku narzędzi wybrana jest konfiguracja *Release*. Dzięki temu aplikacja zostanie skompilowana w zoptymalizowanej wersji, gotowej do udostępnienia.



3. W *Eksploratorze rozwiązań* kliknij prawym przyciskiem myszy nazwę projektu (nie nazwę rozwiązania) w tym przypadku *PierwszyProgramKonsolowy*.



4. Z menu kontekstowego wybierz opcję **Opublikuj**.



5. Pojawi się nowe okno, w którym należy utworzyć profil publikowania. Wybierz opcję **Folder** i kliknij przycisk **Dalej**.



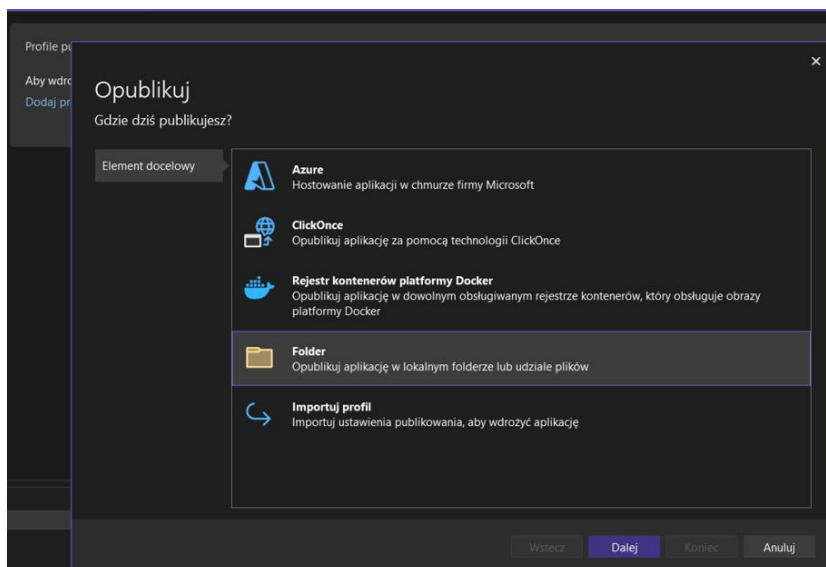
Fundusze Europejskie
dla Rozwoju Społecznego



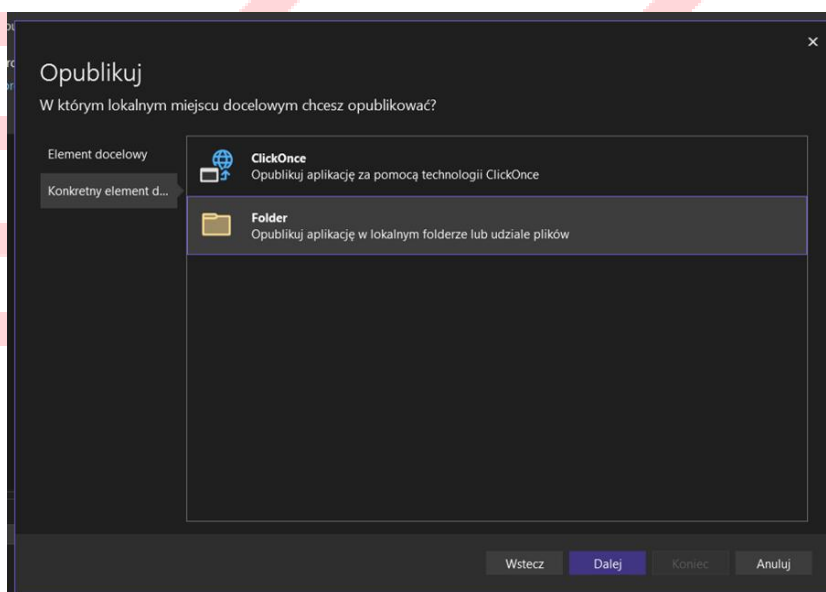
Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską





W kolejnym kroku ponownie wybierz **Folder** i zatwierdź przyciskiem **Dalej**.



- Określ lokalizację, w której mają zostać umieszczone pliki po opublikowaniu, a następnie kliknij **Koniec**.



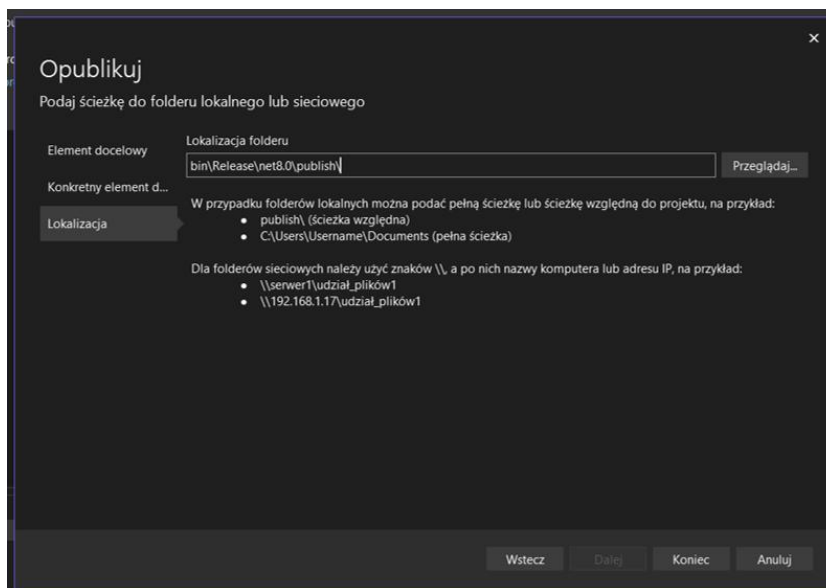
Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

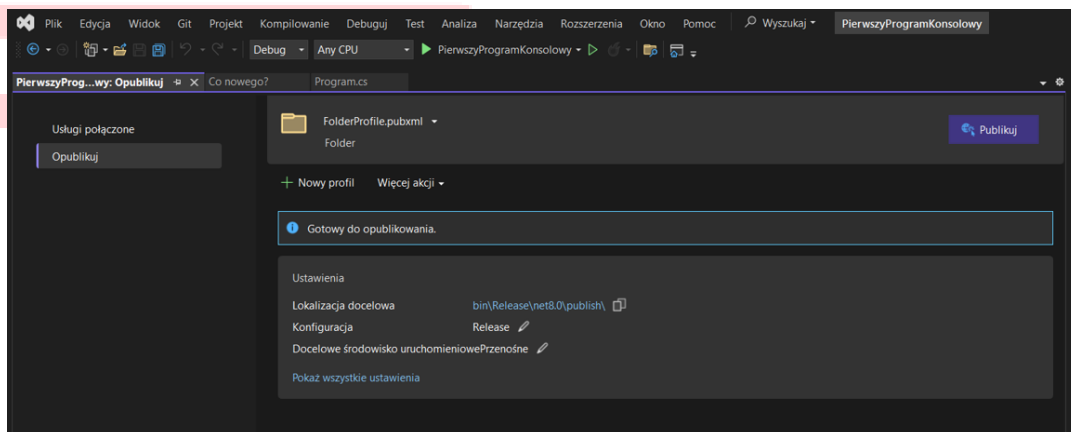
Dofinansowane przez
Unię Europejską





Jeśli profil został poprawnie utworzony, pojawi się komunikat: Utworzono profil publikowania „C:\PierwszyProgramKonsolowy\Properties\Publish Profiles\FolderProfile.pubxml”. Kliknij przycisk **Zamknij**.

7. W projekcie pojawi się nowy element **FolderProfile**. Kliknij **Pokaż wszystkie ustawienia**, aby edytować profil.



W oknie **Ustawienia profilu** możesz określić m.in. docelową platformę (Windows, Linux, macOS) oraz tryb działania (zależny od platformy lub samodzielny).



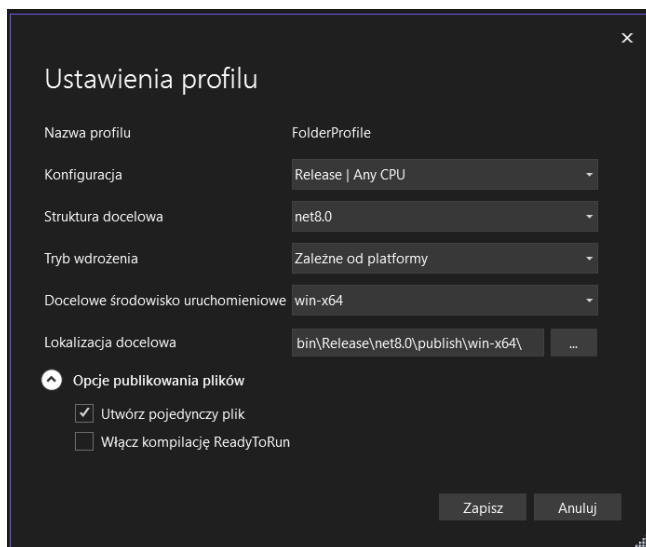
Fundusze Europejskie
dla Rozwoju Społecznego

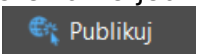


Rzeczpospolita
Polska

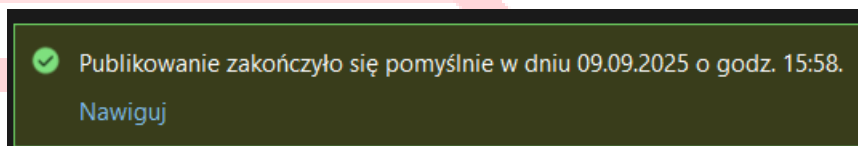
Dofinansowane przez
Unię Europejską



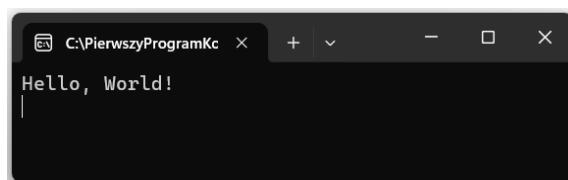


8. Wybierz opcję **win-x64** jako *Docelowe środowisko uruchomieniowe*. Przy tym pojawiają się dodatkowe opcje publikowania plików, m. in. **Utwórz pojedynczy plik** (generowanie jednego pliku wykonywalnego). Po zapisaniu zmian kliknij przycisk , aby wygenerować aplikację zgodnie z ustawionymi parametrami.

9. Po pomyślnym zakończeniu procesu publikacji pojawi się odpowiedni komunikat.



10. Kliknij **Nawiguj**, aby otworzyć folder z utworzonym plikiem wykonywalnym *PierwszyProgramKonsolowy.exe*.
11. Uruchom aplikację *PierwszyProgramKonsolowy.exe*. Na ekranie pojawi się okno konsoli.



Dzięki poleceniu `Console.ReadKey()`; program zakończy się dopiero po wciśnięciu dowolnego klawisza.



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



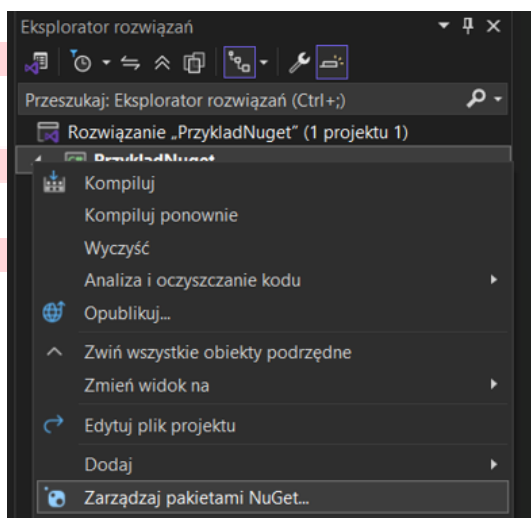
Uwaga! Aby przygotować aplikację dla innego systemu operacyjnego, należy utworzyć osobny profil publikowania i w jego ustawieniach zamiast **win-x64** wskazać odpowiedni identyfikator platformy, np. **linux-x64** lub **osx-x64**.

V. Menedżer pakietów NuGet

NuGet to oficjalny menedżer pakietów dla platformy .NET. Pozwala w prosty sposób dodawać do projektu gotowe komponenty, które ktoś wcześniej przygotował i udostępnił. NuGet automatycznie pobiera i konfiguruje wszystkie zależności, czyli inne pakiety, od których wybrana biblioteka jest uzależniona. Można go używać zarówno z poziomu interfejsu Visual Studio (Menedżer pakietów NuGet), jak i konsoli (np. Install-Package albo dotnet add package).

Przykładowy przebieg instalacji pakietu i jego użycia w projekcie został przedstawiony w kolejnych krokach.

1. Utwórz nowy projekt o nazwie *PrzykladNuget*.
2. W Eksploratorze rozwiązań kliknij prawym przyciskiem myszy na nazwę projektu *PrzykladNuget*, wybierz opcję **Zarządzaj pakietami NuGet....**



3. Przejdź do zakładki **Przeglądaj** i wpisz w wyszukiwarce „Humanizer”.



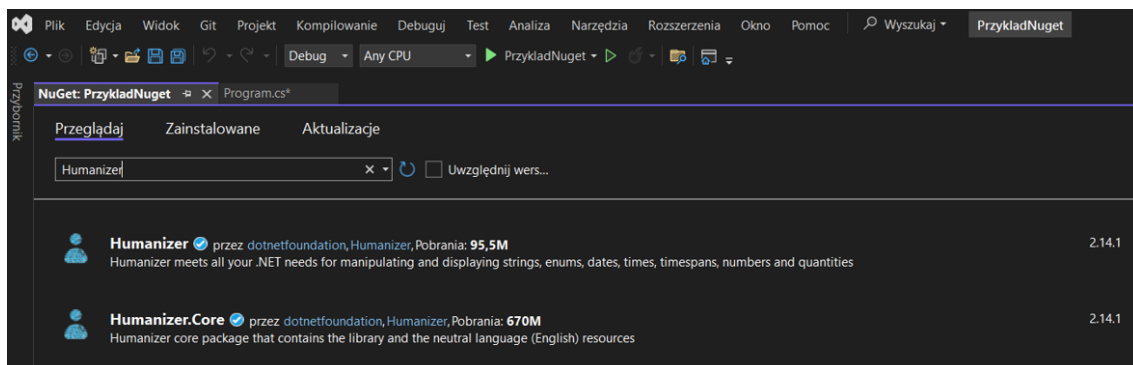
Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



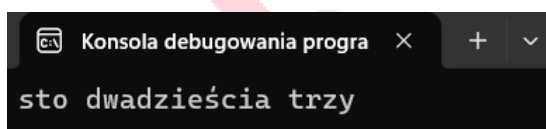


4. Wybierz pakiet **Humanizer.Core.pl** i kliknij **Zainstaluj**. Visual Studio pobierze pakiet i doda go do projektu.
5. Otwórz **Program.cs** i zmień jego zawartość na:

```
using System;
using Humanizer;

namespace PrzykladNuget
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine(123.ToWords()); // Liczby jako słowa
        }
    }
}
```

6. Uruchom program. W wyniku powinien pojawić się następujący napis:



VI. Zmienne i typy danych. Operatory arytmetyczne i logiczne

Przykład 1.1.

W tym przykładzie zadeklarowano i zainicjowano zmienne różnych typów: całkowitych, zmiennoprzecinkowych, dziesiętnych, tekstowych, znakowych i logicznych. Następnie ich wartości zostały wypisane w konsoli za pomocą `Console.WriteLine()`. Dodatkowo zaprezentowano różne sposoby zapisu liczb: w systemie binarnym, szesnastkowym oraz w notacji naukowej. Pokazano też użycie



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



literałów liczbowych z sufiksami (u, L, M, f) oraz możliwość stosowania podkreśleń w dużych liczbach dla poprawy czytelności.

```
int ilosc = 205;
double temperatura = 18.5;
decimal kwota = 200.50M;
float piF = 3.14159f;
char znak = 'C';
string napis = "Podstawy programowania";
bool czyOK = true;
Console.WriteLine("Liczba (int): " + ilosc);
Console.WriteLine("Liczba (double): " + temperatura);
Console.WriteLine("Liczba (decimal): " + kwota);
Console.WriteLine("Liczba (float): " + piF);
Console.WriteLine("Znak: " + znak);
Console.WriteLine("Napis: " + napis);
Console.WriteLine("Wartość logiczna: " + czyOK);

int bin = 0b_0011_1010;
uint hex = 0x3A;
double avogadro = 6.022e23;
uint bezZnaku = 42u;
long duzaLiczba = 9_223_372_036_854_775_807L;
Console.WriteLine("Binarnie: " + bin);
Console.WriteLine("Szesnastkowo: " + hex);
Console.WriteLine("Notacja naukowa: " + avogadro);
Console.WriteLine("uint: " + bezZnaku);
Console.WriteLine("long: " + duzaLiczba);
```

Dane wyjściowe:

```
Liczba (int): 205
Liczba (double): 18,5
Liczba (decimal): 200,50
Liczba (float): 3,14159
Znak: C
Napis: Podstawy programowania
Wartość logiczna: True
Dziesiętnie: 58
Binarnie: 58
Szesnastkowo: 58
Notacja naukowa: 6,022E+23
uint: 42
long: 9223372036854775807
```



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Stałe są przydatne, gdy wartość nie powinna być modyfikowana. W C# deklarujemy je za pomocą słowa kluczowego `const`.

```
const int Miesiace = 12;  
Console.WriteLine("Liczba miesięcy w roku: " + Miesiace);
```

Dane wyjściowe:

```
Liczba miesięcy w roku: 12
```

W przypadku, gdy chcemy skorzystać z dokładniejszej reprezentacji stałych matematycznych, możemy użyć wbudowanych stałych z biblioteki `Math`.

```
double r = 1.5;  
double obwod = 2 * Math.PI * r;  
Console.WriteLine("Obwód koła: " + obwod);  
Console.WriteLine("Stała Pi: " + Math.PI);  
Console.WriteLine("Stała Eulera: " + Math.E);
```

Dane wyjściowe:

```
Obwód koła: 9,42477796076938  
Stała Pi: 3,141592653589793  
Stała Eulera: 2,718281828459045
```

Przykład 1.2.

W tym przykładzie pokazano literały znakowe, napisowe oraz sekwencje ucieczki (`\`, `\n`, `\\`, `\t`).

```
char apostrof = '\\';  
char nowaLinia = '\\n';  
char backslash = '\\\\';  
  
string klasyczny = "Ala\\nma\\tkota";  
string sciezka = @"C:\\Users\\student\\Dokumenty\\plik.txt"; //dosłowny  
string wielowierszowy = ""  
Linia 1  
"cytat" w środku  
Linia 3  
""; // string – wielowierszowy  
  
Console.WriteLine(apostrof);  
Console.WriteLine(nowaLinia);  
Console.WriteLine(backslash);  
Console.WriteLine(klasyczny);  
Console.WriteLine(sciezka);  
Console.WriteLine(wielowierszowy);
```



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Dane wyjściowe:

```
'  
  
\  
Ala  
ma      kota  
C:\Users\student\Dokumenty\plik.txt  
Linia 1  
"cytat" w środku  
Linia 3
```

Przykład 1.3.

W tym przykładzie pokazano sekwencje ucieczki Unicode (\u i \U) użyte w literałach napisowych.

```
using System.Text;  
  
Console.OutputEncoding = Encoding.UTF8;  
  
string polski = "Zażółć gęślą jaźń";  
string strzałka = "\u2192";  
string kciuk = "\U0001F44D";  
  
Console.WriteLine(polski);  
Console.WriteLine(strzałka);  
Console.WriteLine(kciuk);
```

Dane wyjściowe:

```
Zażółć gęślą jaźń  
→  
👍
```

Przykład 1.4.

W tym przykładzie pokazano użycie klasy Convert do zamiany tekstu na liczby całkowite i rzeczywiste. Dla wartości null metoda Convert.ToInt32 zwraca 0, a metoda Convert.ToDouble pozwala przekształcić dane na typ double. Dodatkowo zaprezentowano metodę Convert.ToString, która umożliwia konwersję liczb na tekst, a także wyświetlanie ich w różnych systemach liczbowych (np. binarnym czy szesnastkowym).



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



```
int calkowita = Convert.ToInt32("123");
int zero = Convert.ToInt32(null);
Console.WriteLine("Całkowita: " + calkowita + ", zero: " + zero);

Console.Write("Podaj liczbę rzeczywistą: ");
double rzeczywista = Convert.ToDouble(Console.ReadLine());
Console.WriteLine("Wczytana liczba rzeczywista: " + rzeczywista);

int liczba = 54;
string tekst = Convert.ToString(liczba);
Console.WriteLine("Liczba jako tekst: " + tekst);

string binarnie = Convert.ToString(liczba, 2);
string szesnastkowo = Convert.ToString(liczba, 16);
Console.WriteLine(liczba + " binarnie: " + binarnie);
Console.WriteLine(liczba + " szesnastkowo: " + szesnastkowo);
```

Dane wyjściowe:

```
Całkowita: 123, zero: 0
Podaj liczbę rzeczywistą: 15,5
Wczytana liczba rzeczywista: 15,5
Liczba jako tekst: 54
54 binarnie: 110110
54 szesnastkowo: 36
```

Przykład 1.5.

W tym przykładzie przedstawiono rzutowanie i konwersję typów danych. Jawne rzutowanie z double na int powoduje utratę części ułamkowej, rzutowanie niejawne z int na double odbywa się automatycznie i bez utraty informacji, natomiast metoda Convert.ToInt32 wykonuje konwersję z zaokrągleniem (tzw. zaokrąglenie bankierskie).

```
double d1 = 9.78;
int i1 = (int)d1; // jawne rzutowanie - utrata części ułamkowej
Console.WriteLine(i1);

int i2 = 42;
double d2 = i2; // niejawne rzutowanie int na double
Console.WriteLine(d2);

int i3 = Convert.ToInt32(d1); // konwersja - zaokr. bankierskie
Console.WriteLine(i3);
```



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Dane wyjściowe:

```
9
42
10
```

Przykład 1.6.

Poniższy kod pokazuje użycie operatorów arytmetycznych oraz logicznych.

```
int a = 8, b = 3, c = 5;

Console.WriteLine("Dodawanie: " + (a + b));
Console.WriteLine("Odejmowanie: " + (a - b));
Console.WriteLine("Mnożenie: " + (a * b));
Console.WriteLine("Dzielenie (całkowite): " + (a / b));
Console.WriteLine("Dzielenie (rzeczywiste): " + (double)a / b);
Console.WriteLine("Reszta z dzielenia: " + (a % b));

c += 2;
Console.WriteLine("Wartość c po zwiększeniu o 2: " + c);
c++;
Console.WriteLine("Wartość c po zwiększeniu o 1: " + c);

int wiek = 22;
bool student = true;
bool zniżka = (wiek < 26) && student;
bool dostep = (wiek >= 18) || student;

Console.WriteLine("Czy przysługuje zniżka? " + zniżka);
Console.WriteLine("Czy przysługuje dostęp do systemu? " + dostep);
```

Dane wyjściowe:

```
Dodawanie: 11
Odejmowanie: 5
Mnożenie: 24
Dzielenie (całkowite): 2
Dzielenie (rzeczywiste): 2,6666666666666665
Reszta z dzielenia: 2
Wartość c po zwiększeniu o 2: 7
Wartość c po zwiększeniu o 1: 8
Czy przysługuje zniżka? True
Czy przysługuje dostęp do systemu? True
```



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Przykład 1.7.

Domyślnie elementy enum mają typ `int`, zaczynają się od 0 i rosną o 1. Poniższy kod definiuje typ wyliczeniowy `DniTygodnia`, przypisuje mu wartość i wypisuje ją na ekranie.

```
class Program
{
    enum DniTygodnia { Pon, Wt, Sr, Czw, Pt, Sob, Nd }

    private static void Main(string[] args)
    {
        DniTygodnia dzien = DniTygodnia.Pt;
        Console.WriteLine(dzien);
        Console.WriteLine((int)dzien);
        // Można też rzutować w drugą stronę
        int numerDnia = 0;
        DniTygodnia innyDzien = (DniTygodnia)numerDnia;
        string nazwaDnia = innyDzien.ToString();
        Console.WriteLine(nazwaDnia);
    }
}
```

Dane wyjściowe:

```
Pt
4
Pon
```

Można jawnie wybrać inny typ całkowity oraz przypisać własne wartości:

```
class Program
{
    enum DniTygodnia : byte { Pon = 1, Wt, Sr, Czw, Pt }

    private static void Main(string[] args)
    {
        DniTygodnia dzien = DniTygodnia.Pt;
        Console.WriteLine(dzien);
        Console.WriteLine((int)dzien);
    }
}
```

Dane wyjściowe:

```
Pt
5
```



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Dodatkowo można użyć atrybutu `Description` [z przestrzeni nazw `System.ComponentModel`]. Dzięki metodzie rozszerzającej `GetDescription` można uzyskać opis enuma (np. „Poniedziałek”) zamiast jego nazwy („Pon”).

```
using System.ComponentModel;

enum DniTygodnia
{
    [Description("Poniedziałek")]
    Pon,
    [Description("Wtorek")]
    Wt,
    [Description("Środa")]
    Sr,
    [Description("Czwartek")]
    Czw,
    [Description("Piątek")]
    Pt,
    [Description("Sobota")]
    Sob,
    [Description("Niedziela")]
    Nd
}

class Program
{
    public static string GetDescription(Enum value)
    {
        var field = value.GetType().GetField(value.ToString());
        var attr = Attribute.GetCustomAttribute(field,
        typeof(DescriptionAttribute)) as DescriptionAttribute;
        if (attr != null && attr.Description != null)
            return attr.Description;
        else
            return value.ToString();
    }

    private static void Main(string[] args)
    {
        DniTygodnia dzien = DniTygodnia.Pon;
        Console.WriteLine(dzien);
        Console.WriteLine(GetDescription(dzien));
    }
}
```



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Dane wyjściowe:

Pon
Poniedziałek

Przykład 1.8.

Poniższy kod pokazuje trzy metody łączenia tekstu w C#:

- konkatencję za pomocą operatora +,
- interpolację stringów, czyli wstawianie wartości zmiennych bezpośrednio do tekstu w nawiasach {},
- formatowanie z indeksami [{0}, {1}, ...], w którym zmienne przekazuje się jako kolejne argumenty metody.

```
string imie = "Katarzyna";  
int punkty = 95;  
  
// Konkatenacja  
Console.WriteLine("Brawo, " + imie + "! Masz " + punkty + " pkt.");  
  
// Interpolacja stringów  
Console.WriteLine($"Brawo, {imie}! Masz {punkty} pkt.");  
  
// Formatowanie z indeksami  
Console.WriteLine("Brawo, {0}! Masz {1} pkt.", imie, punkty);
```

Dane wyjściowe:

Brawo, Katarzyna! Masz 95 pkt.
Brawo, Katarzyna! Masz 95 pkt.
Brawo, Katarzyna! Masz 95 pkt.

Przykład 1.9.

W interpolacji stringów możemy użyć specjalnych formatów {wartość:FORMAT}:

- F2 – liczba zostanie wyświetlona z dokładnością do dwóch miejsc po przecinku,
- N2 – działa podobnie jak F2, ale z separacją tysięcy,
- C – waluta z symbolem i regułami danej kultury (domyślnie 2 miejsca po przecinku).
- P1 – procenty (wartość mnożona przez 100 ze znakiem '%').



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



```
using System.Globalization;

double cena = 503145.99;
int ilosc = 3;
double suma = cena * ilosc;
// F2 - dwa miejsca po przecinku (wg bieżącej kultury)
Console.WriteLine($"Suma (F2): {suma:F2} zł");

// N2 - liczba z separatorem tysięcy i 2 miejscami
Console.WriteLine($"Suma (N2): {suma:N2} zł");

// C - waluta (można wymusić kulturę, np. polską)
var pl = new CultureInfo("pl-PL");
Console.WriteLine($"Suma (C): {suma.ToString("C", pl)}");

// P1 - procenty (jedno miejsce po przecinku)
double odsetki = 0.1234;
Console.WriteLine($"Odsetki: {odsetki:P1}"); // 12,3%

// Wyrównanie: szerokość 15 znaków, 2 miejsca po przecinku
Console.WriteLine($"Suma (wyrównanie): |{suma,15:F2}|");
```

Dane wyjściowe:

```
Suma (F2): 1509437,97 zł
Suma (N2): 1 509 437,97 zł
Suma (C): 1 509 437,97 zł
Odsetki: 12,3%
Suma (wyrównanie): |      1509437,97|
```

Przykład 1.10.

W C# do podstawowych obliczeń matematycznych służy wbudowana klasa Math. Zawiera ona wiele przydatnych metod, m.in. pierwiastkowanie, potęgowanie, porównywanie wartości czy zaokrąglanie.

```
Console.WriteLine($"Pierwiastek z 25: {Math.Sqrt(25)}");
Console.WriteLine($"2^4: {Math.Pow(2, 4)}");
Console.WriteLine($"Maksimum z 10 i 20: {Math.Max(10, 20)}");

Console.WriteLine($"Zaokrąglanie (1): {Math.Round(3.5)}");
Console.WriteLine($"Zaokrąglanie (2): {Math.Round(3.14159, 2)}");

Console.WriteLine($"Zaokrąglanie (3): {Math.Round(3.5, 0, MidpointRounding.AwayFromZero)}");
```



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



```
Console.WriteLine($"Zaokrąglanie (4): {Math.Round(2.5, 0,  
MidpointRounding.ToZero)}");
```

Dane wyjściowe:

```
Pierwiastek z 25: 5  
2^4: 16  
Maksimum z 10 i 20: 20  
Zaokrąglanie (1): 4  
Zaokrąglanie (2): 3,14  
Zaokrąglanie (3): 4  
Zaokrąglanie (4): 2
```

Przykład 1.11.

W C# często używa się słowa kluczowego `var`, aby kompilator sam rozpoznał typ zmiennej na podstawie przypisanej wartości. To nie oznacza, że zmienna nie ma typu, jest on wywnioskowany automatycznie.

```
var liczba = 12; //int  
var tekst = "C#"; //string  
Console.WriteLine($"liczba {liczba.GetType().Name} = {liczba}");  
Console.WriteLine($"tekst {tekst.GetType().Name} = {tekst}");
```

Dane wyjściowe:

```
liczba Int32 = 12  
tekst String = C#
```

Przykład 1.12.

Poniższy kod pokazuje, jak pobrać aktualną datę i godzinę, utworzyć własną datę oraz wykonywać proste operacje na datach.

```
DateTime dzis = DateTime.Now; // aktualna data i godzina  
Console.WriteLine($"Aktualna data: {dzis}");  
Console.WriteLine(dzis.ToString("d")); // krótka data  
Console.WriteLine(dzis.ToString("D")); // długa data  
Console.WriteLine(dzis.ToString("g")); // data i czas  
  
DateTime innaData = new DateTime(2000, 5, 10);  
Console.WriteLine($"Inna data: {innaData.ToShortDateString()}");  
//Lub za pomocą wybranego formatowania dd.MM.yyyy  
Console.WriteLine($"Inna data: {innaData:dd.MM.yyyy}");  
  
DateTime jutro = dzis.AddDays(1);  
Console.WriteLine($"Jutrzejsza data: {jutro}");  
Console.WriteLine($"Jutrzejsza data: {jutro:dd.MM.yyyy HH:mm}");
```



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Dane wyjściowe:

```
Aktualna data: 25.09.2025 13:00:17
25.09.2025
czwartek, 25 września 2025
25.09.2025 13:00
Inna data: 10.05.2000
Inna data: 10.05.2000
Jutrzejsza data: 26.09.2025 13:00:17
Jutrzejsza data: 26.09.2025 13:00
```

Innym sposobem na przesunięcie daty o określoną liczbę dni (godzin, minut itp.) jest wykorzystanie struktury `TimeSpan`.

```
//Dołączamy ten fragment do wcześniejszego kodu
TimeSpan liczbaDni = new TimeSpan(5, 3, 0, 0);
DateTime nowaData = dzis.Add(liczbaDni);
Console.WriteLine($"Nowa data: {nowaData}");
```

Dane wyjściowe:

```
Nowa data: 30.09.2025 16:03:50
```

Przykład 1.13.

W tym przykładzie przedstawiono sposób wczytywania danych z klawiatury. Metoda `Console.ReadLine()` pozwala odczytać wprowadzony tekst jako ciąg znaków. W przypadku zmiennej imię wynik jest przypisywany bezpośrednio do typu `string`. Każdy typ liczbowy udostępnia metodę `Parse()`, która umożliwia konwersję z łańcuchów znaków na dany typ. Aby odczytać liczbę całkowitą, wczytany tekst został przekształcony na typ `int` za pomocą metody `int.Parse()`.

```
Console.Write("Podaj imię: ");
string? imię = Console.ReadLine(); //wczytanie tekstu
Console.Write("Podaj wiek: ");
int wiek = int.Parse(Console.ReadLine()); //wczytanie liczby
Console.WriteLine($"Cześć, {imię}! Masz {wiek} lat.");
```

Dane wyjściowe:

```
Podaj imię: Paulina
Podaj wiek: 21
Cześć, Paulina! Masz 21 lat.
```

W powyższym kodzie `int.Parse()` zgłosi wyjątek jeżeli użytkownik wpisze coś, co nie jest liczbą. Aby temu zapobiec warto skorzystać z metody `TryParse()`.

```
Console.Write("Podaj imię: ");
```



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



```
string? imie = Console.ReadLine();

Console.Write("Podaj wiek: ");
if (int.TryParse(Console.ReadLine(), out int wiek))
{
    Console.WriteLine($"Cześć, {imie}! Masz {wiek} lat.");
}
else
{
    Console.WriteLine("Podany wiek nie jest liczbą całkowitą.");
}
```

Dane wyjściowe:

Podaj imię: Marcin
Podaj wiek: 22
Cześć, Marcin! Masz 22 lat.

Dane wyjściowe (z błędem przy wprowadzaniu wieku):

Podaj imię: Anna
Podaj wiek: a
Podany wiek nie jest liczbą całkowitą.

Ćwiczenia samodzielne

Zadanie 1.1.

Napisz program, który obliczy obwód i pole trójkąta o bokach 6,3, 6,7 oraz 7,2. Do obliczenia pola zastosuj wzór Herona. Wyświetl otrzymane wyniki w konsoli z dokładnością do dwóch miejsc po przecinku (:F2).

Wskazówka: do obliczenia pierwiastka kwadratowego użyj funkcji `Math.Sqrt`.

Zadanie 1.2.

Napisz program, który wczyta od użytkownika dwie liczby całkowite. Następnie, korzystając z metody `Math.Max`, wyznacz większą z tych liczb i wypisz ją w konsoli.

Zadanie 1.3.

Napisz program, który wczyta od użytkownika liczbę całkowitą, a następnie wypisze w konsoli jej kwadrat i sześcian.

Wskazówka: do obliczeń możesz użyć mnożenia lub funkcji `Math.Pow(p, w)`, gdzie `p` to podstawa, a `w` – wykładnik.



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Zadanie 1.4.

Napisz program, który wyznaczy obwód i pole prostokąta. Długości boków a oraz b powinny być podane przez użytkownika z klawiatury. Ponadto w rozwiązaniu należy założyć, że zmienne a , b , obwód oraz pole mają typ `double` (rzeczywisty). Wyświetl otrzymane wyniki w konsoli z dokładnością do dwóch miejsc po przecinku (`%.2f`).

Zadanie 1.5.

Napisz program, który wykona następujące obliczenia:

- Liczbę 3456 zwiększy o 5.
- Następnie podzieli powyższą sumę przez 3.
- Obliczy resztę z dzielenia ilorazu przez 5.
- Na koniec wymnoży wynik z dzielenia modulo razy 7.

Po wykonaniu każdego kroku program powinien wypisać w konsoli bieżący wynik.

Zadanie 1.6.

Rozwiąż *zadanie 1.5.* używając operatorów przypisania (np. `+=`, `-=`, `*=`).

Zadanie 1.7.

Napisz program, który wczyta dwie liczby całkowite i obliczy ich sumę, różnicę, iloczyn, iloraz (dzielenie całkowite), resztę z dzielenia oraz średnią arytmetyczną. Program powinien wyświetlić w konsoli wynik każdego działania.

Zadanie 1.8.

Napisz program, który wczyta od użytkownika dowolny napis, a następnie wyświetli go w konsoli zapisany wielkimi literami.

Wskazówka: do zamiany liter na wielkie możesz użyć metody `ToUpper`.

Zadanie 1.9.

Napisz program, który wczyta od użytkownika rok urodzenia. Następnie program powinien określić:

- wiek użytkownika w roku bieżącym,
- ile lat minęło od 18 urodzin,
- wartość logiczną informującą, czy w bieżącym roku wiek jest „okrągły” (podzielny przez 10).



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Zadanie 1.10.

Napisz program, który wczytuje masę ciała m w kilogramach. Następnie wyznacza wartość siły ciężkości działającej na to ciało:

- a) na Ziemi [$g = 9,81 \text{ m/s}^2$],
- b) na Księżycu [$g = 1,62 \text{ m/s}^2$].

Wartość siły ciężkości oblicza się ze wzoru $F = m \cdot g$, gdzie m to masa ciała, zaś g to przyspieszenie ziemskie. Wyświetl otrzymane wartości w konsoli z dokładnością do dwóch miejsc po przecinku (:F2).

Zadanie 1.11.

Napisz program, który wczyta od użytkownika liczbę całkowitą. Następnie sprawdź, czy liczba ta jest parzysta czy nieparzysta. Wynik przedstaw w formie odpowiedniego komunikatu, np.: „Liczba 9 jest nieparzysta.”

Zadanie 1.12.

Napisz program, który wczyta od użytkownika temperaturę w stopniach Celsjusza i przeliczy ją na stopnie Fahrenheita według następującego wzoru:

$$F = \frac{9}{5}C + 32.$$

Zadanie 1.13.

Napisz program, który wczyta od użytkownika trzy znaki (char). Program powinien wypisać Unicode dla tych znaków, a następnie policzyć i wyświetlić średnią wartość tych kodów.

Zadanie 1.14.

Napisz program, który wczyta od użytkownika dwie liczby całkowite a i b . Następnie program powinien zamienić wartości tych liczb miejscami bez użycia dodatkowej zmiennej, wykorzystując kolejno:

- a) dodawanie i odejmowanie (+, -),
- b) mnożenie i dzielenie (*, /),
- c) operator bitowy XOR (^).

Po każdej zamianie program powinien wypisać wartości a i b .



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Zadanie 1.15.

Napisz program, który wczyta od użytkownika liczbę całkowitą. Następnie program powinien wypisać w konsoli:

- a) ostatnią cyfrę
- b) dwie ostatnie cyfry

tej liczby.

Zadanie 1.16.

Napisz program, który pobiera od użytkownika datę w formacie RRRR-MM-DD, a następnie wyznacza i wyświetla dzień tygodnia odpowiadający tej dacie.

Zadanie 1.17.

Napisz program, który pobiera bieżącą datę i godzinę przy użyciu klasy DateTime, a następnie wyświetla poszczególne właściwości tej daty: rok, miesiąc, dzień, dzień tygodnia, dzień roku, godzina, minuta, sekunda oraz milisekunda.

Zadanie 1.18.

Napisz program, w którym zostanie zdefiniowany typ wyliczeniowy:

```
enum Kierunek { Polnoc, Poludnie, Wschod, Zachod }
```

Program powinien wczytać z klawiatury numer kierunku [0 - 3], a następnie wypisać komunikat w postaci „*Idziesz na poludnie*”.

Pytania pomocnicze

- 1) Czym jest algorytm?
- 2) Jaka jest różnica między językiem kompilowanym a interpretowanym?
- 3) Jaką rolę pełni środowisko programistyczne (IDE)?
- 4) Co oznacza, że C# jest językiem wysokiego poziomu?
- 5) Jakie są podstawowe typy danych w języku C#?
- 6) W jaki sposób można sprawdzić, czy liczba jest parzysta?
- 7) Co oznacza pojęcie „zmienna” i jak zadeklarować ją w języku C#?
- 8) Czym różni się rzutowanie jawne od niejawnego?
- 9) Jakie są różnice między operatorami arytmetycznymi i logicznymi?



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Podsumowanie

W tym laboratorium przedstawiono podstawowe pojęcia związane z programowaniem w języku C#. Przedstawiono proces instalacji i konfiguracji środowiska Visual Studio oraz strukturę prostego programu konsolowego. Zaprezentowano zastosowanie zmiennych różnych typów oraz wykorzystanie operatorów arytmetycznych i logicznych.

Literatura

- [1] Michaelis M., *C# 8.0: kompletny przewodnik dla praktyków*, Helion, Gliwice 2021. s. 31-97.
- [2] Albahari J., *C# 12 w pigułce. Kompendium programisty*, Helion, Gliwice, 2024. s. 46-74.
- [3] <https://learn.microsoft.com/pl-pl/dotnet/csharp/>



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Materiały do przedmiotu:

Podstawy programowania

Laboratorium nr 4

**Definiowanie i wywoływanie funkcji, parametry,
wartości zwracane. Organizacja kodu w moduły**

Autor:

mgr Krystyna Kiersztyn



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Spis treści

Wprowadzenie, tematyka zajęć	37
Cel laboratorium	38
Instrukcja laboratoryjna/ćwiczeniowa	38
Ćwiczenia samodzielne	54
Pytania pomocnicze	59
Podsumowanie	59
Literatura	59



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Wprowadzenie, tematyka zajęć

Do tej pory wszystkie instrukcje programów umieszczane były w jednej metodzie `Main`. Takie rozwiązanie jest wystarczające dla prostych przykładów, ale w bardziej złożonych aplikacjach szybko prowadzi do powstania kodu trudnego do utrzymania i mało czytelnego. Rozwiązaniem jest podział programu na mniejsze fragmenty – metody.

Metoda to wydzielona sekwencja instrukcji realizująca określoną operację lub obliczająca wynik. Dzięki stosowaniu metod kod staje się uporządkowany, łatwiejszy do zrozumienia i wielokrotnego wykorzystania. Wydzielanie fragmentów programu do osobnych metod to forma refaktoryzacji, która zmniejsza powtarzalność i zwiększa przejrzystość programu.

Metody mogą przyjmować dane wejściowe w postaci parametrów. *Parametry* to zmienne używane wewnątrz metody, a wartości przekazywane do nich podczas wywołania to *argumenty*. Oprócz przyjmowania danych, metoda może także zwracać wynik obliczeń, którego typ określa się w sygnaturze metody, np. `int`, `string` czy `bool`. Jeżeli metoda nie zwraca żadnej wartości, stosuje się typ `void`. Od wersji C# 7.0 możliwe jest zwracanie wielu wartości jednocześnie za pomocą *krotek*. Sygnatura metody zawiera: modyfikator dostępu (np. `public`, `private`), ewentualny dodatkowy modyfikator (np. `static`, `virtual`), typ zwracanej wartości, nazwę metody oraz listę parametrów.

W języku C# parametry mogą być przekazywane na różne sposoby. Najczęściej stosuje się przekazywanie przez *wartość*. Wówczas metoda otrzymuje kopię zmiennej. Przekazywanie przez *referencję* (`ref`) umożliwia modyfikację wartości w miejscu wywołania. Zmienna musi być zainicjalizowana przed przekazaniem do metody. Parametry wyjściowe (`out`) również są przekazywane przez referencję, ale w odróżnieniu od `ref` muszą mieć przypisaną wartość wewnątrz metody przed jej zakończeniem. Umożliwiają metodzie zwrócenie wielu wartości. Parametr tylko do odczytu (`in`) jest przekazywany przez referencję, ale bez możliwości jego modyfikacji wewnątrz metody. Zmienna musi być zainicjalizowana przed przekazaniem do metody. Dostępne są także *parametry opcjonalne* (z ustaloną wartością *domyślną*), *argumenty nazwane*, a tablica parametrów `params` pozwala przekazać *zmienną liczbę argumentów*. Od wersji C# 6.0 można definiować *metody o ciele wyrażeniowym*, które umożliwiają zwięzłe definiowanie prostych operacji za pomocą operatora `=>`.



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Metody w C# zawsze są powiązane z typem (klasą) i organizowane w przestrzeń nazw. Przestrzeń nazw porządkują typy według obszarów funkcjonalnych, ułatwiają wyszukiwanie elementów i pozwalają uniknąć konfliktów nazw.

Cel laboratorium

Po zrealizowaniu laboratorium student:

- potrafi zdefiniować i wywołać metody,
- potrafi zwrócić pojedynczą wartość oraz wiele wartości przy użyciu krotek,
- rozróżnia przekazywanie parametrów przez wartość i przez referencję,
- stosuje parametry opcjonalne, nazwane oraz tablicowe (params),
- rozumie ideę przeciążania metod,
- wykorzystuje metody do rozwiązywania prostych problemów obliczeniowych i algorytmicznych,
- stosuje podział programu na mniejsze fragmenty w celu poprawy czytelności i ponownego użycia kodu.

Instrukcja laboratoryjna/ćwiczeniowa

Przykład 4.1.

W tym przykładzie przedstawiono metodę statyczną bez parametrów, która nie zwraca żadnej wartości (void), oraz sposób jej wywołania. Metoda wypisuje na ekranie komunikat powitalny.

```
class Program
{
    static void PrzywitajWszystkich()
    {
        Console.WriteLine("Witam wszystkich!");
    }

    private static void Main(string[] args)
    {
        // Wywołanie metody statycznej
        PrzywitajWszystkich();
    }
}
```



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Dane wyjściowe:

Witam wszystkich!

Uwaga! Słowo kluczowe `static` oznacza, że metoda [statyczna] należy do klasy, a nie do konkretnego obiektu. Można ją wywołać bez tworzenia obiektu klasy do Program.

Alternatywnym podejściem do tradycyjnej definicji metody z blokiem instrukcji w nawiasach klamrowych jest użycie *metody z ciałem w postaci wyrażenia*, czyli skróconego zapisu z wykorzystaniem operatora `=>`. Wówczas powyższą metodę `PrzywitajW` możemy zapisać w następujący sposób:

```
static void PrzywitajWszystkich() => Console.WriteLine("Witam  
wszystkich!");
```

Ten zapis jest równoważny z wcześniejszym, ale bardziej zwięzły i stosowany zazwyczaj wtedy, gdy metoda wykonuje tylko jedną prostą instrukcję.

Przykład 4.2.

W tym przykładzie pokazano metodę z jednym parametrem, bez wartości zwracanej (`void`).

```
class Program  
{  
    static void Przywitaj(string imie)  
    {  
        Console.WriteLine($"Cześć, {imie}!");  
    }  
  
    private static void Main(string[] args)  
    {  
        Przywitaj("Magda");  
    }  
}
```

Dane wyjściowe:

Cześć, Magda!

Visual Studio udostępnia wbudowane narzędzie refaktoryzacji, które pozwala szybko przekształcić zwykłą metodę na jej skrócony zapis.

1. Kliknij prawym przyciskiem myszy na nagłówek metody, np. na nazwę `{Przywitaj}`.



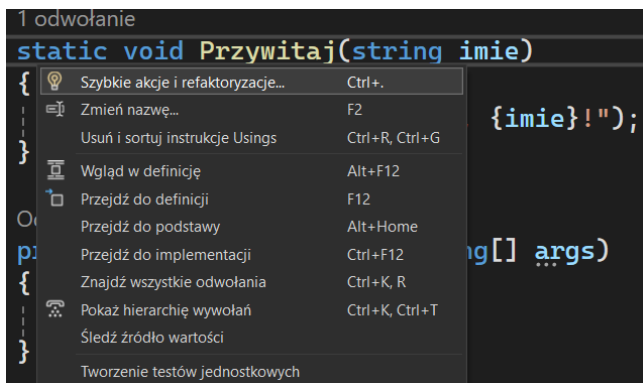
Fundusze Europejskie
dla Rozwoju Społecznego



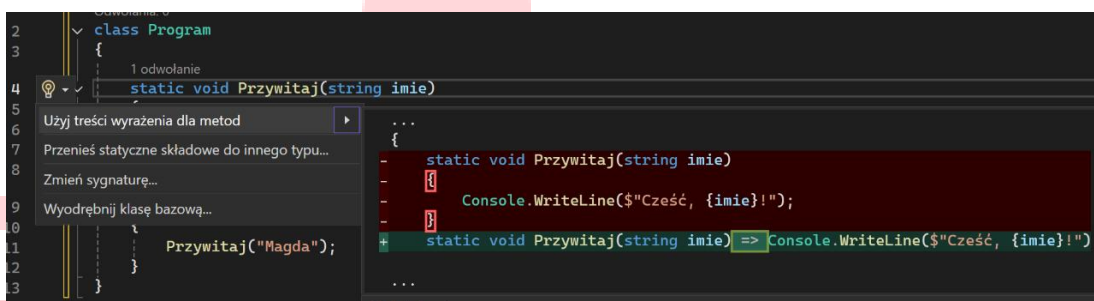
Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską





2. Wybierz opcję **Szybkie akcje i refaktoryzacje**.



3. W menu wybierz opcję **Użyj treści wyrażenia dla metod**. Visual Studio automatycznie zamieni metodę na skrócony zapis.

```
static void Przywitaj(string imie) => Console.WriteLine($"Cześć, {imie}!");
```

Przykład 4.3.

W tym przykładzie przedstawiono metodę z ciałem wyrażeniowym, która zwraca wartość. Metoda przyjmuje temperaturę w stopniach Celsjusza i zwraca jej odpowiednik w stopniach Fahrenheita.

```
class Program
{
    static double PrzeliczStopnie(double c) => (c * 9 / 5) + 32;
    static void Main(string[] args)
    {
        Console.WriteLine($"5°C = {PrzeliczStopnie(5)}°F");
    }
}
```

Dane wyjściowe:

```
5°C = 41°F
```



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Przykład 4.4.

W tym przykładzie zdefiniowano metodę, która nie przyjmuje żadnych argumentów, ale zwraca wartość typu string. Wynik działania metody jest przypisywany do zmiennej i następnie wypisywany w konsoli.

```
class Program
{
    static string Wypisz()
    {
        return "Metoda bez parametrów.";
    }

    private static void Main(string[] args)
    {
        string napis = Wypisz();
        Console.WriteLine(napis);
    }
}
```

Dane wyjściowe:

Metoda bez parametrów.

Przykład 4.5.

W tym przykładzie zdefiniowano metodę, która przyjmuje dwa parametry typu int i zwraca ich sumę (również typu int). Wynik działania metody jest przypisywany do zmiennej, a następnie wyświetlany w konsoli.

```
class Program
{
    static int Dodaj(int a, int b)
    {
        return a + b;
    }

    private static void Main(string[] args)
    {
        int wynik = Dodaj(2, 3);
        Console.WriteLine($"Dodaj(2,3) = {wynik}");
    }
}
```

Dane wyjściowe:

Dodaj(2,3) = 5



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Przykład 4.6.

W tym przykładzie pokazano, że parametry metod w C# są domyślnie przekazywane przez wartość. Oznacza to, że do metody trafia kopia zmiennej. Modyfikacje dokonane na parametrze dotyczą tylko kopii i nie zmieniają oryginalnej zmiennej. Jest to naturalne zachowanie dla *typów wartościowych* (np. int, double).

```
class Program
{
    static void Zwiksz(int x)
    {
        x++;
    }

    private static void Main(string[] args)
    {
        int a = 10;
        Console.WriteLine($"a = {a}");

        Zwiksz(a);
        Console.WriteLine($"Wartość 'a' po wywołaniu Zwiksz(a) = {a}");
    }
}
```

Dane wyjściowe:

```
a = 10
Wartość 'a' po wywołaniu Zwiksz(a) = 10
```

Przykład 4.7.

W tym przykładzie parametr przekazywany jest do metody przez *referencję* (za pomocą słowa kluczowego ref). Oznacza to, że metoda pracuje na oryginalnej zmiennej, a więc modyfikacje wewnątrz metody wpływają bezpośrednio na jej wartość poza metodą.

```
static void Zwiksz(ref int x)
{
    x++;
}

int v = 15;
Console.WriteLine($"v = {v}");

Zwiksz(ref v);
Console.WriteLine($"'v' po wywołaniu Zwiksz(v) = {v}");
```



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Dane wyjściowe:

```
v = 15  
'v' po wywołaniu Zwiększ(v) = 16
```

Uwaga! Zmienna przekazywana za pomocą ref musi być wcześniej zainicjalizowana.

Przykład 4.8.

W tym przykładzie pokazano użycie parametrów przekazywanych przez out. Parametry oznaczone jako out muszą zostać zainicjalizowane wewnątrz metody. Dzięki nim metoda może zwrócić więcej niż jedną wartość.

```
void Podziel(int dzielna, int dzielnik, out int wynik, out int reszta)  
{  
    wynik = dzielna / dzielnik;  
    reszta = dzielna % dzielnik;  
}  
  
int a = 17, b = 5;  
Podziel(a, b, out int w, out int r);  
Console.WriteLine($"{a} / {b} = {w}, reszta = {r}");
```

Dane wyjściowe:

```
17 / 5 = 3, reszta = 2
```

Przykład 4.9.

W tym przykładzie pokazano użycie parametrów przekazywanych przez in. Parametry oznaczone jako in są przekazywane do metody przez referencję, ale nie mogą być w niej modyfikowane. Dzięki temu możliwy jest odczyt wartości bez tworzenia jej kopii.

```
int PoliczKwadrat(in int liczba)  
{  
    return liczba * liczba;  
}  
  
int x = 5;  
int kwadrat = PoliczKwadrat(in x);  
Console.WriteLine($"{x}^2={kwadrat}");
```

Dane wyjściowe:

```
5^2=25
```



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Przykład 4.10.

W tym przykładzie zaprezentowano zastosowanie *parametrów nazwanych*. Dzięki podaniu nazw parametrów podczas wywołania metody można zmieniać kolejność przekazywanych argumentów. Jest to szczególnie przydatne, gdy metoda przyjmuje wiele parametrów tego samego typu lub gdy lista parametrów jest długa.

```
void Formatuj(string imie, string nazwisko, string separator)
{
    Console.WriteLine($"{imie}{separator}{nazwisko}");
}

Formatuj(imie: "Jan", separator: " ", nazwisko: "Kowalski");
```

Dane wyjściowe:

Jan Kowalski

Przykład 4.11.

W tym przykładzie użyto *parametrów opcjonalnych*, czyli takich, które mają przypisane *wartości domyślne* w sygnaturze metody. Jeżeli podczas wywołania nie podamy argumentu, zostanie użyta wartość domyślna. Parametry opcjonalne można również łączyć z parametrami nazwanymi.

```
void PokazUzytkownika(string imie, int wiek = 18, bool aktywny = true)
{
    Console.WriteLine($"{imie}, wiek: {wiek}, aktywny: {aktywny}");
}

// 'wiek' i 'aktywny' przyjmują wartości domyślne
PokazUzytkownika("Jan");

// 'aktywny' przyjmuje wartość domyślną
PokazUzytkownika("Ewa", 22);

// 'wiek' przyjmuje wartość domyślną, 'aktywny' ustawiony na 'false'
PokazUzytkownika("Adam", aktywny: false);
```

Dane wyjściowe:

Jan, wiek: 18, aktywny: True
Ewa, wiek: 22, aktywny: True
Adam, wiek: 18, aktywny: False



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Przykład 4.12.

W tym przykładzie przedstawiono zastosowanie słowa kluczowego `params`, które umożliwia przekazanie do metody *zmiennej liczby argumentów* tego samego typu. Argumenty te są traktowane jako elementy tablicy, dzięki czemu metodę można wywoływać z różną liczbą argumentów, bez konieczności jawnego tworzenia tablicy. Należy pamiętać, że `params` musi być ostatnim parametrem w sygnaturze metody.

```
int Sumuj(params int[] liczby)
{
    int suma = 0;
    foreach (int x in liczby)
        suma += x;
    return suma;
}

Console.WriteLine(Sumuj());
Console.WriteLine(Sumuj(4, 6, 5));
Console.WriteLine(Sumuj(10, 20, 30, 40));
```

Dane wyjściowe:

```
0
15
100
```

Przykład 4.13.

W tym przykładzie pokazano metodę, która zwraca *krotkę* (z trzema wartościami). Elementy krotki mają *nazwane pola* (`suma`, `srednia`, `ilosc`), co umożliwia ich wygodne odczytywanie zarówno poprzez *dekonstrukcję*, jak i z użyciem *notacji kropkowej*.

```
class Program
{
    static (int suma, double srednia, int ilosc) Oblicz(params int[] liczby)
    {
        if (liczby.Length == 0) return (0, 0, 0);

        int suma = 0;
        foreach (var n in liczby) suma += n;

        double srednia = (double)suma / liczby.Length;
        return (suma, srednia, liczby.Length);
    }
}
```



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



```
static void Main()
{
    Console.WriteLine("Użycie krotki z dekonstrukcją:");

    var (suma, srednia, ilosc) = Oblicz(3, 1, 4, 2);
    Console.WriteLine($"suma={suma}, średnia={srednia}, ilość={ilosc}");

    Console.WriteLine("\nAlternatywnie, bez dekonstrukcji:");
    var stat = Oblicz(10, 20);

    Console.WriteLine($"suma = {stat.suma}");
    Console.WriteLine($"średnia = {stat.srednia}");
    Console.WriteLine($"ilość = {stat.ilosc}");
}
```

Dane wyjściowe:

Użycie krotki z dekonstrukcją:
suma=10, średnia=2,5, ilość=4

Alternatywnie, bez dekonstrukcji:
suma = 30
średnia = 15
ilość = 2

Przykład 4.14.

W tym przykładzie zaprezentowano *przeciążanie metod*. W klasie znajduje się kilka metod o tej samej nazwie (Pokaz), ale różniących się listą parametrów. Jedna przyjmuje napis, druga liczbę, trzecia – dwa napisy. Podczas wywołania kompilator automatycznie wybiera odpowiednią wersję metody na podstawie jej sygnatury, czyli liczby oraz typów parametrów.

```
class Program
{
    static void Pokaz(string tekst)
    {
        Console.WriteLine($"Tekst: {tekst}");
    }

    static void Pokaz(int liczba)
    {
        Console.WriteLine($"Liczba: {liczba}");
    }
}
```



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



```
static void Pokaz(string t1, string t2)
{
    Console.WriteLine($"Połączone: {t1} {t2}");
}

private static void Main(string[] args)
{
    Pokaz("Cześć!");
    Pokaz(100);
    Pokaz("Dzień", "dobry");
}
```

Dane wyjściowe:

Tekst: Cześć!
Liczba: 100
Połączone: Dzień dobry

Uwaga! Nie można przeciążyć metody wyłącznie przez zmianę typu wartości zwracanej.

Przykład 4.15.

W tym przykładzie zaprezentowano wykorzystanie *metody lokalnej*. Metoda Powitaj została zdefiniowana wewnątrz metody Main, dzięki czemu jest widoczna tylko w jej obrębie. Takie metody pozwalają uporządkować kod i wydzielić powtarzające się fragmenty logiki, które nie są potrzebne w całej klasie.

```
class Program
{
    static void Main(string[] args)
    {
        void Powitaj(string imie)
        {
            Console.WriteLine($"Witaj, {imie}!");
        }

        Powitaj("Ania");
        Powitaj("Tomek");
    }
}
```

Dane wyjściowe:

Witaj, Ania!
Witaj, Tomek!



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Przykład 4.16.

W tym przykładzie pokazano, czym jest metoda anonimowa. Taką metodę przypisuje się do delegata, czyli typu, który określa, jakie parametry i typ zwracany ma przyjmować metoda. Metody anonimowe są użyteczne, gdy chcemy przekazać krótką funkcję jako argument lub wykonać jednorazową operację bez tworzenia osobnej, nazwanej metody.

```
class Program
{
    delegate int Operacja(int a, int b);

    static void Main(string[] args)
    {
        Operacja Mnozenie = delegate (int x, int y)
        {
            Console.WriteLine("Wykonuję mnożenie.");
            return x * y;
        };
        Console.WriteLine($"Iloczyn: {Mnozenie(3, 6)}");
    }
}
```

Dane wyjściowe:

```
Wykonuję mnożenie.
Iloczyn: 18
```

Przykład 4.17.

W tym przykładzie pokazano użycie wyrażenia lambda, które stanowi krótszy zapis metod anonimowych.

```
class Program
{
    delegate int Operacja(int a, int b);
    static void Main(string[] args)
    {
        Operacja Mnozenie = (x, y) =>
        {
            Console.WriteLine("Wykonuję mnożenie.");
            return x * y;
        };
        Console.WriteLine($"Iloczyn: {Mnozenie(4, 6)}");
    }
}
```



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Dane wyjściowe:

Wykonuję mnożenie.
Iloczyn: 24

Przykład 4.18.

W tym przykładzie pokazano użycie wbudowanego delegata `Func<>`, który określa typy parametrów (może być bez parametrów) oraz typ zwracany metody. Dzięki temu nie trzeba samodzielnie definiować typu delegata.

- a) `Func<int, int, int>` oznacza funkcję przyjmującą dwa parametry typu `int` i zwracającą wartość typu `int`.

```
class Program
{
    static void Main(string[] args)
    {
        Func<int, int, int> Mnozenie = (x, y) =>
        {
            Console.WriteLine("Wykonuję mnożenie.");
            return x * y;
        };

        Console.WriteLine($"Iloczyn: {Mnozenie(7, 8)}");
    }
}
```

Dane wyjściowe:

Wykonuję mnożenie.
Iloczyn: 56

- b) `Func<int>` oznacza funkcję bez argumentów, która zwraca wartość typu `int`.

```
class Program
{
    static void Main(string[] args)
    {
        Func<int> LosujLiczbe = () =>
        {
            Random rand = new Random();
            return rand.Next(1, 101);
        };

        Console.WriteLine($"Wylosowana liczba: {LosujLiczbe()}");
    }
}
```



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Dane wyjściowe:

Wylosowana liczba: {tu będzie wartość z przedziału [1,100]}

Przykład 4.19.

W tym przykładzie pokazano użycie delegata Action, służącego do reprezentowania metod, które wykonują operację, ale nie zwracają żadnej wartości (void).

a) Action<string, int> – funkcja przyjmuje dwa parametry (string i int).

```
class Program
{
    static void Main(string[] args)
    {
        Action<string, int> Powitaj = (imie, wiek) =>
        {
            Console.WriteLine($"Cześć, {imie}! Masz {wiek} lat.");
        };
        Powitaj("Tomasz", 25);
    }
}
```

Dane wyjściowe:

Cześć, Tomasz! Masz 25 lat.

b) Action bez <...> oznacza funkcję bez parametrów.

```
class Program
{
    static void Main(string[] args)
    {
        Action Wyswietl = () =>
        {
            Console.WriteLine("To przykład Action bez parametrów.");
        };
        Wyswietl();
    }
}
```

Dane wyjściowe:

To przykład Action bez parametrów.



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Przykład 4.20.

W poniższych przykładach zaprezentowano, jak ten sam program można zapisać na różne sposoby – od najprostszej wersji, w której cały kod znajduje się w jednej metodzie, aż po wersję uporządkowaną, podzieloną na osobne pliki i klasy.

- a) W tej części pokazano program, w którym cały kod znajduje się w metodzie Main. Zarówno wczytywanie danych, jak i obliczenia pól figur zostały zapisane bezpośrednio w ciele metody.

```
class Program
{
    static void Main(string[] args)
    {
        Console.Write("Podaj bok kwadratu: ");
        double a = double.Parse(Console.ReadLine());
        double poleKw = a * a;
        Console.WriteLine($"Pole kwadratu = {poleKw}\n");

        Console.Write("Podaj bok a: ");
        double x = double.Parse(Console.ReadLine());
        Console.Write("Podaj bok b: ");
        double y = double.Parse(Console.ReadLine());
        double polePr = x * y;
        Console.WriteLine($"Pole prostokąta = {polePr}\n");

        Console.Write("Podaj promień koła: ");
        double r = double.Parse(Console.ReadLine());
        double poleK = Math.PI * r * r;
        Console.WriteLine($"Pole koła = {poleK:F2}\n");
    }
}
```

Dane wyjściowe:

Podaj bok kwadratu: 5
Pole kwadratu = 25

Podaj bok a: 3
Podaj bok b: 4
Pole prostokąta = 12

Podaj promień koła: 1
Pole koła = 3,14



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



b) W tym przykładzie pokazano program z podziałem na metody. Zamiast umieszczać wszystkie obliczenia w Main(), utworzono osobne metody.

```
class Program
{
    static double WczytajDouble(string dane)
    {
        Console.Write(dane);
        return double.Parse(Console.ReadLine());
    }

    static double ObliczPoleKwadratu(double bok) => bok * bok;
    static double ObliczPoleProstokata(double a, double b) => a * b;
    static double ObliczPoleKola(double r) => Math.PI * r * r;

    static void Main(string[] args)
    {
        double a = WczytajDouble("Podaj bok kwadratu: ");
        Console.WriteLine($"Pole kwadratu = {ObliczPoleKwadratu(a)}\n");

        double x = WczytajDouble("Podaj bok a: ");
        double y = WczytajDouble("Podaj bok b: ");
        Console.WriteLine($"Pole prostokata = {ObliczPoleProstokata(x, y)}\n");

        double r = WczytajDouble("Podaj promień koła: ");
        Console.WriteLine($"Pole koła = {ObliczPoleKola(r):F2}");
    }
}
```

Dane wyjściowe:

```
Podaj bok kwadratu: 6
Pole kwadratu = 36

Podaj bok a: 4
Podaj bok b: 7
Pole prostokata = 28

Podaj promień koła: 2
Pole koła = 12,57
```

c) W tym przykładzie pokazano podział programu na osobne pliki i klasy. Utworzono klasę Wzory z metodami do odpowiednich obliczeń i klasę Pomocnicza z metodą do wczytywania danych. Metoda Main korzysta z tych klas, dzięki czemu kod w głównym pliku jest krótki i przejrzysty.



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Plik *Pomocnicza.cs*:

```
namespace Modularnosc
{
    static class Pomocnicza
    {
        public static double WczytajDouble(string komunikat)
        {
            Console.Write(komunikat);

            while (true)
            {
                if (double.TryParse(Console.ReadLine(), out double value) && value > 0)
                {
                    return value;
                }
                else
                {
                    Console.WriteLine("Niepoprawna liczba. Spróbuj ponownie");
                }
            }
        }
    }
}
```

Plik *Wzory.cs*:

```
namespace Modularnosc
{
    static class Wzory
    {
        public static double ObliczPoleKwadratu(double bok) => bok * bok;
        public static double ObliczPoleProstokata(double a, double b) => a * b;
        public static double ObliczPoleKola(double r) => Math.PI * r * r;
    }
}
```

Plik *Program.cs*:

```
using Modularnosc;

class Program
{
    static void Main(string[] args)
    {
        double a = Pomocnicza.WczytajDouble("Podaj bok kwadratu: ");
        Console.WriteLine($"Pole kwadratu = {Wzory.ObliczPoleKwadratu(a)}");
        Console.WriteLine($"Obwód kwadratu = {Wzory.ObliczObwodKwadratu(a)}");

        Console.WriteLine("\nPodaj boki prostokąta.");
    }
}
```



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



```
double x = Pomocnicza.WczytajDouble("a: ");
double y = Pomocnicza.WczytajDouble("b: ");
Console.WriteLine($"Pole prostokąta = {Wzory.ObliczPoleProstokata(x, y)}");

double r = Pomocnicza.WczytajDouble("\nPodaj promień koła: ");
Console.WriteLine($"Pole koła = {Wzory.ObliczPoleKola(r):F2}");
}
}
```

Dane wyjściowe:

Podaj bok kwadratu: 3
Pole kwadratu = 9
Obwód kwadratu = 12

Podaj boki prostokąta.
a: 1
b: 6
Pole prostokąta = 6

Podaj promień koła: 3
Pole koła = 28,27

Ćwiczenia samodzielne

Zadanie 4.1.

Napisz metodę static void Powitaj(string imie = "Użytkowniku", string powitanie = "Witaj"), która wypisuje komunikat w formacie: „powitanie, imie!”. W Main wywołaj metodę:

- a) bez argumentów,
- b) z jednym argumentem,
- c) z dwoma argumentami,
- d) z użyciem parametrów nazwanych.

Zadanie 4.2.

Napisz metodę static void WswietlZnaki(char znak, params int[] liczby), która wypisuje w konsoli wiersze złożone z podanego znaku, przy czym długość każdego wiersza odpowiada kolejnej liczbie z tablicy liczby. Jeżeli nie zostaną przekazane żadne liczby, metoda powinna wyświetlić komunikat: „Nie podano liczb.”



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Zadanie 4.3.

Napisz metodę `static void PowitajWielu(params string[] imiona)`, która przyjmuje dowolną liczbę imion i dla każdego wypisze powitanie w osobnej linii, np. „Witaj, Ania”. Jeżeli do metody nie zostanie przekazane żadne imię, powinna wypisać komunikat: „Nie podano żadnych imion do powitania.”

Zadanie 4.4.

Napisz metodę `double Srednia(params double[] liczby)`, która oblicza średnią arytmetyczną dowolnej liczby argumentów. Jeżeli do metody nie zostaną przekazane żadne liczby, powinna zwrócić `double.NaN`.

Zadanie 4.5.

Napisz metodę `string Formatuj(string imie, string nazwisko, bool odwracaj = false, string separator = " ")`, która zwraca sformatowany napis zawierający imię, nazwisko oddzielone podanym separatorem. Jeżeli parametr `odwracaj` ma wartość `true`, metoda powinna zwracać napis w kolejności „nazwisko separator imie”, a w przeciwnym razie – „imie separator nazwisko”. W Main przetestuj działanie metody.

Zadanie 4.6.

Napisz metodę `bool CzyPierwsza(int n)`, która sprawdza, czy liczba `n` jest liczbą pierwszą. W Main wczytaj liczbę `n` od użytkownika i wypisz odpowiedni komunikat.

Zadanie 4.7.

Napisz metodę `RozwiazRownanieKwadratowe(double a, double b, double c)`, która rozwiązuje równanie kwadratowe i zwraca krotkę postaci (`delta`, `liczbaPierwiastkow`, `x1`, `x2`). Następnie w metodzie Main wczytaj współczynniki od użytkownika, wywołaj tę metodę i wypisz wynik w czytelnej formie.

Zadanie 4.8.

Napisz metodę `CzyMoznaZbudowacTrojkat(double a, double b, double c)`, która sprawdza, czy z odcinków o podanych długościach `a`, `b`, `c` można zbudować trójkąt i zwraca wartość logiczną. Następnie w metodzie Main wczytaj długości boków od użytkownika, wywołaj tę metodę i wypisz wynik w czytelnej formie.

Zadanie 4.9.

Napisz metodę `ZnajdzMiejsceZerowe(double a, double b, out double x0)`, która wyznaczy miejsce zerowe równania liniowego $ax + b = 0$. Jeżeli $a == 0$, metoda powinna zwrócić `false` (bo wtedy brak rozwiązania albo istnieje



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



nieskończenie wiele rozwiązań). W przeciwnym razie ma zwrócić true i wpisać rozwiązanie do parametru x0.

Zadanie 4.10.

Napisz metodę void RozlozNaCyfry(int liczba, out int jednosci, out int dziesiątki, out int setki), która przyjmuje liczbę z zakresu [0, 999] i zwraca przez parametry out jej cyfry jedności, dziesiątek i setek.

Zadanie 4.11.

Napisz metodę ObliczPoleIObwod(double r, out double pole, out double obwod), która dla koła o promieniu r oblicza pole i obwód. W Main wczytaj promień od użytkownika, wywołaj metodę i wypisz otrzymane wyniki.

Zadanie 4.12.

Napisz metodę ObliczCene(double cenaNetto, double vat = 0.23), która oblicza cenę brutto towaru według wzoru: $cenaBrutto = cenaNetto * (1 + vat)$. W Main przetestuj metodę z parametrem domyślnym oraz z inną wartością podatku VAT (np. 8%).

Zadanie 4.13.

Napisz metodę LosujLiczbe(Random rnd, int min, int max), która zwraca losową liczbę całkowitą z przedziału [min, max). W metodzie Main utwórz obiekt klasy Random, a następnie wylosuj 5 liczb z przedziału [1, 50]. Użyj parametrów nazwanych przy wywołaniu metody LosujLiczbe. Wylosowane liczby wypisz w konsoli.

Wskazówka: do losowania liczb możesz wykorzystać klasę Random i metodę Next.

```
Random rnd = new Random();  
int liczba = rnd.Next(a, b); // generuje liczbę z przedziału [a,b)
```

Zadanie 4.14.

Przygotuj prostą grę w zgadywanie liczby. Zaimplementuj metodę Zgadnij(int sekret, int strzał), która zwraca napis „Za mało”, „Za dużo” lub „Trafony” w zależności od wartości podanego strzału. W metodzie Main utwórz obiekt klasy Random, wylosuj liczbę całkowitą z przedziału od 1 do 100, wykorzystując metodę LosujLiczbe z zadania 4.13. Prowadź pętlę pytań do momentu trafienia, zliczając przy tym liczbę prób.



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Zadanie 4.15.

Napisz metodę `void Zamien(ref int a, ref int b)`, która zamienia miejscami wartości dwóch zmiennych. W Main wczytaj od użytkownika dwie liczby, wywołaj metodę i wypisz wartości przed i po zamianie.

Zadanie 4.16.

Napisz metodę `void Przesun(ref int x, ref int y, int dx, int dy)`, która powinna przesunąć punkt o współrzędnych $[x, y]$ o wektor $[dx, dy]$. W wyniku działania metody wartości x i y mają być zmodyfikowane. W metodzie Main wczytaj od użytkownika współrzędne punktu oraz przesunięcia, wywołaj metodę i wypisz nowe współrzędne punktu.

Zadanie 4.17.

Napisz metodę `void Normalizuj(ref double x, ref double y)`, która przeskaluje wektor o współrzędnych $[x, y]$ tak, aby jego długość była równa 1. Jeżeli wektor ma długość równą 0, należy pozostawić go bez zmian. W metodzie Main wczytaj współrzędne wektora od użytkownika, wywołaj metodę i wypisz nowy, znormalizowany wektor.

Wskazówka: aby znormalizować wektor, wystarczy podzielić każdą współrzędną przez jego długość.

Zadanie 4.18.

Napisz metody `string SzyfrCezara(string tekst, int przesuniecie = 3)` oraz `string DeszyfrCezara(string tekst, int przesuniecie = 3)`, które działają wyłącznie na literach alfabetu łacińskiego, zachowując wielkość liter i pozostawiając inne znaki bez zmian. W metodzie Main wczytaj tekst od użytkownika, zaszyfruj go i następnie odszyfruj.

Wskazówki:

- Do sprawdzenia, czy dany znak jest literą, możesz użyć `char.IsLetter(c)`.
- Do rozróżnienia dużych i małych liter możesz użyć `char.IsUpper(c)`.
- Aby zamienić literę na jej pozycję w alfabecie, możesz obliczyć $c - 'A'$ (dla dużych) lub $c - 'a'$ (dla małych).
- Przesuniętą pozycję sprowadzisz z powrotem do zakresu 0 – 25 za pomocą operacji modulo % 26.



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Zadanie 4.19.

Napisz metodę `Odleglosc2D(double x1, double y1, double x2, double y2)` obliczającą odległość euklidesową między dwoma punktami $[x_1, y_1]$, $[x_2, y_2]$ w układzie współrzędnych i pokaż przeciążanie tej metody przyjmując współrzędne jako `int`. Dodaj metodę `bool CzyPunktWKole(double px, double py, double sx, double sy, double r)`, która sprawdza, czy dany punkt (px, py) należy do koła o zadanym środku (sx, sy) i promieniu r .

Zadanie 4.20.

W metodzie `Main` zdefiniuj lokalną metodę `bool CzyParzysta(int liczba)`, która zwraca `true`, jeśli liczba jest parzysta, w przeciwnym razie `false`. Wczytaj liczbę od użytkownika, wywołaj metodę i wypisz odpowiedni komunikat.

Zadanie 4.21.

Utwórz delegata o nazwie `Powitanie`, który przyjmuje jeden parametr typu `string` i zwraca `void`. Następnie przypisz do niego metodę anonimową, która wyświetla komunikat postaci: „Witaj, Ewa!”. Wywołaj metodę dla kilku przykładowych imion. Następnie zrealizuj to samo zadanie, wykorzystując:

- wyrażenie lambda,
- wbudowanego delegata `Action`.

Zadanie 4.22.

Zdefiniuj delegata `Potegowanie`, który przyjmuje dwie liczby całkowite (a, b) i zwraca wartość typu `int`. Przypisz do niego wyrażenie lambda, które oblicza a do potęgi b . Wyświetl wynik w postaci: $2^5 = 32$. Następnie wykonaj to samo zadanie z wykorzystaniem wbudowanego delegata `Func`.

Zadanie 4.23.

Utwórz zmienną typu `Action<string>` o nazwie `WyswietlKomunikat`, która przyjmuje tekst komunikatu i wyświetla go w konsoli z bieżącą datą i godziną. Następnie wywołaj metodę dla kilku przykładowych komunikatów, np.

[2025-10-13 12:02:59] Uruchomiono program...

Zadanie 4.24.

Utwórz zmienną typu `Func<int, bool>` o nazwie `CzyParzysta`, która sprawdza, czy podana liczba jest parzysta. Wyświetl wynik dla kilku przykładowych wartości.



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Pytania pomocnicze

- 1) Jaka jest różnica między parametrem a argumentem metody?
- 2) Na czym polega różnica między przekazywaniem parametrów przez wartość a przez referencję?
- 3) Czym różni się przekazywanie parametrów przez ref oraz out?
- 4) Co oznacza słowo kluczowe in przy parametrze metody?
- 5) Na czym polega przeciążanie metod i w jaki sposób kompilator wybiera odpowiednią wersję metody?
- 6) Czy można przeciążać metody różniące się tylko typem zwracanym?
- 7) Co to są parametry opcjonalne?
- 8) Jak działają parametry nazwane?
- 9) Do czego służy słowo kluczowe params? Dlaczego params musi występować na końcu listy parametrów?
- 10) Co zwraca metoda o typie zwracanym void?
- 11) Czy metoda z parametrem out może zwrócić wartość przez return?
- 12) Jak można zwrócić z metody więcej niż jedną wartość bez użycia out?

Podsumowanie

W tym laboratorium przedstawiono zasady definiowania i wywoływania metod w języku C#. Omówiono różne sposoby przekazywania parametrów: przez wartość, przez referencję (ref, out, in), oraz wykorzystanie parametrów opcjonalnych, nazwanych i tablicowych (params). Pokazano, jak metody mogą zwracać pojedyncze wyniki lub krotki, a także jak stosować przeciążanie metod. Ponadto przedstawiono zastosowanie metod lokalnych, metod anonimowych oraz wyrażeń lambda.

Literatura

- [1] Michaelis M., *C# 8.0: kompletny przewodnik dla praktyków*, Helion, Gliwice 2021. s. 193-229.
- [2] Albahari J., *C# 12 w pigułce. Kompendium programisty*, Helion, Gliwice, 2024. s. 82-88, 116-118, 194-200.

<https://learn.microsoft.com/pl-pl/dotnet/csharp/methods>



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Materiały do przedmiotu:

Podstawy programowania

Laboratorium nr 9

Obsługa wyjątków. Zarządzanie błędami w kodzie

Autor:

mgr Krystyna Kiersztyn



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Spis treści

Wprowadzenie, tematyka zajęć	62
Cel laboratorium	62
Instrukcja laboratoryjna/ćwiczeniowa	63
Ćwiczenia samodzielne	73
Pytania pomocnicze	76
Podsumowanie	76
Literatura	77



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Projekt współfinansowany ze środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU
SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Wprowadzenie, tematyka zajęć

Podczas pisania programów często pojawiają się sytuacje, w których kod może działać niezgodnie z oczekiwaniami. Na przykład użytkownik poda błędne dane, program spróbuje otworzyć nieistniejący plik lub wystąpi próba dzielenia przez zero. Takie sytuacje nazywamy *wyjątkami*.

Mechanizm obsługi wyjątków w języku C# stanowi część środowiska uruchomieniowego CLR i pozwala na bezpieczne reagowanie na błędy w czasie działania programu, zamiast powodować jego natychmiastowe zakończenie. W C# wszystkie wyjątki reprezentowane są przez klasy. Wszystkie wyjątki w C# są reprezentowane przez klasy pochodne od wbudowanej klasy `System.Exception`. Istnieje również możliwość tworzenia własnych klas wyjątków, co umożliwia dopasowanie obsługi błędów do potrzeb konkretnej aplikacji.

Podstawowym celem obsługi wyjątków jest:

- wykrycie błędu,
- poinformowanie użytkownika lub programisty o jego wystąpieniu,
- umożliwienie dalszego działania programu w kontrolowany sposób.

Do obsługi błędów w C# służą następujące konstrukcje:

- `try` – zawiera kod, który może spowodować wystąpienie wyjątku,
- `catch` – przechwytuje i obsługuje wyjątek,
- `finally` – zawiera kod, który zostanie wykonany niezależnie od tego, czy wyjątek wystąpił,
- `throw` – służy do ręcznego zgłaszania wyjątku.

Cel laboratorium

Po zrealizowaniu laboratorium student:

- zna pojęcie wyjątku i rozumie jego znaczenie w procesie obsługi błędów,
- potrafi stosować instrukcje `try`, `catch`, `finally` oraz `throw`,
- potrafi obsługiwać typowe wyjątki występujące w czasie działania programu,
- potrafi tworzyć i stosować własne klasy wyjątków,
- zna zastosowanie instrukcji `checked`.



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Instrukcja laboratoryjna/ćwiczeniowa

Przykład 9.1.

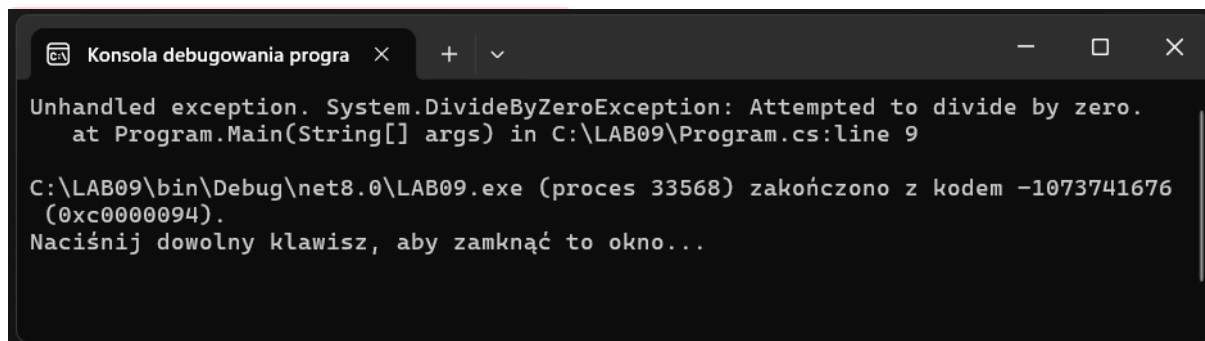
W tym przykładzie wykonywana jest operacja dzielenia liczby a przez b . Ponieważ zmienna b ma wartość 0, podczas wykonywania instrukcji a / b zgłaszany jest wyjątek typu `System.DivideByZeroException`.

```
using System;
class Program
{
    static void Main(string[] args)
    {
        int a = 10;
        int b = 0;

        int wynik = a / b;

        Console.WriteLine($"Wynik dzielenia: {wynik}");
    }
}
```

Program nie zawiera mechanizmu obsługi wyjątków, dlatego jego wykonanie (**bez debugowania**) kończy się błędem i następującym komunikatem w konsoli:



```
Konsola debugowania progra x + v - □ x

Unhandled exception. System.DivideByZeroException: Attempted to divide by zero.
  at Program.Main(String[] args) in C:\LAB09\Program.cs:line 9

C:\LAB09\bin\Debug\net8.0\LAB09.exe (proces 33568) zakończono z kodem -1073741676
(0xc0000094).
Naciśnij dowolny klawisz, aby zamknąć to okno...
```

Podczas debugowania można sprawdzić szczegóły zgłoszonego wyjątku.



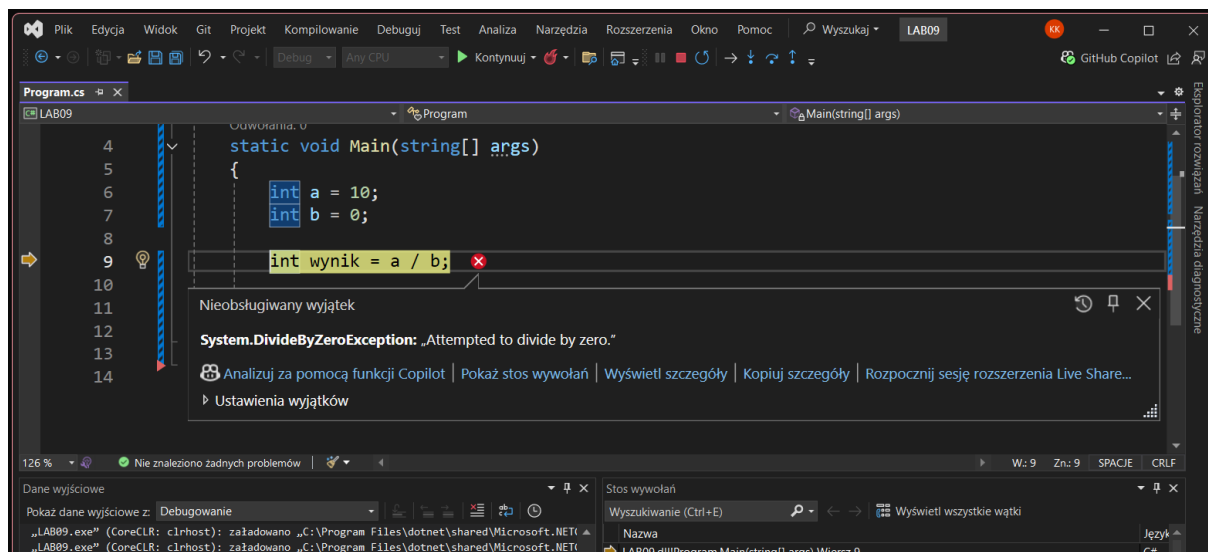
Fundusze Europejskie
dla Rozwoju Społecznego



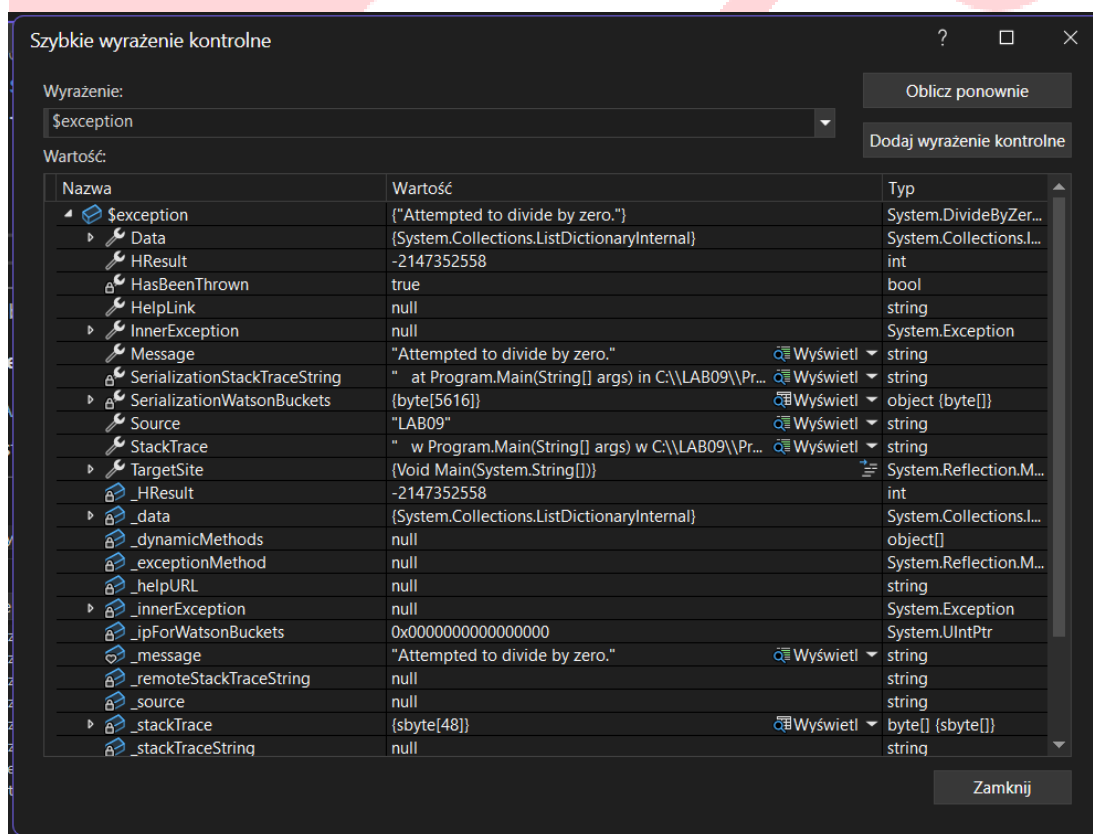
Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską





Po kliknięciu opcji **Wyświetl szczegóły** pojawia się okno **Szybkie wyrażenia kontrolne**, które zawiera dane przechowywane w obiekcie wyjątku.



Warto zwrócić uwagę na kilka istotnych właściwości obiektu wyjątku:



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Projekt współfinansowany ze środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

- **Message** – zawiera opis błędu.
- **StackTrace** – pokazuje ścieżkę wywołań metod prowadzącą do miejsca wystąpienia wyjątku, co ułatwia lokalizację błędu w kodzie.
- **InnerException** – może zawierać inny wyjątek, który spowodował obecny błąd. W tym przypadku ma wartość null, ale czasami może przechowywać inny wyjątek zagnieżdżony w głównym wyjątku.

Kliknięcie przycisku **Wyświetl** przy każdej z właściwości pozwala zobaczyć ich pełny opis.

Przykład 9.2.

W tym przykładzie kod został zabezpieczony blokiem try-catch-finally. W przypadku wystąpienia błędu dzielenia przez zero, program nie przerywa działania, lecz przechodzi do sekcji catch, w której wyświetla komunikat o błędzie. Blok finally wykonuje się zawsze – niezależnie od tego, czy wystąpił wyjątek.

```
using System;
class Program
{
    static void Main(string[] args)
    {
        int a = 10;
        int b = 0;
        try
        {
            int wynik = a / b;
            Console.WriteLine($"Wynik dzielenia: {wynik}");
        }
        catch (DivideByZeroException)
        {
            Console.WriteLine("Błąd: próba dzielenia przez zero!");
        }
        finally
        {
            Console.WriteLine("Zakończono działanie programu.");
        }
    }
}
```

Dane wyjściowe:

```
Błąd: próba dzielenia przez zero!
Zakończono działanie programu.
```



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Przykład 9.3.

W tym przykładzie zaprezentowano obsługę wyjątków przy użyciu dwóch bloków catch. Program wczytuje od użytkownika dwie liczby i wykonuje operację dzielenia. W sytuacji, gdy dzielnik ma wartość 0, zgłaszany jest wyjątek `DivideByZeroException`, natomiast w przypadku podania danych w niepoprawnym formacie – `FormatException`. Każdy z wyjątków jest przechwytywany w odpowiednim bloku catch. Kolejność bloków catch ma znaczenie. Bardziej szczegółowe typy wyjątków powinny znajdować się przed wyjątkami ogólnymi.

```
class Program
{
    static void Main()
    {
        try
        {
            Console.Write("Podaj liczbę a: ");
            int a = int.Parse(Console.ReadLine());
            Console.Write("Podaj liczbę b: ");
            int b = int.Parse(Console.ReadLine());

            int wynik = a / b;
            Console.WriteLine($"Wynik dzielenia: {wynik}");
        }
        catch (DivideByZeroException)
        {
            Console.WriteLine("Błąd: próba dzielenia przez zero!");
        }
        catch (FormatException)
        {
            Console.WriteLine("Błąd: niepoprawny format liczby!");
        }
    }
}
```

Dane wyjściowe:

```
Podaj liczbę a: 5
Podaj liczbę b: 2
Wynik dzielenia: 2
```

Dane wyjściowe:

```
Podaj liczbę a: a
Błąd: niepoprawny format liczby!
```



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Dane wyjściowe:

Podaj liczbę a: 55
Podaj liczbę b: 0
Błąd: próba dzielenia przez zero!

Przykład 9.4.

W tym przykładzie zaprezentowano sposób zapisywania informacji o wyjątkach do pliku tekstowego. Program wykonuje prostą operację dzielenia dwóch liczb wprowadzonych przez użytkownika, a w przypadku wystąpienia błędu (np. dzielenia przez zero lub błędnego formatu danych) przechwytuje wyjątek w bloku catch.

Dodatkowo, w każdej obsłudze błędu wywoływana jest metoda ZapiszDoPliku(), która za pomocą klasy StreamWriter zapisuje do pliku *log.txt* szczegóły zgłaszanego wyjątku (jego typ, opis, ślad stosu oraz datę wystąpienia).

```
using System.IO;
class Program
{
    static void Main()
    {
        try
        {
            Console.Write("Podaj liczbę a: ");
            int a = int.Parse(Console.ReadLine());

            Console.Write("Podaj liczbę b: ");
            int b = int.Parse(Console.ReadLine());

            int wynik = a / b;
            Console.WriteLine($"Wynik dzielenia: {wynik}");
        }
        catch (DivideByZeroException ex)
        {
            Console.WriteLine("Błąd: próba dzielenia przez zero!");
            ZapiszDoPliku(ex);
        }
        catch (FormatException ex)
        {
            Console.WriteLine("Błąd: niepoprawny format liczby!");
            ZapiszDoPliku(ex);
        }
    }
}
```



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



```
static void ZapiszDoPliku(Exception ex)
{
    using (StreamWriter sw = new StreamWriter("log.txt", true))
    {
        sw.WriteLine($"[{DateTime.Now}] Wystąpił wyjątek:");
        sw.WriteLine($"Typ: {ex.GetType().Name}");
        sw.WriteLine($"Opis: {ex.Message}");
        sw.WriteLine($"Ślad stosu:\n{ex.StackTrace}");
        sw.WriteLine(new string('-', 60));
    }
}
```

Dane wyjściowe:

Podaj liczbę a: x
Błąd: niepoprawny format liczby!

Po uruchomieniu programu z powyższymi danymi w pliku *log.txt* pojawiły się następujące informacje:

```
[6.10.2025 16:04:43] Wystąpił wyjątek:
Typ: FormatException
Opis: The input string 'x' was not in a correct format.
Ślad stosu:
   at System.Number.ThrowFormatException[TChar](ReadOnlySpan`1 value)
   at System.Int32.Parse(String s)
   at Program.Main() in C:\LAB09\Program.cs:line 9
-----
```

Przykład 9.5.

W tym przykładzie zaprezentowano zastosowanie instrukcji `checked`, która umożliwia kontrolowanie przepełnienia typów liczbowych w czasie działania programu. Domyślnie w języku C# przepełnienie zmiennych typu całkowitego nie powoduje błędu, a nadmiarowa wartość jest „obcinana” i wynik ulega zniekształceniu. Instrukcja `checked` wymusza sprawdzanie, czy wynik operacji mieści się w dopuszczalnym zakresie typu (`Int32`), a w przypadku przekroczenia granicy zgłaszany jest wyjątek `OverflowException`.

```
class Program
{
    static void Main()
    {
        try
        {
```



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



```
Console.Write("Podaj liczbę całkowitą: ");
int n = int.Parse(Console.ReadLine());

int wynik = 1;

checked // sprawdzanie przepełnienia
{
    for (int i = 2; i <= n; i++)
        wynik *= i;
}

Console.WriteLine($"Silnia z {n} = {wynik}");
}

catch (OverflowException)
{
    Console.WriteLine("Błąd: wynik przekroczył maksymalną " +
        "wartość typu Int32 (przepełnienie).");
}
catch (FormatException)
{
    Console.WriteLine("Błąd: niepoprawny format liczby!");
}
}
```

Dane wyjściowe:

Podaj liczbę całkowitą: 12
Silnia z 12 = 479001600

Dane wyjściowe:

Podaj liczbę całkowitą: 15
Błąd: wynik przekroczył maksymalną wartość typu Int32 (przepełnienie).

Przykład 9.6

W tym przykładzie zaprezentowano sposób zgłaszania wyjątku z poziomu metody za pomocą instrukcji throw. Metoda Wyświetl sprawdza poprawność przekazanego parametru imię. Jeżeli parametr ten jest pusty lub zawiera jedynie białe znaki, za pomocą instrukcji throw zostaje zgłoszony wyjątek ArgumentNullException. Do konstruktora wyjątku przekazywane są dwa argumenty: nazwa parametru uzyskana przy użyciu operatora nameof oraz komunikat opisujący błąd.



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



W metodzie Main() wywołanie Wyswietl("") powoduje zgłoszenie wyjątku, który następnie zostaje przechwycony w bloku catch.

```
class Program
{
    private static void Main(string[] args)
    {
        try
        {
            Wyswietl("");
        }
        catch (ArgumentNullException ex)
        {
            Console.WriteLine($"Przechwycono wyjątek.");
            Console.WriteLine(ex.Message);
        }
    }

    static void Wyswietl(string imie)
    {
        if (string.IsNullOrEmpty(imie))
            throw new ArgumentNullException(nameof(imie),
                "Imię nie może być puste.");
        Console.WriteLine($"Podano imię: {imie}");
    }
}
```

Dane wyjściowe:

```
Przechwycono wyjątek.
Imię nie może być puste. (Parameter 'imie')
```

Przykład 9.7.

W tym przykładzie zaprezentowano sposób definiowania i zgłaszania własnego wyjątku. Własny wyjątek tworzony jest poprzez zdefiniowanie klasy dziedziczącej po klasie bazowej Exception. W konstruktorze można przekazać do klasy bazowej komunikat opisujący błąd.

Program pobiera od użytkownika wiek, sprawdza poprawność danych i w przypadku wprowadzenia liczby ujemnej zgłasza wyjątek NegativeAgeException za pomocą instrukcji throw.



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



```
public class NegativeAgeException : Exception
{
    public NegativeAgeException(int wiek) : base($"Podano niepoprawny " +
        $"wiek: {wiek}. Wiek nie może być liczbą ujemną.")
    { }
}

class Program
{
    static void Main()
    {
        try
        {
            Console.Write("Podaj swój wiek: ");
            int wiek = int.Parse(Console.ReadLine());

            if (wiek < 0)
                throw new NegativeAgeException(wiek);

            Console.WriteLine($"Twój wiek to: {wiek} lat.");
        }
        catch (NegativeAgeException ex)
        {
            Console.WriteLine($"Błąd wartości: {ex.Message}");
        }
        catch (FormatException)
        {
            Console.WriteLine("Błąd: niepoprawny format liczby!");
        }
    }
}
```

Dane wyjściowe:

Podaj swój wiek: -18
Błąd wartości: Podano niepoprawny wiek: -18. Wiek nie może być liczbą ujemną.

Dane wyjściowe:

Podaj swój wiek: 18
Twój wiek to: 18 lat.

Przykład 9.8.

W tym przykładzie przedstawiono zastosowanie bloku finally w sytuacjach, gdy konieczne jest wykonanie określonych działań (np. zamknięcie pliku, zamykanie



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



połączeń z bazą danych itp.) niezależnie od wystąpienia błędu. Program próbuje otworzyć plik *dane.txt*, a jeżeli plik nie zostanie znaleziony, zgłaszany jest wyjątek `FileNotFoundException`. W bloku `finally` następuje zamknięcie pliku (jeżeli został wcześniej otwarty) oraz wyświetlenie komunikatu potwierdzającego wykonanie tego bloku.

```
class Program
{
    static void Main(string[] args)
    {
        StreamReader reader = null;
        try
        {
            reader = new StreamReader("dane.txt");
            string zawartosc = reader.ReadToEnd();
            Console.WriteLine(zawartosc);
        }
        catch (FileNotFoundException ex)
        {
            Console.WriteLine("Błąd: Plik nie został znaleziony.");
            Console.WriteLine(ex.Message);
        }
        finally
        {
            if (reader != null)
            {
                reader.Close();
                Console.WriteLine("Plik został zamknięty.");
            }
            Console.WriteLine("Blok finally został wykonany.");
        }
    }
}
```

Dane wyjściowe (plik nie istnieje):

Błąd: Plik nie został znaleziony.
Could not find file 'C:\LAB09\bin\Debug\net8.0\dane.txt'.
Blok finally został wykonany.

Dane wyjściowe (plik *dane.txt* zawiera liczby od 1 do 6):

1 2 3 4 5 6
Plik został zamknięty.
Blok finally został wykonany.



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Ćwiczenia samodzielne

Zadanie 9.1.

Napisz program, który pobiera od użytkownika liczbę całkowitą. Jeżeli użytkownik wprowadzi coś innego niż liczba, zgłaszany jest wyjątek `FormatException`. Program powinien przechwycić ten wyjątek i poinformować użytkownika o błędzie. Dodatkowo w bloku `finally` należy wyświetlić komunikat: „*Program zakończył działanie.*”

Zadanie 9.2.

Napisz program, który pobiera od użytkownika liczbę całkowitą x , a następnie obliczy jej odwrotność $1/x$. Program powinien obsługiwać wyjątki `DivideByZeroException` oraz `FormatException`. W przypadku wystąpienia błędu program powinien wyświetlić odpowiedni komunikat w konsoli i zapisać szczegóły wyjątku (datę, rodzaj wyjątku i treść komunikatu) do pliku `exceptions.txt`.

Zadanie 9.3.

Napisz program, który utworzy tablicę pięciu elementów typu `int`, a następnie pobierze od użytkownika indeks elementu tej tablicy, który ma zostać wyświetlony. Jeżeli użytkownik poda indeks spoza zakresu, zgłaszany jest wyjątek `IndexOutOfRangeException`. Program powinien przechwycić ten wyjątek i wyświetlić odpowiedni komunikat.

Zadanie 9.4.

Napisz program, który w pętli nieskończonej wczytuje od użytkownika liczby całkowite. Pętla powinna zostać przerwana w momencie wpisania słowa „*stop*”. Poprawnie wprowadzone liczby powinny zostać zapisane do pliku `liczby.txt`, każda w nowej linii.

Jeżeli użytkownik wprowadzi dane, które nie mogą zostać przekonwertowane na liczbę całkowitą, program powinien przechwycić wyjątek `FormatException` i wyświetlić stosowny komunikat. W przypadku błędów podczas zapisu należy obsługiwać wyjątek `IOException`. Po zakończeniu działania programu plik powinien zostać zamknięty w bloku `finally`.

Zadanie 9.5.

Napisz program, który wczyta dwie liczby oraz znak określający operację (+, -, *, /). Na podstawie wprowadzonego operatora program powinien obliczyć wynik wybranego działania i wyświetlić go w konsoli. Program powinien obsługiwać następujące błędy: nieznany operator (zdefiniuj własny wyjątek `InvalidOperator`),



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



błędny format danych [FormatException] oraz dzielenie przez zero [DivideByZeroException].

Zadanie 9.6.

Napisz program, który pobiera od użytkownika liczbę całkowitą i oblicza jej pierwiastek kwadratowy. Jeżeli użytkownik poda liczbę ujemną, program powinien zgłosić [throw] i obsłużyć wyjątek `ArgumentOutOfRangeException`, informując, że nie można obliczyć pierwiastka z liczby ujemnej.

Uwaga! Metoda `Math.Sqrt()` zwraca wartość typu `double` i nie zgłasza wyjątku dla niepoprawnych argumentów (<0). W takich przypadkach wynik przyjmuje specjalne wartości, np. `Double.NaN` lub `Double.PositiveInfinity`. Aby w tym zadaniu wyjątek został zgłoszony, należy samodzielnie sprawdzić dane wejściowe i zgłosić wyjątek za pomocą `throw`.

Zadanie 9.7.

Napisz program, który próbuje otworzyć plik tekstowy o nazwie *dane.txt* z folderu `C:\Pliki` oraz wyświetlić jego zawartość w konsoli. Program powinien przechwycić wyjątek `DirectoryNotFoundException` w sytuacji, gdy wskazany katalog nie istnieje, `FileNotFoundException` – jeżeli plik o podanej nazwie nie zostanie odnaleziony, oraz `IOException`, gdy wystąpi błąd podczas operacji wejścia-wyjścia. Program powinien przechwycić te wyjątki i wyświetlić odpowiednie komunikaty.

Zadanie 9.8.

Napisz program, który odczyta liczby z pliku tekstowego *liczby.txt* i obliczy ich średnią arytmetyczną (każda liczba znajduje się w osobnej linii). Program powinien obsłużyć wyjątek `FileNotFoundException`, jeżeli plik nie istnieje oraz `FormatException`, jeżeli plik zawiera dane, których nie można przekonwertować na liczby. W przypadku wystąpienia błędu należy wyświetlić odpowiedni komunikat w konsoli.

Zadanie 9.9.

Napisz metodę `ObliczSrednia(int[] liczby)`, która przyjmuje w parametrze tablicę liczb całkowitych i zwraca średnią arytmetyczną jej elementów. W przypadku, gdy przekazana tablica ma wartość `null`, metoda powinna zgłosić wyjątek `NullReferenceException` z komunikatem „*Tablica nie może być niezainicjalizowana*”. Jeżeli natomiast tablica jest pusta, metoda powinna zgłosić wyjątek `InvalidOperationException` z komunikatem „*Tablica nie może być pusta*”. W metodzie `Main` należy utworzyć przykładową tablicę i przetestować działanie



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



metody ObliczSrednia dla różnych przypadków, w tym: poprawnie zainicjalizowanej tablicy z danymi, wartości null oraz pustej tablicy.

Zadanie 9.10.

Napisz program, który pobiera od użytkownika liczbę całkowitą. Program powinien sprawdzić, czy liczba mieści się w zakresie od 0 do 1000. W przypadku wprowadzenia wartości spoza tego zakresu należy zgłosić wyjątek `ArgumentOutOfRangeException` i poinformować użytkownika o błędnym wprowadzeniu danych.

Zadanie 9.11.

Napisz metodę `SprawdzHaslo(string haslo)`, która sprawdzi, czy długość przekazanego w parametrze hasła wynosi co najmniej 8 znaków. Jeżeli hasło jest krótsze, metoda powinna zgłosić własny wyjątek `TooShortPasswordException`. W metodzie `Main()` pobierz od użytkownika hasło, wywołaj metodę `SprawdzHaslo()`, przechwyc wyjątek i wyświetl odpowiedni komunikat.

Zadanie 9.12.

Utwórz klasę `Konto`, zawierającą właściwości `Saldo` (typu `decimal`) oraz `NrKonta` (typu `string`). Zaimplementuj metodę `Wyplac(decimal kwota)`, która zmniejsza saldo o podaną w parametrze wartość. Jeżeli na koncie nie ma wystarczających środków (saldo jest mniejsze od podanej kwoty), metoda powinna zgłosić wyjątek `InvalidOperationException` z komunikatem „*Niewystarczające środki na koncie*”.

W metodzie `Main()` utwórz instancję klasy `Konto`, wczytaj od użytkownika kwotę do wypłaty, a następnie wywołaj metodę `Wyplac()` dla poprawnej oraz błędnej kwoty. Obsłuż wyjątek i wyświetl odpowiedni komunikat.

Zadanie 9.13.

Zdefiniuj klasę `Produkt`, zawierającą właściwości `Nazwa`, `Cena` i `StanMagazynowy`. Zaimplementuj metodę `Sprzedaj(int liczbaSztuk)`, która w przypadku próby sprzedaży większej liczby produktów niż jest dostępna, zgłasza wyjątek `InvalidOperationException` z komunikatem „*Brak wystarczającej liczby produktów w magazynie*”.

W metodzie `Main()` utwórz obiekt klasy `Produkt`, pobierz od użytkownika liczbę sztuk do sprzedaży i obsłuż wyjątek, wyświetlając odpowiedni komunikat.

Zadanie 9.14.

Utwórz klasę `Pracownik` zawierającą właściwości `Imie`, `Stanowisko` oraz `Pensja`. W klasie należy zdefiniować metodę `ZmienPensje(double nowaPensja)`, która



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



w przypadku, gdy przekazana wartość jest mniejsza od zera, zgłasza wyjątek `ArgumentException` z komunikatem „*Pensja nie może być ujemna.*” W metodzie `Main()` należy obsłużyć ten wyjątek i wyświetlić odpowiedni komunikat.

Zadanie 9.15.

Napisz metodę `Multiply(int a, int b)`, zwracającą iloczyn dwóch liczb. Wewnątrz metody użyj instrukcji `checked`, aby sprawdzić, czy nie dochodzi do przepełnienia. Obsłuż wyjątek `OverflowException` w metodzie `Main()` i wyświetl komunikat: „*Błąd: wynik mnożenia przekroczył dopuszczalny zakres.*”

Pytania pomocnicze

- 1) Czym jest wyjątek i w jakich sytuacjach może wystąpić?
- 2) Co dzieje się, gdy w programie wystąpi wyjątek, który nie został obsłużony?
- 3) W jaki sposób można przechwycić kilka różnych typów wyjątków?
- 4) Jak można samodzielnie zgłosić wyjątek w programie?
- 5) W jakich przypadkach warto tworzyć własne klasy wyjątków i jak to zrobić?
- 6) Jaka jest rola bloku `finally` i kiedy jest on wykonywany?
- 7) W jaki sposób można zapisywać informacje o wyjątkach do pliku?
- 8) Jakie wyjątki mogą wystąpić podczas pracy z plikami i jak je obsłużyć?
- 9) Dlaczego nie należy przechwytywać wszystkich wyjątków za pomocą jednego bloku `catch(Exception)`?

Podsumowanie

W tym laboratorium przedstawiono zasady obsługi wyjątków i zarządzania błędami. Omówiono pojęcie wyjątku jako mechanizmu umożliwiającego reagowanie na błędy występujące podczas działania programu. Zaprezentowano strukturę bloków `try`, `catch`, `finally` oraz sposób ręcznego zgłaszania wyjątków przy użyciu instrukcji `throw`. Omówiono obsługę typowych wyjątków, takich jak `DivideByZeroException`, `FormatException`, `IndexOutOfRangeException` czy `OverflowException`. Dodatkowo przedstawiono zasady definiowania własnych klas wyjątków oraz ich praktyczne zastosowanie w programach.



Fundusze Europejskie
dla Rozwoju Społecznego



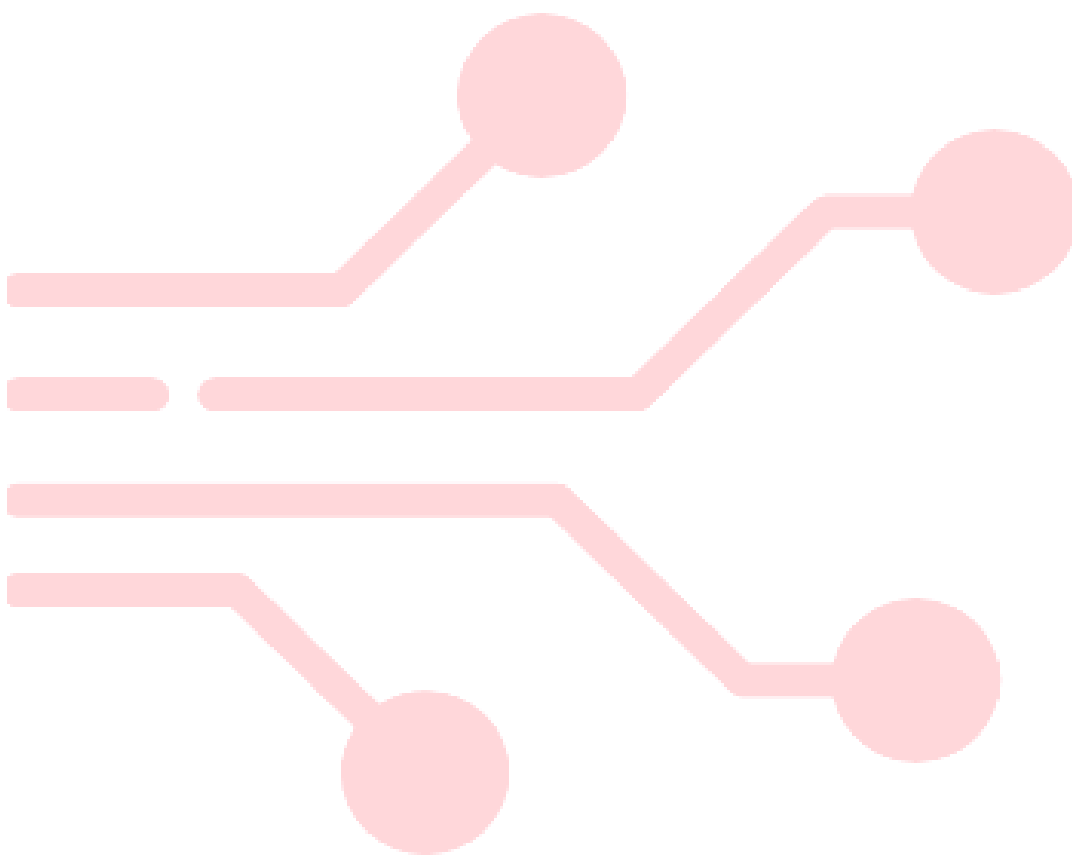
Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Literatura

- [1] Michaelis M., *C# 8.0: kompletny przewodnik dla praktyków*, Helion, Gliwice 2021. s. 229-241.
- [2] Albahari J., *C# 12 w pigułce. Kompendium programisty*, Helion, Gliwice, 2024. s. 201-208.
- [3] <https://learn.microsoft.com/pl-pl/dotnet/csharp/fundamentals/exceptions>



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską

