

# Programowanie pod Windows

## Zbiór zadań

Uwaga: zbiór zadań jest w fazie ciągłego rozwoju. Wszelkie prawa autorskie zastrzeżone. Dokument może być rozpowszechniany wyłącznie w celach edukacyjnych, z wyłączeniem korzyści materialnych.

**Wiktor Zychla**  
Instytut Informatyki  
Uniwersytetu Wrocławskiego

Wersja 2023.10.01



# Wprowadzenie

Szanowni Państwo!

Niniejszy zbiór zadań przeznaczony jest dla słuchaczy wykładu **Programowanie pod Windows .NET**, który mam przyjemność prowadzić w Instytucie Informatyki Uniwersytetu Wrocławskiego w kolejnych semestrach letnich od roku akademickiego 2002/2003. Celem wykładu jest zapoznanie słuchaczy z praktyką programowania systemów operacyjnych rodziny Windows.

Zbiór zadań stanowi uzupełnienie podręcznika, pozycji *Windows oczami programisty* [1], dostępnej w wersji akademickiej jako skrypt *Programowanie pod Windows*.

Zadania są pogrupowane w zestawy, ogłaszane na kolejnych zajęciach. Dodatkowe, niepunktowane zadania umieszczone w ostatniej części zbioru, stanowią zaproszenie dla Czytelnia do własnej pracy.

Rozwinięciem materiału wykładu **Programowanie pod Windows** jest prowadzony w semestrze zimowym wykład **Projektowanie aplikacji ASP.NET**, który jest w całości poświęcony podsystemowi ASP.NET, dedykowanemu rozwijaniu aplikacji internetowych. Z tego też powodu wykład **Programowanie pod Windows** świadomie całkowicie pomija ten obszar technologiczny.

Naturalną kontynuacją każdego wykładu technologicznego, w tym tego, jest również wykład dotyczący projektowania obiektowego. Język programowania i technologia są bowiem tylko narzędziami tworzenia aplikacji, ale można ich używać lepiej lub gorzej. Stąd serdecznie zachęcam do wysłuchania wykładów takich jak **Projektowanie obiektowe oprogramowania**, dla których miejsce w którym kończy się **Programowanie pod Windows** jest początkiem opowieści o tym jak współcześnie projektuje i wytwarza się oprogramowanie.

Wiktor Zychla, październik 2023  
wzychla@uwr.edu.pl



# Spis treści

<b>1</b>	<b>Zestaw 1, Język C# - rozgrzewka</b>	<b>7</b>
<b>2</b>	<b>Zestaw 2, Język C# - podstawowe elementy</b>	<b>9</b>
<b>3</b>	<b>Zestaw 3, Język C# - refleksja, typy generyczne</b>	<b>13</b>
<b>4</b>	<b>Zestaw 4, Język C# 3.0</b>	<b>17</b>
<b>5</b>	<b>Zestaw 5, Język C#4, C#5</b>	<b>21</b>
<b>6</b>	<b>Zestaw 6, System.Windows.Forms</b>	<b>23</b>
<b>7</b>	<b>Zestaw 7, System.Windows.Forms (2)</b>	<b>25</b>
<b>8</b>	<b>Zestaw 8, Windows Presentation Foundation</b>	<b>27</b>
<b>9</b>	<b>Zestaw 9, Elementy biblioteki standardowej</b>	<b>29</b>
<b>10</b>	<b>Zestaw A, Komunikacja z bazą danych</b>	<b>31</b>
<b>11</b>	<b>Zestaw B, Projekt podsumowujący</b>	<b>33</b>
<b>A</b>	<b>Varia</b>	<b>35</b>
A.1	Poziom łatwy . . . . .	35
A.1.1	Dziwna kolekcja . . . . .	35
A.1.2	Rekurencyjne zmienne statyczne . . . . .	35
A.1.3	Rozterki kompilatora . . . . .	36
A.1.4	Składowe prywatne . . . . .	36
A.1.5	Nieoczekiwany błąd kompilacji . . . . .	36
A.1.6	Wywołanie metody na pustej referencji . . . . .	37
A.2	Poziom średniozaawansowany . . . . .	37
A.2.1	Zamiana wartości dwóch zmiennych . . . . .	37
A.2.2	Operacje na zbiorach (1) . . . . .	37
A.2.3	Operacje na zbiorach (2) . . . . .	38
A.2.4	Operacje na zbiorach (3) . . . . .	38
A.3	Poziom trudny . . . . .	38
A.3.1	Specyficzne ograniczenie generyczne . . . . .	38
A.3.2	Zasięg zmiennej w domknięciu . . . . .	39



# Rozdział 1

## Zestaw 1, Język C# - rozgrzewka

Liczba punktów do zdobycia: **6/6**

1. (**1p**) Napisać program, który wyznacza zbiór wszystkich liczb naturalnych 1 a 100000, które są podzielne zarówno przez każdą ze swoich cyfr z osobna jak i przez sumę swoich cyfr.
2. (**1p**) Przygotować rozwiązanie (Solution) które składa się co najmniej z czterech projektów (Project): dwu aplikacji konsolowych i dwu bibliotek.

W każdej z bibliotek umieścić po jednej klasie z jedną metodą. Dodać referencje do bibliotek z każdej aplikacji konsolowej. Nie dodawać referencji pomiędzy aplikacjami konsolowymi.

W każdej z aplikacji konsolowych napisać fragment kodu, który wywołuje kod z obu bibliotek.

Pokazać jak z poziomu Visual Studio uruchomić jedną z aplikacji konsolowych, potem drugą (Set as startup project...) a potem obie naraz.

Pokazać jak w każdym z tych sposobów uruchomienia kodu można umieszczać pułapki w kodzie i debugować kod.

3. (**1p**) Zdokumentować (przez umieszczenie odpowiednich komentarzy w kodzie) jeden dowolny program z bieżącej sekcji.

Wygenerować dokumentację w postaci pliku XML podczas kompilacji. Użyć narzędzia SandCastle Help File Builder (<https://github.com/EWSoftware/SHFB>) do zbudowania pomocy w obsługiwanych przez SandCastle stylach (np. Website).

4. (**1p**) Napisać w C# dowolny program demonstrujący użycie klas (metod, pól, propercji, indeksów, delegacji i zdarzeń) oraz podstawowych konstrukcji składniowych (pętle, instrukcje warunkowe, `switch`) i zdekompilować go za pomocą narzędzia **ILSpy** (<http://ilspy.net/>).

Otrzymany kod skompilować (`ilasm`), aby otrzymać plik wynikowy. Plik ten następnie zdekompilować na powrót do języka C#.

Porównać otrzymane w ten sposób pliki z kodem źródłowym. Jak objawiają się i z czego wynikają różnice?

5. (**2p**) Zaimplementować klasę siatki dwuwymiarowej, `Grid`, z dwoma indeksami:

- jednowymiarowym, zwracającym listę elementów zadanego wiersza tablicy, tak aby klient klasy mógł napisać:

```
...
Grid grid = new Grid( 4, 4 );
int[] rowdata = grid[1]; // akcesor "get"
```

- dwuwymiarowym, zwracającym określony element tablicy, tak aby klient klasy mógł napisać:

```
...
Grid grid = new Grid( 4, 4 );

elem[2, 2] = 5;           // akcesor "set"
int elem = grid[1, 4]; // akcesor "get"
```

Oba indeksy powinny przyjmować jako parametry liczby całkowite. Konstruktor klasy powinien przyjmować jako parametry liczbę wierszy i liczbę kolumn siatki.



## Rozdział 2

# Zestaw 2, Język C# - podstawowe elementy

Liczba punktów do zdobycia: **10/16**

1. (**1p**) Zademonstrować w praktyce następujące kwalifikatory dostępu do składowych (na przykładzie dostępu do pól lub metod)

- `public`
- `protected`
- `internal`
- `private`

2. (**2p**) Zademonstrować w praktyce i rozumieć sens następujących elementów języka. Jeśli w nawiasie po elemencie języka występuje kilka możliwych elementów do których może on się odnosić, proszę wybrać co najmniej jedną z propozycji, ale niekoniecznie wszystkie.

- modyfikator `static` dla klas
- modyfikator `static` dla składowych klas (pól, metod)
- modyfikator `sealed` dla klas
- modyfikator `abstract` dla klas
- modyfikator `abstract` dla składowych klas (metod)
- słowa kluczowe `virtual` i `override` dla składowych klas (metod)
- słowo kluczowe `partial` w definicji klasy
- słowo kluczowe `readonly` w deklaracji pola klasy
- modyfikatory `in`, `ref` oraz `out` na liście parametrów metod

3. (**1p**) Zaprezentować w praktyce mechanizm przeciążania sygnatur funkcji (funkcje o tej samej nazwie ale różnych sygnaturach).

Czy typ zwracany z dwóch lub więcej funkcji przeciążonych może być różny czy zawsze musi być taki sam?

Pokazać jak funkcja przeciążona może wywoływać inną funkcję przeciążoną zamiast dostarczać własnej implementacji. Pokazać jak funkcja może wywołać funkcję z klasy bazowej zamiast dostarczać własnej implementacji. Pokazać jak przeciążać konstruktory klasy.

4. **(2p)** Czym różni się mechanizm finalizerów (zwanymi dawniej destruktorami) od mechanizmu uwalniania zasobów za pomocą implementacji interfejsu `IDisposable`?

Zaprezentować oba w praktyce: przygotować klasę która ma finalizer i inną klasę implementującą interfejs `IDisposable`. W obu podejściach wywołać `Console.WriteLine` z metody sprzątajacej.

Zaprezentować lukier syntaktyczny opakowujący użycie obiektu implementującego `IDisposable` w blok ze słowem kluczowym `using`.

Zaobserwować, że w przypadku interfejsu `IDisposable` programista ma pełną kontrolę nad momentem w którym wykonuje się metoda `Dispose`. Zaobserwować że programista nie ma wpływu na to kiedy wykona się finalizer klasy.

Czy można wymusić wywołanie metody sprzątajacej pamięć (odsćmiecać)? Czy to dobry pomysł, żeby wymuszać to we własnym kodzie?

5. **(1p)** Pokazać jak definiować właściwości (ang. *properties*)
- właściwość z polem kopii zapasowej (ang. *property with backing field*)
  - właściwość implementowana automatycznie (ang. *auto-implemented property*)
6. **(2p)** Rozszerzyć poprzedni przykład o demonstrację właściwości posiadających skutki uboczne. Formalnie, niech będzie dana klasa

```
public class Person
{
    public string Name { get; set; }
    public string Surname { get; set; }
}
```

Klasę zmodyfikować tak, żeby udostępniała zdarzenie (ang. *event*) informujące subskrybenta o tym że zmieniła się wartość którejś z właściwości.

Czy potrzebne są do tego dwa osobne zdarzenia? Które podejście jest lepsze - jedno zdarzenie o ogólniejszej sygnaturze czy wiele osobnych zdarzeń, po jednym zdarzeniu dla każdego pola?

Formalnie, klient chciałby z powiadamiania korzystać w sposób przedstawiony poniżej - tak zmodyfikować kod klasy `Person` żeby było to możliwe.

Zaprezentować dwa warianty

- powiadomienie pojawia się zawsze kiedy nadana jest wartość właściwości (formalnie - kiedy wywołuje się akcesor `set`)
- powiadomienie pojawia się tylko wtedy kiedy nowa wartość pola jest różna od poprzedniej

```
static void Main( string[] args )
{
    Person person = new Person();
    person.PropertyValueChanged += Person_PropertyValueChanged;
    person.Name = "Jan";

    Console.ReadLine();
}

private static void Person_PropertyValueChanged(
    object source,
    string propertyName,
    object propertyValue )
```

```
{  
    Console.WriteLine(  
        "właściwość {0}, nowa wartość {1}",  
        propertyName,  
        propertyValue );  
}
```

7. (**1p**) Pokazać jak przeciąża się operatory - zdefiniować klasę wektora dwuwymiarowego i dodać do niej standardowe operatory arytmetyki na wektorach.



## Rozdział 3

# Zestaw 3, Język C# - refleksja, typy generyczne

Liczba punktów do zdobycia: 10/26

1. **(2p)** W trakcie wykładu przedstawiono szkic ogólnego generatora zapytań SQL/struktury XML/struktury JSON dla dowolnych obiektów. Państwa zadaniem będzie odtworzyć ten przykład na przykładzie generatora struktury XML. W pierwszym podejściu - przy pomocy interfejsu za pomocą którego można z obiektu dla którego generuje się XML pobrać informację o jego strukturze.

Formalnie: generator to klasa z metodą generującą

```
public class XMLGenerator
{
    public string GenerateXML( IClassInfo dataObject )
    {
        // uzupełnić implementację
        throw new NotImplementedException();
    }
}
```

W celu pobrania informacji o strukturze obiektu dla którego ma zostać wygenerowany XML, generator wykorzysta interfejs `IClassInfo`:

```
public interface IClassInfo
{
    string[] GetFieldNames();
    object GetFieldValue( string fieldName );
}
```

Proszę zwrócić uwagę jak zaprojektowany jest ten interfejs: jedna z jego metod zwraca listę wszystkich pól klasy, druga zwraca wartość konkretnego pola.

Jeśli teraz ktoś chciałby wygenerować XML dla zadanej klasy, na przykład takiej:

```
public class Person
{
    public string Name { get; set; }
    public string Surname { get; set; }
}
```

to po pierwsze, klasa musiałaby implementować interfejs `IClassInfo`

```

public class Person : IClassInfo
{
    public string Name { get; set; }
    public string Surname { get; set; }

    public string[] GetFieldNames()
    {
        return new[] { "Name", "Surname" };
    }

    public object GetFieldValue( string fieldName )
    {
        switch ( fieldName )
        {
            case "Name":
                return this.Name;
            case "Surname":
                return this.Surname;
            default:
                return null;
        }
        throw new NotImplementedException();
    }
}

```

a po drugie - należałoby właśnie (co jest treścią zadania!) zaimplementować metodę `GenerateXML` generatora.

Wtedy można by napisać fragment kodu:

```

Person person =
    new Person()
    {
        Name = "Jan",
        Surname = "Kowalski"
    };

XMLGenerator generator = new XMLGenerator();

string xml = generator.GenerateXML( person );

```

2. (2p) W drugim podejściu do generatora XML luzuje się wymagania - zakładamy że klasa która ma być zapisywana do XML (w poprzednim przykładzie klasa `Person`) nie musi implementować żadnego interfejsu.

Jak w takim razie generator ma dostać się do listy pól w klasie i wartości konkretnych pól?

Za pomocą refleksji.

Formalnie, zmieniamy definicję generatora

```

public class XMLGenerator
{
    public string GenerateXML( object dataObject )
    {
        // uzupełnić implementację
        throw new NotImplementedException();
    }
}

```

i nadal chcemy móc napisać

```

Person person =
    new Person()
    {
        Name = "Jan",
        Surname = "Kowalski"
    }

```

```

    };

XMLGenerator generator = new XMLGenerator();

string xml = generator.GenerateXML( person );

```

3. (1p) Generator oparty na refleksji ma pewną wadę - refleksja podczas enumeracji składowych klasy uwzględnia wszystkie składowe o takiej samej charakterystyce (na przykład wszystkie pola publiczne). A co jeśli chciałoby się **pominać** jakieś pole?

Należałoby je oznaczyć atrybutem.

Formalnie, chcemy móc zdefiniować atrybut pozwalający pominąć pole podczas generacji:

```

public class Person
{
    public string Name { get; set; }

    [IgnoreInXML]
    public string Surname { get; set; }
}

```

a kod generatora zmodyfikować w taki sposób żeby podczas enumeracji składowych klasy wykrywał właściwości znakowane tym konkretnym atrybutem i pomijał je w trakcie generowania XML

4. (1p) Zademonstrować w działaniu metody `ConvertAll`, `FindAll`, `ForEach`, `RemoveAll` i `Sort` klasy `List<T>` używając anonimowych delegacji o odpowiednich sygnaturach.
5. (1p) We własnej klasie `ListHelper` zaprogramować statyczne metody `ConvertAll`, `FindAll`, `ForEach`, `RemoveAll` i `Sort` o semantyce zgodnej z odpowiednimi funkcjami z klasy `List<T>` i sygnaturach rozszerzonych względem odpowiedników o instancję obiektu `List<T>` na którym mają operować.

```

public class ListHelper
{
    public static List<TOutput> ConvertAll<T, TOutput>(
        List<T> list,
        Converter<T, TOutput> converter );
    public static List<T> FindAll<T>(
        List<T> list,
        Predicate<T> match );
    public static void ForEach<T>( List<T>, Action<T> action );
    public static int RemoveAll<T>(
        List<T> list,
        Predicate<T> match );
    public static void Sort<T>(
        List<T> list,
        Comparision<T> comparison );
}

```

6. (3p) Napisać klasę `BinaryTreeNode<T>`, która będzie modelem dla węzła drzewa binarnego. Węzeł powinien przechowywać informację o danej typu `T` oraz swoim lewym i prawym synu.

Klasa powinna zawierać dwa enumeratory, dla przechodzenia drzewa w głąb i wszerz, zaprogramowane z wykorzystaniem słowa kluczowego `yield`.

*Wskazówka: choć implementacja bez yield może wydawać się trudna, w rzeczywistości jest również stosunkowo prosta. Należy wykorzystać pomocnicze struktury danych, przechowującą informację o odwiedzanych węzłach. Każdy MoveNext ogląda bieżący węzeł, a jego podwęzły, lewy i prawy, umieszcza w pomocniczej strukturze danych. Każdy Current*

*usuwa bieżący węzeł z pomocniczej struktury i zwraca jako wynik. Strukturę danych dobiera się w zależności od tego czy chce się implementować przechodzenie wszerek czy wgłęb (jakie struktury danych należy wybrać dla każdego z tych wariantów?)*



## Rozdział 4

# Zestaw 4, Język C# 3.0

Liczba punktów do zdobycia: **10/36**

*Uwaga! W zadaniach w których polecenie brzmi "dany jest plik tekstowy ..." należy sobie we własnym zakresie przygotować taki przykładowy plik tekstowy.*

1. (1p) Zaimplementować metodę `bool IsPalindrome()` rozszerzającą klasę `string`. Implementacja powinna być niewrażliwa na białe znaki i znaki przestankowe występujące wewnątrz napisu ani na wielkość liter. Klient tej metody powinien wywołać ją tak:

```
string s = "Kobyła ma mały bok.";
bool ispalindrome = s.IsPalindrome();
```

2. (1p) Dany jest plik tekstowy zawierający zbiór liczb naturalnych w kolejnych liniach. Napisać wyrażenie LINQ, które odczyta kolejne liczby z pliku i wypisze tylko liczby większe niż 100, posortowane malejąco.

```
from liczba in [liczby]
where ...
orderby ...
select ...
```

Przeformułować wyrażenie LINQ na ciąg wywołań metod LINQ to Objects:

```
[liczby].Where( ... ).OrderBy( ... )
```

Czym różnią się parametry operatorów **where/orderby** od parametrów funkcji **Where, OrderBy**?

3. (1p) Dany jest plik tekstowy zawierający zbiór nazwisk w kolejnych liniach. Napisać wyrażenie LINQ, które zwróci zbiór **pierwszych** liter nazwisk uporządkowanych w kolejności alfabetycznej. Na przykład dla zbioru (Kowalski, Malinowski, Krasicki, Abaciki) wynikiem powinien być zbiór (A, K, M).

*Wskazówka: zgodnie z tytułem zadania użyć operatora `GroupBy`*

4. (**1p**) Napisać wyrażenie LINQ, które dla zadanego foldera wyznaczy sumę długości plików znajdujących się w tym folderze.

Do zbudowania sumy długości plików użyć funkcji **Aggregate**. Listę plików w zadanym folderze wydobyć za pomocą odpowiednich metod z przestrzeni nazw **System.IO**.

5. (**1p**) Dane są dwa pliki tekstowe, pierwszy zawierający zbiór danych osobowych postaci (Imię, Nazwisko, PESEL), drugi postaci (PESEL, NumerKonta). Kolejność danych w zbiorach jest przypadkowa.

Napisać wyrażenie LINQ, które połączy oba zbiory danych i zbuduje zbiór danych zawierający rekordy postaci (Imię, Nazwisko, PESEL, NumerKonta). Do połączenia danych należy użyć operatora **join**.

6. (**2p**) Rejestr zdarzeń serwera IIS ma postać pliku tekstowego, w którym każda linia ma postać:

```
08:55:36 192.168.0.1 GET /TheApplication/WebResource.axd 200
```

gdzie poszczególne wartości oznaczają czas, adres klienta, rodzaj żądania HTTP, nazwę zasobu oraz status odpowiedzi.

Napisać aplikację która za pomocą jednego (lub wielu) wyrażeń LINQ wydobędzie z przykładowego rejestru zdarzeń IIS listę adresów IP trzech klientów, którzy skierowali do serwera aplikacji największą liczbę żądań.

Wynikiem działania programu powinien być przykładowy raport postaci:

```
12.34.56.78 143
23.45.67.89 113
123.245.167.289 89
```

gdzie pierwsza kolumna oznacza adres klienta, a druga liczbę zarejestrowanych żądań.

7. (**1p**) Listy generyczne ukonkretniamy typem elementów:

```
List<int>    listInt;
List<string> listString;...
```

Z drugiej strony, w C# 3.0 mamy typy anonimowe, które nie są nigdy jawnie nazwane:

```
var item = new { Field1 = "The value", Field2 = 5 };
Console.WriteLine( item.Field1 );
```

Czy możliwe jest zadeklarowanie i korzystanie z listy generycznej elementów typu anonimowego?

```
var item = new { Field1 = "The value", Field2 = 5; };
List<?> theList = ?
```

W powyższym przykładzie, jak utworzyć listę generyczną, na której znalazłby się element **item** w taki sposób, by móc następnie do niej dodawać nowe obiekty takiego samego typu?

*Obiekty typu anonimowego mają ten sam typ, jeśli mają tę samą liczbę składowych tego samego typu w tej samej kolejności.*

8. (2p) Cechą charakterystyczną anonimowych delegacji, bez względu na to czy zdefiniowano je przy użyciu słowa kluczowego **delegate**, czy też raczej jako lambda wyrażenia, jest brak "nazwy", do której można odwołać się w innym miejscu kodu.

Zadanie polega na zaproponowaniu takiego tworzenia anonimowych delegacji, żeby w jednym wyrażeniu możliwa była rekursja. W szczególności, poniższy fragment kodu powinien się kompilować i zwracać wynik zgodny ze specyfikacją.

```
List<int> list = new List<int>() { 1,2,3,4,5 };

foreach ( var item in
    list.Select( i => [...] ) )

    Console.WriteLine( item );
}
```

W powyższym fragmencie kodu, puste miejsce ([...]) należy zastąpić definicją ciała anonimowej delegacji określonej rekursywnie:

$$f(i) = \begin{cases} 1 & i \leq 2 \\ f(i-1) + f(i-2) & i > 2 \end{cases}$$

*Wskazówka* W języku C# można z powodzeniem zaimplementować operator punktu stałego **Y**, wykorzystywany do definicji funkcji rekurencyjnych. Zadanie to można rozwiązać więc definiując taki operator i za jego pomocą implementując funkcję rekurencyjną. Istnieje jednak zaskakujący i o wiele prostszy sposób rozwiązania wymagający jednak trochę nagięcia specyfikacji. Oba rozwiązania będą przyjmowane.



## Rozdział 5

# Zestaw 5, Język C#4, C#5

Liczba punktów do zdobycia: 8/44

1. (3p) Przeprowadzić testy porównawcze szybkości kodu, w którym metoda będzie miała parametr raz typu konkretnego, a drugi raz - dynamicznego. Jak bardzo wolniejsze jest wykonywanie kodu dynamicznego w tym konkretnym przypadku?

```
public int DoWork1( int x, int y )
{
    // jakieś obliczenia na x i y, np. x + y
}

public dynamic DoWork2( dynamic x, dynamic y )
{
    // te same obliczenia na x i y
}
```

Do przeprowadzenia testów użyć biblioteki **Benchmark.NET** (<https://benchmarkdotnet.org/>), którą proszę zainstalować w projekcie za pomocą menedżera pakietów NuGet.

Czy i jak wyniki zmieniają się w zależności od tego jak bardzo złożone jest wyrażenie używające zmiennych?

2. (3p) Zademonstrować w praktyce możliwość dynamicznej implementacji operacji na obiekcie dziedziczącym z **DynamicObject**. Konkretnie - pokazać jak używać

- TryGetMember, TrySetMember
- TryGetIndex, TrySetIndex
- TryInvoke, TryInvokeMember
- TryUnaryOperation, TryBinaryOperation

3. (1p) Powtórzyć przykład z wykładu, w którym zaprezentowano w jaki sposób można osiągnąć efekt, w którym "oczekiwalny" (ang. *awaitable*) może być dowolny obiekt. Konkretnie, chodzi o przykład w którym można napisać

```
Console.WriteLine( "1" );

await 2000; // (1)

Console.WriteLine( "1" );
```

gdzie instrukcja (1) powoduje dwusekundowe oczekiwanie przed wykonaniem kolejnej linii.

4. (1p) Na podobieństwo poprzedniego zadania - dostarczyć takiego rozszerzenia "oczekiwanego" napisu, które spowoduje, że

```
Console.WriteLine( await "https://www.google.com" ); //
```

spowoduje pobranie zawartości witryny spod wskazanego adresu i zwrócenie napisu - zawartości witryny. Formalnie, chodzi o implementację metody

```
public static TaskAwaiter<string> GetAwaiter( this string url )  
{  
    // uzupełnić  
}
```

Do pobrania zawartości witryny o wskazanym adresie użyć obiektu **HttpClient**. Uwaga! Użyć go poprawnie, w szczególności:

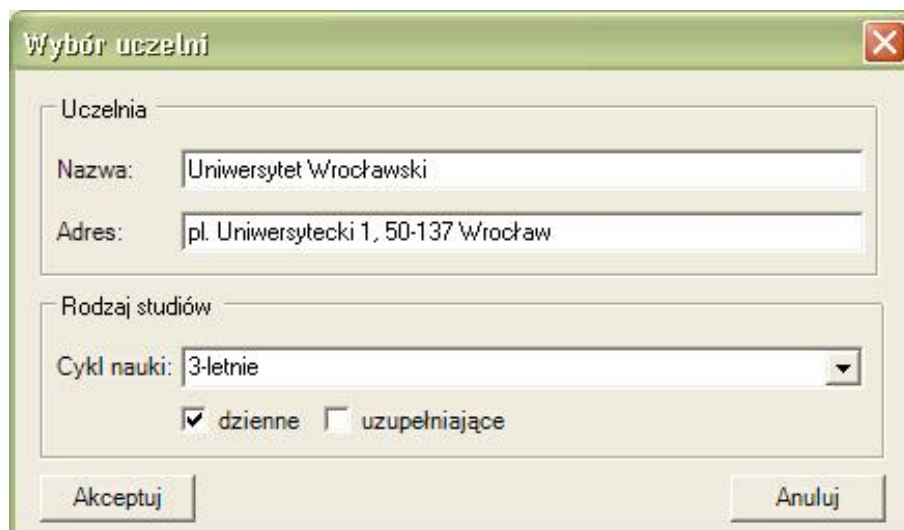
<https://www.aspnetmonsters.com/2016/08/2016-08-27-httpclientwrong/>

## Rozdział 6

# Zestaw 6, System.Windows.Forms

Liczba punktów do zdobycia: **6/50**

1. **(2p)** Napisać program, który odtworzy następujący wygląd okna z rysunku 8.1.  
Okno zawiera dwie ramki grupujące (*GroupBox*). Pierwsza ramka zawiera dwa pola tekstowe (*TextBox*), druga zawiera pole wyboru (*ComboBox*) oraz dwa przyciski stanu (*CheckBox*).  
Lista rozwijalna pola wyboru powinna być wypełniona przykładowymi nazwami.  
Po wybraniu przez użytkownika przycisku **Akceptuj**, wybór powinien zostać zaprezentowany w oknie informacyjnym (rysunek 8.2).  
Naciśnięcie przycisku **Anuluj** powinno zakończyć program.  
*Uwaga! Komunikat w oknie informacyjnym zależy oczywiście od danych wprowadzonych przez użytkownika na formularzu głównym*
2. **(2p)** Napisać program, który zademonstruje działanie następujących formantów biblioteki standardowej
  - **MenuStrip**
  - **ContextMenuStrip**
  - **ToolStrip**
  - **ToolTip**
  - **TabControl**
  - **SplitContainer**
  - **Panel**
  - **FlowLayoutPanel**
3. **(1p)** Napisać program, który zademonstruje działanie następujących formantów biblioteki standardowej
  - **OpenFileDialog**
  - **SaveFileDialog**
  - **FolderBrowserDialog**
4. **(1p)** Pokazać jak w aplikacji korzystać z pliku konfiguracyjnego aplikacji - do pliku konfiguracyjnego zapisać parametry typu **string** (literal), **int** (liczba) oraz **bool** (wartość logiczna) a w aplikacji poprawnie je odczytać i pokazać wartości.



Wybór uczelni

Uczelnia

Nazwa: Uniwersytet Wrocławski

Adres: pl. Uniwersytecki 1, 50-137 Wrocław

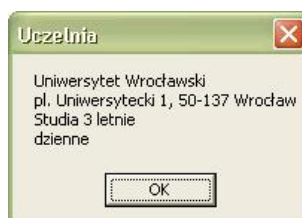
Rodzaj studiów

Cykl nauki: 3-letnie

☒ dzienne ☐ uzupełniające

Akceptuj Anuluj

Rysunek 6.1: Wygląd okna do zadania [1]



Uczelnia

Uniwersytet Wrocławski  
pl. Uniwersytecki 1, 50-137 Wrocław  
Studia 3 letnie  
dzienne

OK

Rysunek 6.2: Informacja dla użytkownika do zadania [1]



## Rozdział 7

# Zestaw 7, System.Windows.Forms (2)

Liczba punktów do zdobycia: 8/58

1. (3p) Przygotować aplikację, która wykorzystuje omówiony na wykładzie podsystem GDI+ do rysowania w oknie zegara analogowego prezentującego bieżący czas, zgodny z zegarem systemowym.

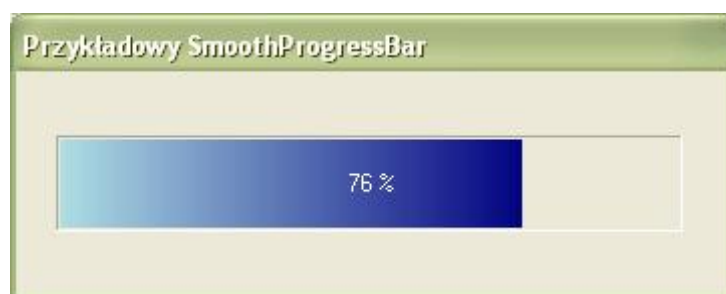
Rysowany widok powinien poprawnie dostosowywać się do wielkości okna podczas zmiany jego rozmiarów przez użytkownika.

2. (2p) Zaimplementować własny komponent `SmoothProgressBar`, który będzie imitować zachowanie standardowego komponentu `ProgressBar` (pasek postępu).

Komponent powinien mieć co najmniej 3 właściwości: `Min`, `Max` i `Value`, pozwalające określić odpowiednio minimalną, maksymalną i bieżącą wartość paska postępu. Mając te informacje, `SmoothProgressBar` w zdarzeniu `Paint` powinien rysować gładki (w przeciwieństwie do oryginalnego, który jest złożony z "kafelków") pasek postępu o odpowiedniej długości (według zadanych proporcji).

3. (2p) Zademonstrować w praktyce działanie klasy `BackgroundWorker` oraz jej zdarzenia `ProgressChanged` do delegowania długiego zadania do przetwarzania w tle.

Formalnie - wątek w tle niech testuje jakiś zakres liczb na przykład testem pierwszości. Zakres dobrać tak, aby obliczenia trwały nie krócej niż kilkanaście sekund. Postęp obliczeń powinien być raportowany w interfejsie użytkownika za pomocą formantu paska postępu.



Rysunek 7.1: Przykładowy `SmoothProgressBar`

Dla porównania - przygotować wersję która zamiast `BackgroundWorker` użyje po prostu wątku (obiekt typu `Thread`) który w trakcie obliczeń spróbuje aktualizować pasek postępu. Jaka trudność pojawia się w tym drugim podejściu (jest to również odpowiedź na pytanie co wnosi `BackgroundWorker` w stosunku do takiego naiwnego podejścia)?

4. (1p) Zaprezentować na niewielkim przykładzie zastosowanie rozszerzeń języka w obszarze programowania asynchronicznego (`async/await`).

Bardziej formalnie - pokazać że w aplikacji okienkowej użycie nieblokującej metody asynchronicznej `HttpClient::ReadStringAsync` do pobrania zawartości z zewnętrznego zasobu sieciowego nie spowoduje zablokowania wątku głównego aplikacji, w którym przetwarzana jest pętla obsługi komunikatów. W tej samej aplikacji zademonstrować synchroniczne, blokujące wywołanie metody `WebClient::DownloadString`.

## Rozdział 8

# Zestaw 8, Windows Presentation Foundation

Liczba punktów do zdobycia: **5/63**

1. (**2p**) Napisać program, który odtworzy następujący wygląd okna z rysunku 8.1. Jest to identyczna specyfikacja jak jedno z zadań z poprzednich zestawów, różnica dotyczy technologii - tym razem jest to WPF.

Okno zawiera dwie ramki grupujące (*GroupBox*). Pierwsza ramka zawiera dwa pola tekstowe (*TextBox*), druga zawiera pole wyboru (*ComboBox*) oraz dwa przyciski stanu (*CheckBox*).

Lista rozwijalna pola wyboru powinna być wypełniona przykładowymi nazwami.

Po wybraniu przez użytkownika przycisku **Akceptuj**, wybór powinien zostać zaprezentowany w oknie informacyjnym (rysunek 8.2).

Naciśnięcie przycisku **Anuluj** powinno zakończyć program.

2. (**3p**) Napisać najprostszą możliwą wersję gry kółko-krzyżyk w WPF. Interfejs zbudować przy pomocy komponentu `Grid`, który należy odpowiednio podzielić na 3 wiersze i 3 kolumny równego rozmaru (`SharedSizeGroup`). W każdej komórce grid'a umieścić przycisk, który będzie odpowiednio reagować na zdarzenie kliknięcia.

W jednym dodatkowym wierszu siatki umieścić komponent który zajmie całą szerokość wiersza (`ColumnSpan`) i będzie prezentować informacje na temat gry.

Wybór uczelni

Uczelnia

Nazwa: Uniwersytet Wrocławski

Adres: pl. Uniwersytecki 1, 50-137 Wrocław

Rodzaj studiów

Cykl nauki: 3-letnie

☒ dzienne ☐ uzupełniające

Akceptuj Anuluj

Rysunek 8.1: Wygląd okna do zadania [1]

Uczelnia

Uniwersytet Wrocławski  
pl. Uniwersytecki 1, 50-137 Wrocław  
Studia 3 letnie  
dzienne

OK

Rysunek 8.2: Informacja dla użytkownika do zadania [1]

## Rozdział 9

# Zestaw 9, Elementy biblioteki standardowej

Liczba punktów do zdobycia: 10/73

1. (1p) Zademonstrować użycie następujących klas do obsługi strumienia

- **FileStream**
- **StreamReader**, **StreamWriter**
- **BinaryReader**, **BinaryWriter**
- **StringBuilder**, **StringWriter**

Zademonstrować właściwość podsystemu strumieni, w której parametrem konstruktora strumienia może być inny strumień (taki wzorzec organizacji klas nosi nazwę **Dekorator**): napisać program, który zawartość wskazanego pliku tekstowego zapisze do **zaszyfrowanego** algorytmem AES **skompresowanego** strumienia GZip (klasy **CryptoStream** i **GZipStream**).

W dalszej części kodu pokazać jak odczytać dane z takiego pliku.

2. (3p) Napisać konsolowy program, który rozwiązuje klasyczny problem golibrody lub problem "palaczy tytoniu" za pomocą którejkolwiek z metod synchronizacji wątków udostępnianej przez bibliotekę standardową (semafony, muteksy, zdarzenia, sekcja krytyczna).
3. (1p) Zademonstrować działanie klas **FtpWebRequest**, **HttpWebRequest**, **WebClient**, **HttpClient**, **HttpListener**, **TcpListener**, **TcpClient**, **SmtpClient**.

Zwrócić uwagę na te funkcje z interfejsów powyższych klas, których metody pobierania danych są zaimplementowane jako asynchroniczne (zwracają **Task**).

4. (1p) Napisać program, który korzystając z informacji z odpowiedniej instancji obiektu **CultureInfo** wypisze pełne i skrótowe nazwy miesięcy i dni tygodnia oraz bieżącą datę w językach: angielskim, niemieckim, francuskim, rosyjskim, arabskim, czeskim i polskim.

Uwaga: jeśli konsola tekstowa nie obsługuje pewnych czcionek to zamiast konsoli tekstowej użyć okna informacyjnego konsoli okienkowej (**MessageBox.Show**).

5. (2p) Napisać usługę systemową (*System Service*), która będzie co minutę zapisywać listę uruchomionych aplikacji do pliku tekstowego.

*Uwaga! Po skompilowaniu usługa musi zostać zarejestrowana w systemie za pomocą programu `installutil.exe`. Zarządzanie usługami odbywa się z poziomu panelu **Zarządzanie komputerem**, sekcja **Usługi i aplikacje**.*

6. (1p) Umieścić dowolny plik w zasobach aplikacji (w projekcie plik powinien mieć właściwość *Embedded Resource*). Następnie napisać klasę, która po podaniu nazwy zasobu umożliwi wydobyć pliku z zasobów zestawu.

Osadzanie plików (tekstowych, binarnych) w zasobach aplikacji przydaje się wtedy kiedy aplikacja jest dystrybuowana do środowiska klienckiego. Zamiast plików wykonywalnych i dodatkowych plików zasobów, klient dostaje pliki wykonywalne w zasobach których zaszyte są pliki z danymi.

7. (1p) Nauczyć się korzystać z którejś z bibliotek do logowania informacji diagnostycznych (np. **log4net**, **nlog** czy **serilog**). Pokazać jak konfigurować sposób odkładania informacji diagnostycznych (konsola/plik itp.)

## Rozdział 10

# Zestaw A, Komunikacja z bazą danych

Liczba punktów do zdobycia: 11/84

1. (1p) Zainstalować SQL Server w dowolnej wersji dla programisty (Developer, Express, LocalDB). Zorientować się w dokumentacji czym charakteryzują się poszczególne wersje.
2. (1p) Przygotować bazę danych Microsoft SQL Server zawierającą dane osobowe i adresy przykładowej grupy studentów.

Model bazy danych zawiera trzy tabele

- tabelę **Student** z polami ID, Imię, Nazwisko, DataUrodzenia
- tabelę **Adres** z polami ID, Ulica, NrDomu, NrMieszkania, KodPocztowy, ID\_MIEJSCOWOSC
- tabelę **Miejscowosc** z polami ID, Nazwa

Tabele **Student** i **Adres** powinny być połączone relacją wiele-wiele (to wymaga pomocniczej tabeli!). Tabele **Adres** i **Miejscowosc** powinny być połączone relacją jeden-wiele

3. (2p) Użyć ADO.NET do połączenia do bazy danych, pobierania danych, dodawania, modyfikacji i usuwania. Ściśle - pokazać pracę z bazą danych za pomocą obiektów **SqlConnection**, **SqlCommand**, **SqlDataReader**.

W przypadku dodawania danych - oprogramować scenariusz dodawania studenta z adresem i miejscowością. Formalnie - przygotować funkcję o takiej liczbie parametrów jaka jest niezbędna do rejestracji pełnych danych (Imię, Nazwisko, DataUrodzenia, Ulica, NrDomu, ..., Nazwa):

- funkcja sprawdza czy osoba o podanym imieniu, nazwisku i dacie urodzenia już istnieje w bazie, jeśli tak to pomija dodawanie
- funkcja sprawdza czy w tabeli Miejscowości znajduje się miejscowość o podanej nazwie. Jeśli nie - dodaje, jeśli tak, pobiera identyfikator
- funkcja używa identyfikatora miejscowości żeby zarejestrować adres
- funkcja dodaje dane osobowe studenta i powiązanie studenta z nowo dodanym adresem

4. (**1p**) W poprzednim zadaniu kryje się pewien problem - w sytuacji kiedy funkcja wykonuje się równolegle dwa lub więcej razy, mimo sprawdzenia może zdarzyć się, że w bazie pojawią się duplikaty (te same dane w dwu różnych rekordach).

W jakich przypadkach mogłoby zdarzyć się że mimo sprawdzenia, te same dane zostaną zarejestrowane wiele razy? Jak najlepiej zabezpieczyć się przed dodawaniem takich duplikatów?

Pokazać stosowną modyfikację kodu aplikacji lub struktury bazy danych, która uniemożliwia powstawanie duplikatów.

5. (**1p**) Inny problem kryjący się w kodzie który wykonuje zmiany na kilku różnych tabelach, to problem odczytu przez inny wątek takich danych, których proces dodawania jeszcze się nie zakończył (na przykład już dodano adres ale jeszcze nie dodano studenta). W celu wykluczenia możliwości wystąpienia takich sytuacji, baza danych ma możliwość używania tzw. transakcji.

Pokazać w jaki sposób rozpoczynać, kończyć lub anulować (BEGIN, COMMIT, ROLL-BACK) transakcje z poziomu kodu C#.

6. (**1p**) Zadanie 3 powtórzyć w technologii Dapper (używając tych samych zapytań do danych)
7. (**1p**) Zadanie 3 powtórzyć w technologii Linq2SQL
8. (**1p**) Zadanie 3 powtórzyć w technologii Dapper.SimpleCRUD (nie używając żadnych zapytań tylko polegając na automatycznym tworzeniu zapytań przez Dapper.SimpleCRUD).
9. (**1p**) Zadanie 3 powtórzyć w technologii Entity Framework 6.

**Uwaga!** Entity Framework 6 działa prawidłowo zarówno w .NET Framework jaki i .NET Core (5, 6, 7).

10. (**1p**) Pokazać jak zarządzać migracjami w Entity Framework. Formalnie - wygenerować migrację początkową. Pokazać jak baza danych tworzy się automatycznie na podstawie definicji migracji. Zmienić model obiektowy (na przykład dodać jakieś pole w klasie). Wygenerować migrację aktualizującą strukturę bazy danych.



## Rozdział 11

# Zestaw B, Projekt podsumowujący

Liczba punktów do zdobycia: **16/100**

1. (**16p**) Celem zadania jest połączenie zdobytej do tej pory wiedzy z obszaru języka i technologii. Zadanie polega na przygotowaniu aplikacji okienkowej umożliwiającej wgląd/modyfikację w wybrany obszar danych.

Interfejs użytkownika: System.Windows.Forms, WPF lub MAUI. Aplikacja powinna obojętnie używać kontrolki drzewa (TreeView) i listy danych. Drzewo prezentuje dane w postaci hierarchicznej i znajduje się po lewej stronie okna głównego, lista danych prezentuje fragment danych wskazany na drzewie i znajduje się w centralnej części okna (układ wzorowany np. na systemowym Eksploratorze plików). Wybór funkcji dodania nowego elementu lub edycji istniejącego powinien skutkować otwarciem danych do edycji w osobnym oknie.

- w przypadku WPF można próbować używać zewnętrznych bibliotek np. MvvmToolkit (<https://learn.microsoft.com/en-us/dotnet/communitytoolkit/mvvm/>), czy Caliburn (<https://caliburnmicro.com/>)
- kontrolka drzewa dla MAUI może pochodzić z zewnętrznej biblioteki komponentów, np. UraniumUI (<https://github.com/enisn/UraniumUI>)

Dostęp do danych (albo/albo):

- relacyjna baza danych (SQLServer/PostgreSQL)
- plik XML/JSON
- system plików (opis niżej)

Specyfikacja wersji z bazą danych / plikiem XML:

- aplikacja przechowuje w bazie danych rejestr danych osobowych studentów (imię, nazwisko, data urodzenia), listę zajęć oraz przypisanie studenta do zajęć w roku akademickim
- na drzewie pojawiają się trzy kategorie główne - lista studentów, lista lat akademickich, lista zajęć
- lata akademickie można dodawać/edytować, zajęcia można dodawać/edytować
- studentów można dodawać, edytować, po wskazaniu studenta na liście przechodzi się do okna widoku danych jednego studenta z co najmniej dwiema zakładkami - zakładką danych osobowych i zakładką przypisań do zajęć. Na każdej z zakładek można edytować dane (edytować dane osobowe lub listę przypisań do zajęć)

Specyfikacja wersji z systemem plików:

- aplikacja jest bardzo podstawową, uproszczoną wersją systemowego Eksploratora plików
- nad drzewem pojawia się wybór dysku systemowego
- na drzewie pojawiają się foldery na wybranym dysku
- jeżeli folder ma podfolery, to na drzewie użytkownik może rozwijać węzły drzewa rekursywnie aż do najgłębiej zagnieżdżonego podfoldera
- na liście pojawiają się pliki w wybranym na drzewie folderze
- użytkownik może dwukliknąć każdy plik i aplikacja próbuje wtedy wykonać domyślną akcję dla tego pliku (poczytać o właściwości `UseShellExecute` obiektu `ProcessStartInfo`)
- wybrane pliki są edytowalne z poziomu samej aplikacji - na przykład tylko pliki o rozszerzeniu \*.txt - ich wybór zamiast wykonać domyślną akcję dla pliku powinien otwierać osobne okno z prostym edytorem opartym o kontrolkę typu **RichTextBox**

# Dodatek A

## Varia

Niniejszy rozdział zbioru zadań ma charakter uzupełniający i zawiera zadania dodatkowe, niepunktowane, często o charakterze nieszablonowym, nietypowym, których rozwiązanie pozwala na pełniejsze zrozumienie wybranych mechanizmów języka i środowiska uruchomieniowego. Zadania z tego rozdziału pochodzą z bloga autora, gdzie były publikowane w latach 2008-2013.

### A.1 Poziom łatwy

#### A.1.1 Dziwna kolekcja

Czy to możliwe, że w bibliotece standardowej istnieje kolekcja, która sama tworzy elementy o zadanych kluczach kiedy tylko zostanie o nie poproszona? Wygląda na to, że tak - poniżej zaprezentowano kod, w którym dopiero co utworzona instancja kolekcji raportuje że zawiera element o losowo wybranym kluczu, mimo że taki element nie został tam wcześniej dodany.

```
namespace ConsoleApplication
{
    class Program
    {
        static void Main( string[] args )
        {
            NameValueCollection Collection = new NameValueCollection();

            Console.WriteLine(
                "Does NameValueCollection magically create items? " +
                Collection["foo"] != null ? " Yes, it does!" : "No, it doesn't."
            );
        }
    }
}
```

Zadaniem Czytelnika jest wskazanie błędu w powyższym kodzie, prowadzącego do takiego nieoczekiwanego zachowania.

#### A.1.2 Rekurencyjne zmienne statyczne

Czy dwie zmienne statyczne, które odwołują się nawzajem do siebie, spowodują powstanie nieskończonej rekursji?

```
public class A
{
    public static int a = B.b + 1;
}

public class B
```

```

{
    public static int b = A.a + 1;
}

public class MainClass
{
    public static void Main()
    {
        Console.WriteLine( "A.a={0}, B.b={1}", A.a, B.b );
    }
}

```

### A.1.3 Rozterki kompilatora

Reguły semantyczne języka muszą precyzyjnie rozstrzygać przypadki "brzegowe". W poniższym przykładzie kompilator ma dwie możliwości - wybrać metodę z klasy bazowej bez konwersji argumentu lub metodę z tej samej klasy ale z konwersją argumentu. Która reguła obowiązuje w przypadku języka C#? Czy wybór przeciwnej strategii byłby dopuszczalny?

```

class A
{
    public void Foo( int n )
    {
        Console.WriteLine( "A::Foo" );
    }
}

class B : A
{
    /* note that A::Foo and B::Foo are not related at all */
    public void Foo( double n )
    {
        Console.WriteLine( "B::Foo" );
    }
}

static void Main( string[] args )
{
    B b = new B();
    /* which Foo is chosen? */
    b.Foo( 5 );
}

```

### A.1.4 Składowe prywatne

Czy możliwe jest że klasa **A** ma dostęp do prywatnych składowych klasy **B**?

"Oczywiście, że nie, to wbrew regule enkapsulacji" - to zwyczajowa odpowiedź. Niemniej, jest co najmniej jeden przypadek, gdy jest to możliwe, co więcej, jest to dość ważna właściwość języka.

Pytanie brzmi więc: w jakich okolicznościach w języku C# klasa **A** może mieć pełen dostęp do **prywatnych** składowych innej klasy **B**.

### A.1.5 Nieoczekiwany błąd kompilacji

Rozważmy poniższy kod

```

using System;

class Foo
{
    private Foo() { }
}

```

```
class Program : Foo
{
    static void Main( string[] args )
    {
    }
}
```

Próba jego kompilacji kończy się komunikatem

```
Foo() is inaccessible due to its protection level
```

Jest to dość nieoczekiwane, w żadnym miejscu kodu nie ma próby utworzenia nowej instancji typu `Foo`. Ba, w kodzie nie ma w ogóle ani jednego wywołania operatora `new`. Wydaje się więc, że nie powinno mieć żadnego znaczenia czy konstruktor `Foo` jest dostępny czy nie.

Czytelnik proszony jest o wyjaśnienie powyższego paradoksu.

### A.1.6 Wywołanie metody na pustej referencji

Czy możliwe jest wywołanie metody na pustej referencji? Oczywista odpowiedź, to "nie".

Czyżby?

```
static void Main( string[] args )
{
    Foo _foo = null;

    // will throw NullReferenceException
    Console.WriteLine( _foo.Bar() );

    Console.ReadLine();
}
```

Uzasadnić, że powyższy kod nie musi wcale powodować wyjątku, przeciwnie, może zachować się całkowicie poprawnie i wypisać na konsoli wynik wywołania metody `Bar`.

## A.2 Poziom średniozaawansowany

### A.2.1 Zamiana wartości dwóch zmiennych

Następujący kod bywa wykorzystywany w językach C/C++ do zamiany wartości dwóch zmiennych **bez** użycia zmiennej pomocniczej.

```
int x, y;

x ^= y ^= x ^= y;
```

Nieoczekiwanie jednak, mimo wspólnych korzeni składni języka, powyższy kod nie działa poprawnie w języku C#. Zadaniem Czytelnika jest wyjaśnienie dlaczego tak się dzieje.

### A.2.2 Operacje na zbiorach (1)

Czy Czytelnik potrafi przewidzieć wynik działania poniższego kodu (zawartość która zostanie wypisana na konsoli) **bez** faktycznego uruchomienia?

```
List<int> list = new List<int>() { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

list.FindAll( i => { Console.WriteLine( i ); return i < 5; } );
```

### A.2.3 Operacje na zbiorach (2)

Po rozwiązaniu poprzedniego zadania Czytelnik z pewnością bez trudu przewidzi również wynik działania poniższego kodu **bez** faktycznego uruchomienia?

```
List<int> list = new List<int>() { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
list.Where( i => { Console.WriteLine( i ); return i < 5; } );
```

### A.2.4 Operacje na zbiorach (3)

Po rozwiązaniu dwóch poprzednich zadań, przewidzenie wyniku działania poniższego kodu **bez** faktycznego uruchomienia powinno być już łatwe.

```
List<int> list = new List<int>() { 1, 2, 3 };
list.GroupBy ( i => { Console.Write( "X" ); return i; } );
list.ToLookup( i => { Console.Write( "X" ); return i; } );
```

## A.3 Poziom trudny

### A.3.1 Specyficzne ograniczenie generyczne

Założmy następującą definicję interfejsu generycznego

```
public interface IGenericInterface<TValue>
{
    ... interface contract
}
```

Taki interfejs może być implementowany przez różne klasy z różną wartością argumentu generycznego

```
class Foo : IGenericInterface<Bar>
{
    ...
}

class Bar : IGenericInterface<Baz>
{
    ...
}
```

Czy możliwe jest w języku C# takie ograniczenie generycznego argumentu w definicji interfejsu, żeby jedynym dozwolonym ukonkretnieniem tego argumentu był typ implementujący interfejs?

Mówiąc inaczej, taka i tylko taka definicja typu powinna być dozwolona

```
class Foo : IGenericInterface<Foo>
{
}
```

a taka (i podobne) powinna powodować **błąd kompilacji**

```
class Foo : IGenericInterface<Bar>
{
}
```

### A.3.2 Zasięg zmiennej w domknięciu

W poniższym kodzie pętla wewnętrzna tworzy 10 instancji funkcji anonimowych, które ”łapią” zmienną lokalną w domknięciu. Wynik działania kodu jest jednak zgoła nieoczekiwany:

```
// create array of 10 functions
static Func<int>[] constfuncs()
{
    Func<int>[] funcs = new Func<int>[10];

    for ( var i = 0; i < 10; i++ )
    {
        funcs[i] = () => i;
    }

    return funcs;
}

...

var funcs = constfuncs();
for ( int i = 0; i < 10; i++ )
    Console.WriteLine( funcs[i]() );

// output:
// 10
// 10
// ...
// 10
```

Z konstrukcji kodu można bowiem naiwnie oczekiwać, że skoro *i*-ta funkcja powinna, zgodnie z definicją, zwracać wartość *i*. Tak się jednak nie dzieje.

Zadaniem Czytelnika jest nie tylko wyjaśnić powód takiego zachowania się domknięcia, ale również zaproponowanie eleganckiego rozwiązania, w którym nie naruszając zasady ”*i*-ta funkcja zwraca wartość *i*”, wynikiem działania

```
for ( int i = 0; i < 10; i++ )
    Console.WriteLine( funcs[i]() );
```

będzie

```
0
1
2
3
4
5
6
7
8
9
```





# Bibliografia

- [1] Wiktor Zychla *Windows oczami programisty*, Mikom
- [2] Archer T., Whitechapel A. *Inside C#*, Microsoft Press
- [3] Eckel B. *Thinking in C#*, <http://www.bruceeckel.com>
- [4] Gunnerson E. *A Programmer's Introduction to C#*
- [5] Lidin S. *Inside Microsoft .NET IL Assembler*, Microsoft Press
- [6] Petzold Ch. *Programming Windows*, Microsoft Press
- [7] Solis D. *Illustrated C#*