

Ćwiczenie pomocnicze

Aplikacje C#

Autor: Mariusz Fraś

Niniejsza instrukcja jest opracowana dla Visual Studio 2013

1 Cele ćwiczenia

Celem ćwiczenia jest:

1. Zapoznanie się ze środowiskiem deweloperskim Visual Studio dla wybranych aplikacji w języku C#.
2. Poznanie ogólnej architektury wybranych aplikacji w języku C#.

2 Środowisko deweloperskie

2.1 Wymagania wstępne

Standardowe (i zarazem wymagane) oprogramowanie do tworzenia i testowania aplikacji w języku C#. stosowane w laboratorium to:

- System operacyjny Windows 7 lub wyższy
- Visual Studio 2013 lub wyższy (w laboratorium może być wersja 2019).

2.2 Instalacja środowiska

Standardowa instalacja wg instrukcji Microsoft.

3 Zadanie – część I

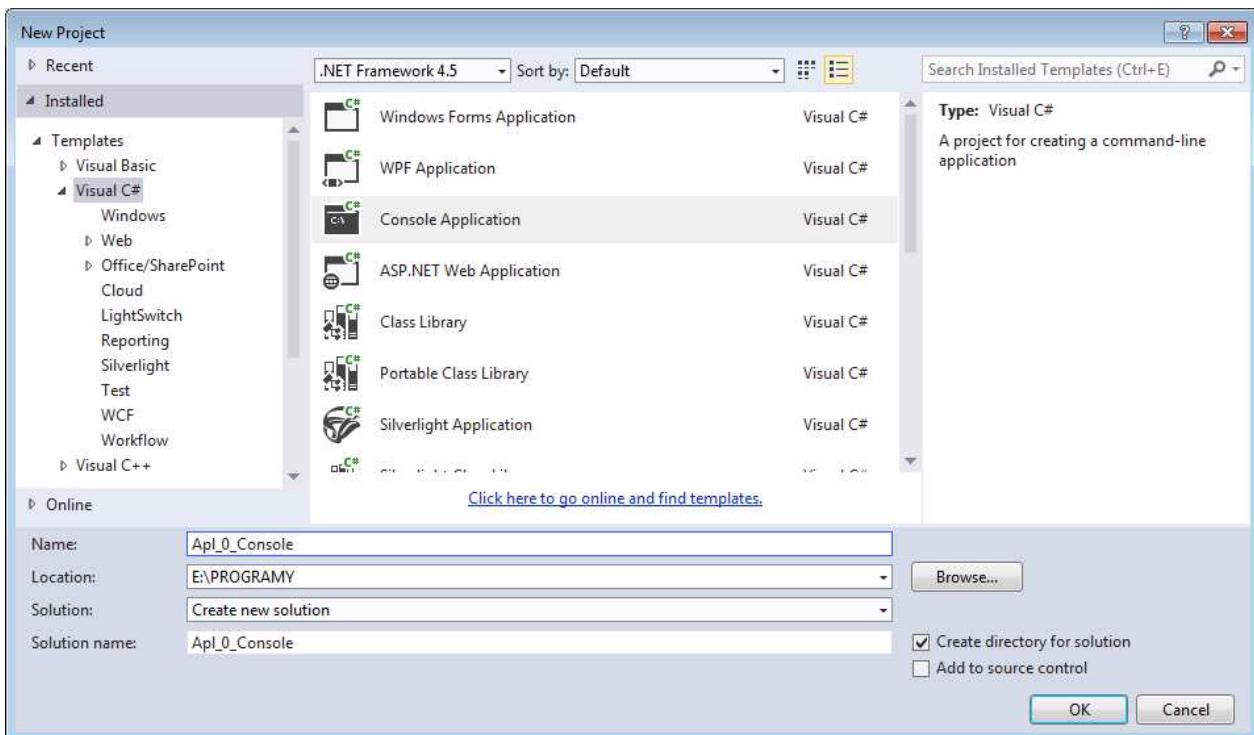
Podstawowe elementy tworzenia aplikacji konsolowych, Windows Forms i WPF

1. Uruchomienie środowiska

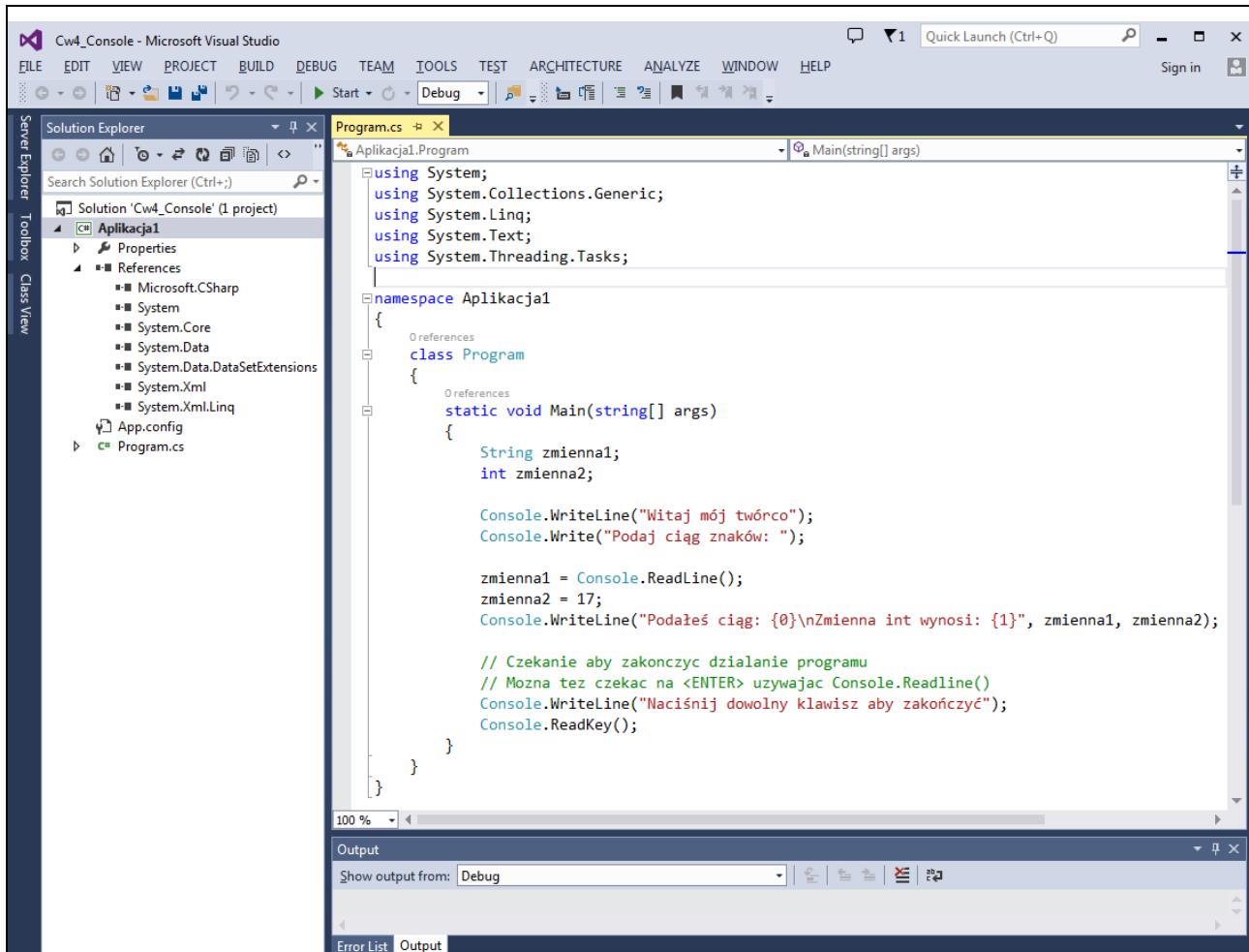
- Przygotuj katalog roboczy, w którym będzie tworzona aplikacja.
- Uruchom platformę Visual Studio 2013.
- Przejrzyj dostępne podstawowe opcje programu, a zwłaszcza dostępne szablony dla tworzenia różnych aplikacji..

2. Aplikacja konsolowa

- Wybierz nowy projekt aplikacji dla szablonu **Visual C# Console Application** nadając mu własną nazwę.



- Przejrzyj zawartość projektu. Zwróć uwagę na następujące elementy:
 - kod programu w pliku Program.cs
 - okna środowiska: deweloperskie (kod lub projekt wizualny), Solution (solucja - cały projekt), Properties (własności wybranego elementu).
- Zwróć uwagę na kod programu, w tym:
 - funkcję Main
 - dołączane klasy z odpowiednich przestrzeni nazw,
 - własną definicję przestrzeni (domyślnie jak nazwa projektu),a następnie dopisz kod jak na rys.



```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Aplikacja1
{
    class Program
    {
        static void Main(string[] args)
        {
            String zmienna1;
            int zmienna2;

            Console.WriteLine("Witaj mój twórco");
            Console.Write("Podaj ciąg znaków: ");

            zmienna1 = Console.ReadLine();
            zmienna2 = 17;
            Console.WriteLine("Podaleś ciąg: {0}\nZmienna int wynosi: {1}", zmienna1, zmienna2);

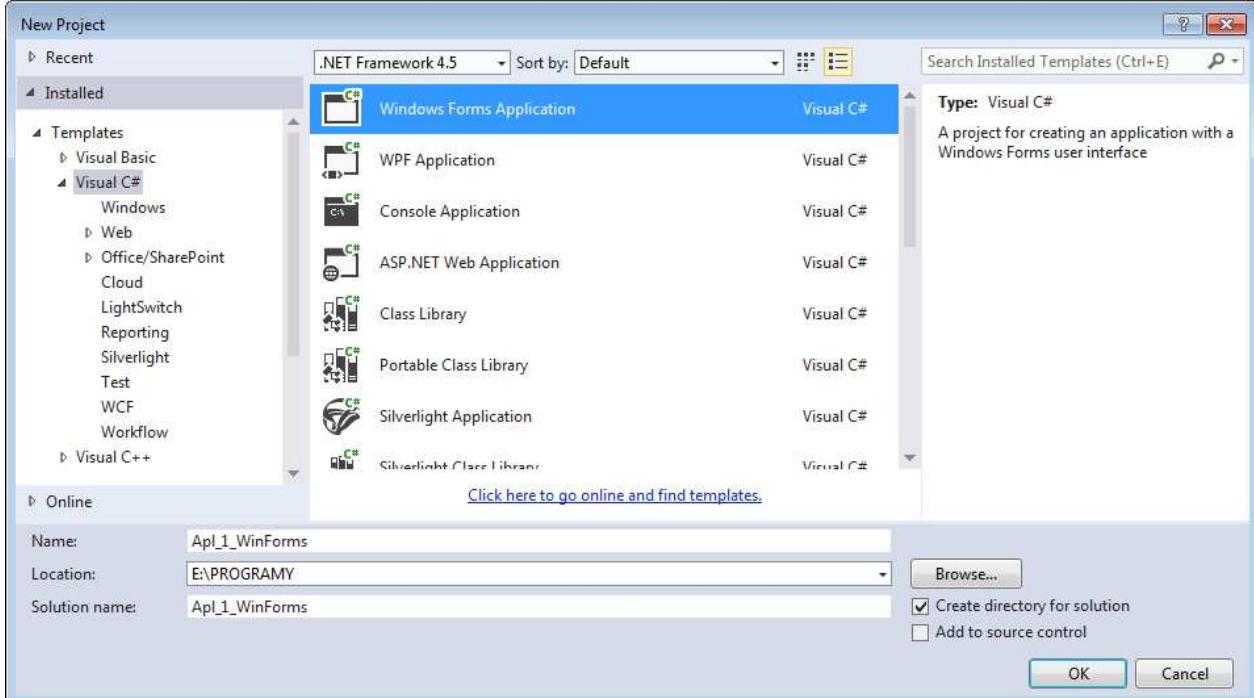
            // Czekanie aby zakończyć działanie programu
            // Można też czekać na <ENTER> używając Console.ReadLine()
            Console.WriteLine("Naciśnij dowolny klawisz aby zakończyć");
            Console.ReadKey();
        }
    }
}

```

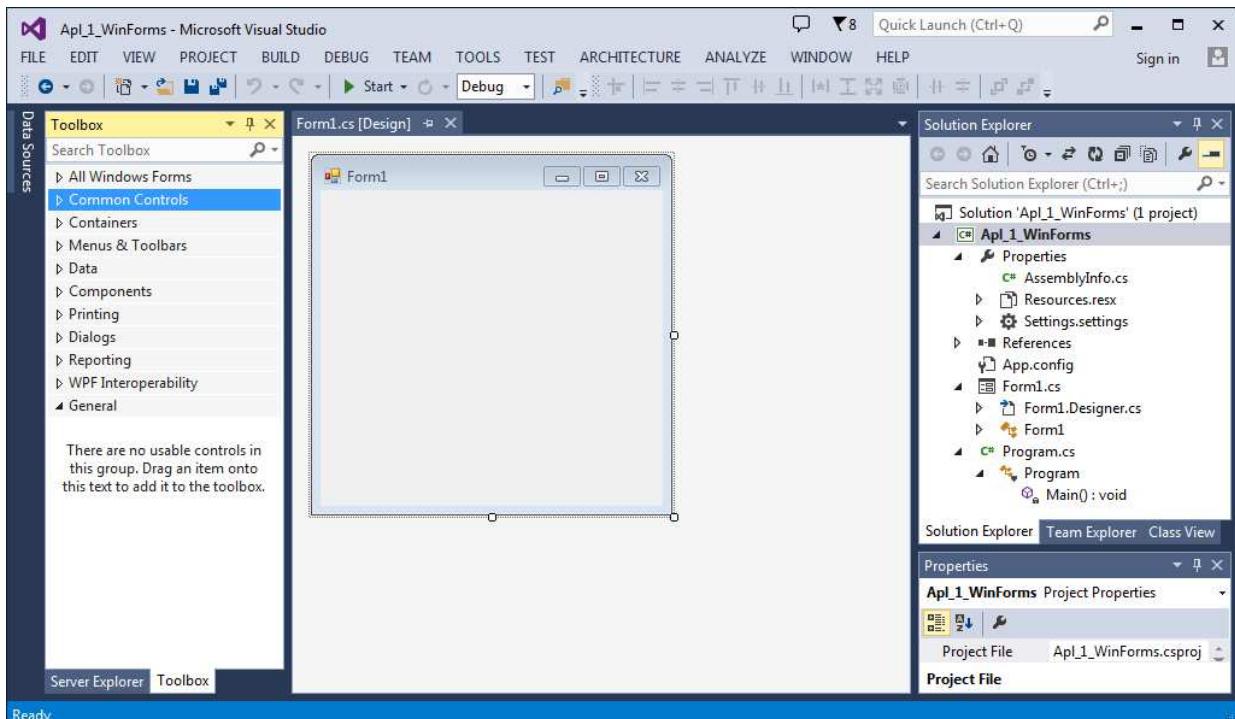
- Zwróć uwagę na sposób:
 - wypisywania komunikatów na ekranie w konsoli,
 - dodatkowego wypisywania wartości zmiennych (dyrektywa **{x}**, gdzie x – numer wypisywanego kolejnego parametru),
 - sposób wypisywania w nowej linii – znak **\n**,
 - wczytywania danych wejściowych (podawanych w konsoli),
 - sposobu wstrzymywania zakończenia działania aplikacji,
- Zbuduj/skompiluj program (opcja **BUILD**).
 - **BUILD → Build Nazwa_projektu** (po wybraniu/otwarciu pliku kodu) – za pierwszym razem.
 - **BUILD → Rebuild Nazwa_projektu** (kolejne powtórzenia operacji).
 - **BUILD → Build Solution** – dotyczy całej solucji / wszystkich projektów (w solucji może być kilka projektów) – za pierwszym razem.
 - **BUILD → Rebuild Solution** (kolejne powtórzenia operacji).
- Uruchom program i sprawdź działanie aplikacji.

3. Aplikacja Windows Forms

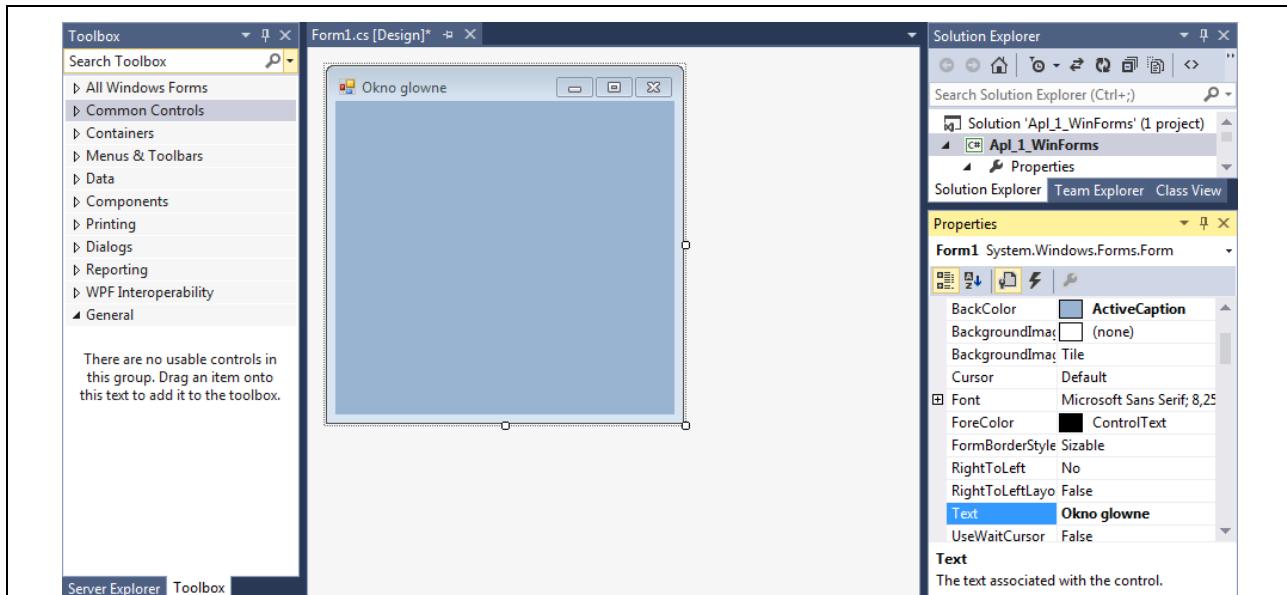
- Wybierz nowy projekt aplikacji dla szablonu **Visual C# Windows Forms Application** nadając mu własną nazwę.



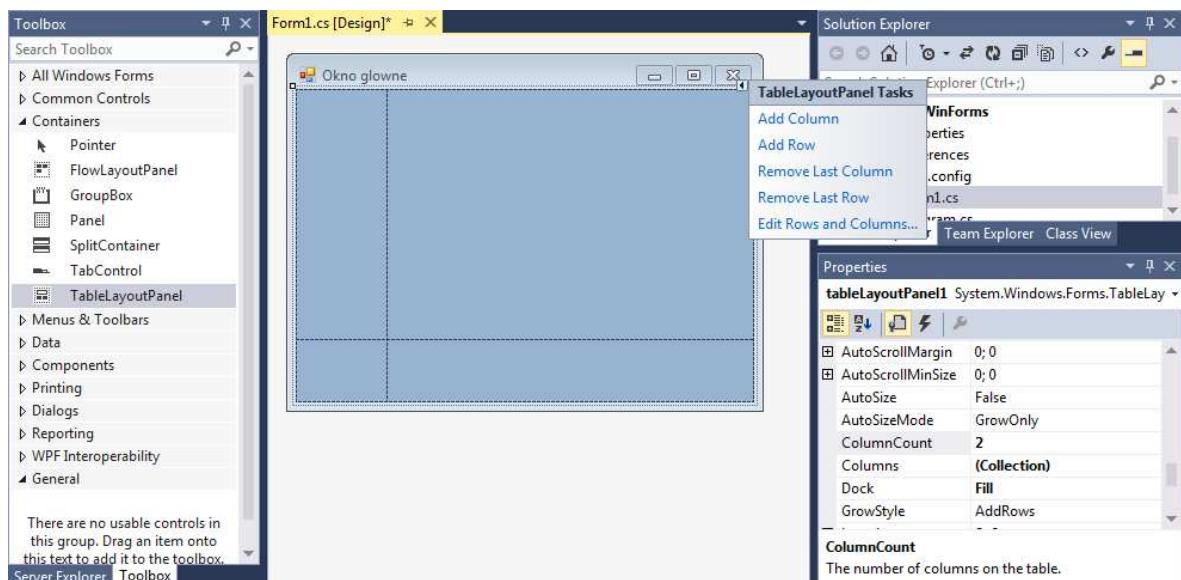
- Po utworzeniu projektu powinno pojawić się okno jak poniżej.
Zwróć uwagę na rozwijalne okna z lewej strony.



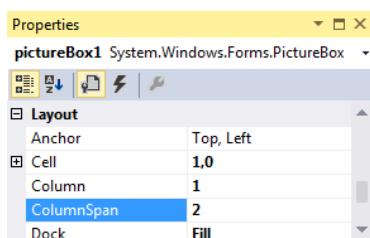
- W oknie właściwości zmień kilka parametrów (np. nazwę okna, kolor tła) jak na rysunku.



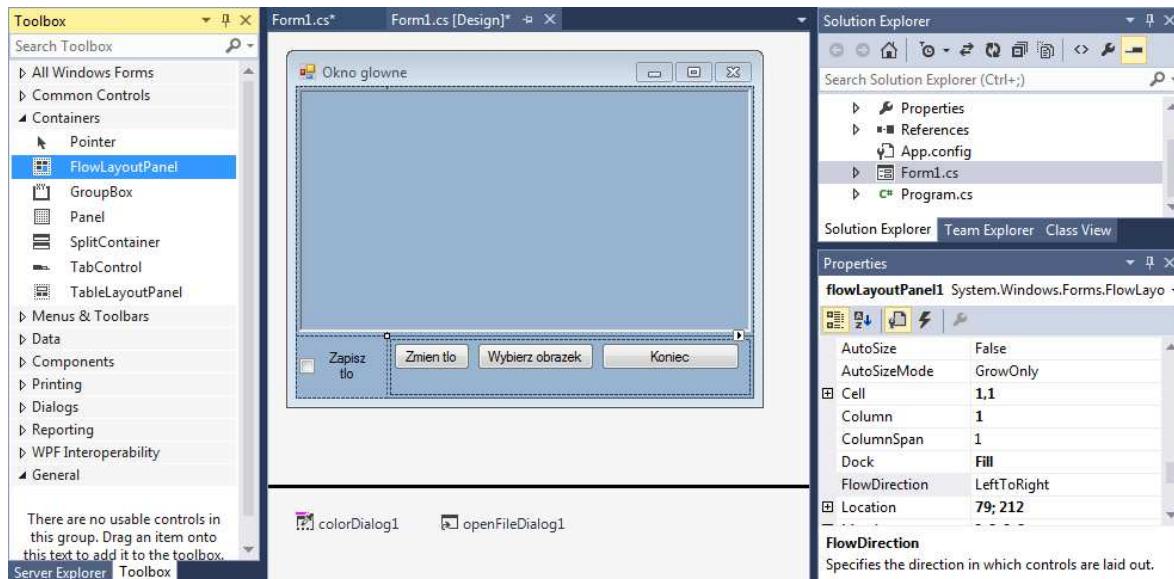
- Rozwiń okno **Toolbox** i gałąź **Containers**. Dodaj obiekt formatujący układ graficzny okna aplikacji (podwójne kliknięcie lub przeciągnięcie) **TableLayoutPanel** jak na rys.
- Ustaw pokrycie dla całości okna (parametr **Dock**)
- Rozwiń mały trójkąt w rogu panelu, wybierz edycję rzędów i kolumn i ustaw proporcje 20%/80%.



- Dodaj z gałęzi **Common Controls** Toolbox'a kontrolkę :**PictureBox**.
 - zadokuj ją w elemencie nadzależnym – z menu z małego trójkąta w rogu elementu lub właściwość **Dock = Fill**,
 - rozciagnij kontrolkę na dwie kolumny – właściwość **ColumnSpan = 2**,



- Dodaj kolejno następujące elementy:
 - CheckBox** z gałęzi **Common Controls** Toolbox'a,
 - FlowLayoutPanel** z gałęzi **Containers** Toolbox'a
(FlowLayoutPanel umożliwia umieszczanie w nim kontrolek obok siebie).



- Umieść 3 przyciski w FlowLayoutPanel.
- Umieść w kontrolkach odpowiednie tekst i sformatuj je aby osiągnąć efekt jak na rysunku powyżej.
- Nadaj przyciskom odpowiednie nazwy aby łatwo było je skojarzyć. Następnie wygeneruj kod obsługi zdarzenia wykonania akcji na kontrolkach poprzez dwukrotne kliknięcie każdej z nich.

Po tej czynności w pliku Form1.cs powinien być podobny kod jak na rysunku poniżej.

```
Form1.cs*  ✎ X Program.cs
Apł_1_WinForms.Form1  Form1()
using ...
namespace Apł_1_WinForms
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void przyciskTlo_Click(object sender, EventArgs e)
        {

        }

        private void przyciskObrazek_Click(object sender, EventArgs e)
        {

        }

        private void przyciskKoniec_Click(object sender, EventArgs e)
        {

        }

        private void checkBox1_CheckedChanged(object sender, EventArgs e)
        {
    }
}
```

- Zauważ, że obiekt klasy formularza jest wywoływany z funkcji Main w pliku Program.cs (wygenerowany automatycznie podczas tworzenia formularza).

```

Form1.cs* Program.cs ✘ X
Apl_1_WinForms.Program Main()
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Apl_1_WinForms
{
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new Form1());
        }
    }
}

```

- Uruchom aplikację i przetestuj wprowadzone zmiany.
- Dodaj do aplikacji 2 okna dialogowe z gałęzi **Dialogs** Toolbox'a:
 - ColorDialog,
 - OpenFileDialog.
- Nadaj oknom odpowiednie tytuły.
- Dla okna otwarcia pliku ustaw własność **Filtr** na wartość:
**JPEG Files (*.jpg)|*.jpg|PNG Files (*.png)|*.png|
BMP Files (*.bmp)|*.bmp|All files (*.*)|*.***
- Dopisz następujący kod do odpowiednich funkcji obsługi:

```

private void przyciskKoniec_Click (object sender, EventArgs e)
{
    this.Close();
}

private void przyciskTlo_Click(object sender, EventArgs e)
{
    if (colorDialog1.ShowDialog() == DialogResult.OK)
        pictureBox1.BackColor = colorDialog1.Color;
}

private void przyciskObrazek_Click (object sender, EventArgs e)
{
    if (openFileDialog1.ShowDialog() == DialogResult.OK)
    {
        pictureBox1.Load(openFileDialog1.FileName);
    }
}

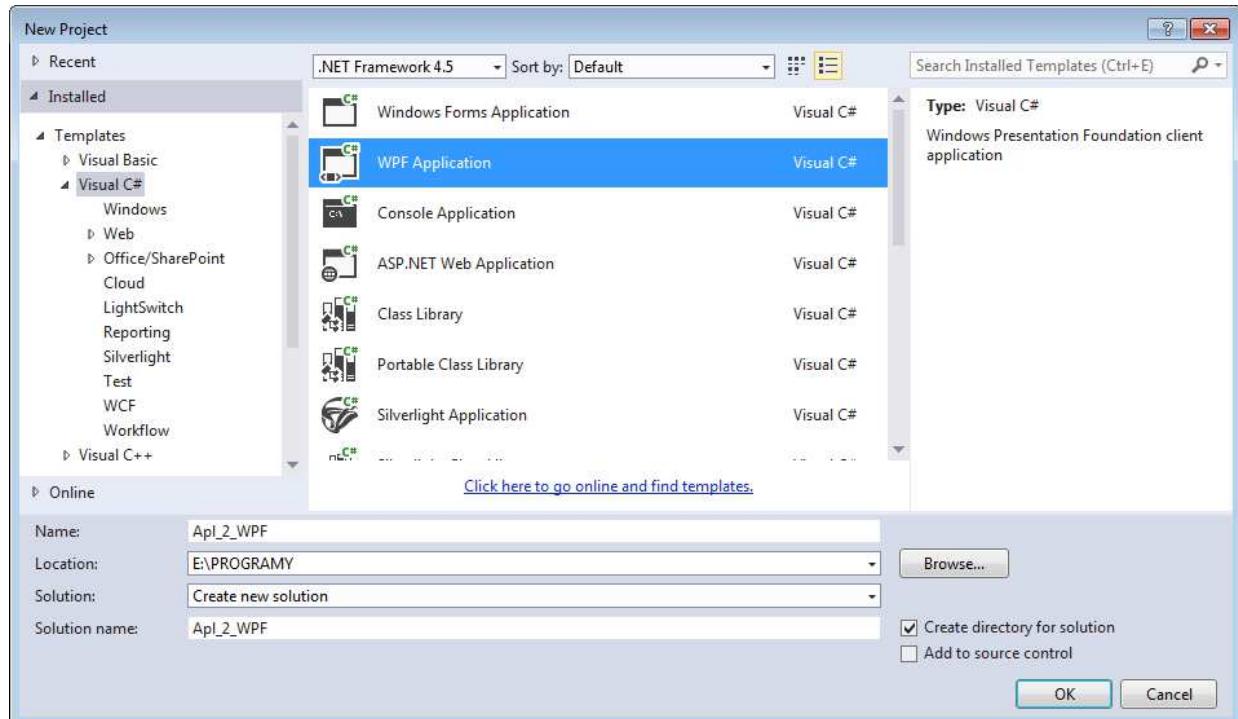
private void checkBox1_CheckedChanged(object sender, EventArgs e)
{
    //...do realizacji we własnym zakresie
}

```

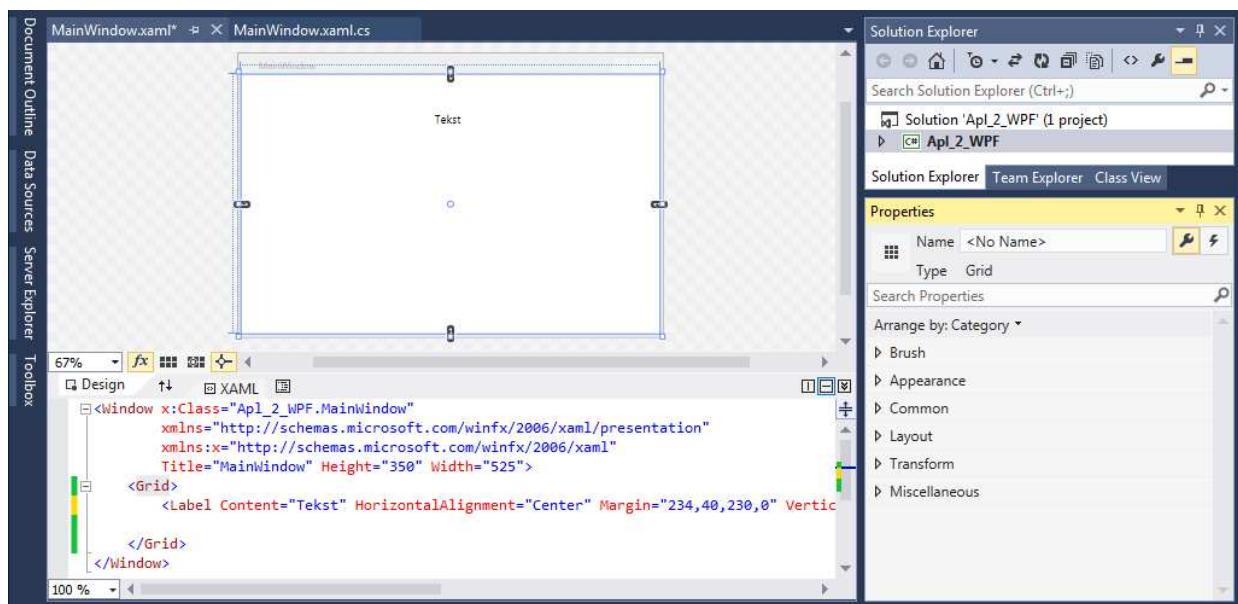
- Uruchom i przetestuj działanie aplikacji.

4. Aplikacja WPF (Windows Presentation Foundation)

- Wybierz nowy projekt aplikacji dla szablonu **Visual C# WPF Application** nadając mu własną nazwę.

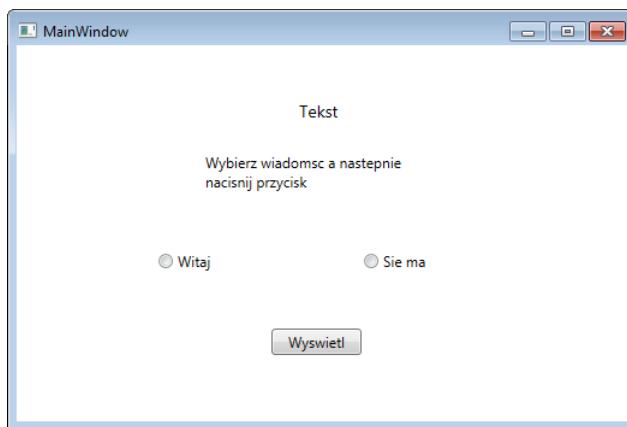


- Zwróć uwagę na nowy sposób realizacji aplikacji: okno wizualnego projektowania i plik XAML (Extensible Application Markup Language). XAML jest to język opisu interfejsu oparty na XML.



- W pliku XAML są zdefiniowane obiekty i ich właściwości, które są wyświetlane w aplikacji,

- Podobnie jak w poprzednim przykładzie utwórz aplikację o interfejsie jak na rysunku poniżej.



- Przeglądaj zawartość pliku XAML, który powinien być podobny do poniższego.

Design XAML

```
<Window x:Class="Apl_2_WPF.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="MainWindow" Height="350" Width="525">
    <Grid>
        <Label Content="Tekst" HorizontalAlignment="Center" Margin="234,40,230,0" VerticalAlignment="Top" Width="53" FontSize="14"/>
        <TextBlock HorizontalAlignment="Left" Height="82" Margin="157,89,0,0" TextWrapping="Wrap" VerticalAlignment="Top" Width="142"/>
        <RadioButton x:Name="przycisk1" Content="Witaj" HorizontalAlignment="Left" Margin="118,171,0,0" VerticalAlignment="Top"/>
        <RadioButton x:Name="przycisk2" Content="Sie ma" HorizontalAlignment="Left" Margin="289,171,0,0" VerticalAlignment="Top"/>
        <Button Content="Wyswietl" HorizontalAlignment="Left" Margin="212,235,0,0" VerticalAlignment="Top" Width="75" Click="Button_Click_1"/>
    </Grid>
</Window>
```

- Spróbuj edytować plik XAML i sprawdź tego efekty.
- Zwróć uwagę na składowe projektu aplikacji, w tym:
 - MainWindow.xaml – opis okna głównego aplikacji
 - MainWindow.xaml.cs – kod wsparcia dla okna głównego
 - App.xaml, App.xaml.cs – główna składowa inicjalizująca aplikację

Solution Explorer

```
Aplication 'Apl_2_WPF' (1 project)
  + Aplication
    - Properties
    - AssemblyInfo.cs
    - Resources.resx
    - Settings.settings
    - References
    - App.config
    - App.xaml
    - App.xaml.cs
    - MainWindow.xaml
    - MainWindow.xaml.cs
```

Properties

App.xaml

```
<Application x:Class="Apl_2_WPF.App"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        StartupUri="MainWindow.xaml">
    <Application.Resources>
        ...
    </Application.Resources>
</Application>
```

- Dopolacz kod obslugi przycisku:

```
Private void Button_Click(object sender, RoutedEventArgs e)
{
    if (przycisk1.IsChecked == true)
    {
        MessageBox.Show("Witaj.");
    }
    else
    {
        przycisk2.IsChecked = true;
        MessageBox.Show("Sie ma.");
    }
}
```

- Przetestuj dzialanie aplikacji

ĆWICZENIE POMOCNICZE 1

Podstawy programowania aplikacji w języku Java

Mariusz Fraś

Cele ćwiczenia

1. Zapoznanie się ze środowiskiem deweloperskim aplikacji Java.
2. Poznanie/przypomnienie ogólnej architektury i podstawowych elementów implementacji aplikacji Java (obiektów, dziedziczenia, obsługi zdarzeń, wyjątków itp.).

1. Środowisko deweloperskie

1.1. Wymagania wstępne

Standardowe (i zarazem wymagane) oprogramowanie do tworzenia i testowania aplikacji Java stosowane w laboratorium to:

- System operacyjny Windows (min. wersja 7)
- Oracle Java 2 Software Development Kit (Oracle Open JDK) lub podobne.
- Platforma programistyczna Eclipse lub IntelliJ IDEA.

1.2. Instalacja środowiska

- Instalacja platformy deweloperskiej jest standardowa wg opisu witryny www.eclipse.org lub www.jetbrains.com/idea/.

Uwaga1: należy uwzględniać z jakiej wersji JDK korzystamy. Nowsze wersje Java są natywnie obsługiwane bez dodatkowych zabiegów od odpowiedniej wersji Eclipse.

Uwaga2: W przypadku stosowania szczególnego oprogramowania do ochrony systemu (zapory systemu (firewall), oprogramowanie antywirusowe, itp.) może być potrzeba odblokowania komunikacji dla tworzonych aplikacji.

2. Podstawowe elementy programowania aplikacji Java

W ćwiczeniu zostanie zaimplementowana prosta aplikacja (działająca w oknie konsoli tekstowej) zawierająca następujące elementy:

- a. Utworzenie projektu i głównej klasy aplikacji
- b. Utworzenie klasy realizującej pewne przetwarzanie (tu: obliczenia)
- c. Zdefiniowanie interfejsu i zastosowanie go.
- d. Utworzenie klasy pochodnej wykorzystującej mechanizm dziedziczenia.
- e. Ilustracja obsługi wyjątków.
- f. Zastosowanie rzutowania
- g. Podstawowa obsługa wejścia/wyjścia.

Uwagi:

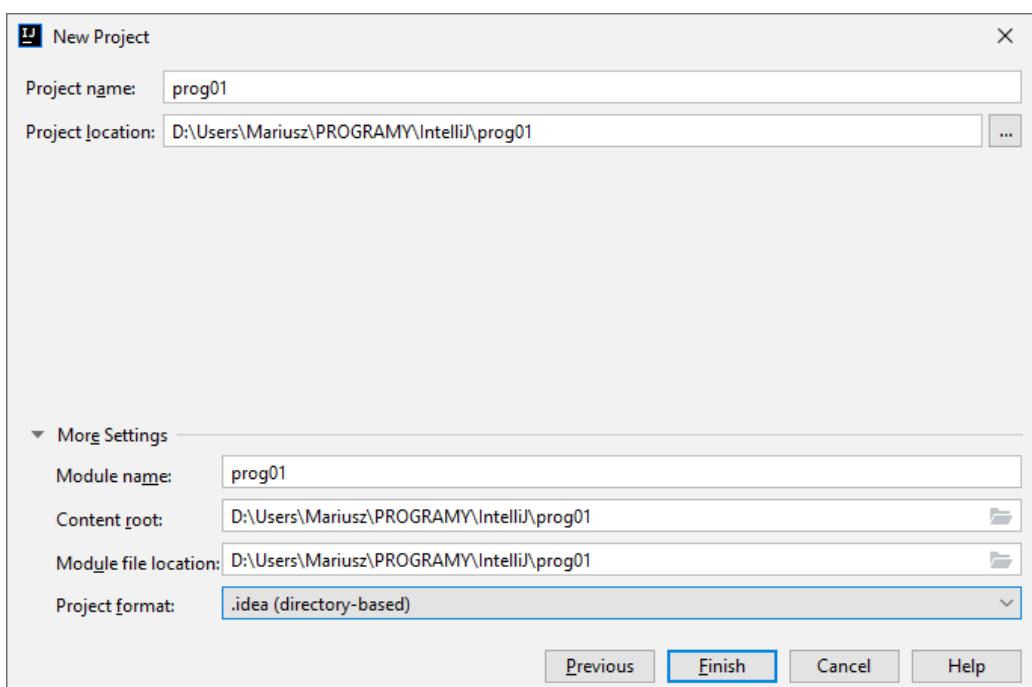
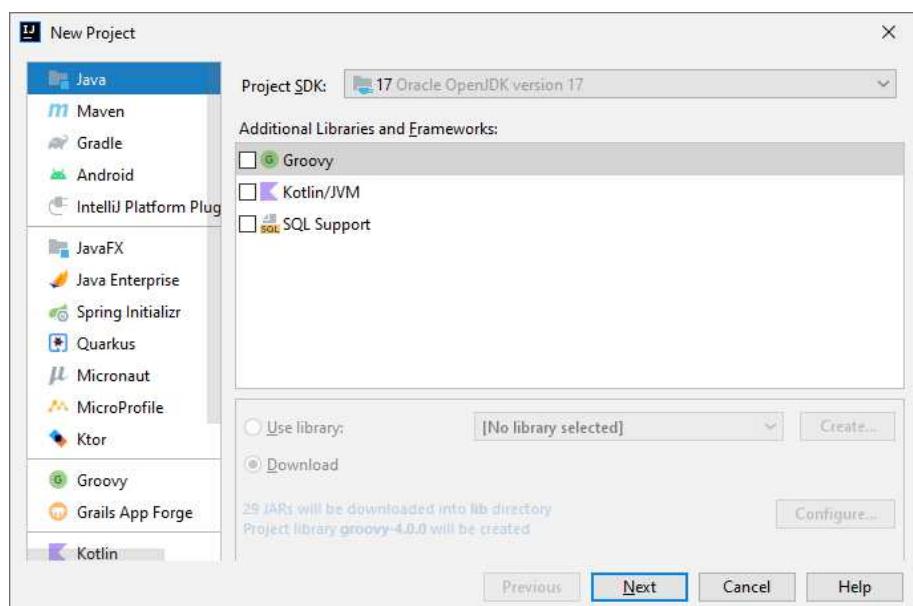
- We wszystkich ćwiczeniach podawane są podstawowe instrukcje dla tworzenia kodu. Kompilator może sygnalizować czasem błędy ze względu na brak jakiejś czynności – np. nie zimportowanie potrzebnej biblioteki). We wszystkich przypadkach należy skorygować sygnalizowane błędy przez kompilator (np. importując odpowiednie klasy itp.) wykorzystując podpowiedzi kompilatora..
- W całej instrukcji wykropkowanie fragmenty - ... lub [...] – oznaczają pominiętą w opisie część kodu – kod do uzupełnienia samemu lub przedstawiony w instrukcji wcześniej.

2.1. Wstępne przygotowanie środowiska

- Przygotować katalog roboczy, w którym będzie tworzona aplikacja – np. **D:\Users\[katalog_użytkownika]\Programy** lub w innym miejscu – w zależności od potrzeb.
- Uruchomić program Eclipse lub IntelliJ IDEA. Podczas uruchamiania lub potem wybrać katalog roboczy dla projektów aplikacji (w ustawieniach).

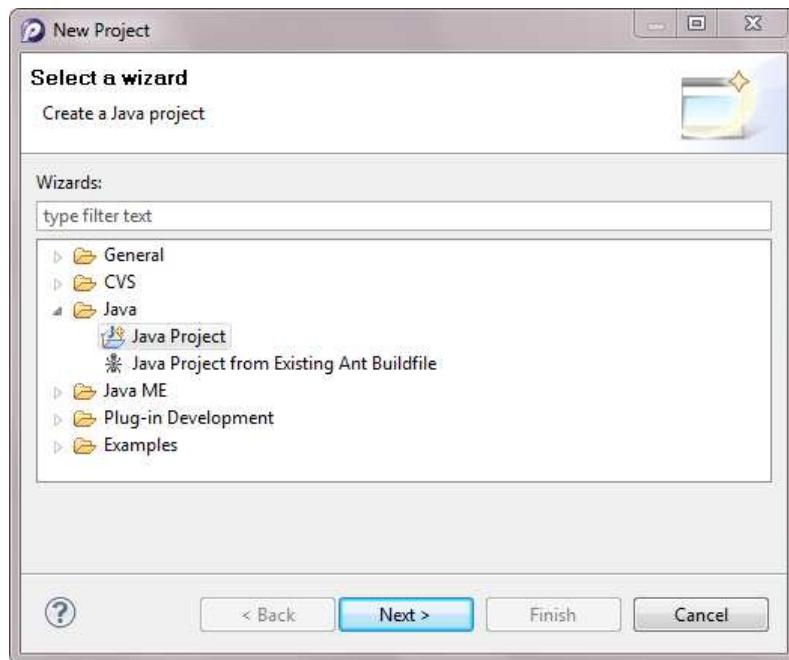
2.2. Utworzenie pierwszej aplikacji – klasa główna

- Utwórz w platformie projekt Java. Wybierz w menu New Project ustaw dane (nazwę projektu – tu **prog01**).:
 - a) w IntelliJ IDEA:
Projects → New Project, w oknie New Project wybierz opcję Java.

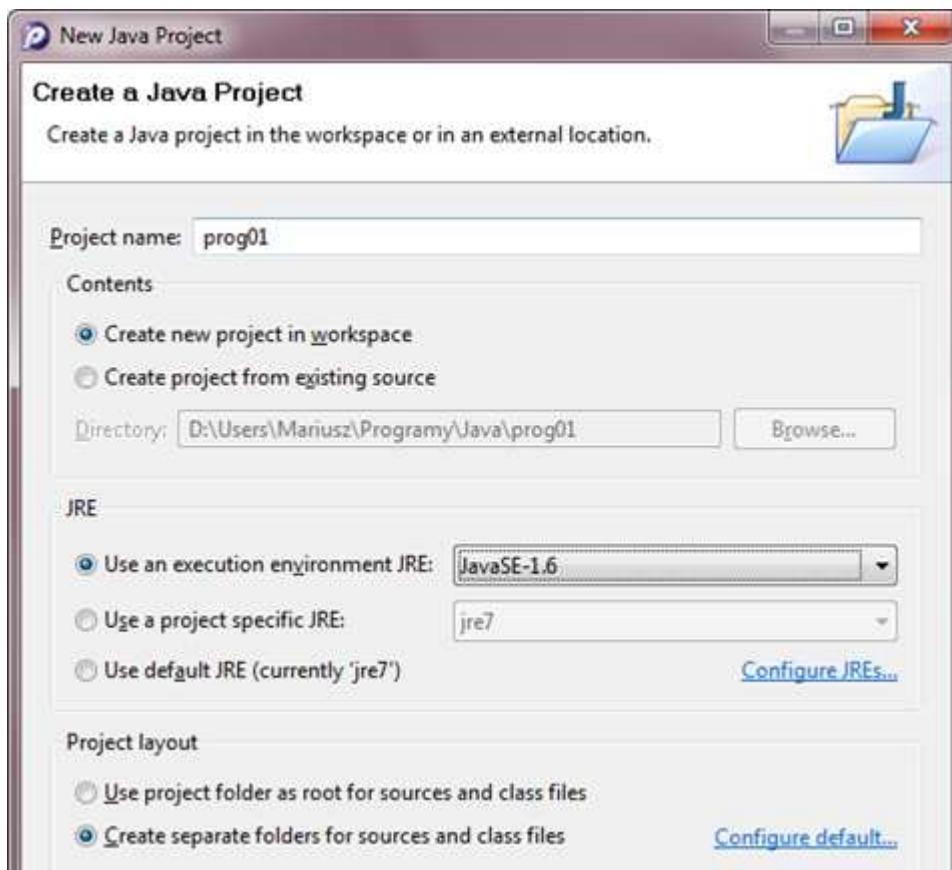


b) w Eclipse

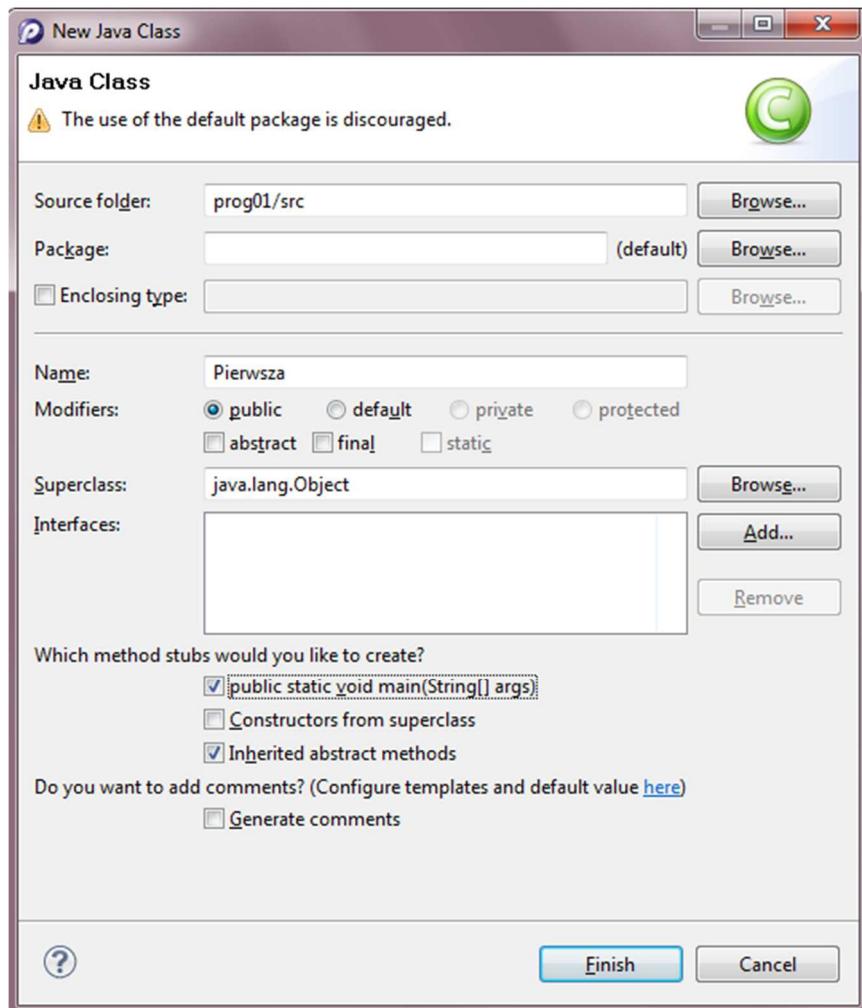
File → New... → Project..., w oknie New Project wybierz opcję Java\Java Project.



W Eclipse w oknie New Java Project w polu Project name wpisz nazwę projektu i wybierz odpowiednie środowisko wykonawcze JRE – najlepiej najnowszą wersję Java SE. Upewnij się, że będą używane oddzielne katalogi dla plików źródłowych i wykonawczych i zatwierdź.

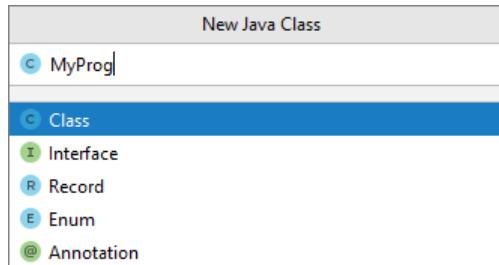


- Utwórz nową klasę – klasę główną aplikacji:
 - a) W Eclipse:
 - zaznacz po lewej stronie w oknie *Package explorer* węzeł *src*,
 - prawym klawiszem myszki wybierz opcję dodaj klasę *New→Class*
 - w oknie *New Java Class* wyczyść pole *Package* – to ułatwi pracę z pakietami Java,
 - wpisz nazwę klasy w polu *Name* (tu: Pierwsza),
 - zaznacz pole *public static void main(...)* – metoda niezbędna w jednej klasie (głównej) aplikacji.



- b) W IntelliJ IDEA utwórz nową klasę Java za pomocą menu kontekstowego:





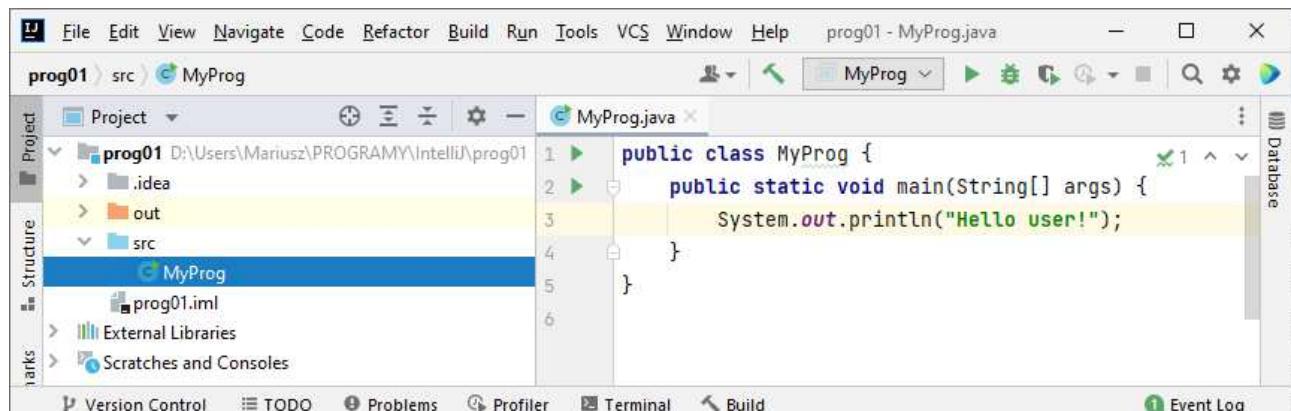
Metoda **main** klasy głównej jest funkcją wejściową – tym miejscem, w którym zaczyna się wykonywanie kodu aplikacji. Metoda **main** może być tylko jedna, w głównej klasie aplikacji.

W aplikacji można używać metod klas z bibliotek dołączanych automatycznie (biblioteki systemowe z często wykorzystywany klasami w aplikacjach) lub dołączanych jawnie przez programistę. Metody specyfikuje się podając klasę w której są zdefiniowane.

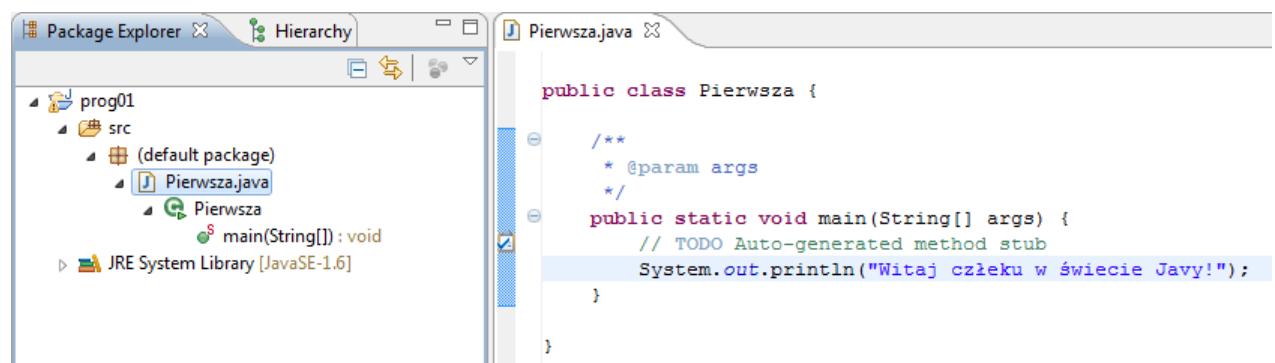
- W oknie edycji dopisz w metodzie **main** kod wyświetlający napis w standardowym wyjściu (tu na ekranie) za pomocą klasy **System**, jej pola **out** (statyczne typu **PrintStream**) i metody **println**:

```
System.out.println("Hello user!");
```

IntelliJ IDEA:

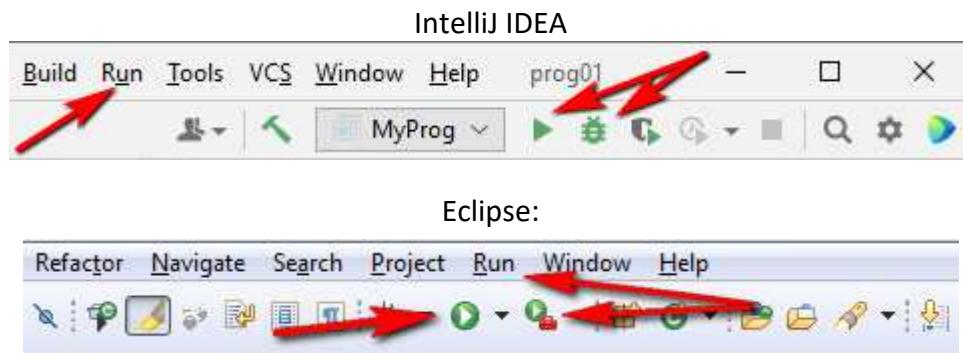


Eclipse:



- Uruchom aplikację (opcja w menu **Run** → **Run ...** lub **Run** → **Run As** lub ikona uruchomienia w pasku narzędzi) i sprawdź jej działanie.

Uwaga: za pierwszym razem może być trzeba najpierw dodać "konfigurację" specyfikującą rodzaj elementu uruchamianego jako aplikację java.



Odtąd momentu w instrukcji rozróżnianie platformy deweloperskiej będzie z reguły pomijane.

2.3. Utworzenie klasy realizującej obliczenia na liczbach całkowitych

- Podobnie jak poprzednio utwórz nową klasę (tu: CalcInt).
(w Eclipse nie zaznaczaj opcji *public static void main(...)* – ta metoda już jest zdefiniowana)
- Dopusz kod dwóch metod wykonujących działania dodawania i odejmowania, zwracających wartość typu int (całkowitoliczbową) i mające dwa parametry typu int:

```
public int add(int a, int b)
public int subst(int a, int b)
```

- Dopusz kod pustego konstruktora klasy:

```
public CalcInt() { }
```

The screenshot shows the IntelliJ IDEA code editor with a single file named 'CalcInt.java'. The code defines a class 'CalcInt' with a constructor and two methods: 'add' and 'subst'. The 'add' method returns the sum of its parameters, and the 'subst' method returns their difference. The code is color-coded according to Java syntax.

```
1 public class CalcInt
2 {
3     public CalcInt() {}
4
5     public int add(int a, int b) {
6         return (a+b);
7     }
8     public int subst(int a, int b) {
9         return a-b;
10    }
11}
```

Konstruktor to metoda wywoływana przy tworzeniu obiektu danej klasy. Pusty konstruktor jest tworzony automatycznie (nie trzeba go definiować jak w przykładzie). Natomiast można tworzyć konstruktory realizujące różne przydatne działania przy tworzeniu obiektu danej klasy – np. inicjalizacja danych obiektu.

- W klasie głównej zadeklaruj i utwórz obiekt (zmienną) typu właśnie utworzonej klasy:

```
CalcInt k1 = new CalcInt();
```

Uwaga: zasięg (ważność) zmiennych jest ograniczony do miejsca deklaracji – określony przez nawiasy klamrowe – i obejmuje także kod zagnieżdżony (zagnieżdżone nawiasy).

- Dopisz kod który wywoła odpowiednie metody klasy CalcInt i przypisze zmiennej typu int (tu: zmiennej result) sumę dwóch liczb, a potem różnicę, i wyświetli wyniki operacji:

```
int result;
result = k1.add(2,5);
System.out.println("First result is " + result);
result = k1.subst(2,5);
System.out.println("Second result is " + result);
```

```
MyProg.java
public class MyProg {
    public static void main(String[] args)
    {
        System.out.println("Hello user!");

        CalcInt k1 = new CalcInt();
        int result;
        result = k1.add( a: 2, b: 5 );
        System.out.println("First result is " + result);
        result = k1.subst( a: 2, b: 5 );
        System.out.println("Second result is " + result);
    }
}
```

- Uruchom i przetestuj działanie aplikacji.

2.4. Dodanie interfejsu narzucającej zachowanie klasy

- Dopisz do klasy głównej kod używający nowej metody multiply:

```
result = k1.multiply(2,5);
System.out.println(result);
```

- Sprawdź jaki jest sygnalizowany błąd implementacji.

Dla uniknięcia takich błędów stosuje się interfejsy. Wskazanie implementacji interfejsu przez klasę narzuca konieczność implementacji odpowiednich metod w klasie. Tak można kontrolować aby tworzona klasa musiała mieć odpowiedni kod (tu: potrzebne metody).

- Zdefiniuj interfejs (tu o nazwie IOperations) w podobny sposób jak nową klasę – tym razem wybierz opcję *New→Interface*.
- Dopisz w interfejsie deklaracje trzech metod

```
public int add(int a, int b);
public int subst(int a, int b);
public int multiply(int a, int b);
```

```
IOperations.java
public interface IOperations {
    public int add(int a, int b);
    public int subst(int a, int b);
    public int multiply(int a, int b);
}
```

- Zapisz zawartość okien edycyjnych.
Przejdź do okna klasy realizującej obliczenia (tu: CalcInt)
- Dopisz w deklaracji klasy implementację intefesju IOperations.

```
public class CalcInt implements IOperations
```

- Sprawdź sygnalizowany błąd: naprowadź kurSOR na czerwone oznaczenie błędu, przeczytaj informację, potem kliknij w te oznaczenie. Wybierz opcję naprawy dodającą definicję brakującej metody.
- Do metody **multiply**, która została wygenerowana dopisz kod realizujący działanie i zwracający jego wynik.

```
1 public class CalcInt implements IOperations
2 {
3     public CalcInt() {}
4
5     public int add(int a, int b) { return (a+b); }
6     public int subst(int a, int b) { return a-b; }
7
8     @Override
9     public int multiply(int a, int b) {
10         return a*b;
11     }
12 }
```

2.5. Dziedziczenie klas - utworzenie klasy pochodnej

- Utwórz nową klasę (tu: ExtCalcInt). Klasa będzie rozszerzać klasę CalcInt (dziedziczyć po niej) ale realizować odejmowanie w nieco innym sposobie (tu jako przykład: zwracać tylko dodatnie wyniki). Klasa wykorzysta klasę macierzystą.

```
public class ExtCalcInt extends CalcInt
```

Dopisz kod metody **subst** zwracającą wartość typu **int** i przyjmującą jako dwa parametry wartości **int**. Metoda wywołuje działanie klasy macierzystej (tu: **subst(...)**) za pomocą specyfikatora **super** (czyli **super**.**subst(a, b)**) i jeśli wynik jest niedodatni to zwraca wartość 0.

```
public class ExtCalcInt extends CalcInt {
    public int subst(int a, int b) {
        int result = super.subst(a,b);
        if(result>0)
            return result;
        else
            return 0;
    }
}
```

- W klasie głównej zadeklaruj i utwórz obiekt zdefiniowanego typu:

```
ExtCalcInt k2 = new ExtCalcInt();
```

- Dopisz kod który przypisze zmiennej typu **int** (tu: zmiennej **result**) sumę dwóch liczb a potem różnicę i wyświetl wyniki operacji:

```
System.out.println("Second class CalcInt");
ExtCalcInt k2 = new ExtCalcInt();
System.out.println("First result is " + result);
result = k2.subst(2,5);
System.out.println("Second result is " + result);
result = k2.multiply(2,5);
System.out.println(result);
```

- Uruchom i przetestuj działanie aplikacji. Wynik powinien być podobny do poniższego.

```
Hello user!
The class CalcInt:
Add result is 7
Subst result is -3
Multiply result is 10
Second class ExtCalcInt:
Add result is 10
Subst result is 0

Process finished with exit code 0
```

2.6. Obsługa wyjątków (ang. Exception)

"Exception" jest skrótem dla "exceptional event" – jest to zdarzenie, które następuje w trakcie realizacji programu, które zakłóca normalny przebieg instrukcji programu (np. gdy wykonywane jest dzielenie przez zero lub odwołujemy się do pliku, który nie istnieje). Gdy wystąpi błąd, tworzony i przekazywany jest (zgłaszanym) do systemu wykonawczego obiekt wyjątku. Wyjątki są automatycznie rzucane, lub możemy je sami rzucać dla sygnalizowania błędów. Wyjątki można przechwytywać i stosownie do rodzaju wyjątku podejmować odpowiednie działania w programie. Jest wiele klas wyjątków – np. *IO Exception*, *ClassNotFoundException*, itp. Wszystkie są pochodnymi klasami **Exception**.

- W klasie metod obliczeniowych dopisz metodę dzielenia całkowitego **divide**:

```
public int divide(int a,int b) {
    return (a/b);
```

- W klasie głównej wywołaj metodę dzielenia a za nią dopisz instrukcję wyświetlającą wynik:

```
int result2 = k2.divide(x,y);
System.out.println("Divide result is " + result2);
```

W miejsce **x** i **y** wpisz najpierw wartości 5 i 2 a następnie 5 i 0.

Zwróć uwagę na zakończenie programu z błędem (podane są różne informacje z wykonania – m.in. nr linii programu powodującą błąd):

```
Exception in thread "main" java.lang.ArithmetricException Create breakpoint : / by zero
at ExtCalcInt.divide(ExtCalcInt.java:10)
at MyProg.main(MyProg.java:24)

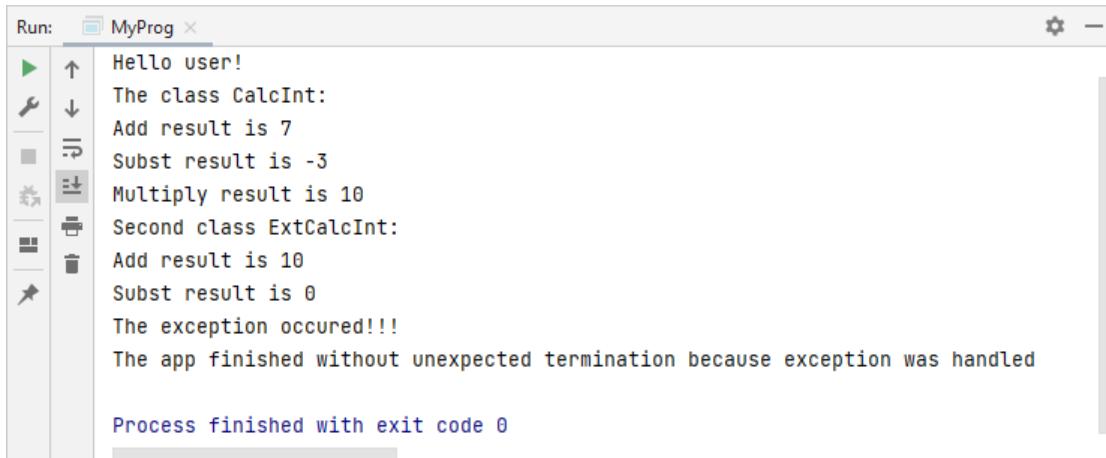
Process finished with exit code 1
```

- W klasie głównej wywołaj metodę dzielenia tym razem w bloku **try...catch...**, w którym będzie obsługiwana kontrola wyjątków:

```
[...]
try {
    int result2 = k2.divide(5,0);
    System.out.println("Divide result is " + result2);
}
catch (Exception e) {
    System.out.println("The exception occurred! Division by zero!");
}

System.out.println("The app finished without termination because exception
                  was handled");
[...]
```

- Uruchom i przetestuj działanie aplikacji.



- Można też wyświetlić informację opisującą błąd (jak poprzednio) z użyciem metody **printStackTrace()**:

```
[...]
catch (Exception e) {
    e.printStackTrace();
}
[...]
```

2.7. Konwersja i rzutowanie, wykorzystanie typu Object

Klasa **Object** jest korzeniem drzewa hierarchii dziedziczenia wszystkich klas.

Konwersja jest działaniem zmieniającym typ jednego rodzaju na inny (np. int na float).

Konwersja niejawną jest realizowana automatycznie w różnych sytuacjach. Np. gdy zmiennej typu long przypisujemy wartość zmiennej typu int, to dla wartości tej pierwszej wykonywana jest najpierw konwersja z int na long.

Rzutowanie (ang. casting) jest działaniem pozwalającym zmienić typ jednego rodzaju na drugi w sposób jawnym (jawną konwersja). Realizowane jest za pomocą podanej w nawiasach nazwy typu przed rzutowaną zmienną.

Konwersje niejawne są bezpieczne. Podczas konwersji jawnej może dochodzić do utraty części danych (np. gdy zachodzi konwersja z float na int). Dla obiektów konwersja jest dozwolona tylko z typu wyższego w hierarchii dziedziczenia na niższy (z pochodnego na podstawowy). Rzutowanie jest potrzebne, gdy świadomie chcemy dokonać konwersji na typ "bardziej ograniczony" (np. z float na int).

- W klasie głównej dopisz deklaracje dwóch zmiennych typu int i double. Przypisz zmiennej double wartość niecałkowitą, a następnie stosując rzutowanie przypisz zmiennej int wartość zmiennej double. Wyświetl wartości obu zmiennych.

```
int i;  
double f = 3.14;  
  
i = (int)f;  
System.out.println(f);  
System.out.println(i);
```

- Usuń rzutowanie (int) i zobacz reakcję kompilatora.
Przywróć na powrót rzutowanie.
- W klasie głównej dopisz deklaracje dwóch zmiennych typu Object.
- Utwórz obiekt typu CalcInt i przypisz referencję do niego dla pierwszej zmiennej typu Object. Wywołaj jakąś metodę klasy posługując się rzutowaniem. Wyświetl wynik.
- Utwórz obiekt typu ExtCalcInt i przypisz referencję do niego dla drugiej zmiennej typu Object. Wywołaj jakąś metodę klasy posługując się rzutowaniem. Wyświetl wynik.

```
Object o1, o2;  
o1 = new CalcInt();  
result = ((CalcInt) o1).subst(1, 9);  
System.out.println(result);  
  
o2 = new ExtCalcInt();  
result = ((CalcInt) o2).subst(1, 9);  
System.out.println(result);
```

- Uruchom i przetestuj działanie aplikacji.

Zwróć uwagę na różne efekty dla różnych prób rzutowania.

2.8. Obsługa wejścia (konsoli)

Obsługę wejścia (tu: wczytywanie danych podawanych z konsoli/klawiatury) można zrealizować na różne sposoby. Dwa z nich to:

- skorzystanie z klasy Scanner
- skorzystanie z buforowanego Reader'a strumieni.

Aby wczytywać dane za pomocą klasy Scanner należy:

- zimportować klasę Scanner,
- zadeklarować zmienną (obiekt) typu Scanner,
- utworzyć obiekt tego typu dla standardowego strumienia wejścia,
- wywołać odpowiednią metodę odczytu tej klasy

- Dopisz zmienną globalną typu **Scanner** dla klasy głównej (bezpośrednio w ciele klasa – poza funkcją main):

```
private static Scanner skaner;
```

- Ponieważ to wymagana zainportuj klasę z biblioteki:

```
import java.util.Scanner;
```

To samo proponuje kompilator dla naprawy błędu.

- Dopisz kod dla wczytania wartości i wyświetlenia ich:

```
[...]
skaner = new Scanner(System.in);
System.out.println("Enter a string: ");
String s = skaner.next();
System.out.println("You have entered: "+s);
System.out.println("Enter integer value: ");
int i2 = skaner.nextInt();
System.out.println("You have entered: "+i2);
[...]
```

Aby wczytywać dane z użyciem buforowanego strumienia należy:

- zainportować klasę **BufferedReader**,
(jeśli używamy inne strumienie to te również zainportować),
- utworzyć obiekt tego typu,
- wywołać metodę **readLine** tej klasy

- Dopisz kod dla wczytania wartości i wyświetlenia ich:

```
...
BufferedReader br = new BufferedReader(
                    new InputStreamReader(System.in));
System.out.println("Podaj ciąg znaków: ");
String s2 = br.readLine();
System.out.println("Podaj liczbę całkowitą: ");
int i3 = Integer.parseInt(br.readLine());
```

- Zainportuj odpowiednie klasy:

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
```

Uwaga: operacje tej klasy wymagają większej kontroli i obsługi wyjątków!

- Dodaj wykonanie instrukcji w bloku **try ... catch...** lub rzucanie wyjątku przez całą metodę w której wywoływane są instrukcje – tu:

```
public static void main(String[] args) throws IOException {
    ...
}
```

- Dopisz w programie jego kończenie dopiero jak użytkownik naciśnie klawisz <Enter> (po wczytaniu czegokolwiek) – np. dopisz na końcu:

```
System.out.println("Press <ENTER>...");
br.readLine();
```

2.9. Kompilacja i uruchamianie w konsoli

- Otwórz konsolę (polecenie cmd) i utwórz katalog roboczy dla programu, a następnie skopiuj pliki źródłowe programu *.java do tego katalogu.
- Skompiluj program w konsoli polecienniem javac
 - jako parametr należy podać nazwę klasy głównej (wraz z rozszerzeniem .java).
- Uruchom program z konsoli polecienniem java w oddzielnym oknie
 - należy poprzedzić polecenie polecienniem start (czyli start java)
 - jako parametr trzeba podać nazwę pliku dla klasy głównej (bez rozszerzenia .class).

3. Zadanie – część II.

Zapoznaj się z typami danych stosowanych w języku Java. Opanuj techniki tworzenia aplikacji Java w jak największym stopniu.

Ćwiczenie 1

XML-RPC

Autor: Mariusz Fraś

1 Cele ćwiczenia

Celem ćwiczenia jest:

1. Poznanie ogólnej architektury aplikacji XML-RPC.
2. Opanowanie wybranej technologii tworzenia aplikacji XML-RPC.

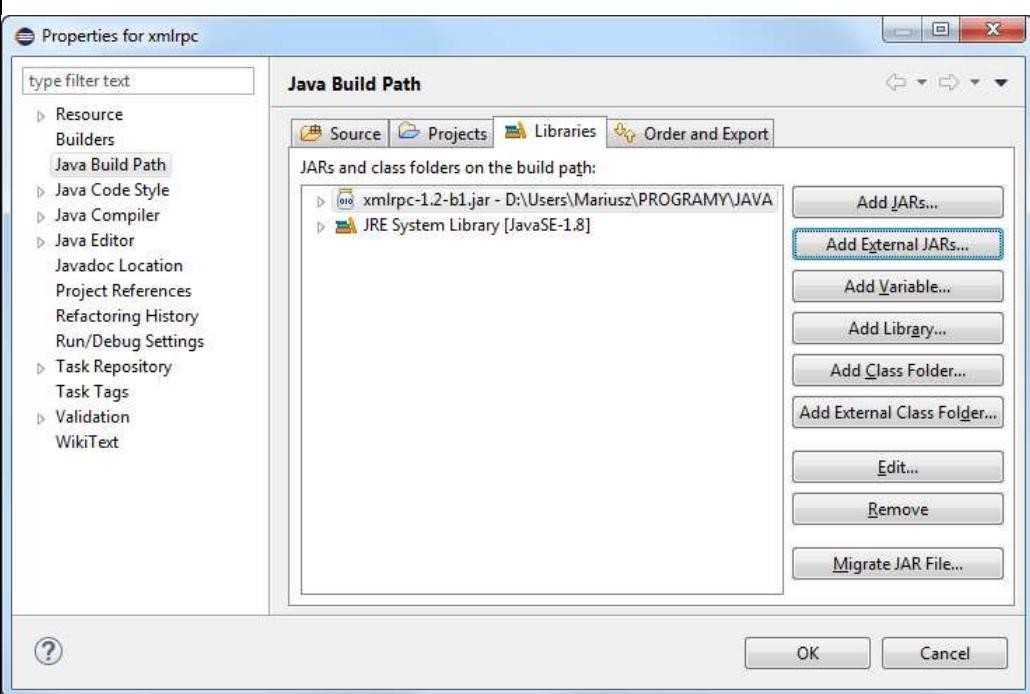
2 Środowisko deweloperskie

Standardowe (i zarazem wymagane) oprogramowanie do tworzenia i testowania aplikacji XML-RPC stosowane w laboratorium to:

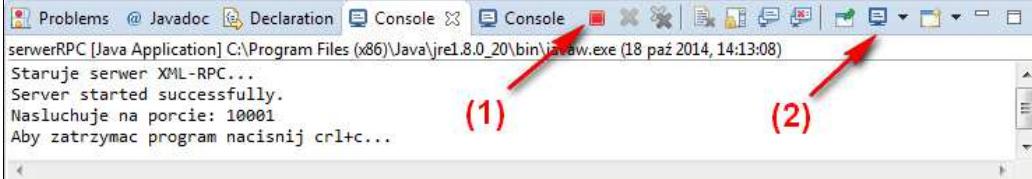
- System operacyjny Windows 7 lub wyższy
- Java 2 Software Development Kit (JDK).
- Środowisko programistyczne Java – np. Eclipse lub IntelliJ IDEA.
- Pakiet **org.apache.xmlrpc**
Informacje o pakiecie są w kursie na eportalu.

3 Zadanie – część I

Podstawowe elementy tworzenia aplikacji XML-RPC

1. Wstępne przygotowanie środowiska	<ul style="list-style-type: none"> • Przygotuj katalog roboczy, w którym będzie tworzona aplikacja. • Z eportalu pobierz pakiet xmlrpc-1.2-b1.jar do katalogu roboczego • Przygotuj komplikację programów Java z linii komend <ul style="list-style-type: none"> – sprawdź czy jest w systemie ścieżka dostępu do katalogu bin JDK – użyj poleceń set lub path, lub echo %path%. <i>/ ścieżka powinna być odpowiednio ustawiona /</i> – jeśli nie ma jej to ustaw ścieżkę dostępu do katalogu bin JDK w zmiennej PATH <ul style="list-style-type: none"> - dodanie nowej ścieżki dostępu wykonuję się komendą: set path=%path%;nowa_ścieżka_dostępu - lub w zmiennych środowiskowych, w zaawansowanych ustawieniach apletu System, w Panelu sterowania. <p>Uwaga: błędne wykonanie komendy może wykasować wszystkie ścieżki, niezbędne do prawidłowej pracy systemu operacyjnego.</p> • Przejdz w oknie poleceń do katalogu roboczego i sprawdź efekty przez uruchomienie z wiersza poleceń kompilatora poleceniem: javac (wywołanie kompilatora potwierdzi poprawność ustawień).
2. Utworzenie kodu serwera RPC	<ul style="list-style-type: none"> • Utwórz w platformie Eclipse projekt Java o własnej nazwie (tu: xmlrpcserwer) • Podczas tworzenia projektu lub tuż po utworzeniu dodaj do projektu pakiet xmlrpc-1.2-b1.jar (właściwości projektu → <i>Java Build Path</i> → zakładka <i>Libraries</i> → <i>Add External JARs...</i>) – patrz rysunek. 

	<ul style="list-style-type: none"> Utwórz nową klasę serwera RPC (tu nazwa: serwerRPC) Podczas tworzenia serwera zaznacz opcję tworzenia metody statycznej public static main(...). Zimportuj klasę serwera RPC z biblioteki RPC – dopisz na początku kodu: <pre><code>import org.apache.xmlrpc.WebServer;</code></pre> <ul style="list-style-type: none"> Dopisz do klasy metodę publiczną o nazwie echo, która będzie zwracała sumę dwóch zmiennych całkowitych podanych jako parametry wywołania metody, np. <pre><code>public Integer echo(int x, int y) { return new Integer(x+y); }</code></pre> <ul style="list-style-type: none"> W metodzie main w bloku: <pre><code>try { ... } catch (Exception exception) { System.out.println("Serwer XML-RPC: " + exception); }</code></pre> <p>dodaj kod (w miejsce kropek):</p> <pre><code>System.out.println("Startuje serwer XML-RPC..."). int port = xxx; WebServer server = new WebServer(port); //ponizej tworzy się obiekt swojej klasy serwera // i uruchomia się go: server.addHandler("MojSerwer", new serwerRPC()); server.start(); System.out.println("Serwer wystartował pomyslnie."); System.out.println("Nasluchuje na porcie: " + port); System.out.println "Aby zatrzymać serwer nacisnij crtl+c");</code></pre> <p>zamiast xxx podaj wartość 10000+nr komputera w laboratorium.</p> <ul style="list-style-type: none"> Sprawdź poprawność kodu.
3. Utworzenie kodu klienta RPC	<ul style="list-style-type: none"> Utwórz w Eclipse drugi projekt Java o własnej nazwie (tu: xmlrpcklient) Podczas tworzenia projektu lub tuz po utworzeniu dodaj do projektu pakiet xmlrpc-1.2-b1.jar – tak jak dla serwera. Utwórz nową klasę klienta RPC Podczas tworzenia serwera zaznacz opcję tworzenia metody statycznej public static main(...).

	<ul style="list-style-type: none"> Zimportuj klasę serwera RPC z biblioteki RPC – dopisz na początku kodu: <pre><code>import org.apache.xmlrpc.XmlRpcClient;</code></pre> <ul style="list-style-type: none"> W metodzie main w bloku <pre><code>try { ... }</code></pre> <p>dodaj:</p> <ul style="list-style-type: none"> - kod utworzenia obiektu wywołania metody serwera, - upakowania parametrów dla wywołania metody echo, - wywołania samej metody, <p>t.j.:</p> <pre><code>XmlRpcClient srv = new XmlRpcClient("http://localhost:xxx"); Vector<Integer> params = new Vector<Integer>(); params.addElement(new Integer(13)); params.addElement(new Integer(21)); Object result = srv.execute("MojSerwer.echo", params);</code></pre> <p>Zamiast xxx podaj port = 10000 + numer komputera w laboratorium.</p> <ul style="list-style-type: none"> Dodaj także wyświetlanie otrzymanej wartości. W tym celu najpierw przekształć rezultat na wartość o odpowiednim typie: <pre><code>int wynik = ((Integer) result).intValue();</code></pre> <p>i wyświetl wynik na ekranie;</p>
4. Testowanie działania aplikacji	<ul style="list-style-type: none"> Uruchom serwer w Eclipse Uruchom klienta w Eclipse Skontroluj w konsolach wyniki działania. Zatrzymaj serwer. <p>Uwaga: w Eclipse zatrzymuje się serwer nie za pomocą sekwencji klawiszy ctrl-c (jak w konsoli) ale naciśnięciem ikony (1) okna konsoli jak na rysunku poniżej</p>  <p>Przełączanie między konsolami jest możliwe za pomocą przycisku (2) lub wybierając odpowiednią zakładkę.</p>

5. Procedura asynchroniczna	<ul style="list-style-type: none">• W projekcie klienta utwórz nową klasę, która będzie zawierała metody wywoływane gdy na serwerze zakończy się przetwarzanie procedury wywoływanej asynchronicznie.<ul style="list-style-type: none">◦ Utwórz w projekcie klasę◦ Zdefiniuj klasę publiczną implementującą interfejs AsyncCallback (tu klasę AC).<pre>public class AC implements AsyncCallback { ... }</pre>◦ Dopisz wymagane przez interfejs dwie metody<pre>public void handleResult(Object resultat, URL url, String metoda)</pre><p>oraz:</p><pre>public void handleError(Exception e, URL url, String metoda)</pre><ul style="list-style-type: none">◦ W obu metodach dopisz wyświetlanie na ekranie odpowiednich (łatwo rozpoznawalnych) komunikatów.• W kodzie klienta (w klasie głównej, w funkcji main) dopisz kod<ul style="list-style-type: none">◦ deklaracja i utworzenie obiektu typu implementującego interfejs AsyncCallback (tu obiektu typu AC):<pre>AC cb = new AC();</pre>◦ utworzenia obiektu parametrów (tu: vector) dla wywołania nowej metody asynchronicznej execAsy,◦ wywołania metody asynchronicznej execAsy z parametrami: nazwa procedury, obiekt parametrów procedury, wskazanie na obiekt którego metody będą wywołane po zakończeniu procedury.<pre>... Vector<Integer> params2 = new Vector<Integer>(); params2.addElement(new Integer(3000)); srv.executeAsync("MojSerwer.execAsy", params2, cb); System.out.println("Wywolano asynchronicznie"); ...</pre>• Zwróć potem uwagę na komunikaty po wywołaniu procedury.• Dopisz w kodzie serwera jeszcze jedną metodę – wywoływaną asynchronicznie procedurę execAsy. Procedura będzie teoretycznie długo coś wykonywać, co zasymulowane będzie wstrzymaniem wykonywania na okres x milisekund (x – parametr podawany w wywołaniu procedury) metodą Thread.sleep(x).
-----------------------------	--

	<pre>public int execAsy(int x) { System.out.println("... wywołano asy - odliczam"); try { Thread.sleep(x); } catch(InterruptedException ex) { ex.printStackTrace(); Thread.currentThread().interrupt(); } System.out.println("... asy - koniec odliczania"); return (123); }</pre>
6. Testowanie działania aplikacji	<ul style="list-style-type: none"> Uruchom serwer a następnie klienta w Eclipse Zwróć uwagę na kolejność i momenty wypisywania komunikatów. Zauważ, że po wywołaniu metody asynchronous, natychmiast wykonywane są kolejne instrukcje w kliencie i dopiero po pewnym czasie wywoływana jest zwrotnie metoda handleResult i wypisywany jest jej komunikat. Można zmienić kolejność wywoływanie metod w kliencie (lub dopisać kolejne) aby zaobserwować działanie aplikacji.
7. Kompilacja, uruchamianie i testowanie aplikacji w konsoli Windows	<ul style="list-style-type: none"> Przejdź do konsoli Windows (Wiersz polecenia) Skopiuj do katalogu roboczego następujące pliki: <ul style="list-style-type: none"> - xmlrpc-1.2-b1.jar - pliki *.java (źródła aplikacji) Skompiluj najpierw serwer, a następnie klienta poleceniem javac <ul style="list-style-type: none"> – aby dodać bibliotekę jar wykorzystaj parametr -classpath podając po parametrze nazwę pliku jar poprzedzoną znakami ":" (człysłów nie podawać). Uruchom serwer w oddzielnym oknie poleceniem start java <ul style="list-style-type: none"> - parametr classpath użyj analogicznie jak wyżej. Uruchom klienta poleceniem java <ul style="list-style-type: none"> - parametr classpath użyj analogicznie jak wyżej. <p>Podpowiedź: w poleceniu javac podaje się rozszerzenia plików, w poleceniu java nie podaje się rozszerzenia plików.</p>
8. Testowanie w sieci	<p style="color: red;">Ta część ćwiczenia może być zmodyfikowana przez prowadzącego.</p> <ul style="list-style-type: none"> Skompiluj klienta i serwera tak, aby można je było uruchomić na dwóch różnych komputerach w sieci (podając adres IP) <i>lub</i> Ustal z wybraną osobą wykonującą ćwiczenie wzajemne wykorzystanie utworzonych serwerów. Skompiluj odpowiednio klienta tak, jak dla pierwszej opcji. Uruchom klienta i serwer na dwóch różnych komputerach w laboratorium i przetestuj działanie.

4 Zadanie – część II

Szczegółowe wymagania i wybór zadania (A, B lub inne) określa prowadzący grupy.

A. Przykładowe zadanie do przećwiczenia.

Napisać aplikację XML-RPC (może to być aplikacja "konsolowa" (wiersza poleceń)) spełniającą następujące wymagania:

1. Serwer RPC wykonuje kilka działań (oferuje kilka procedur), **inne niż w podanej instrukcji ćwiczenia**, wymagających podania **więcej niż jednego parametru**.
2. Przynajmniej jedna procedura zawiera **parametry różnego typu**. Wykorzystać w tym celu w kliencie np. wektor zmiennych typu Object.
3. Przynajmniej jedna procedura jest **wywoływana asynchronicznie**.
4. Serwer zawiera usługę/procedurę **show**, która podaje informacje o dostępnych procedurach – wyświetla ich listę z opisem (nazwa procedury, parametry, krótki opis)
5. Dodać w aplikacji funkcjonalność:

przy uruchamianiu klienta można podać adres dostępowy usług serwera – adres IP lub DNS i port – i dopiero potem łączyć się ze wskazanym serwerem.

B. Zastanowić się jak zaimplementować aplikację, żeby klientowi wystarczyła informacja o adresie serwera i metodzie *show()* – aby można było wywoływać dowolną usługę serwera bez wcześniejszej znajomości tych usług oprócz znajomości usługi *show()*.

A więc, tak żeby nie trzeba było na sztywno kodować ich wywołania w kliencie wg specyfikacji.

Czyli nie znamy wcześniej usług (oprócz adresu i usługi *show()*), i chcemy móc z nich skorzystać bez powtórnego kodowania klienta. Poznajemy usługi tylko dzięki usłudze *show()*.

Uwaga: na serwerze nie powinna to być jedna sparametryzowana procedura z parametrami typu string (proste rozwiążanie ale nie to jest celem), ale oddzielne procedury dla każdej usługi.

C. Przygotować się do napisania na zajęciach aplikacji o podobnych funkcjonalnościach (testu z samodzielnego napisania klienta i serwera według podanych wskazówek).

EXERCISE 1

XML-RPC

Mariusz Fraś

Objectives of the exercise

1. Familiarization with the development environment for XML-RPC applications.
2. Understanding the general architecture of the XML-RPC application.
3. Mastering the techniques of performing synchronous and asynchronous XML-RPC calls.

1. Development environment

1.1. Prerequisites

The standard (and required) software for creating and testing XML-RPC applications used in the laboratory is:

- Windows 7 (or higher version) Operating System.
- Oracle Java 2 Software Development Kit (Oracle Open JDK) or similar.
- Java development environment (e.g. Eclipse or IntelliJ IDEA development platform).
- The **org.apache.xmlrpc** package

Information about the package is in the course on **eportal**.

2. Exercise – Part I

The basic elements of creating XML-RPC application

In the exercise, a simple XML-RPC client-server application will be implemented containing the following elements:

- a. RPC server containing object with remotely callable procedure.
- b. The base client that calls procedure synchronously.
- c. A part of the client that handles server response asynchronously.

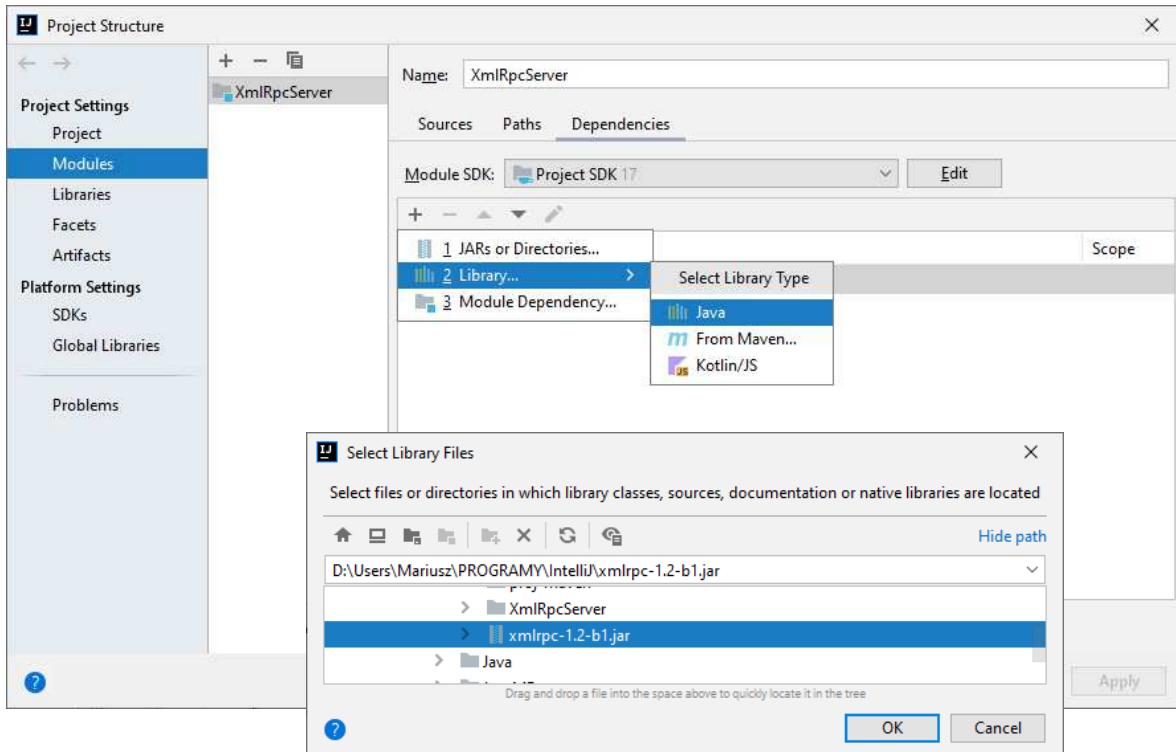
Remark: Throughout the manual, the dotted lines -... or [...] - denote a part of the code omitted in the description - code to be completed by yourself or presented in the manual earlier.

2.1. Initial preparation of the environment

- Prepare a working directory where the application will be created - eg **D:\Users\[user_dir]\Programs** or elsewhere - depending on your needs.
- Download from eportal the **xmlrpc-1.2-b1.jar** package to the working directory.
- Prepare the system environment for compilation of Java programs
 - Note: 1) incorrect execution of the below command can delete all paths necessary for the proper operation of the operating system.
 - 2) In the class lab the whole environment should already be set up and you probably don't need to do anything else.
 - run system console and check if there is a path to the JDK bin directory in the system - use the path or echo %path% command
 - if it is not already set, add the path to the bin JDK directory
 - in Windows you can do it by setting up environment variable in Advanced settings of the System applet in the Control Panel, or using *Change environment variables* option in *User Account configuration*.
- Go to the working directory in the command console window and check the effects by running the compiler using command **javac --version** – displaying info by the compiler will confirm the correctness of the settings.

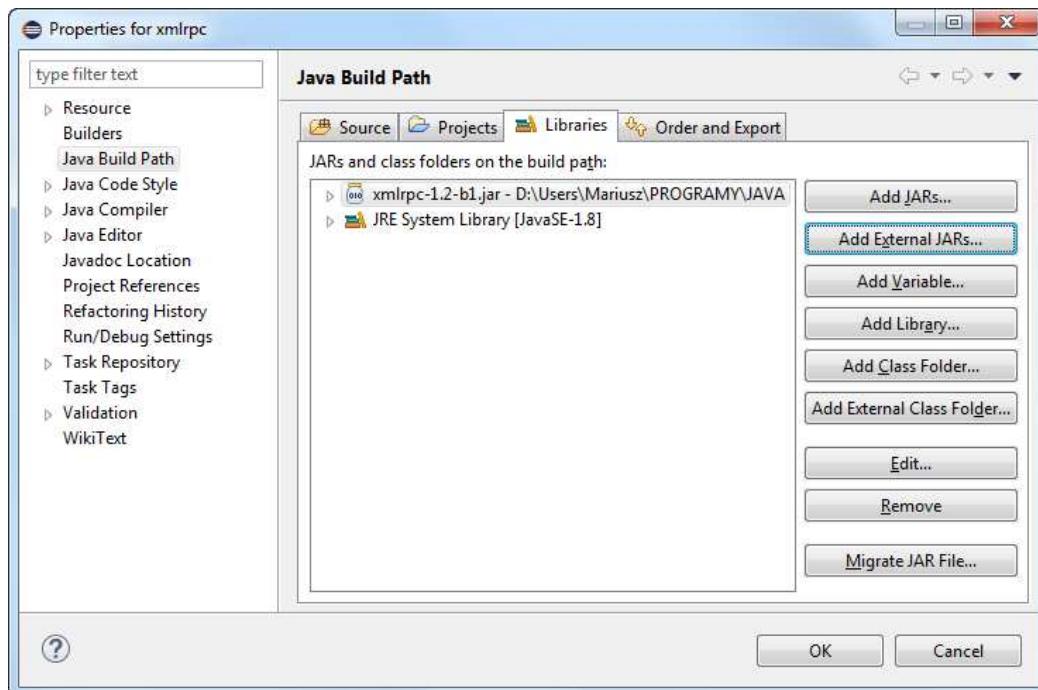
2.2. Building the RPC server code

- Create a Java project with its own name (here, **XmIRpcServer** name is used).
For the server and client it can be separate projects or you can create one XmIRpcApp project and two separate modules (e.g. XmIRpcServer and XmIRpcClient) for each app component – both approaches works.
- Add **xmlrpc-1.2-b1.jar** library package to the project.
 - a) in IntelliJ IDEA:
File → Project Structure... , in the Project Structure window select Modules node, Dependencies tab, and add (+ sign) Library... - select xmlrpc-1.2-b1.jar from file system.



b) in Eclipse

In Project Properties → Java Build Path → tab Libraries → Add External JARs...



- Create a new class - the main class of the server application – with **main** method.
- Create a new class (here named **MyService**) – the class with procedures to be called remotely. It can used the same main class but separate one makes the code clear and readable.
- Add the code of public **compute** method, which will return the sum of two integer variables given as method's call parameters – e.g.:

```
public class MyService {
    int i = 1;
    public Integer compute(int x, int y) {
        System.out.println(i++);
        return x+y;
    }
}
```

- In the main class **RpcServer**, in **main** method, in the block:

```
try {
[...]
} catch (Exception exception) {
    System.err.println("XML-RPC Server: " + exception);
}
```

Add the code (instead of dots):

```
System.out.println("Starting XML-RPC server...");
int port = xxx;

WebServer server = new WebServer(port);

//Below server object with name MyService is created and run:
server.addHandler("MyService", new MyService());
server.start();
System.out.println("Server started successfully.");
System.out.println("Listening on port: " + port);
System.out.println("Press <ENTER> to stop the server.");

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
br.readLine();

server.shutdown();
```

Instead **xxx** use the value **10000 + computer number in the lab** or any free port on your own computer.

- Check the correctness of the code. Correct if necessary.

2.3. Building the RPC client code

- Create a second Java project (here named: **XmlRpcClient**) or new module in the one shared project
- Add **xmlrpc-1.2-b1.jar** package to the project – just like for the server.
- Create a new client **main** class (here named **RpcClient**) with **main** method.
- In the **main** method, in the block:

```
try {
[...]
} catch (Exception exception) {
    System.err.println("XML-RPC Client: " + exception);
}
```

add the code (instead of dots):

- code to create the object that is used to call server methods (procedures),
- packing parameters for the server procedure call,
- calling the method itself,

i.e.:

```
XmlRpcClient srv = new XmlRpcClient("http://localhost:xxx");
Vector<Integer> params = new Vector<Integer>();
params.addElement(13);
params.addElement(21);

Object result = srv.execute("MyService.compute", params);
System.out.println("result = " + result);
```

Instead **xxx** use the value **10000 + computer number in the lab** or any free port on your own computer.

- Check the correctness of the code. Correct if necessary.
- Also add the code to display the result on the screen.

2.4. Testing the operation of the application

- Start the server in the development platform.
- Start the client in the development platform.

Notes:

- in the platform (as well IntelliJ IDEA as Eclipse) you can switch between consoles of apps,
- be aware that you shouldn't run the server if the previous run was not finished (and the port is still used).
- Check the results of the operation on the consoles.

Testing with Web API test tools – here Postman.

- Run the Postman (or similar tool – e.g. SoapUI) to send HTTP request.
- In the tool window set up the HTTP POST request:
 - enter the address of the server service – here localhost and proper port,
 - add **Content-type** header and set it as **text/xml**,

Params	Authorization	Headers (9)	Body	Pre-request Script	Tests	Settings	Cookies
<input checked="" type="checkbox"/> Content-Type			text/plain				
<input checked="" type="checkbox"/> Content-Length			<calculated when request is sent>				Go to body
<input checked="" type="checkbox"/> Host			<calculated when request is sent>				
<input checked="" type="checkbox"/> User-Agent			PostmanRuntime/7.29.0				
<input checked="" type="checkbox"/> Accept			*/*				
<input checked="" type="checkbox"/> Accept-Encoding			gzip, deflate, br				
<input checked="" type="checkbox"/> Connection			keep-alive				
<input checked="" type="checkbox"/> Content-Type			text/xml				

- enter the body as valid **xml-rpc** request:

```
<?xml version="1.0"?>
<methodCall>
    <methodName>MyServer.echo</methodName>
    <params>
        <param>
            <value><int>8</int></value>
        </param>
        <param>
            <value><int>13</int></value>
        </param>
    </params>
</methodCall>
```

You may select *raw/XML* format of body window.

- Check the results of the operation. Test what happen if you enter wrong address or wrong names, or wrong parameters.
- Stop applications.

2.5. Asynchronous procedure

- Add in the server (in **MyService**) the code of one more method **asyProc** which will be called asynchronously.

The procedure will theoretically be performed for a long time, which will be simulated by suspending execution for the period of x milliseconds (x – the parameter given in the procedure call) by the `Thread.sleep(x)` method.

```
public int asyProc(int x) {
    System.out.println("...asyProc called - processing");
    try {
        Thread.sleep(x);
    } catch(InterruptedException ex) {
        ex.printStackTrace();
        Thread.currentThread().interrupt();
    }
    System.out.println("... asyProc - finished");
    return (123);
}
```

- In the client project/module, create a new class that will contain methods called when the processing of the procedure invoked asynchronously by the client completes on the server.
 - Create.
 - Define a new public class in the project (here named **AC**) that implements the interface `AsyncCallback` (from `xml-rpc` library).
 - Add two methods required by the interface:

```
public void handleResult(Object result, URL url, String procName)
```

and:

```
public void handleError(Exception e, URL url, String procName)
```

- In both methods, add the code that display appropriate (easily recognizable) messages on the screen (including the result).
- In the client's code (in the main class, in the `main` function) add:
 - declaration and creation of an object (here named `cb`) implementing the interface `AsyncCallback` (here object of type `AC`):
 - creation of the call parameters object (vector) for the new asynchronous method `execAsy`,
 - insert parameters into the vector for the method being called `execAsy`,
 - calling the method itself (with the third parameter indicating the object whose methods will be called after the called procedure will be finished).

```
AC cb = new AC();
Vector<Integer> params2 = new Vector<Integer>();
params2.addElement(1000);
srv.executeAsync("MyService.asyProc", params2, cb);
System.out.println("Called asynchronously");
```

Later, pay attention to the messages after calling the procedure.

- Add the code to NOT finish the client immediately – to be able to restive the callback after some time.

2.6. Testing the operation of the application

- Start first the server and then the client.
- Pay attention to the order and moments of displaying messages..
Note that after invoking the method asynchronously, subsequent instructions are executed immediately in the client and only after a certain time the `handleResult` method is called back and its message printed.
- Change the order of calling procedures in the client (or add another one) to observe the operation of the application.

2.7. Compilation, running and testing of applications in the console

- Go to the Windows console (Command Prompt)
- Copy the following files to the selected working directory:
 - `xmlrpc-1.2-b1.jar`
 - `*.java` files (application source code)

You may work on existing folder but remember about proper pointing folders.
- Compile the server first and then the client with the `javac` command
 - to add a jar library, use the `-classpath` parameter preceded by the string `".;"` (do not specify the quotation marks).

Hints:

- 1) in the `javac` command, file name extensions should be given,
- 2) in the `java` command no file extension should be given.
- 3) the `-classpath` (or `-cp` for short) parameter specifies class search path of directories and zip/jar files

- Start the server in a separate window with **start java** command
 - use the **classpath** parameter in the same way as above.
- start the client with the **java** command
 - use the **classpath** parameter in the same way as above.

2.8. Network test

Note: this item is to test if possible (e.g. in the lab class) and can be modified by the teacher.

- Compile the client and server so that they can be run on two selected computers on the network (e.g. by providing an IP address)
- Run the client and server on two different computers in the lab and test the operation.

3. Exercise – part II.

Selection of the A, B or the other one is determined by the teacher.

- A. Prepare to write a program in the classroom, containing elements with similar functionalities, according to the teacher's instructions.
- B. Write an XML-RPC application that meets the following requirements:
 - 1. The RPC server performs several actions (it offers several procedures), other than in the given exercise instruction, that require more than one parameter.
 - 2. At least one procedure contains parameters of various types. For this purpose, use a vector of Object type variables in the client.
 - 3. At least one procedure is called and serviced asynchronously.
 - 4. The server includes a show service which gives information about available procedures - displays their list with description (procedure name, parameters, short description).
 - 5. Add the following functionality in the application:
 - a. When starting the client, you can specify the address of the server service – IP address or DNS address, and port.
 - b. Invoke each service by entering the service name (procedure name).
 - c. For invoked services, enter the call parameters from the command line.
- C. Think about how to implement the application so that it would be enough for the client to only receive information about the server's address and the *show()* procedure – so that any server service (procedure) can be called without prior knowledge of these services (apart from the *show()* service). So that you do not have to hard encode their calls in the client according to the server specification.

In other words we do not know the services in advance (except address and *show()* procedure), and we would like to be able to call them without re-coding the client. We get to know the available procedures thanks to the *show()* service only.

Note: the solution should not be based on one parameterized procedure (method) with string parameters (trivial but improper solution). I should be separate method.

The detailed and final requirements for Part II are set by the teacher.

Ćwiczenie 2

gRPC

(*tymczasowo w języku angielskim*)

Autor: Mariusz Fraś

1 Objectives of the exercise

The purpose of the exercise is:

1. Familiarization with the development environment for gRPC applications.
2. Understanding the general architecture of the gRPC application.
3. Mastering the basic techniques of creating gRPC applications.

2 Development environment

The developing of gRPC applications can be performed using various platforms. Here is considered the following environment:

- Windows 7 (or higher) Operating System (eventually MacOS).
- Visual Studio 2017 or 2019 (eventually Java programming platform (also with SpringBoot)),
- Proper NuGet packages.

3 Exercise – Part I

Creating simple gRPC application project.

3.1 Visual Studio

Visual Studio 2017 has got quite good support for building gRPC applications with help of NuGet packages management. Visual Studio 2019 has got dedicated gRPC app project.

Note:

From [nugget.org](https://www.nuget.org/): "*NuGet is the package manager for .NET. The NuGet client tools provide the ability to produce and consume packages. The NuGet Gallery is the central package repository used by all package authors and consumers.*"

NuGet is a package manager aimed to enable developers to share reusable code.

NuGet's client (nuget.exe) is a free and open-source command-line app that can create and consume packages. NuGet is distributed as a Visual Studio extension, and it can natively consume NuGet packages.

A. Visual Studio 2017.

The following steps describe developing simple gRPC application (client and server) with VS 2017.

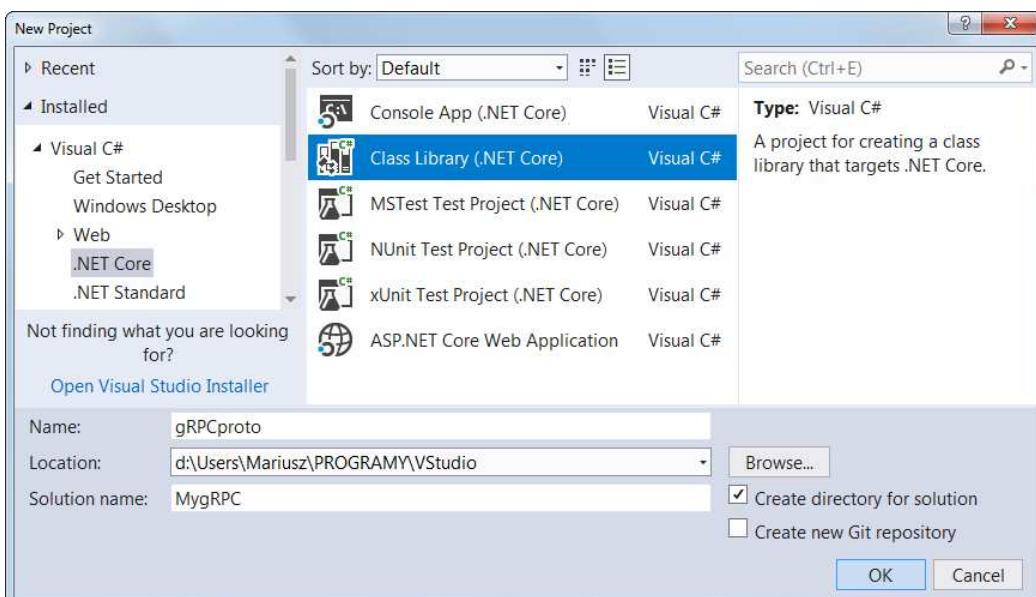
In this instruction one solution with 3 projects will be created to develop simple gRPC server and gRPC client.

1. Create a new C# project (new solution) for proto interface.

This will be for creating code of stubs for server and client – after compilation the proper source code for server and client will be generated.

It can be used Class Library (.NET Core) or . Class Library (.NET Standard) project – both will work. Here we use the first one:

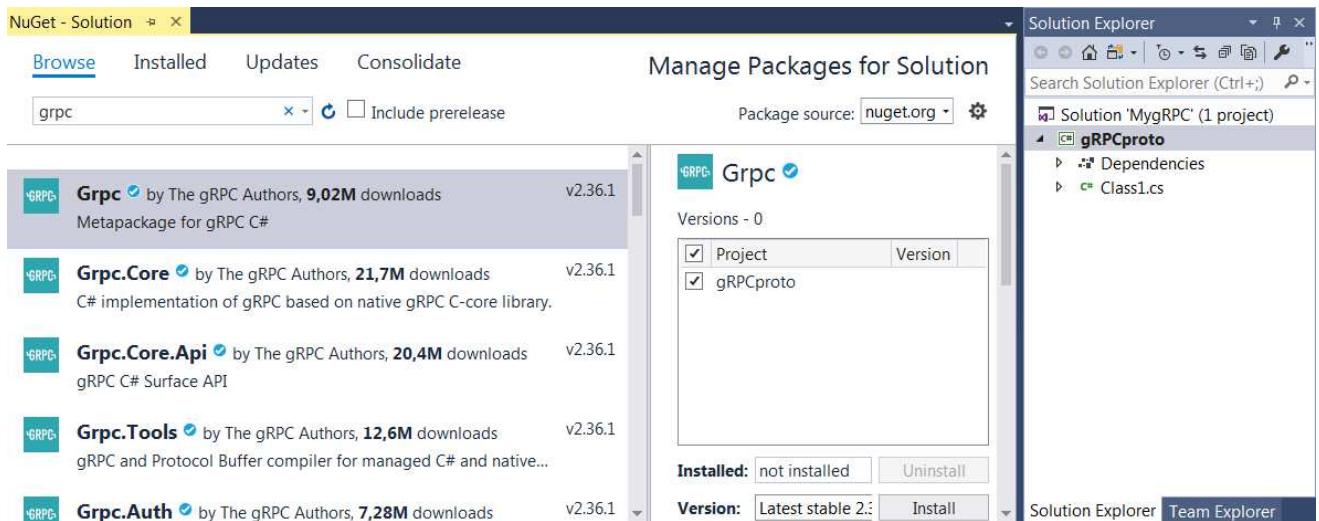
- Create Class Library (.NET Core) project in a new solution (here named gRPCproto in MygRPC solution).



2. Configure the project (here named gRPCproto) for generation of interface stubs.

In order to compile proto files it must be included several packages with use of NuGet manager.

- Select the project in Solution Explorer window, and from platform menu or context menu select *Manage NuGet Packages* option.
- Select *Browse* tab, enter **grpc** in *Search* field.



- Select and add the following packages to the project:
 - Grpc (metapackage for gRPC) – this will also add other necessary packages,
 - Grpc.Tools (gRPC and Protocol Buffers compiler)

- Enter **protobuf** in *Search* field.

Select and add the following package:

- Google.Protobuf (C# runtime library for Protocol Buffers)

- You may delete initially created Class1.cs file – it will be not used.

3. Create proto file and interface defined in proto file.

Here the interface for creating stubs for server and client will be defined in proto file.

The remote procedure **addInt** with two parameters that adds two integers will be defined. The procedure returns the result and some comment (string value).

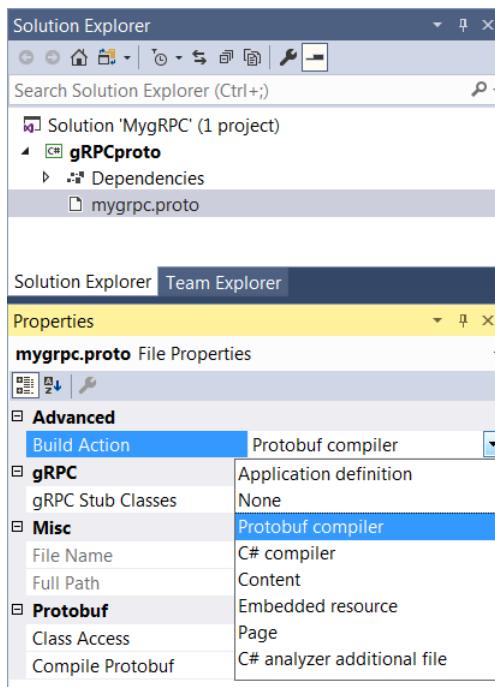
- Create manually or using platform options (*Add New Item* in menu) a text proto file with proto filename extension – here named mygrpc.proto).
- Enter the following content that defines:
 - Protocol Buffers version,
 - service name, remote procedure name, output, and input parameters (messages),
 - input (request) message – parameters in call,
 - output (response) message – replay for call.

```

syntax = "proto3";
package mygrpcproto;
// Service definition.
service MyGrpcSrv {
    rpc addInt (AddIntRequest) returns (AddIntReply) {}
}
// The request message
message AddIntRequest {
    int32 num1 = 1;
    int32 num2 = 2;
}
// The response message
message AddIntReply {
    int32 result = 1;
    string comment = 2;
}

```

- Select the proto file in *Solution Explorer* window and bellow in *Properties* window select in *Build Action* field the *Protobuf compiler* action.

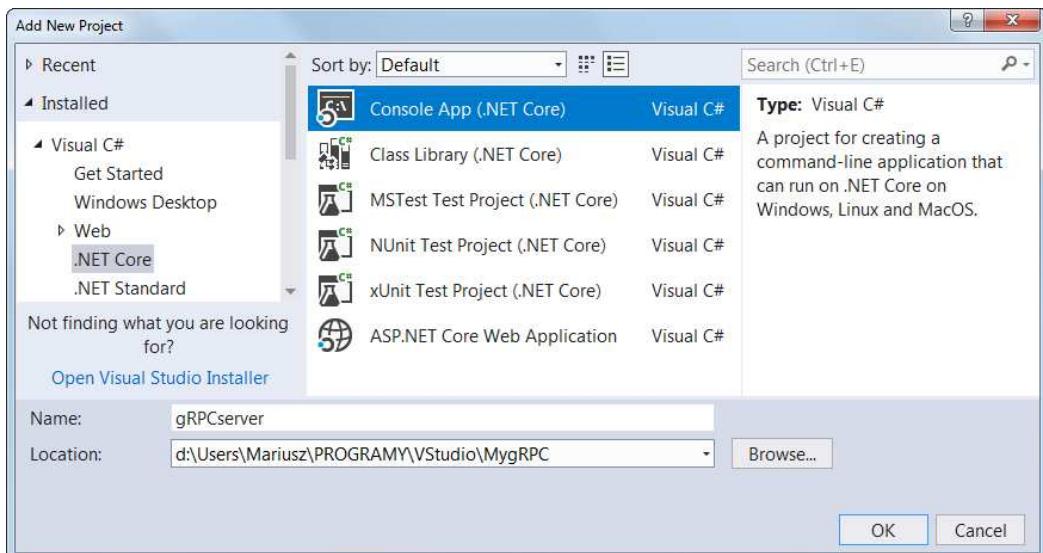


- You may compile (build option) the project to check that all works well.
Pay attention for generated files with classes in **obj** directory of the project – among the others source code for server and client stubs.

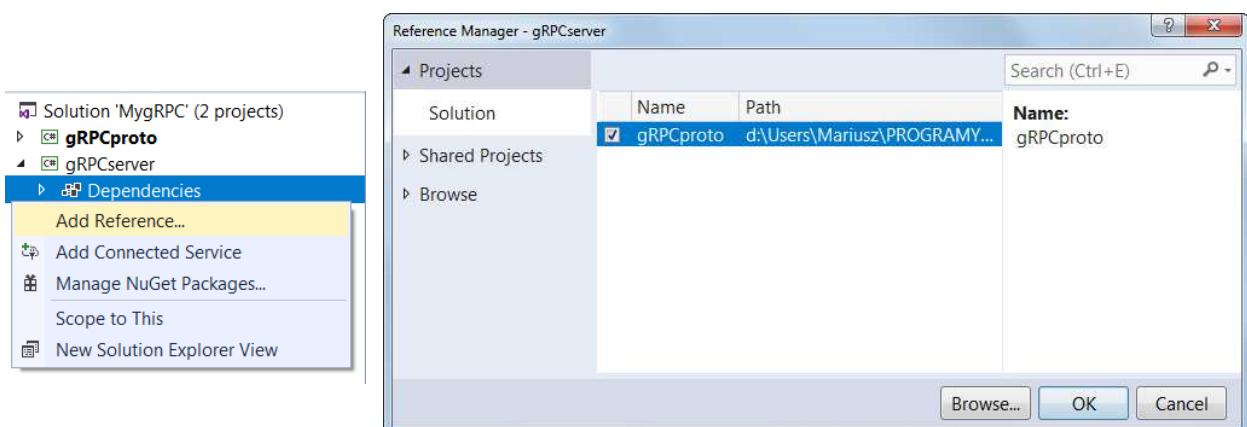
4. Create in the same solution the gRPC server project.

Here the server .Net console application will be created.

- Add to the solution the Console App (.NET Core) project – here named gRPCserver.



- Add the dependency (reference) to gRPCproto project (to use their generated classes). Select *Add Reference* from context menu (see the figure below), or edit *Project Dependencies*.



- Add the class implementing the remote procedure (the interface defined in proto file). The class must extend the base class with the name of ***service name in proto file*** tagged with ***Base*** word. This class is defined in the class generated from proto file named as ***service name in proto file***.

```
namespace gRPCserver {
    class MyGrpcSrvImpl : MyGrpcSrv.MyGrpcSrvBase {
        public override Task<AddIntReply> addInt(AddIntRequest req,
                                                    ServerCallContext ctx) {
            string comment;
            int result;

            ...
            return Task.FromResult(new AddIntReply { Result = result,
                                                    Comment = comment });
        }
        ...
    }
}
```

Set (calculate) result and comment values properly (e.g. comment the sign of the result).

Note:

- here the Task<> is used – this allow asynchronous processing,
- the ServerCallContext is not used here (is for potential future use).

- Add the code in Program class to create the server object and to run the server listening on given port:

```
namespace gRPCserver {
    ...
    class Program {
        const int port = 10000;
        static void Main(string[] args) {
            Console.WriteLine("Starting Hello gRPC server");
            Server myServer = new Server
            {
                Services = { MyGrpcSrv.BindService(new MyGrpcSrvImpl()) },
                Ports = { new ServerPort("localhost", port,
                                         ServerCredentials.Insecure) }
            };
            myServer.Start();
            Console.WriteLine("Hello gRPC server listening on port " + port);
            Console.WriteLine("Press any key to stop the server...");
            Console.ReadKey();
            myServer.ShutdownAsync().Wait();
        }
    }
}
```

Note:

- for the Server object the available services are defined (here our one service implementation) and ports where services calls come to,
 - the services are created with BindService method,
 - here we specify localhost and port=10000 as the procedure endpoint,
 - for simplicity we use not secure connection (hint: on Mac OS there are problems to use secure connection for gRPC).
- Correct compiler warnings/errors properly (usually by importing definitions – with using keyword). Make a note that name for proto file (name of package) start with capital letter (not like in name in proto file).

5. Create in the same solution the gRPC client project.

Here the client .Net console application will be created.

- Add to the solution the new Console App (.NET Core) project – here named gRPCclient.

- Add the dependency (reference) to gRPCproto project (to use their generated classes). Select *Add Reference* from context menu.
- Implement in Main method:
 - Creating a channel which will be used to communicate with server
 - specifying the address and port (here we run the server on localhost),
 - specifying no security mechanisms (for simplicity).
 - Creating client object – to make requests.

The client class is generated from the interface defined in proto file. This class is defined in the class named as ***service name in proto file***. The client class name is ***service name in proto file*** tagged with ***Client*** word.

- Calling remote procedure – here named **addInt** – with use of **AddIntRequest** object (from the class generated from proto file).
- Closing the channel when no more used.

```
static void Main(string[] args) {
    Console.WriteLine("Starting gRPC Client");
    Channel channel = new Channel("127.0.0.1:10000",
                                   ChannelCredentials.Insecure);
    var client = new MyGrpcSrv.MyGrpcSrvClient(channel);
    String str;
    int num1, num2;
    Console.Write("Enter number 1: ");
    str = Console.ReadLine();
    if (int.TryParse(str, out num1))
    {
        Console.Write("Enter number 2: ");
        str = Console.ReadLine();
        if (int.TryParse(str, out num2))
        {
            var reply = client.addInt(new AddIntRequest { Num1 = num1,
                                                          Num2 = num2 });
            Console.WriteLine("From server: " + reply.Comment + reply.Result);
        }
        else
            Console.WriteLine("Wrong value!");
    }
    else
        Console.WriteLine("Wrong value!");
    Console.WriteLine("Stopping gRPC Client");
    channel.ShutdownAsync().Wait();
}
```

6. Build the solution (compile/build all projects).

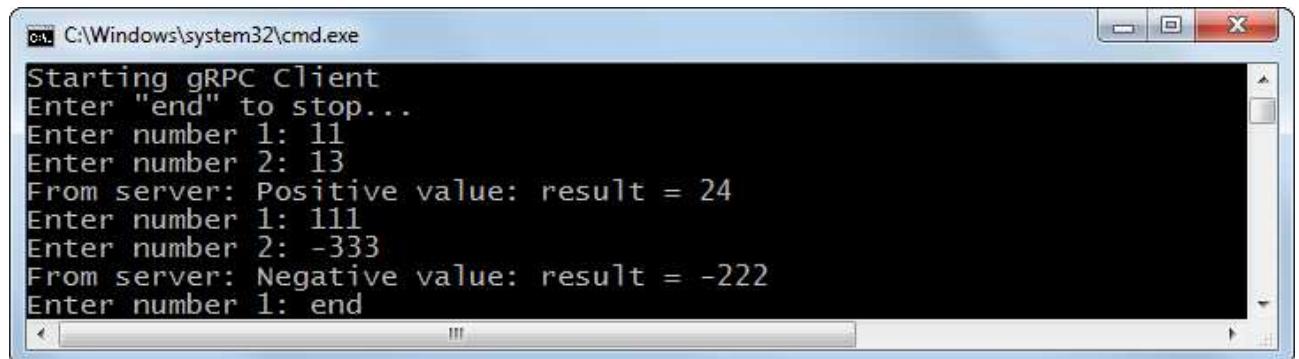
- Check the operation of the application.
- The result should be similar to below figures:

Server running in separate window (one process):



```
C:\Windows\system32\cmd.exe - dotnet gRPCserver.dll
Starting Hello gRPC server
Hello gRPC server listening on port 10000
Press any key to stop the server...
Called addInt (1)
Called addInt (2)
```

Client running in separate window (other process):



```
C:\Windows\system32\cmd.exe
Starting gRPC Client
Enter "end" to stop...
Enter number 1: 11
Enter number 2: 13
From server: Positive value: result = 24
Enter number 1: 111
Enter number 2: -333
From server: Negative value: result = -222
Enter number 1: end
```

Note:

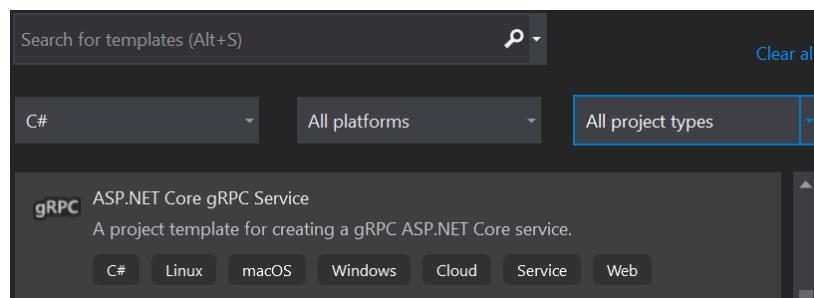
After proper creating and building the solution (all projects), the VS 2017 platform sometimes shows (after re-opening the solution (projects)) that errors exist for uses directive (as if proto namespace was not visible and classes generated from it). Anyway all works well (after some time errors may disappear) and the solution can be rebuilt without errors. This behavior of Visual Studio 2017 is strange and hard to explain.

B. Visual Studio 2019.

Visual Studio 2019 has got dedicated gRPC app project. It is recommended also to use several different NuGet packages.

1. Creation of C# project (new solution) for gRPC service (server).

- To create new gRPC app select *ASP.NET Core gRPC Service*. You can use Search field find the project faster.



After selection of names (in next window) you can specify *Target Framework* (default is .NET core 3.1), and eventually enable *Docker* option (don't do it now).

2. The project structure

The initial default project consist of:

- the initial proto file – **greet.proto**,
- the initial gRPC service - **GreeterService.cs**,
- the service configuration code – **Startup.cs**,
- the code for service hosting (by default Kestrel web server is used) – **Program.cs**,
- **appSettings.json** – contains configuration data
- the necessary dependences (including NuGet packadges).

To create your own solution you can as well edit *.cs and proto files as rename or add new ones.

Note the following:

- The default main NuGet package for the default gRPC service project is *GrpcAspNetCore* package which include:
Google.Protobuf, *Grpc.Tools*, and *GrpcAspNetCore.Server* packages.
- The project includes files with methods called by the runtime. It includes:
 - **Startup** class that configures services and the app's request pipeline. It has two default methods:
 - **Configure** method to create the app's request processing pipeline,
 - **ConfigureServices** optional method to configure the app's services. Services are registered here.
 - Enabling gRPC service
gRPC is enabled with the **AddGrpc** method in **ConfigureServices** method.
 - Routing
Routing in ASP..NET Core is responsible for matching incoming requests (especially HTTP ones) and dispatching those requests to the app's service endpoints.

The service hosted by ASP.NET Core gRPC, should be added to the routing pipeline with use of the **UseRouting()**, the **UseEndpoints()**, and the **MapGrpcService<TService>()** methods called in the **Configure** method. It adds endpoints to the gRPC service.

Startup class is typically specified by calling the **UseStartup<TStartup>** method when the app's host is built.

- Hosting the service

Host is the component that among the others encapsulates (and run) the services – when it is run it calls **StartAcync** on each service implementation..

- **Program** class (with **Main** method) is used to create and run the host for gRPC service.

- The default host for gRPC service is Kestrel – a cross-platform web server for ASP.NET Core (other host options are also available). Kestrel doesn't have some of the advanced features but provides the best performance and memory utilization. It requires HTTP/2 transport and should be secured with TLS.

Attention: MacOS doesn't support ASP.NET Core gRPC with TLS.

- Creation and running the host

In the main method it is built and run the host with builder method:

```
CreateHostBuilder(args).Build().Run();
```

```
public static void Main(string[] args)
{
    CreateHostBuilder(args).Build().Run();
}

public static IHostBuilder CreateHostBuilder(string[] args) =>
    Host.CreateDefaultBuilder(args)
        .ConfigureWebHostDefaults(webBuilder =>
    {
        webBuilder.UseStartup<Startup>();
    }
);
```

- Defining the host

The default HTTP host (Kestrel Web server) is built using:

```
CreateDefaultBuilder(args).ConfigureWebHostDefaults(...);
```

In the method it is specified Startup as the startup class with use of UseStartup<TStartup> method.

3. The gRPC service (gRPC procedure)

- The service code is similar as described for VS 2017.
- The proto file format is obviously exactly the same (but other operation and messages are defined). Here, ensure that namespace in proto file is the same as for gRPC service implementation.

4. The gRPC client

The client can be developed similarly. Here different building channel will be presented and calling the service procedure asynchronously. The difference is using one different NuGet package.

- Add to the solution (or create in new solution) a new *Console App (.NET Core)* project.
- Add (similarly as for VS 2017) NuGet packages:
 - Grpc.Net.Client – note: here is the difference,
 - Grpc.Tools,
 - Google.Protobuf.

- Add/create new folder (e.g. Protos) in project and copy/create **greet.proto** proto file. Ensure (change) the namespace to the same as for the client.
- Select the client project node in *Solution Explorer*, select the context menu *Edit Project File*, and ensure (alternatively enter/correct) the in the file is included section:

```
<ItemGroup>
  <Protobuf Include="Protos\greet.proto" GrpcServices="Client" />
</ItemGroup>
```

Especially check if **Client** is set as GrpcServices value.

- Enter the code of client Main method in which you will:
 - create the channel for communication with server (use the same port number as in server),
 - create client,
 - call the remote procedure asynchronously.

```
static async Task Main(string[] args)
{
    Console.WriteLine("Starting gRPC Hello Client");
    using var channel = GrpcChannel.ForAddress("https://localhost:5001");
    var client = new Greeter.GreeterClient(channel);
    String str = Console.ReadLine();
    var reply = await client.SayHelloAsync(new HelloRequest { Name = str });
    Console.WriteLine("From server: " + reply.Message);
    Console.WriteLine("Press any key to exit...");
    Console.ReadKey();
}
```

The class and message names are as the default ones in project.

5. Build the solution (compile/build all projects) and run the application (server first, and then client).
- Check the operation of the application.

3.2 Visual Studio Code

Maybe will be completed later.

3.3 Java based platform

Maybe will be completed later.

4 Exercise – Part II

The detailed and final requirements for Part II are set by the teacher.

- A. Develop or prepare to develop a program according to the teacher's instructions.

Ćwiczenie 2

gRPC

(*tymczasowo w języku angielskim*)

Autor: Mariusz Fraś

1 Objectives of the exercise

The purpose of the exercise is:

1. Familiarization with the development environment for gRPC applications.
2. Understanding the general architecture of the gRPC application.
3. Mastering the basic techniques of creating gRPC applications.

2 Development environment

The developing of gRPC applications can be performed using various platforms. Here is considered the following environment:

- Windows 7 (or higher) Operating System (eventually MacOS).
- Visual Studio 2019 (or 2017),
- Proper NuGet packages.
- Java programming platform (optionally).

3 Exercise – Part I

Creating simple gRPC application project.

3.1 Visual Studio

Visual Studio 2019 contains dedicated gRPC service application project. It is also possible to use several NuGet packages to build gRPC application components.

Note (from nugget.org):

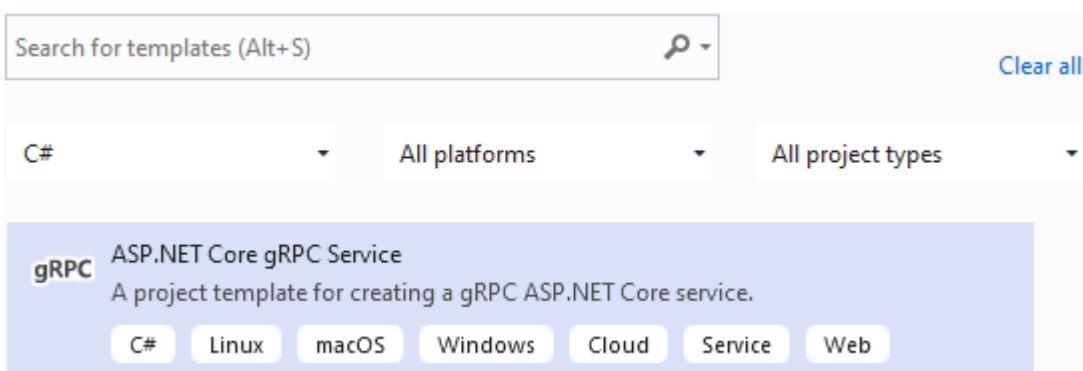
"NuGet is the package manager for .NET. The NuGet client tools provide the ability to produce and consume packages. The NuGet Gallery is the central package repository used by all package authors and consumers."

NuGet is a package manager aimed to enable developers to share reusable code. NuGet's client (nuget.exe) is a free and open-source command-line app that can create and consume packages. NuGet is distributed as a Visual Studio extension, and it can natively consume NuGet packages.

A. Visual Studio 2019.

3.1.1 C# project (new solution) for gRPC service (server).

- To create new gRPC app select *ASP.NET Core gRPC Service*. You can use Search field to find the project faster.

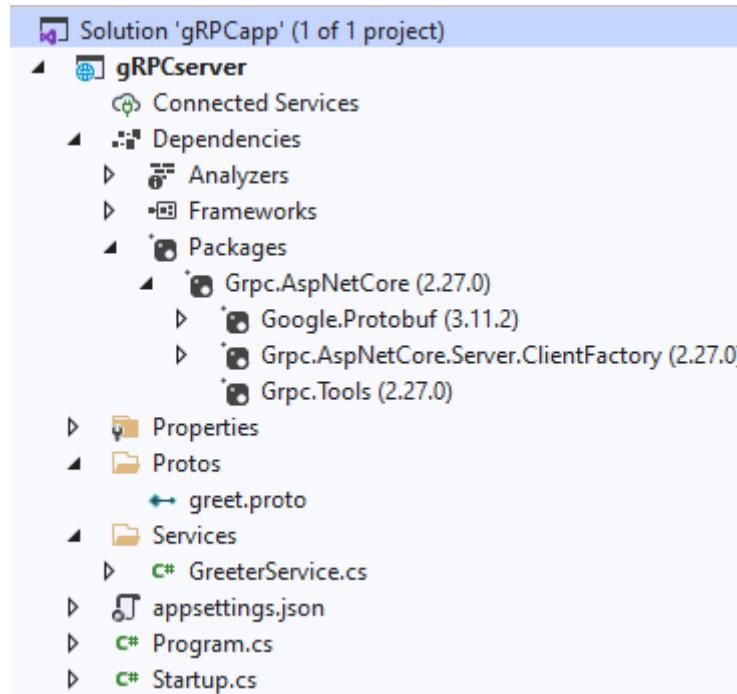


- After selection of names (in next window – here project name gRPCserver in gRPCapp solution) you can specify *Target Framework* (.Net 5.0 or .NET Core 3.1), and eventually enable *Docker* option (don't do it now).

3.1.2 The project structure

The initial default project consist of (among the others):

- already included necessary packages – **Grpc.AspNetCore**,
- the initial proto file (file defining procedure interface) – **greet.proto**,
- the initial gRPC service - **GreeterService.cs**,
- **appSettings.json** – contains configuration data
- the code for service host (by default Kestrel web server is used) – **Program.cs**,
- the service configuration code – **Startup.cs**,



To create your own solution you can as well edit *.cs and proto files as rename or add new ones.

Note the following:

- The default main NuGet package for the default gRPC service project is *GrpcAspNetCore* package which include: *Google.Protobuf*, *Grpc.Tools*, *Grpc/AspNetCore.Server*, and some other packages.
- The project includes files with methods called by the runtime. It includes:
 - **Startup** class that configures services and the app's request pipeline. It has two default methods:
 - **Configure** method to create the app's request processing pipeline,
 - **ConfigureServices** optional method to configure the app's services. Services are registered here.
 - Enabling gRPC service
gRPC is enabled with the **AddGrpc** method in **ConfigureServices** method.
 - Routing
Routing in ASP..NET Core is responsible for matching incoming requests (especially HTTP ones) and dispatching those requests to the app's service endpoints.

The service hosted by ASP.NET Core gRPC, should be added to the routing pipeline with use of the **UseRouting()**, the **UseEndpoints()**, and the **MapGrpcService<TService>()** methods called in the **Configure** method. It adds endpoints to the gRPC service.

- Startup class is typically called by `UseStartup<TStartup>` method when the app's host is built.
- Hosting the service

Host is the component that, among the others, encapsulates (and run) the services – when it is run it calls `StartAcync` on each service implementation..

- **Program** class (with `Main` method) is used to create and run the host for gRPC service.
- The default host for gRPC service is Kestrel – a cross-platform web server for ASP.NET Core (other host options are also available). Kestrel doesn't have some of the advanced features but provides the best performance and memory utilization. It requires HTTP/2 transport and should be secured with TLS.

Attention: MacOS doesn't support ASP.NET Core gRPC with TLS.

- Creation and running the host

In the main method it is built and run the host with builder method:

```
CreateHostBuilder(args).Build().Run();

public static void Main(string[] args)
{
    CreateHostBuilder(args).Build().Run();
}

public static IHostBuilder CreateHostBuilder(string[] args) =>
    Host.CreateDefaultBuilder(args)
        .ConfigureWebHostDefaults(webBuilder =>
    {
        webBuilder.UseStartup<Startup>();
    }
);
```

- Defining the host

The default HTTP host (Kestrel Web server) is built using:

```
CreateDefaultBuilder(args).ConfigureWebHostDefaults(...);
```

In the method it is specified `Startup` as the startup class with use of `UseStartup<TStartup>` method.

3.1.3 gRPC procedure interface defined in proto file

In proto file the interface used for creating stubs for server and client is defined. In the example the remote procedure named `GrpcProc` with two parameters is defined. The procedure returns some calculation result and some message (string value).

- Access the `proto.file` and enter (or modify) the following content that defines:
 - Protocol Buffers version,
 - service and procedure definition (with input and output parameters (messages)),

- input (request) message – parameters in call,
- output (response) message – response for call.

```

syntax = "proto3";
option csharp_namespace = "gRPCserver";
package mygrpc;
// Service definition.
service GrpcService {
    rpc GrpcProc (GrpcRequest) returns (GrpcResponse);
}
// The request message
message GrpcRequest {
    string name = 1;
    int32 age = 2;
}
// The response message
message GrpcResponse {
    string message = 1;
    int32 days = 2;
}

```

- Ensure that namespace in proto file is the same as for gRPC service implementation. When changing names in proto file they must be changed in server implementation accordingly.

3.1.4 The gRPC service (gRPC procedure)

In the solution's Services node skeleton code of remote gRPC procedure is generated.

The class must extend the base class with the name of ***service name defined in proto file*** tagged with ***Base*** word. This class is defined in the class generated from proto file named as ***service name defined in proto file***. (i.e. ***SrvName.SrvNameBase*** class where ***SrvName*** is the name of the service defined in ***proto file***.

In the class the method of remote procedure (with the name as in proto file) takes the request and returns the Task object with response.

Note:

- the Task<...> is used to allow asynchronous processing,
- the ServerCallContext is not used here (is for potential future use).

- Add/modify the code of GrpcProc that returns some message and number of days for provided years:

```

public class MyGrpcService : GrpcService.GrpcServiceBase
{
    ...
    public override Task<GrpcResponse> GrpcProc(GrpcRequest request,
                                                ServerCallContext context)
    {

```

```

        string msg;
        int val;
        val = request.Age * 12 * 365;
        msg = "Hello "+request.Name+" being "+request.Age+" years old.";
        return Task.FromResult(new GrpcResponse { Message=msg, Days=val });
    }
}

```

Pay attention that request and response field names are with capital first letter!

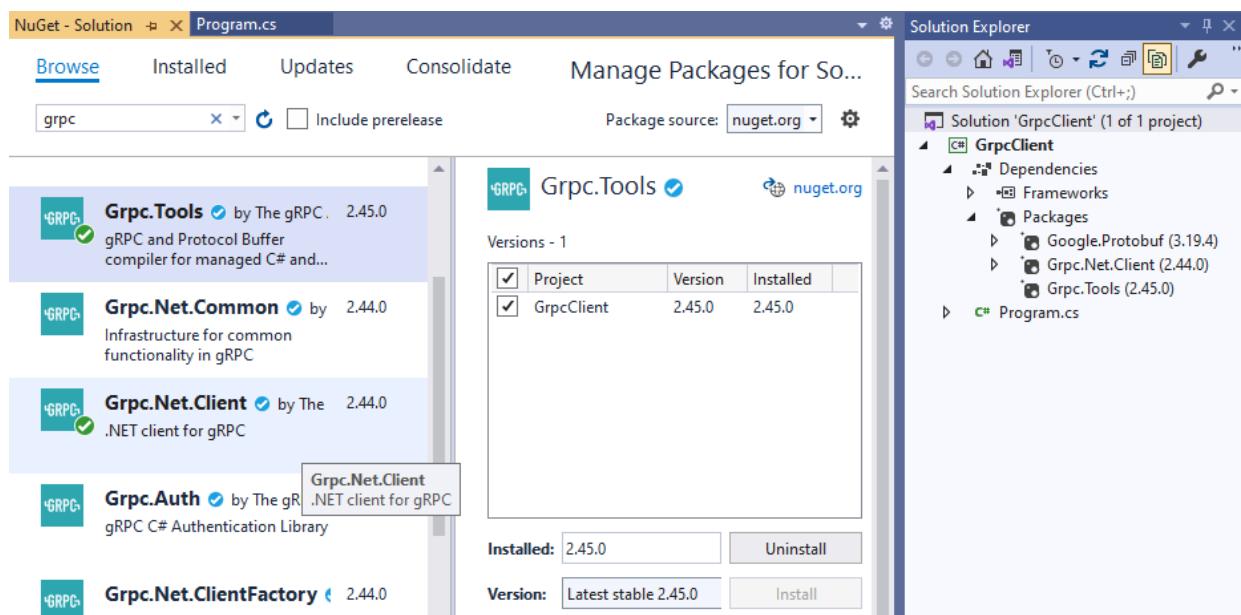
- Build and run the application to verify the correctness.

3.1.5 The gRPC client

Here the client .Net console application will be created.

In order to compile proto files it must be included in the project several packages with use of NuGet manager.

- Create a new project (eventually add one to previous) a *Console Application (.NET Core)* project – here named **gRPCclient**.
- Add NuGet packages: Grpc.Net.Client, Grpc.Tools, Google.Protobuf.
 - Select the project in Solution Explorer window, and from platform menu or context menu select *Manage NuGet Packages* option (or select this option from bar menu).
 - Select *Browse* tab, enter **grpc** in *Search* field.



- Select and add the following packages to the project:
 - Grpc.Net.Client – .Net client for gRPC,
 - Grpc.Tools – gRPC and Protocol Buffers compiler.
- Enter **protobuf** in *Search* field.
Select and add the following package:
 - Google.Protobuf – C# runtime library for Protocol Buffers.

- Add/create new folder (e.g. Protos) in project and copy/create **greet.proto** proto file. Ensure (change) the namespace to the same as for the client.
- Select the client project node in *Solution Explorer*, select the context menu *Edit Project File*, and ensure (alternatively enter/correct) the in the file is included section:

```
<ItemGroup>
  <Protobuf Include="Protos\mygrpc.proto" GrpcServices="Client" />
</ItemGroup>
```

Especially check if **Client** is set as GrpcServices value.

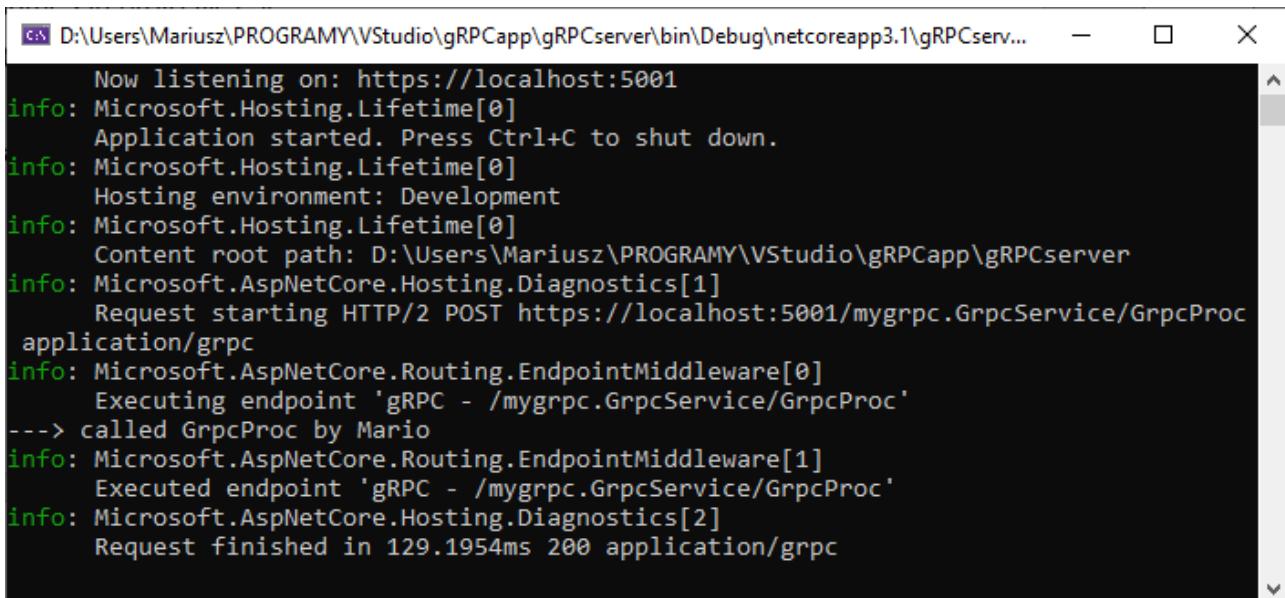
- Enter the code of client Main method in which you will:
 - create a channel for communication with the server (specifying the address and the same port as in server (here we run the server on localhost)),
 - create client object – to make requests,
 - call the remote procedure asynchronously passing input parameters,
 - display results,
 - close the channel when no more used.

```
static async Task Main(string[] args)
{
    Console.WriteLine("Starting gRPC Client");
    using var channel = GrpcChannel.ForAddress("https://localhost:5001");
    var client = new GrpcService.GrpcServiceClient(channel);
    Console.Write("Enter the name: ");
    String str = Console.ReadLine();
    int val = 21;
    var reply = await client.GrpcProcAsync(new GrpcRequest { Name=str , Age=val});
    Console.WriteLine("From server: " + reply.Message);
    Console.WriteLine("From server: "+val+" years = "+reply.Days+" days");
    Console.WriteLine("Press any key to exit...");
    Console.ReadKey();
    channel.ShutdownAsync().Wait();
}
```

The client class is generated from the interface defined in proto file. This class is defined in the class named as **service name in proto file**. The client class name is **service name in proto file** tagged with **Client** word. Here the client object is built with the `GrpcService.GrpcServiceClient` class generated from proto file (names correspond to names in proto file).

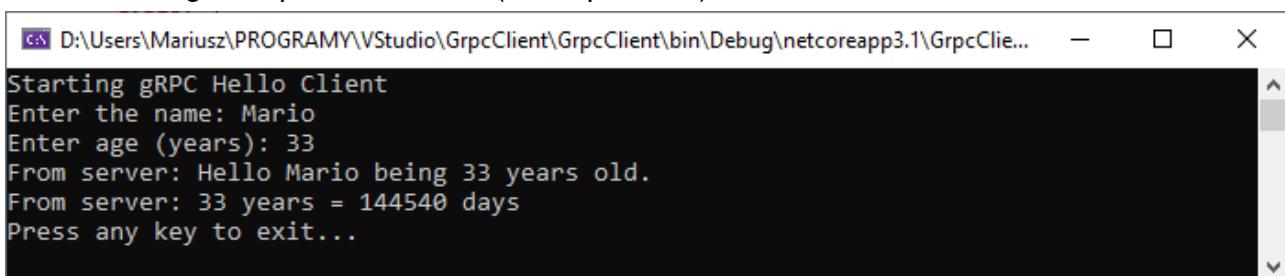
1. Build the solution (compile/build all projects) and run the application (server first, and then client).
 - Check the operation of the application.
 - The result should be similar to below figures:

Server running in separate window (one process):



```
D:\Users\mariusz\PROGRAMY\VStudio\gRPCapp\gRPCserver\bin\Debug\netcoreapp3.1\gRPCserv...
Now listening on: https://localhost:5001
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Development
info: Microsoft.Hosting.Lifetime[0]
      Content root path: D:\Users\mariusz\PROGRAMY\VStudio\gRPCapp\gRPCserver
info: Microsoft.AspNetCore.Hosting.Diagnostics[1]
      Request starting HTTP/2 POST https://localhost:5001/mygrpc.GrpcService/GrpcProc
application/grpc
info: Microsoft.AspNetCore.Routing.EndpointMiddleware[0]
      Executing endpoint 'gRPC - /mygrpc.GrpcService/GrpcProc'
---> called GrpcProc by Mario
info: Microsoft.AspNetCore.Routing.EndpointMiddleware[1]
      Executed endpoint 'gRPC - /mygrpc.GrpcService/GrpcProc'
info: Microsoft.AspNetCore.Hosting.Diagnostics[2]
      Request finished in 129.1954ms 200 application/grpc
```

Client running in separate window (other process):



```
D:\Users\mariusz\PROGRAMY\VStudio\GrpcClient\GrpcClient\bin\Debug\netcoreapp3.1\GrpcClien...
Starting gRPC Hello Client
Enter the name: Mario
Enter age (years): 33
From server: Hello Mario being 33 years old.
From server: 33 years = 144540 days
Press any key to exit...
```

3.2 Visual Studio 2017

Visual Studio 2017 has got slightly worse, however still quite good support for building gRPC applications with help of NuGet packages management.

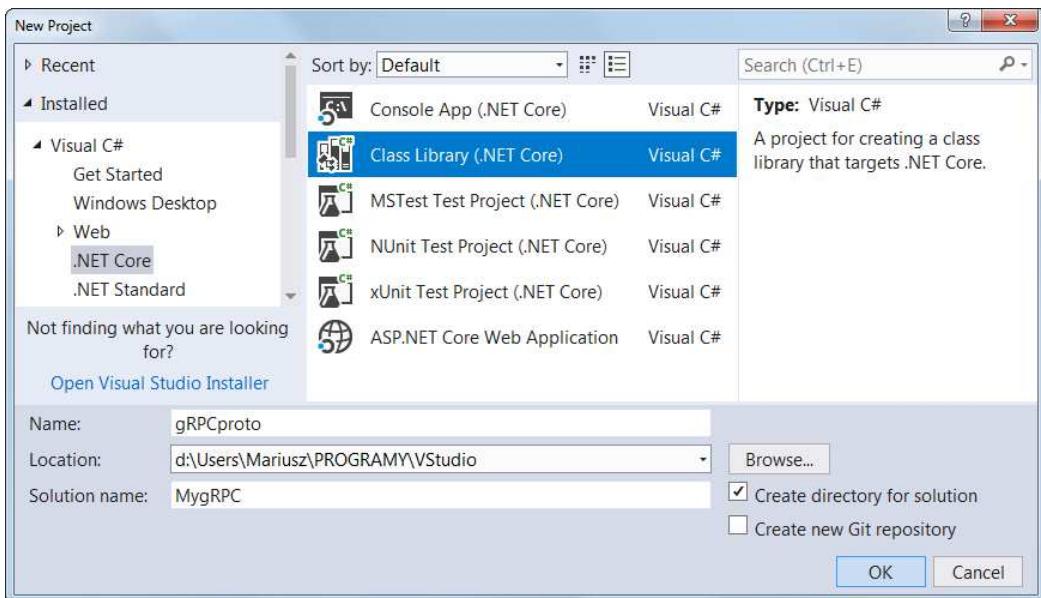
The following steps describe developing simple gRPC application (client and server) with VS 2017. In this instruction one solution with 3 projects will be created to develop simple gRPC server and gRPC client.

1. Create a new C# project (new solution) for proto interface.

This will be for creating code of stubs for server and client – after compilation the proper source code for server and client will be generated.

It can be used Class Library (.NET Core) or . Class Library (.NET Standard) project – both will work. Here we use the first one:

- Create Class Library (.NET Core) project in a new solution (here named gRPCproto in MygRPC solution).



2. Configure the project (here named gRPCproto) for generation of interface stubs.

In order to compile proto files it must be included several packages with use of NuGet manager.

- Select the project in Solution Explorer window, and from platform menu or context menu select *Manage NuGet Packages* option.
- Select *Browse* tab, enter **grpc** in *Search* field.

- Select and add the following packages to the project:
 - Grpc (metapackage for gRPC) – this will also add other necessary packages,
 - Grpc.Tools (gRPC and Protocol Buffers compiler)
- Enter **protobuf** in *Search* field.
Select and add the following package:
 - Google.Protobuf (C# runtime library for Protocol Buffers)
- You may delete initially created Class1.cs file – it will be not used.

3. Create proto file and interface defined in proto file.

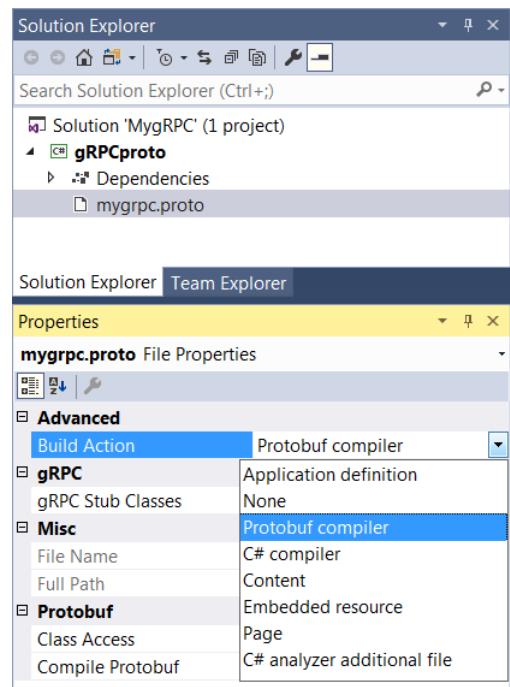
Here the interface for creating stubs for server and client will be defined in proto file.

The remote procedure **addInt** with two parameters that adds two integers will be defined. The procedure returns the result and some comment (string value).

- Create manually or using platform options (*Add New Item* in menu) a text proto file with proto filename extension – here named mygrpc.proto).
- Enter the following content that defines:
 - Protocol Buffers version,
 - service name, remote procedure name, output, and input parameters (messages),
 - input (request) message – parameters in call,
 - output (response) message – replay for call.

```
syntax = "proto3";
package mygrpcproto;
// Service definition.
service MyGrpcSrv {
    rpc addInt (AddIntRequest) returns (AddIntReply) {}
}
// The request message
message AddIntRequest {
    int32 num1 = 1;
    int32 num2 = 2;
}
// The response message
message AddIntReply {
    int32 result = 1;
    string comment = 2;
}
```

- Select the proto file in *Solution Explorer* window and bellow in *Properties* window select in *Build Action* field the *Protobuf compiler* action.

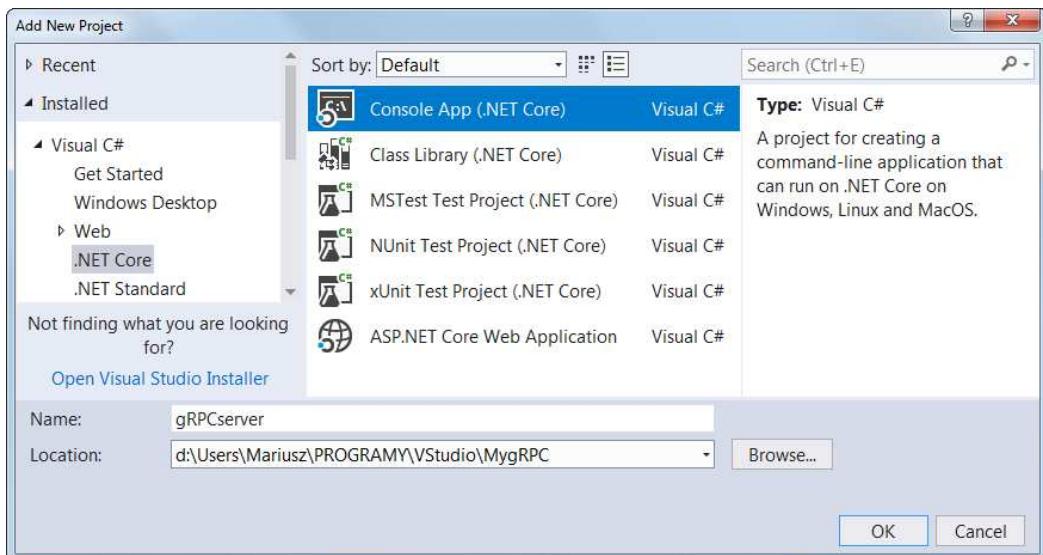


- You may compile (build option) the project to check that all works well.
Pay attention for generated files with classes in **obj** directory of the project – among the others source code for server and client stubs.

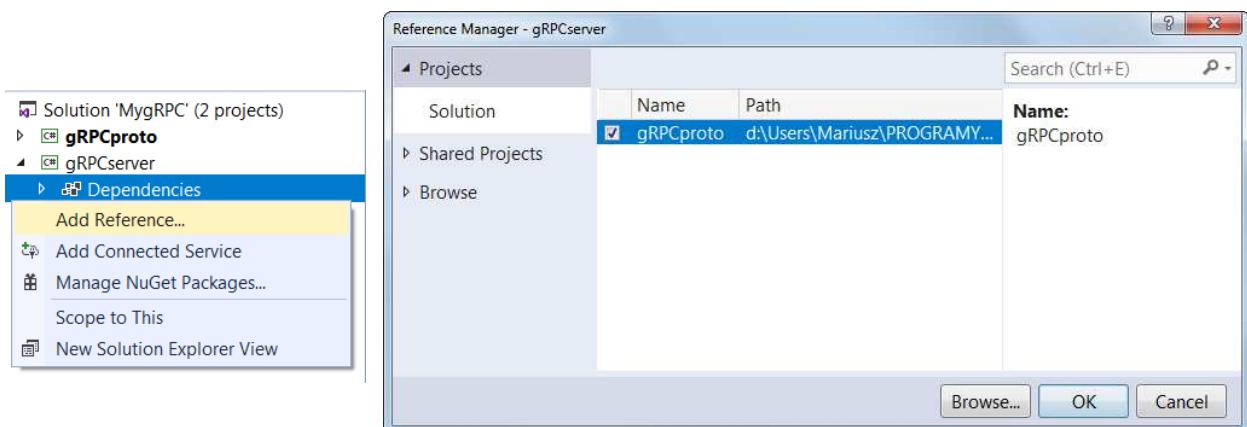
4. Create in the same solution the gRPC server project.

Here the server .Net console application will be created.

- Add to the solution the Console App (.NET Core) project – here named gRPCserver.



- Add the dependency (reference) to gRPCproto project (to use their generated classes). Select *Add Reference* from context menu (see the figure below), or edit *Project Dependencies*.



- Add the class implementing the remote procedure (the interface defined in proto file). The class must extend the base class with the name of **service name in proto file** tagged with **Base** word. This class is defined in the class generated from proto file named as **service name in proto file**.

```
namespace gRPCserver {
    class MyGrpcSrvImpl : MyGrpcSrv.MyGrpcSrvBase {
        public override Task<AddIntReply> addInt(AddIntRequest req,
            ServerCallContext ctx) {
```

```

        string comment;
        int result;

        ...
        return Task.FromResult(new AddIntReply { Result = result,
Comment = comment });
    } }

...
}

```

Set (calculate) result and comment values properly (e.g. comment the sign of the result).

Note:

- here the Task<> is used – this allow asynchronous processing,
- the ServerCallContext is not used here (is for potential future use).

- Add the code in Program class to create the server object and to run the server listening on given port:

```

namespace gRPCserver {

    ...
    class Program {
        const int port = 10000;
        static void Main(string[] args) {
            Console.WriteLine("Starting Hello gRPC server");
            Server myServer = new Server
            {
                Services = { MyGrpcSrv.BindService(new MyGrpcSrvImpl()) },
                Ports = { new ServerPort("localhost", port,
ServerCredentials.Insecure) }
            };
            myServer.Start();
            Console.WriteLine("Hello gRPC server listening on port " + port);
            Console.WriteLine("Press any key to stop the server...");
            Console.ReadKey();
            myServer.ShutdownAsync().Wait();
        }
    }
}

```

Note:

- for the Server object the available services are defined (here our one service implementation) and ports where services calls come to,
- the services are created with BindService method,
- here we specify localhost and port=10000 as the procedure endpoint,
- for simplicity we use not secure connection (hint: on Mac OS there are problems to use secure connection for gRPC).

- Correct compiler warnings/errors properly (usually by importing definitions – with using keyword). Make a note that name for proto file (name of package) start with capital letter (not like in name in proto file).

5. Create in the same solution the gRPC client project.

Here the client .Net console application will be created.

- Add to the solution the new Console App (.NET Core) project – here named gRPCclient.
- Add the dependency (reference) to gRPCproto project (to use their generated classes). Select *Add Reference* from context menu.
- Implement in Main method:
 - Creating a channel which will be used to communicate with server
 - specifying the address and port (here we run the server on localhost),
 - specifying no security mechanisms (for simplicity).
 - Creating client object – to make requests.

The client class is generated from the interface defined in proto file. This class is defined in the class named as ***service name in proto file***. The client class name is ***service name in proto file*** tagged with ***Client*** word.

- Calling remote procedure – here named **addInt** – with use of **AddIntRequest** object (from the class generated from proto file).
- Closing the channel when no more used.

```
static void Main(string[] args) {
    Console.WriteLine("Starting gRPC Client");
    Channel channel = new Channel("127.0.0.1:10000",
        ChannelCredentials.Insecure);
    var client = new MyGrpcSrv.MyGrpcSrvClient(channel);
    String str;
    int num1, num2;
    Console.Write("Enter number 1: ");
    str = Console.ReadLine();
    if (int.TryParse(str, out num1))
    {
        Console.Write("Enter number 2: ");
        str = Console.ReadLine();
        if (int.TryParse(str, out num2))
        {
            var reply = client.addInt(new AddIntRequest { Num1 = num1,
                Num2 = num2 });
            Console.WriteLine("From server: " + reply.Comment + reply.Result);
        }
        else
            Console.WriteLine("Wrong value!");
    }
}
```

```
    }
    else
        Console.WriteLine("Wrong value!");
        Console.WriteLine("Stopping gRPC Client");
        channel.ShutdownAsync().Wait();
}
```

6. Build the solution (compile/build all projects).

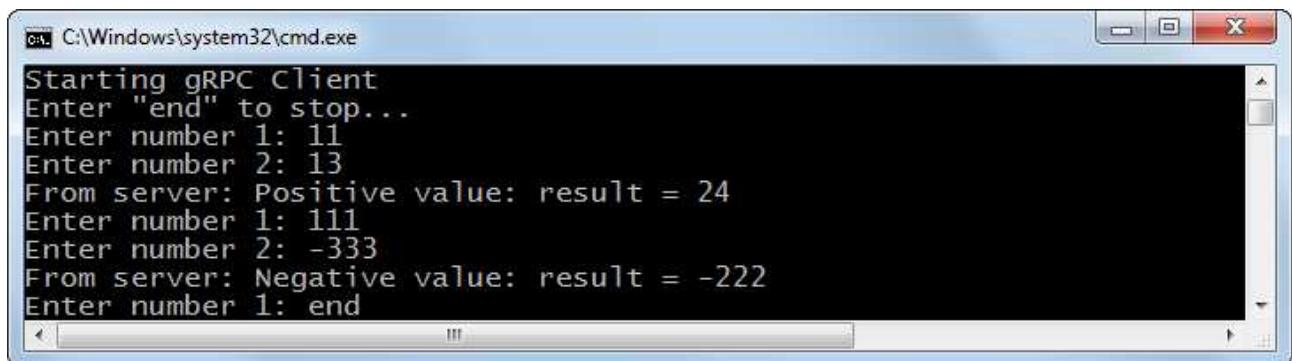
- Check the operation of the application.
- The result should be similar to below figures:

Server running in separate window (one process):



```
C:\Windows\system32\cmd.exe - dotnet gRPCserver.dll
Starting Hello gRPC server
Hello gRPC server listening on port 10000
Press any key to stop the server...
Called addInt (1)
Called addInt (2)
```

Client running in separate window (other process):



```
C:\Windows\system32\cmd.exe
Starting gRPC Client
Enter "end" to stop...
Enter number 1: 11
Enter number 2: 13
From server: Positive value: result = 24
Enter number 1: 111
Enter number 2: -333
From server: Negative value: result = -222
Enter number 1: end
```

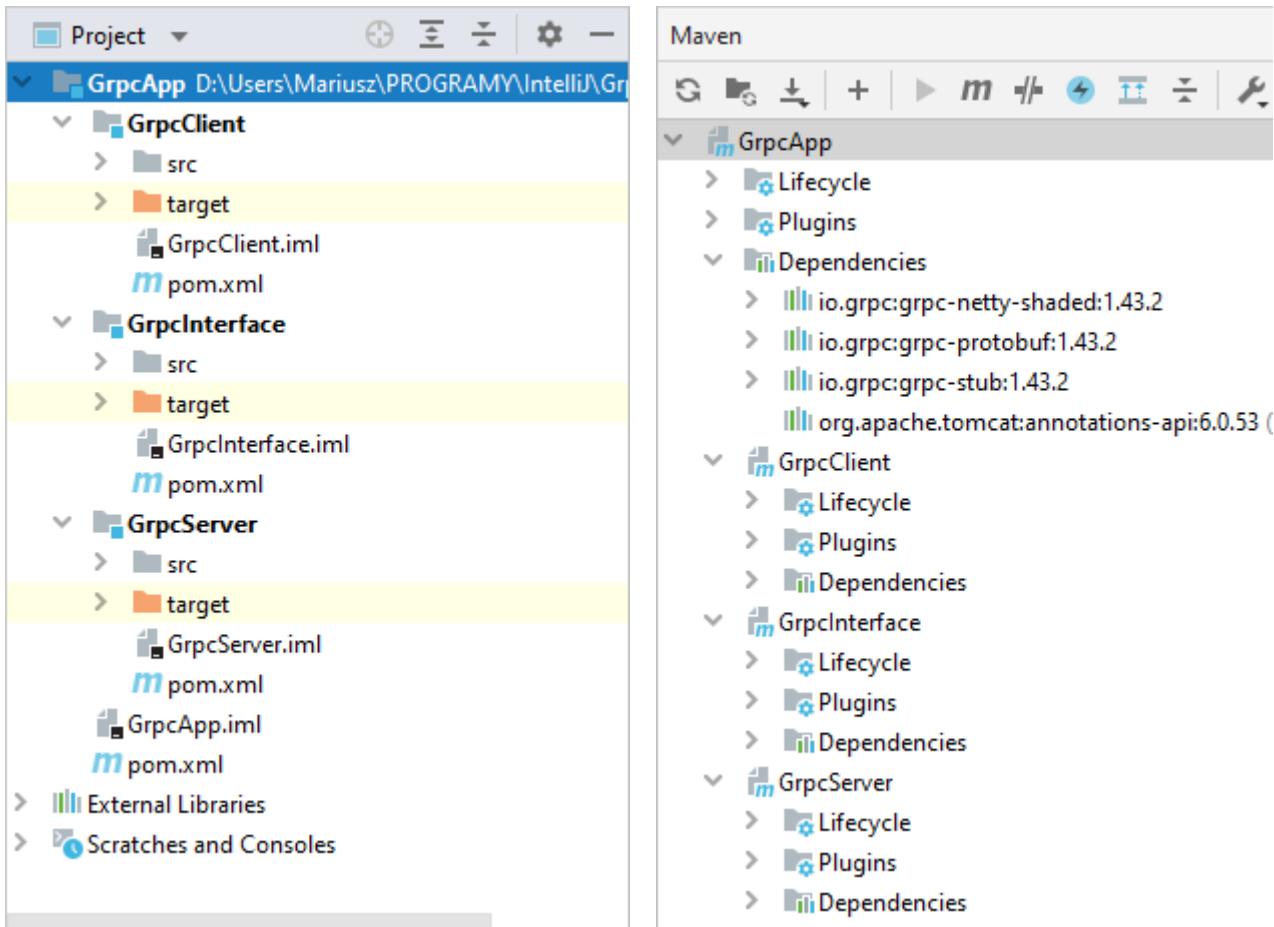
Note:

After proper creating and building the solution (all projects), the VS 2017 platform sometimes shows (after re-opening the solution (projects)) that errors exist for uses directive (as if proto namespace was not visible and classes generated from it). Anyway all works well (after some time errors may disappear) and the solution can be rebuilt without errors. This behavior of Visual Studio 2017 is strange and hard to explain.

3.3 Java based platform – maven project

Detailed description will be completed later.

3.3.1 General project structure



3.3.2 POMs

Main POM

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>pl.edu.pwr.wit.rsi</groupId>
    <artifactId>GrpcApp</artifactId>
    <packaging>pom</packaging>
    <version>1.0-SNAPSHOT</version>
    <modules>
        <module>GrpcServer</module>
        <module>GrpcClient</module>
        <module>GrpcInterface</module>
    </modules>
```

```

<properties>
    <maven.compiler.source>17</maven.compiler.source>
    <maven.compiler.target>17</maven.compiler.target>
    <grpc.version>1.43.2</grpc.version>
    <protobuf.maven.plugin.version>0.6.1</protobuf.maven.plugin.version>
    <protobuf.version>3.19.4</protobuf.version>
    <os.maven.plugin.version>1.6.2</os.maven.plugin.version>
</properties>

<dependencies>
    <dependency>
        <groupId>io.grpc</groupId>
        <artifactId>grpc-netty-shaded</artifactId>
        <version>${grpc.version}</version>
    </dependency>
    <dependency>
        <groupId>io.grpc</groupId>
        <artifactId>grpc-protobuf</artifactId>
        <version>${grpc.version}</version>
    </dependency>
    <dependency>
        <groupId>io.grpc</groupId>
        <artifactId>grpc-stub</artifactId>
        <version>${grpc.version}</version>
    </dependency>
    <dependency> <!-- necessary for Java 9+ -->
        <groupId>org.apache.tomcat</groupId>
        <artifactId>annotations-api</artifactId>
        <version>6.0.53</version>
        <scope>provided</scope>
    </dependency>
</dependencies>

</project>

```

Interface POM:

```

<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <parent>
        <artifactId>GrpcApp</artifactId>
        <groupId>pl.edu.pwr.wit.rsi</groupId>
        <version>1.0-SNAPSHOT</version>
    </parent>
    <modelVersion>4.0.0</modelVersion>

    <artifactId>GrpcInterface</artifactId>

    <build>
        <extensions>
            <extension>
                <groupId>kr.motd.maven</groupId>
                <artifactId>os-maven-plugin</artifactId>
                <version>${os.maven.plugin.version}</version>
            </extension>
        </extensions>
    </build>

```

```

<plugins>
    <plugin>
        <groupId>org.xolstice.maven.plugins</groupId>
        <artifactId>protobuf-maven-plugin</artifactId>
        <version>${protobuf.maven.plugin.version}</version>
        <configuration>
            <protocArtifact>com.google.protobuf:protoc:${protobuf.version}:
                exe:${os.detected.classifier}</protocArtifact>
            <pluginId>grpc-java</pluginId>
            <pluginArtifact>io.grpc:protoc-gen-grpc-java:${grpc.version}:
                exe:${os.detected.classifier}</pluginArtifact>
        </configuration>
        <executions>
            <execution>
                <goals>
                    <goal>compile</goal>
                    <goal>compile-custom</goal>
                    <goal>test-compile</goal>
                    <goal>test-compile-custom</goal>
                </goals>
            </execution>
        </executions>
    </plugin>
</plugins>
</build>

</project>

```

Server POM:

```

<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <parent>
        <artifactId>GrpcApp</artifactId>
        <groupId>pl.edu.pwr.wit.rsi</groupId>
        <version>1.0-SNAPSHOT</version>
    </parent>
    <modelVersion>4.0.0</modelVersion>

    <artifactId>GrpcServer</artifactId>

    <dependencies>
        <dependency>
            <groupId>pl.edu.pwr.wit.rsi</groupId>
            <artifactId>GrpcInterface</artifactId>
            <version>1.0-SNAPSHOT</version>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-assembly-plugin</artifactId>
                <version>3.3.0</version>
            </plugin>
        </plugins>
    </build>

```

```

<configuration>
    <descriptorRefs>
        <descriptorRef>jar-with-dependencies</descriptorRef>
    </descriptorRefs>
    <archive>
        <manifest>
            <mainClass>GrpcServer</mainClass>
        </manifest>
    </archive>
</configuration>
<executions>
    <execution>
        <id>make-assembly</id>
        <phase>package</phase>
        <goals>
            <goal>single</goal>
        </goals>
    </execution>
</executions>
</plugin>
</plugins>
</build>

</project>

```

Client POM:

```

<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <parent>
        <artifactId>GrpcApp</artifactId>
        <groupId>pl.edu.pwr.wit.rsi</groupId>
        <version>1.0-SNAPSHOT</version>
    </parent>
    <modelVersion>4.0.0</modelVersion>

    <artifactId>GrpcClient</artifactId>

    <dependencies>
        <dependency>
            <groupId>pl.edu.pwr.wit.rsi</groupId>
            <artifactId>GrpcInterface</artifactId>
            <version>1.0-SNAPSHOT</version>
            <scope>compile</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-assembly-plugin</artifactId>
                <version>3.3.0</version>
                <configuration>
                    <descriptorRefs>
                        <descriptorRef>jar-with-dependencies</descriptorRef>
                    </descriptorRefs>

```

```

        <archive>
            <manifest>
                <mainClass>GrpcClient</mainClass>
            </manifest>
        </archive>
    </configuration>
    <executions>
        <execution>
            <id>make-assembly</id>
            <phase>package</phase>
            <goals>
                <goal>single</goal>
            </goals>
        </execution>
    </executions>
    </plugin>
</plugins>
</build>

</project>
```

3.3.3 Proto file

In main\proto directory/node.

```

syntax = "proto3";
option java_multiple_files = true;
option java_outer_classname = "GrpcAppProto";
option objc_class_prefix = "GAP";

// The greeting service definition.
service GrpcService {
    // Greeting procedure
    rpc grpcProcedure (GrpcRequest) returns (GrpcResponse) {}
}

// The request message containing the user's name and age.
message GrpcRequest {
    string name = 1;
    int32 age = 2;
}

// The response message containing the greetings
message GrpcResponse {
    string message = 1;
}
```

3.3.4 Server code

```

public class GrpcServer {
    public static void main(String[] args) {
        int port = 50001;
        System.out.println("Starting server...");
        Server server = ServerBuilder
            .forPort(port)
            .addService(new GrpcServiceImpl()).build();
        try {
            server.start();
            System.out.println("...Server started");
            server.awaitTermination();
        } catch (IOException e) {
            e.printStackTrace();
```

```

        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

    static class GrpcServiceImpl extends GrpcServiceGrpc.GrpcServiceImplBase {
        public void grpcProcedure(GrpcRequest req,
                                  StreamObserver<GrpcResponse> responseObserver) {
            String msg;
            System.out.println("...called GrpcProcedure");
            if (req.getAge() > 18)
                msg = "Mr/Ms " + req.getName();
            else
                msg = "Boy/Girl";
            GrpcResponse response = GrpcResponse.newBuilder()
                .setMessage("Hello " + msg).build();
            responseObserver.onNext(response);
            responseObserver.onCompleted();
        }
    }
}

```

3.3.5 Client code

```

public class GrpcClient {
    public static void main(String[] args) {
        String address = "localhost";
        int port = 50001;
        System.out.println("Running grpc client...");
        ManagedChannel channel = ManagedChannelBuilder.forAddress(address, port)
            .usePlaintext()
            .build();

        GrpcServiceGrpc.GrpcServiceBlockingStub stub =
            GrpcServiceGrpc.newBlockingStub(channel);
        GrpcRequest request = GrpcRequest.newBuilder()
            .setName("Mariusz")
            .setAge(21)
            .build();
        GrpcResponse response = stub.grpcProcedure(request);
        System.out.println(response);
        channel.shutdown();
    }
}

```

4 Exercise – Part II

The detailed and final requirements for Part II are set by the teacher.

- A. Develop or prepare to develop a program according to the teacher's instructions.

Ćwiczenie 3

Java RMI

Autor: Mariusz Fraś

1 Cele ćwiczenia

Celem ćwiczenia jest:

1. Poznanie ogólnej architektury aplikacji Java RMI.
2. Opanowanie podstawowych technik tworzenia aplikacji Java RMI.

- **Pierwsza część zadania** jest do wykonania według podanej instrukcji i ewentualnych poleceń prowadzącego zajęcia laboratoryjne.
- **Druga część zadania jest do przygotowania i oddania lub do wykonania na kolejnych zajęciach.**

2 Środowisko deweloperskie

Standardowe oprogramowanie do tworzenia i testowania aplikacji Java RMI stosowane w laboratorium to:

- System operacyjny Windows 7 lub wyższy.
- Java 2 Software Development Kit (JDK).
- Środowisko programistyczne Java (np. Eclipse).

3 Zadanie – część I

Podstawowe elementy tworzenia aplikacji Java RMI

W zadaniu zostanie zrealizowane:

- Zdefiniowanie interfejsu zdalnego obiektu oraz kodu klasy zdalnego obiektu implementującego ten interfejs.
- Utworzenie kodu serwera – włączenie zezwoleń poprzez uruchomienie menadżera zabezpieczeń, utworzenie instancji obiektu zdalnego i rejestracja obiektu zdalnego w rejestrze.
- Utworzenie kodu klienta – wyszukanie obiektu zdalnego i wywołanie metody zdalnego obiektu.
- Zdefiniowanie uprawnień aplikacji, uruchomienie rejestru i uruchomienie aplikacji.
- Dodanie do aplikacji (zdefiniowanie) obiektu zdalnego przyjmującego jako parametr typ obiektowy i zwracający typ obiektowy.
- Dodanie do klienta kodu wykorzystującego obiekt zdalny z metodą o parametrach obiektowych i zwracający obiekt.

1. Utworzenie projektu i kodu zdalnego obiektu	<ul style="list-style-type: none"> • Utwórz w platformie Eclipse projekt Java o własnej nazwie (np. <code>MojSerwerRMI</code>) • Zdefiniuj w oddzielnym pliku nowy interfejs <code>CalcObject</code> dla klasy obiektu zdalnego (który będzie dostępny zdalnie dla klienta). Interfejs rozszerza klasę <code>java.rmi.Remote</code> i zawiera metodę, która musi rzucić wyjątek <code>RemoteException</code>. <pre><code>public interface CalcObject extends Remote { public double calculate(double a, double b) throws RemoteException; }</code></pre> • Zdefiniuj w oddzielnym pliku klasę <code>CalcObjImpl</code> implementującą interfejs <code>CalcObject</code>, z metodą realizującą obliczenia. Obiekty tego typu będą dostępne zdalnie. Klasa rozszerza biblioteczną klasę <code>UnicastRemoteObject</code>: <pre><code>import java.rmi.RemoteException; import java.rmi.server.UnicastRemoteObject; public class CalcObjImpl extends UnicastRemoteObject implements CalcObject { private static final long serialVersionUID = 101L; public CalcObjImpl() throws RemoteException { super(); } public double calculate(double a, double b) throws RemoteException { double wynik = a + b; return wynik; } }</code></pre>
--	--

<p>2. Utworzenie kodu serwera</p>	<ul style="list-style-type: none"> • Zdefiniuj klasę główną serwera MyServer, w której będzie: <ul style="list-style-type: none"> – pobieranie danych o adresie usługi (adresie zdalnego obiektu) z parametru wywołania programu, – włączenie zezwoleń poprzez utworzenie/ustawienie systemowego menadżera bezpieczeństwa, – w bloku try ... catch ... utworzenie instancji obiektu zdalnego CalcObjImpl, – w bloku try ... catch ... rejestracja obiektu-serwera w rejestrze (rmiregistry) za pomocą klasy i metody Naming.rebind, – zakończenie procesu (metody main) – obiekt serwera jest uruchamiany w oddzielnym wątku. • Podczas tworzenia klasy serwera zaznacz opcję tworzenia metody statycznej public static main(...). <pre><code>public class MyServer { public static void main(String[] args) { if (args.length == 0) { System.out.println("You have to enter RMI object address in the form: //host_address/service_name"); return; } if (System.getSecurityManager() == null) System.setSecurityManager(new SecurityManager()); } try { CalcObjImpl implObiekta = new CalcObjImpl(); java.rmi.Naming.rebind(args[0], implObiekta); System.out.println("Server is registered now :-)"); System.out.println("Press Ctrl+C to stop..."); } catch (Exception e) { System.out.println("SERVER CAN'T BE REGISTERED!"); e.printStackTrace(); return; } } }</code></pre> <ul style="list-style-type: none"> • Sprawdź poprawność kodu. Skoryguj ewentualne błędy.
<p>3. Utworzenie kodu klienta</p>	<ul style="list-style-type: none"> • Utwórz drugi projekt Java o własnej nazwie (np. MojKlientRMI) • Zdefiniuj w oddzielnym pliku interfejs CalcObject dla klasy obiektu zdalnego, identycznie jak dla serwera. • Zdefiniuj klasę główną klienta MyClient, w której będzie: <ul style="list-style-type: none"> – deklaracja odpowiednich zmiennych, w tym zmiennej/referencji obiektu zdalnego (interfejsu), – pobieranie danych o adresie usługi (adresu zdalnego obiektu) z parametru wywołania programu,

	<ul style="list-style-type: none"> – utworzenie/ustawienie systemowego menadżera bezpieczeństwa (uwaga: nie jest to wymagane we wszystkich przypadkach – tu w kliencie jest zakomentowane – w razie potrzeby zdalnego pobierania definicji klas trzeba zdjąć ten komentarz), – pobieranie referencji do zdalnego obiektu za pomocą klasy i metody Naming.lookup, – w bloku try ... catch ... wywołanie usługi (metody zdalnego obiektu) i wyświetlenie odpowiedzi. <pre><code>public class MyClient { public static void main(String[] args) { double wynik; CalcObject zObiekty; if (args.length == 0) { System.out.println("You have to enter RMI object address in the form: // host_address/service_name "); return; } String adres = args[0]; // //use this if needed // if (System.getSecurityManager() == null) // System.setSecurityManager(new SecurityManager()); try { zObiekty = (CalcObject) java.rmi.Naming.lookup(adres); } catch (Exception e) { System.out.println("Nie mozna pobrac referencji do "+adres); e.printStackTrace(); return; } System.out.println("Referencja do "+adres+" jest pobrana."); try { wynik = zObiekty.calculate(1.1, 2.2); } catch (Exception e) { System.out.println("Blad zdalnego wywolania."); e.printStackTrace(); return; } System.out.println("Wynik = "+wynik); return; } }</code></pre> <ul style="list-style-type: none"> • Sprawdź poprawność kodu. Skoryguj ewentualne błędy.
4. Uruchomienie komponentów systemu i aplikacji	<ul style="list-style-type: none"> • Utwórz plik tekstowy srv.policy z definicją uprawnień dla aplikacji – tu ustawiamy wszystkie uprawnienia dla aplikacji: <pre><code>grant { permission java.security.AllPermission; };</code></pre>

	<ul style="list-style-type: none"> Umieść plik w folderze bin serwera (uwaga: w przypadku innej lokalizacji bajtkodu serwera trzeba odpowiednio zmienić wywołania w dalszych komendach). Utwórz i umieść analogiczny plik w folderze klienta. Uruchom konsolę (w systemie Windows wiersz poleceń) i przejdź do folderu bin z klasami serwera. Uruchom z konsoli rejestr rmiregistry polecaniem: <code>start rmiregistry</code> Uwaga: w przypadku problemów ze znalezieniem programu należy odpowiednio użyć lub ustawić ścieżkę dostępu. Uruchom proces serwera specyfikując ustawienie uprawnień: <code>java -Djava.security.policy=srv.policy MyServer //x/y</code> W miejsce x i y wpisz odpowiednio adres hosta i nazwę usługi. Uruchom kolejną konsolę i przejdź do folderu bin z klasami klienta. Uruchom proces klienta specyfikując ustawienie uprawnień: <code>java MyClient //x/y</code> W miejsce x i y wpisz odpowiednio adres serwera i nazwę usługi. Skontroluj w konsolach wyniki działania. Zatrzymaj wszystkie procesy.
5. Dodanie w serwerze klas i obiektów dla drugiego obiektu zdalnego z metodą z obiektem jako parametrem, zwracającą także obiekt	<ul style="list-style-type: none"> Zdefiniuj w projekcie serwera, w oddzielnym pliku, klasę InputType która będzie typem parametru zdalnie wywoływanej metody klasy obiektu zdalnego (klasy CalcObject2). Klasa parametru implementuje <code>java.io.Serializable</code> i zawiera zadanie/operację do obliczeń: <pre>public class InputType implements Serializable { private static final long serialVersionUID = 101L; String operation; public double x1; public double x2; public double getx1() { return x1; } public double getx2() { return x2; } }</pre> Obiekt tego typu to <i>obiekt-zadanie</i> – obiekt zawierający zadanie do wykonania przez zdalny obiekt. Zdefiniuj w projekcie serwera, w oddzielnym pliku, klasę ResultType która będzie typem wyniku obliczeń zwracanego przez metodę zdalnego obiektu. Klasa wyniku także musi implementować <code>java.io.Serializable</code>.

```
public class ResultType implements Serializable {
    private static final long serialVersionUID = 102L;
    String result_description;
    public double result;
}

• Zdefiniuj w oddzielnym pliku nowy interfejs CalcObject2 dla klasy drugiego obiektu zdalnego:

public interface CalcObject2 extends Remote {
    public ResultType calculate(InputType inputParam)
        throws RemoteException;
}

• Zdefiniuj w oddzielnym pliku klasę CalcObjImpl2 implementującą interfejs CalcObject2 w której:
– jest metoda wywoływana zdalnie calculate,
– w niej pobierane są dane z obiektu-zadania (z parametru metody),
– stosownie do danych wykonywane jest przetwarzanie – tu: wykonywanie operacji add lub sub,
– tworzony jest obiekt do zwrotu wyniku,
– zwracany jest ten obiekt.

public class CalcObjImpl2 extends UnicastRemoteObject
    implements CalcObject2
{
    public CalcObjImpl2() throws RemoteException {
        super();
    }

    public ResultType calculate(InputType inParam)
        throws RemoteException {
        double zm1, zm2;
        ResultType wynik = new ResultType();

        zm1 = inParam.getx1();
        zm2 = inParam.getx2();
        wynik.result_description = "Operacja "+inParam.operation;

        switch (inParam.operation) {
        case "add" :
            wynik.result = zm1 + zm2;
            break;
        case "sub" :
            wynik.result = zm1 - zm2;
            break;
        default :
            wynik.result = 0;
            wynik.result_description = "Podano zla operacje";
            return wynik;
        }

        return wynik;
    }
}
```

	<ul style="list-style-type: none"> Zmodyfikuj główną klasę serwera MyServer tak aby: <ul style="list-style-type: none"> Wprowadzane i sprawdzane były dwa parametry wywołania programu – obsługę drugiego parametru zrealizować tak jak pierwszego ale dla adresu drugiego obiektu zdalnego, W bloku try ... catch ... utworzyć instancję drugiego obiektu i utworzyć wiązanie metodą Naming.rebind. Sprawdź poprawność kodu. Skoryguj ewentualne błędy.
6. Dodanie w kliencie klas, zmiennych i obiektów dla wywołania metody drugiego zdalnego obiektu z parametrem w postaci obiektu	<ul style="list-style-type: none"> Zdefiniuj w kliencie te same klasy co w serwerze: InputType, ResultType i CalcObject2. Zmodyfikuj ogólna klasę klienta MyClient tak aby: <ul style="list-style-type: none"> wprowadzane i sprawdzane były dwa parametry wywołania programu – obsługę drugiego parametru zrealizować tak jak pierwszego ale dla adresu drugiego obiektu zdalnego, dodać odpowiednie zmienne dla obsługi drugiego obiektu zdalnego utworzyć obiekt-zadania (typu InputType) – parametr dla metody zdalnej, dla obiektu-zadania zainicjować składowe x1 i x2 jakimiś wartościami, a zmiennej operation przypisać ciąg "add" lub "sub", <pre> ... CalcObject2 zObiekt2; ResultType wynik2; InputType inObj; ... inObj = new InputType(); inObj.x1= xxx; inObj.x2= yyy; inObj.operation="add"; //lub "sub" ... </pre> <p>W miejsce xxx i yyy podać jakieś wartości zmiennoprzecinkowe.</p> <ul style="list-style-type: none"> w bloku try ... catch ... utworzyć wiązanie dla zmiennej typu CalcObject2 do drugiego obiektu zdalnego, za pomocą klasy i metody Naming.lookup – podobnie jak wcześniej do pierwszego obiektu, w bloku try ... catch ... wywołać metodę obliczeń (calculate) drugiego obiektu zdalnego, podobnie jak wcześniej dla pierwszego obiektu, ale podając odpowiednie parametry i odbierając wynik w postaci obiektu rezultatu, wyświetlić wynik dodatkowo z opisem (polem description) jako komentarzem. Sprawdź poprawność kodu. Skoryguj ewentualne błędy.

7. Dodanie kodu tworzenia rejestru	<p>Zamiast ręcznie konfigurować i uruchamiać rejestr można ten etap zrealizować programowo o ile rejestr i serwer są uruchamiane na tym samym komputerze.</p> <ul style="list-style-type: none">W kodzie serwera, przed tworzeniem implementacji obiektów zdalnych i ich rejestracją, dodaj kod utworzenia rejestru dla serwera (pozbycie się potrzeby uruchamiania rejestrów oddzielnie w konsoli): <pre>try { Registry reg = LocateRegistry.createRegistry(1099); } catch (RemoteException e1) { e1.printStackTrace(); }</pre> <p>Podany tu port nr 1099 jest taki jak domyślny port dla rejestrów RMI.</p>
8. Uruchomienie aplikacji	<ul style="list-style-type: none">Uruchom proces serwera z linii komend tak jak poprzednio, ale z dodatkowym parametrem (dla drugiej usługi) i bez uruchamiania rejestrów z linii komend (czyli bez komendy <code>start rmiregistry</code>). Uwaga: drugi parametr (adres usługi) musi mieć taki sam adres hosta ale inną nazwę!Uruchom klienta z linii komend tak jak poprzednio, ale z dodatkowym parametrem (dla drugiej usługi).Skontroluj działanie aplikacji.

4 Zadanie – część II

Szczegółowe wymagania i wybór zadania (A, B lub inne) określa prowadzący grupy.

A. Przygotować się do napisania na zajęciach aplikacji zawierającej elementy o podobnych funkcjonalnościach według wskazówek prowadzącego.

B. Przykładowe zadanie do wykonania:

Napisać aplikację Java RMI będącą częściową implementacją modelu farmer-worker – tu spełniającą następujące wymagania:

1. Serwer-worker jest aplikacją która przetwarza/oblicza przekazane mu zadanie.

Zadanie jest przekazywane jako obiekt – parametr wywołania metody workera.

Wskazówki:

- a. Klasa serwera-workera implementuje interfejs rozszerzający `java.rmi.Remote`,
- b. Interfejs (a więc i worker) ma metodę `oblicz/compute/calculate` lub podobnie.
- c. Parametr tej metody to obiekt-zadanie (zdefiniowany typ klasy-zadania).
- d. Klasa zadania także ma metodę `oblicz/compute/calculate` lub podobnie, realizującą jakieś konkretne przetwarzanie/obliczenia.
- e. Metoda workera `oblicz/compute/...` wywołuje metodę obliczeń przekazanego mu obiektu-zadania i zwraca wyniki wykonania.
- f. Worker rejestruje się w rejestrze RMI.

2. Zadanie do przetwarzania:

a. Obiekt-zadanie musi być serializowany. W tym celu:

b. Zdefiniować interfejs rozszerzający `java.io.Serializable`, z metodą `oblicz/compute/calculate` lub podobnie.

Ten interfejs jest parametrem wywoływanej metody workera.

c. Zdefiniować klasę zadania, która implementuje ten interfejs i realizuje obliczenia.

d. Zadanie zwraca wynik przetwarzania w postaci obiektu (ogólnie wynik może być bardziej złożony niż pojedyncza wartość).

e. Metoda workera zwraca ten sam typ co zadanie (wynik obliczeń/przetwarzania).

3. Klient bezpośrednio współpracuje z serwerami-workerami (rola farmera jest pominięta, a właściwie częściowo zawarta w kliencie).

/ ***Uwaga:*** jest tu uproszczenie pełnego modelu farmer-worker. Farmer powinien być pośrednikiem dla klienta, który dzieli zadanie na podzadania i rozdziela je do workerów, i powinien być również dostępny zdalnie. Tu pomijamy te elementy. /

a. Klient tworzy obiekty-zadania.

b. Klient lokalizuje serwery-workery.

c. Wywołuje ich metody (obliczenia) przekazując w parametrze podzadanie do wykonania i zbiera wyniki wykonania, oraz wyświetla całość.

d. Używa co najmniej dwóch workerów.

4. Wymyślić własne zadanie obliczeniowe, które można podzielić na podzadania i przekazać do obliczeń dla workerów (obliczanie czegoś wg jakiejś procedury, wyszukiwanie czegoś, itp.).

5. Wyniki wykonania zadania są przekazywane jako obiekty.

6. Dla uproszczenia workery można wywoływać sekwencyjnie

(w rzeczywistości model zakłada wykonywanie zadań przez workery równolegle) .

Ćwiczenie 4.1

WCF – podstawy i konfigurowanie usługi

Autor: Mariusz Fraś

1 Cele ćwiczenia

Celem ćwiczenia jest:

1. Poznanie podstawowej architektury aplikacji WCF
 2. Zapoznanie się z podstawami tworzenia usługi z opisem WSDL i dostępnej protokołem SOAP (tu: usługi WCF), i klienta takiej usługi (tu: klienta WCF).
 3. Poznanie opcji konfigurowania usług – punktów końcowych, transportu, sposobu działania usługi.
-
- **Pierwsza część zadania** jest do wykonania według podanej instrukcji i ewentualnych polecen prowadzącego zajęcia laboratoryjne.
 - **Druga część zadania jest do przygotowania i oddania lub do wykonania wg. polecen prowadzącego na kolejnych zajęciach.**

2 Zadanie – część I

Konstrukcja podstawowej aplikacji WCF

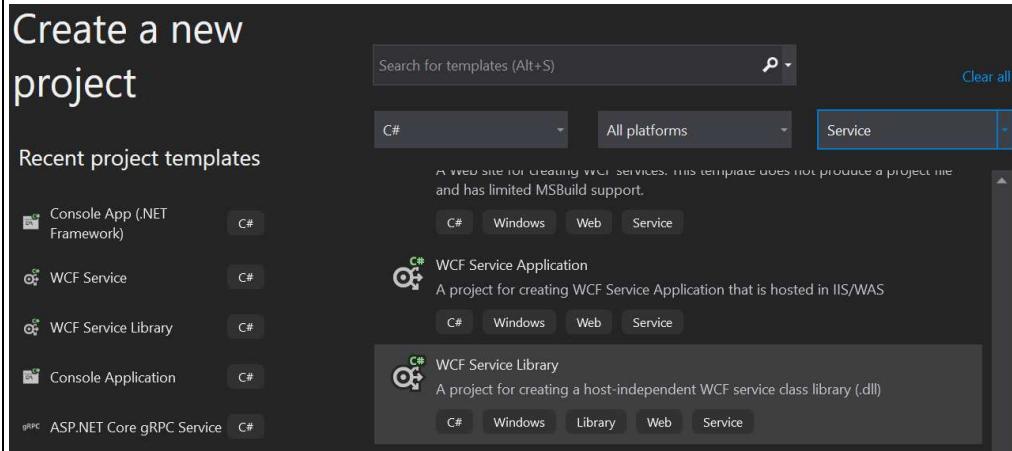
W zadaniu zostanie zrealizowane rozwiązanie obejmujące: **a)** usługę uruchamianą jako odrębna aplikacja i **b)** klienta korzystającego z tej usługi. Usługa i klient zostaną zrealizowane w narzędziem Visual Studio (dalej w skrócie VS) jako usługa WCF. Zadanie realizacji podstawowej aplikacji WCF klient-serwer składa się z kilku etapów:

1. Zdefiniowanie kontraktu usługi.
2. Implementacja kontraktu usługi (implementacja usługi).
3. Utworzenie aplikacji hostującej usługę
 - tu: jest to aplikacja konsolowa – tzw. self-hosting service.
4. Utworzenie aplikacji klienckiej i w niej proxy klienta (ang, *proxy client*).
5. Skonfigurowanie proxy klienta.
6. Implementacja aplikacji klienckiej.

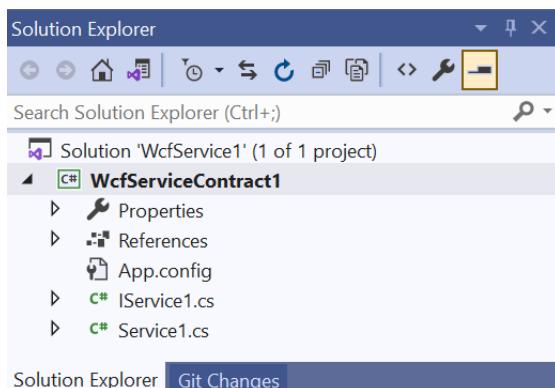
UWAGA: *wszelkie zmiany (np. nazw klas itp.) w już automatycznie wygenerowanym przez platformę kodzie najlepiej realizować poprzez opcję refaktoryzacji narzędziami platformy*

- W VS2013 **opcja Refactor w menu kontekstowym** (po kliknięciu prawego klawisza myszy).
- W VS2015÷2019 **odpowiednia opcja w menu kontekstowym** (np. Rename) - brak jest wyróżnionej opcji Refactor.

1. Wstępne przygotowanie środowiska	<ul style="list-style-type: none"> • Przygotuj katalog roboczy, w którym będą tworzona aplikacja WCF. • Uruchom platformę Visual Studio. • Sprawdź i ewentualnie ustaw foldery w których będą tworzone projekty w opcji TOOLS→Options...→Projects and Solutions→Locations (lub General we wcześniejszych wersjach VS).
2. Definiowanie kontraktu usługi WCF	<ul style="list-style-type: none"> • Utwórz nowąację i projekt aplikacji z szablonu Visual C# WCF Service Library nadając własne nazwy solucji i projektowi (tu: WcfService1 i WcfServiceContract1).



- Przejrzyj zawartość projektu. Zwróć uwagę na następujące elementy:
 - **IService1.cs** – plik deklaracji (interfejs) kontraktu,
 - **Service1.cs** – implementacja kontraktu,
 - **App.config** – konfiguracja własności i dostępności implementacji kontraktu.



Uwaga: w najnowszej wersji VS nazwy plików mogą się zmienić po zmianie nazwy interfejsu lub klasy z domyślnej (jak wyżej) na własną – na taką jak własna nazwa klasy lub interfejsu. Takie zachowanie jest konfigurowalne (można to wyłączyć).

- Otwórz plik **IService1.cs** i zdefiniuj kontrakt usługi – interfejs **ICalculator** zawierający metody Add, Sub, i Multiply:
 - Usuń nieużywany kod.
 - Dopisz lub przerób kod do postaci:

Uwaga: nazwa przestrzeni nazw (namespace) jest ustawiana taka jak nazwa projektu – nie trzeba jej zmieniać.

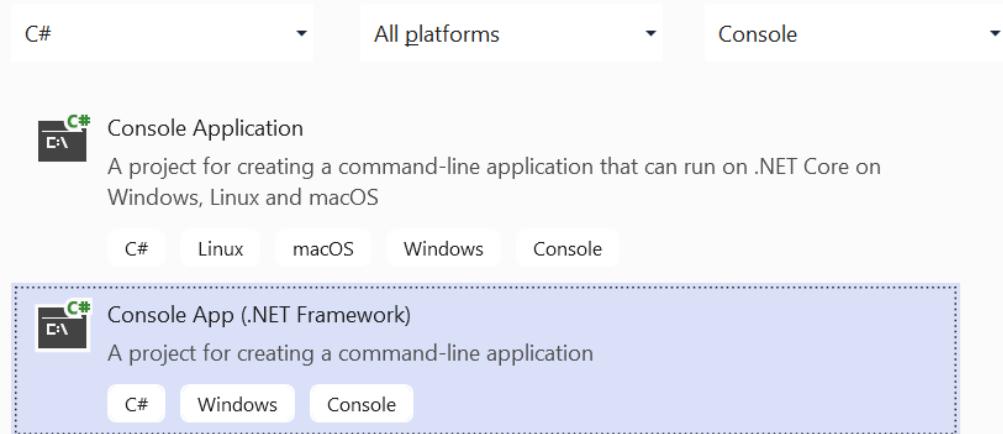
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.Text;
```

	<pre> namespace WcfServiceContract1 { [ServiceContract(ProtectionLevel = ProtectionLevel.None)] public interface ICalculator { [OperationContract] double Add(double n1, double n2); [OperationContract] double Sub(double n1, double n2); [OperationContract] double Multiply(double n1, double n2); } } </pre> <p>Uwaga: tu po zmianie nazwy z <i>IService1</i> na ICalculator (opcją menu – nie manualnie!) nazwa pliku może zmienić się w projekcie.</p> <ul style="list-style-type: none"> • Własność (<i>behavior</i>) ProtectionLevel (ustawioną na None) dodano w celu uniknięcia kwestii oprogramowania autoryzacji przy dostępie do usług.
3. Implementacja kontraktu usługi WCF	<ul style="list-style-type: none"> • Otwórz plik Service1.cs. Wpisz kod klasy MyCalculator implementującej interfejs ICalculator: Zaimplementuj każdą z wymaganych metod: <pre> using System; using System.Collections.Generic; using System.Linq; using System.Runtime.Serialization; using System.ServiceModel; using System.Text; </pre> <pre> namespace WcfServiceContract1 { public class MyCalculator : ICalculator { public double Add(double n1, double n2) { ... } public double Sub(double n1, double n2) { ... } public double Multiply(double n1, double n2) { ... } } } </pre> • W miejsce kropek ... dopisz odpowiedni kod dla każdej metody: <ul style="list-style-type: none"> - wykonanie odpowiedniego działania, - wyświetlenie informacji w konsoli co jest wywołane, co otrzymano w wywołaniu i co jest zwracane, - zwrócenie odpowiedniej wartości.

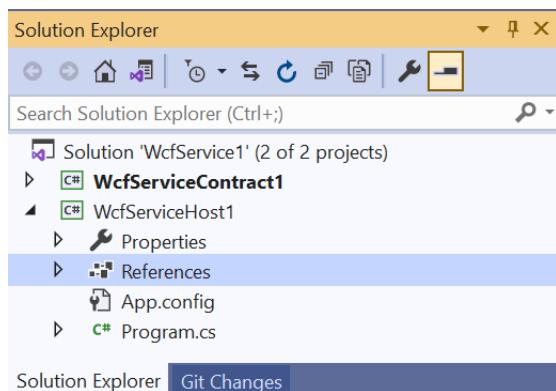
4. Hostowanie usługi WCF

Utwórz aplikację konsolową hostującą usługę WCF (Host usługi).

- Dodaj do dotychczasowej solucji drugi projekt aplikacji konsolowej nadając mu własną nazwę (tu: WcfServiceHost1)
 - opcja: **Add... → New Project.**



- Sprawdź (i ewentualnie ustaw) wersję Framework'a aplikacji.
 - Przy tworzeniu lub zaznacz projekt w **Solution Explorer**, prawym klawiszem wywołaj menu kontekstowe Właściwości (**Properties**) i sprawdź wartość w opcji **Application → Target Framework**.
- Dodaj w projekcie referencję do projektu kontraktu usługi WCF:
 - Zaznacz w Solution Explorer folder **References** i wybierz opcję **Add Reference**.

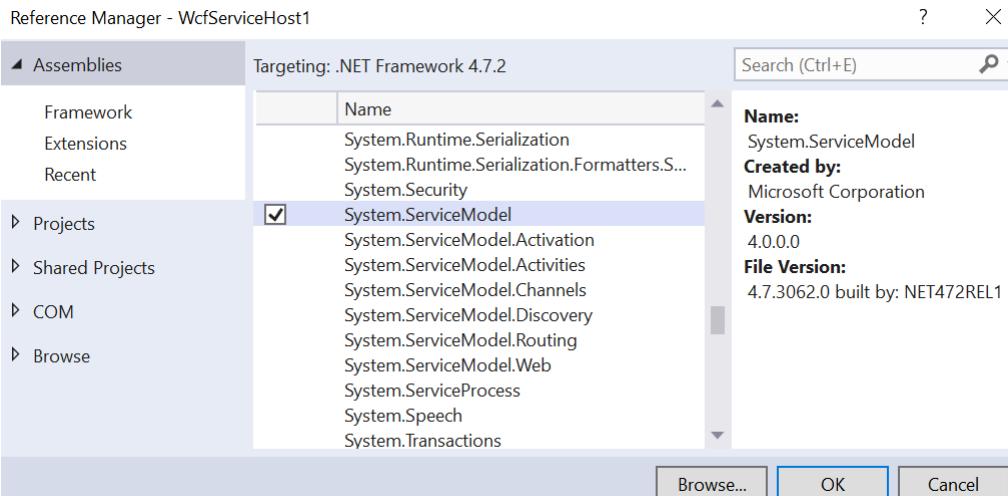


- W oknie menadżera referencji wybierz **Solution/Project**, zaznacz projekt kontraktu usługi WCF i zatwierdź:

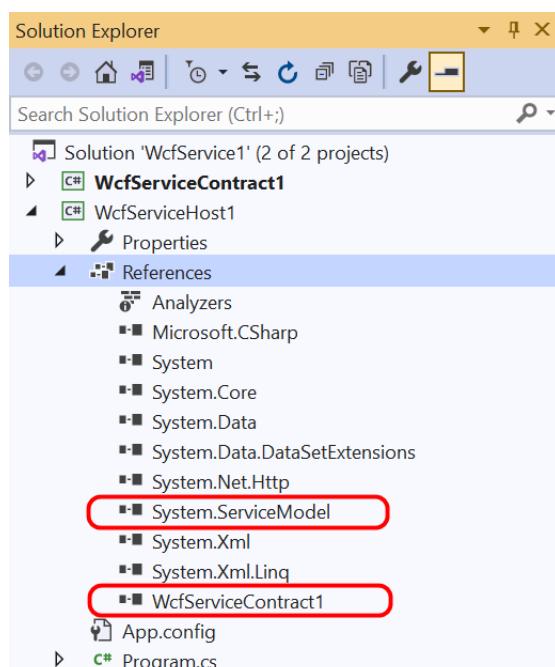


- Dodaj w projekcie referencję do **System.ServiceModel**:
 - Kliknij w Solution Explorer prawym klawiszem folder **References** i wybierz opcję **Add Reference**.

- W oknie menadżera referencji wybierz **Assemblies/Framework**, zaznacz **System.ServiceModel** i zatwierdź.



- Zweryfikuj pojawienie się dodatkowych referencji w projekcie jak na rysunku poniżej:



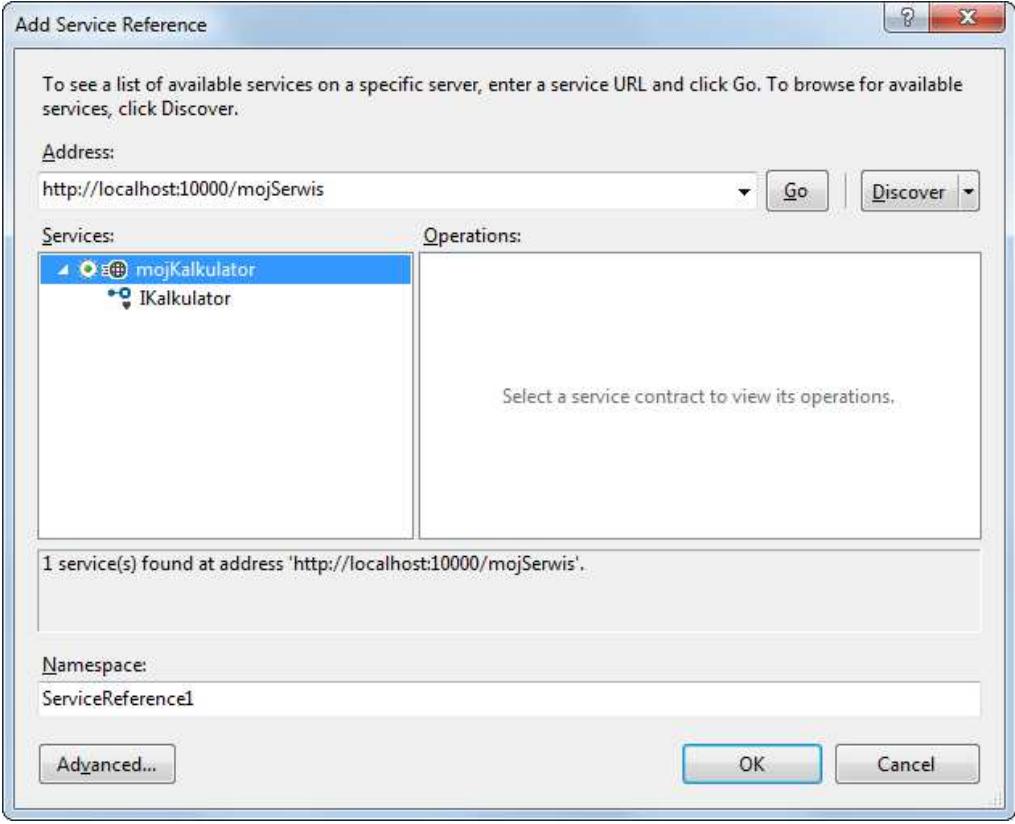
- Otwórz plik **Program.cs** i wpisz kod realizujący następujące funkcje:
 - Utworzenie URI z bazowym adresem serwisu.
 - Utworzenie instancji serwisu.
 - Dodanie punktu końcowego serwisu.
 - Umożliwienie wymiany metadanych (informacji o serwisie).
 - Uruchomienie serwisu (i na koniec zamknięcie serwisu).

Zamiast **xxx** podaj port **10000+nr stanowiska w laboratorium**.

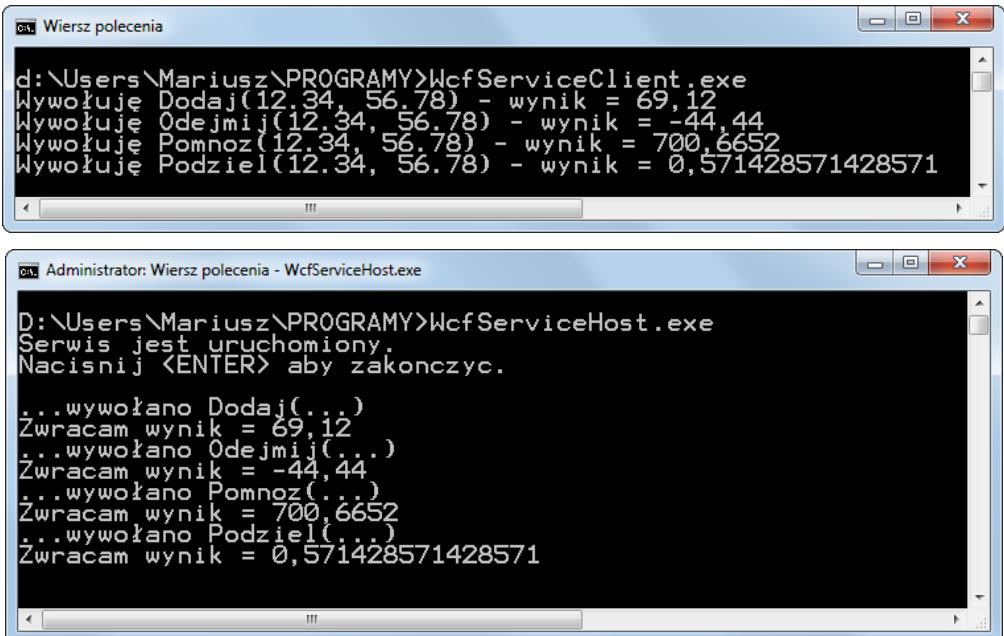
Zamiast **NazwaBazowa** (nazwa serwisu) wpisz własną nazwę swojej usługi.

	<pre> using System; using System.Collections.Generic; using System.Linq; using System.Text; using System.Threading.Tasks; namespace WcfServiceHost { class Program { static void Main(string[] args) { // Krok 1 Utworz URI dla bazowego adresu serwisu. Uri baseAddress = new Uri("http://localhost:xxx/NazwaBazowa"); // Krok 2 Utworz instancje serwisu. ServiceHost myHost = new ServiceHost(typeof(MyCalculator), baseAddress); // Krok 3 Dodaj endpoint. WSHttpBinding myBinding = new WSHttpBinding(); ServiceEndpoint endpoint1 = myHost.AddServiceEndpoint (typeof(ICalculator), myBinding, "endpoint1"); // Krok 4 Ustaw włączenie metadanych. ServiceMetadataBehavior smb = new ServiceMetadataBehavior(); smb.HttpGetEnabled = true; myHost.Description.Behaviors.Add(smb); try { // Krok 5 Uruchom serwis. myHost.Open(); Console.WriteLine("Serwis jest uruchomiony."); Console.WriteLine("Nacisnij <ENTER> aby zakończyć."); Console.ReadLine(); // aby nie kończyć natychmiast: myHost.Close(); } catch (CommunicationException ce) { Console.WriteLine("Wystąpił wyjątek: {0}", ce.Message); myHost.Abort(); } } } } </pre> <ul style="list-style-type: none"> Usuń błędy poprzez dodanie importu odpowiednich bibliotek - po zaznaczeniu kursorem, opcja Quick Actions and Refactorings... w menu kontekstowym lub Show potential fixes. <ul style="list-style-type: none"> Najczęściej będzie to dodanie dyrektywy importu nazw using.
5. Testowanie działania aplikacji	<p><i>UWAGA: aby uruchomić serwis trzeba mieć uprawnienia administratora w systemie Windows. W innym przypadku trzeba system dodatkowo odpowiednio skonfigurować.</i></p> <ul style="list-style-type: none"> Przetestuj poprawność działania aplikacji

	<ul style="list-style-type: none"> ○ Zbuduj kod wykonywalny aplikacji – użyj jednej z opcji: <ul style="list-style-type: none"> - BUILD → Build Nazwa_projektu (zbudowanie aplikacji – zwykle za pierwszym razem). - BUILD → Rebuild Nazwa_projektu (kolejne powtórzenia – zwykle po modyfikacjach). - BUILD → Build Solution (dotyczy całej solucji / wszystkich projektów – zwykle za pierwszym razem) - BUILD → Rebuild Solution (zwykle kolejne powtórzenia). ○ Uruchom z linii komend aplikację hostującą serwis WCF  <ul style="list-style-type: none"> ○ Uruchom przeglądarkę i spróbuj połączyć się z adresem: http://localhost:xxx/NazwaBazowa <i>Połączenie z hostem i wyświetlenie strony z odpowiednim opisem oznacza poprawność działania aplikacji.</i> ● Przejdź do strony z opisem WSDL usługi. Zamknij węzły <i>Policy</i> (są tu nieistotne) i zidentyfikuj ważne części opisu usługi: typy, wiadomości, operacje, punkt dostępu, itp. ● Uruchomienie hosta usługi z poziomu VS automatycznie uruchamia wbudowanego klienta umożliwiającego przetestowanie działania usługi. Kliknij w tym kliencie którąś operację i sprawdź postać wysyłanego komunikatu XML (SOAP).
6. Tworzenie klienta usługi.	<p>Utworzenie aplikacji klienta usługi i proxy klienta (client proxy) z użyciem funkcji Visual Studio: Add Service Reference.</p> <ul style="list-style-type: none"> ● Dodaj do solucji trzeci projekt aplikacji z szablonu Visual C# Console Application nadając mu własną nazwę. ● Sprawdź (i ewentualnie ustaw) wersję Framework'a aplikacji (tak samo jak w drugim projekcie). ● Dodaj w projekcie referencję do System.ServiceModel (tak samo jak w drugim projekcie). ● Dodaj referencję serwisową do zdefiniowanej usługi: (Ilustracja dalej na rysunku) <ul style="list-style-type: none"> ○ Uruchom najpierw aplikację hostującą usługę WCF! ○ Zaznacz w Solution Explorer prawym klawiszem folder References i wybierz opcję Add Service Reference. ○ W oknie Add Service Reference, w polu Address wpisz adres usługi (endpoint):

	<p>http://localhost:xxx/NazwaBazowa</p> <p>Zamiast xxx podaj odpowiedni nr portu.</p> <ul style="list-style-type: none"> ○ Naciśnij przycisk Go i powinna pojawić się dostępna usługa na podanym punkcie dostępu (patrz rys. dalej). Wybranie kontraktu (interfejsu) dodatkowo pokaże dostępne operacje (metody). ○ Zatwierdź wybór przyciskiem OK. <p><i>W powyższy sposób generowany jest kod proxy klienta (client proxy) – dołączanego modułu, realizującego wywołania usług.</i></p> 
7. Konfigurowanie klienta	<p>Konfiguracja klienta zawarta jest, w utworzonym poprzez referencję serwisową, pliku konfiguracyjnym App.config projektu klienta.</p> <ul style="list-style-type: none"> • Otwórz plik App.config klienta i przeanalizuj jego zawartość. • Zwróć uwagę zwłaszcza na sekcję i nazwę i typ wiązania (binding), sekcję i nazwę punktu dostępu (endpoint) oraz specyfikację kontraktu (contract). • Jego zawartość powinna być podobna do tej na ilustracji.

	 <pre> <?xml version="1.0" encoding="utf-8" ?> <configuration> <startup> <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5" /> </startup> <system.serviceModel> <bindings> <wsHttpBinding> <binding name="WSHttpBinding_IKalkulator" /> </wsHttpBinding> </bindings> <client> <endpoint address="http://localhost:10000/mojSerwis/endpoint1" binding="wsHttpBinding" bindingConfiguration="WSHttpBinding_IKalkulator" contract="ServiceReference1.IKalkulator" name="WSHttpBinding_IKalkulator"> <identity> <userPrincipalName value="DOMAF\Adsuro" /> </identity> </endpoint> </client> </system.serviceModel> </configuration> </pre>
8. Implementacja klienta	<ul style="list-style-type: none"> Otwórz plik Program.cs i wpisz kod klienta realizujący działania: <ul style="list-style-type: none"> Utworzenie instancji klienta (WCF proxy). Wywołanie operacji usługi z klienta (proxy). Zamknięcia klienta <pre> using System; using System.Collections.Generic; using System.Linq; using System.Text; using System.Threading.Tasks; namespace WcfServiceClient1 { class Program { static void Main(string[] args) { //Krok 1: Utworzenie instancji WCF proxy. CalculatorClient myClient = new CalculatorClient(); // Krok 2: Wywolanie kolejnych operacji uslugi. ... double result = myClient.Add(value1, value2); Console.WriteLine(...); ... result = myClient.Sub(value1, value2); Console.WriteLine(...); ... result = myClient.Multiply(value1, value2); Console.WriteLine(...); // Krok 3: Zamknięcie klienta zamknie połączenie i czyszcza zasoby. myClient.Close(); } } } </pre> <ul style="list-style-type: none"> W miejsce kropek wstaw wypisanie odpowiednich komunikaty W miejsce value1 i value2 wpisz jakieś liczby. Usuń błędy poprzez dodanie importu odpowiednich nazw.

<p>9. Testowanie działania aplikacji</p>	<ul style="list-style-type: none"> Uruchom serwis (aplikację hostującą usługę WCF) w jednym oknie konsoli. Uruchom klienta w drugim oknie konsoli. Skontroluj wyniki działania. Efekt działania klienta i serwisu powinien być podobny do zamieszczonych ilustracji. 
<p>10. Modyfikacja kontraktu</p>	<ul style="list-style-type: none"> Dopisz w interfejsie kontraktu jeszcze jedną operację: <code>[OperationContract] double Summarize(double n1);</code> Dopisz w implementacji kontraktu zmienną globalną w klasie: <code>double sum=0;</code> Zdefiniuj w implementacji kontraktu operację "trwałego" sumowania <code>public double Summarize(double n1) { sum += n1; return sum; }</code>
<p>11. Modyfikacja hosta serwisu</p>	<ul style="list-style-type: none"> W projekcie hosta w pliku konfiguracyjnym zdefiniuj dodatkowy punkt końcowy dla serwisu o podanym wiązaniu. Otwórz plik App.config i wpisz w węźle <configuration>, za węzłem <startup>, kod (kolejno): <ul style="list-style-type: none"> Usługę (service) – węzeł <service> – a w nim: Punkt końcowy (endpoint) usługi – węzeł <endpoint>,

	<pre> <system.serviceModel> <services> <service name="WcfServiceContract1.MyCalculator"> <endpoint name="myEndpoint3" address="/endpoint3" binding="wsHttpBinding" contract="WcfServiceContract1.ICalculator"> </endpoint> </service> </services> </system.serviceModel> </pre> <p><i>W tym pliku można konfigurować wszelkie elementy serwisu.</i></p> <ul style="list-style-type: none"> • Otwórz plik Program.cs hosta i dopisz kod realizujący następujące funkcje: <ul style="list-style-type: none"> ◦ Dodanie kolejnego endpointa (dla transportu BasicHttp). ◦ Wyświetlenie informacji o kontraktcie. • W bloku try {...} przed uruchomieniem serwisu (przed funkcją <code>Open()</code>) utwórz obiekt wiązania <code>BasicHttpBinding</code> (dla transportu Basic Http) i dodaj dodatkowy punkt końcowy/endpoint (jednocześnie pobierając go), oraz pobierz punkt końcowy serwisu zdefiniowany w pliku <code>app.config</code>. <p>Za endpointem 1 dopisz kod:</p> <pre> BasicHttpBinding binding2 = new BasicHttpBinding(); ServiceEndpoint endpoint2 = myHost.AddServiceEndpoint(typeof(IKalkulator), binding2, "endpoint2"); ServiceEndpoint endpoint3 = myHost.Description.Endpoints.Find(new Uri("http://localhost:xxx/MyService/endpoint3")); </pre> <ul style="list-style-type: none"> • Następnie dopisz kod wyświetlający informacje o endpointach (jak poniżej dla endpointa 1), powielając go dla każdego endpointa: <pre> Console.WriteLine("\n---> Endpointy:"); Console.WriteLine("\nService endpoint {0}:", endpoint1.Name); Console.WriteLine("Binding: {0}", endpoint1.Binding.ToString()); Console.WriteLine("ListenUri: {0}", endpoint1.ListenUri.ToString()); </pre>
12.Testowanie działania serwisu	<ul style="list-style-type: none"> • Przebuduj (opcja Rebuild) kontrakt usługi i host usługi. • Uruchom z konsoli serwis i sprawdź działanie. • Zapoznaj się z wyświetlonymi danymi w konsoli hosta.
13.Modyfikacja klienta dla działania z nowym hostem	<ul style="list-style-type: none"> • Upewnij się, że działa host i zaktualizuj referencje serwisu w aplikacji klienta: <ul style="list-style-type: none"> ◦ zaznacz w oknie solucji prawym klawiszem myszy węzeł Service References→ServiceReference1 i wybierz opcję Update Service Reference. • Sprawdź jakie zmiany pojawiły się w pliku App.config klienta.

	<p><i>Kod poniżej wykonuje się ponieważ proxy klienta można tworzyć bez specyfikacji endpointu tylko gdy jest jeden endpoint danego typu. W innym przypadku trzeba zawsze specyfikować endpoint dla proxy.</i></p> <ul style="list-style-type: none"> W pliku Program.cs zakomentuj instrukcję tworzenia proxy klienta bez specyfikacji endpointu w konstruktorze <pre>//CalculatorClient myClient = new CalculatorClient();</pre> W zamian dopisz instrukcje utworzenia proxy klientów dla operacji za pomocą różnych punktów końcowych. Przy tworzeniu klientów specyfikuj nazwę punktu odpowiednio do tych wygenerowanych w pliku App.config. <pre>CalculatorClient client1 = new CalculatorClient("WSHttpBinding_ICalculator"); CalculatorClient client2 = new CalculatorClient("BasicHttpBinding_ICalculator"); CalculatorClient client3 = new CalculatorClient ("myEndpoint3");</pre> <p>Nazwy punktów końcowych (klient 3) oraz nazwy wiązań (klient 1 i 2) muszą być zgodne z nazwami widocznymi w pliku App.config.</p> <ul style="list-style-type: none"> Stosownie do powyższego zakomentuj instrukcję wywołania operacji Dodaj(...), Odejmij(...), itd. poprzedniego proxy klienta tj. <code>myClient</code> i w zamian dopisz wywołania dla nowych klientów: <pre>result = clientx.operacja(...);</pre> <p>gdzie x- nr klienta, <i>operacja(...)</i> – to wywołanie Add(...), Sub(...), Multiply(...)... </p> <ul style="list-style-type: none"> Dopisz wywołanie operacji Summarize dwukrotnie dla każdego proxy klienta (czyli przez wszystkie punkty końcowe). <pre>result = clientx.Summarize(jakaś_wartość);</pre> <p>gdzie x- nr klienta, <i>jakaś_wartość</i> – wartość liczbowa.</p> <ul style="list-style-type: none"> Dopisz wyświetlanie odpowiednich komentarzy i wyników działań w konsoli klienta.
14. Testowanie działania usługi	<ul style="list-style-type: none"> Uruchom w konsoli klienta (pamiętaj wcześniej uruchomić serwis). Zwróć uwagę na poszczególne wartości wyników sumowania (czyli operacji <code>Summarize(...)</code>). Zatrzymaj serwis i klienta. W pliku Service1.cs kontraktu usługi (implementacji kontraktu) dopisz tuż przed definicją klasy instrukcję specyfikującą zachowanie serwisu jako pojedynczej instancji dla wszystkich klientów: <pre>[ServiceBehavior(InstanceContextMode = InstanceContextMode.Single)]</pre> <ul style="list-style-type: none"> Przebuduj (opcja Rebuild) kontrakt i host usługi.

- | | |
|--|--|
| | <ul style="list-style-type: none">• Uruchom z konsoli serwis.• Uaktualnij referencję serwisową klienta i przebuduj klienta.• Uruchom w konsoli klienta (dwukrotnie) i ponownie zwróć uwagę na poszczególne wartości wyników sumowania – zmianę w porównaniu do poprzedniej wersji usługi. |
|--|--|

3 Zadanie – część II

- A. Przećwiczyć technikę tworzenia serwisów i klientów WCF.
1. Definiowanie i implementacja kontraktów.
 2. Tworzenie hosta usługi.
 3. Definiowanie punktów końcowych (endpoint) o określonych adresach.
 4. Definiowanie sposobu komunikacji (BasicHttp, WSHttp, NetTCP).
 5. Tworzenie klienta, wiązanie i wywoływanie operacji usług.
- B. Przygotować się do napisania na zajęciach aplikacji o podobnych funkcjonalnościach według wskazówek prowadzącego.

Ćwiczenie 4.2

WCF – kontrakty danych, operacje jednokierunkowe i zwrotne (callbacks).

Autor: Mariusz Fraś

1 Cele ćwiczenia

Celem ćwiczenia jest:

1. Zapoznanie się z kontraktami danych - specyfikacja przekazywania w komunikatach złożonych zestawów danych.
 2. Poznanie kolejnych opcji konfigurowania serwisu i typu operacji usług, w tym operacji jednokierunkowych.
 3. Poznanie sposobu definiowania i obsługi usług typu callback (lub inaczej duplex) – usługi ze zwrotnym wywołaniem procedury obsługi odpowiedzi na żądanie.
-
- **Pierwsza część zadania** jest do wykonania według podanej instrukcji i ewentualnych poleceń prowadzącego zajęcia laboratoryjne.
 - **Druga część zadania jest do przygotowania i oddania lub do wg wykonania wg. poleceń prowadzącego na kolejnych zajęciach.**

2 Zadanie – część I

Zadanie polega na realizowaniu usługi wykorzystującej kontrakty danych (**DataContract**) oraz usługi realizującej operacje synchroniczne i asynchroniczne wykonywane w serwisie wielowątkowo. Ponadto zadanie polega na zrealizowaniu usługi asynchronicznej zwracającej wynik wykonania operacji, ze zwrotnym wywołaniem – tzw. kontrakt typu **Callback**.

Zadanie składa się z kilku następujących etapów:

1. Utworzenie usługi zawierającej kontrakt danych i klienta korzystającego z niej.
2. Definicja serwisu z operacją typu żądanie-odpowiedź i operacji jednokierunkowej (**OneWay** - żądanie bez odpowiedzi), oraz deklaracja serwisu jako wielowątkowego.
3. Następnie rozbudowa klienta dla wywołania dodatkowych operacji.
4. Zdefiniowanie usługi z kontraktem typu **Callback**, z usługami jednokierunkowymi.
5. Rozbudowa klienta o uchwyty (ang. *handler*) z operacjami, które będą wywoływane przez serwer po zakończeniu usługi, w celu zwrócenia rezultatu działania.

Uwaga: aplikacje należy uruchamiać z poziomu konsoli – to pozwoli łatwiej przyswoić materiał (zwłaszcza kwestie konfigurowania serwisów).

1. Wstępne przygotowanie solucji (projektów)	<ul style="list-style-type: none"> • Utwórz nowąację, a w niej kolejno 3 projekty: <ul style="list-style-type: none"> - kontrakt usługi – projekt WCF Library, - host usługi – projekt aplikacji konsolowej, - klient usługi – projekt aplikacji konsolowej. • Skonfiguruj wstępnie kolejno 3 projekty (podobnie jak w poprzednim ćwiczeniu): <ul style="list-style-type: none"> - referencję w hoście do System.ServiceModel i do kontraktu, - referencję w kliencie do System.ServiceModel. <p>Uwaga: nie konfiguruj Connected Services References dopóki kontrakt i host nie będą gotowe.</p>
2. Definiowanie kontraktu usługi i kontraktu danych	<p>Zdefiniuj w projekcie kontraktu – w pliku opisującym interfejs usługi IService1.cs – kontrakt usługi mającej operację (metodę) przyjmującej parametry typu złożonego (złożony zestaw danych) i zwracającą parametr typu złożonego – typ opisany kontraktem danych (DataContract). W prezentowanym przykładzie są to dane opisujące liczbę zespoloną (mającą część rzeczywistą i urojoną).</p> <ul style="list-style-type: none"> • Wpisz opis interfejsu (kontraktu usługi) IComplexCalc zawierający metodę addCNum: <pre>[ServiceContract] public interface IComplexCalculator { [OperationContract] ComplexNum addCNum(ComplexNum n1, ComplexNum n2); }</pre>

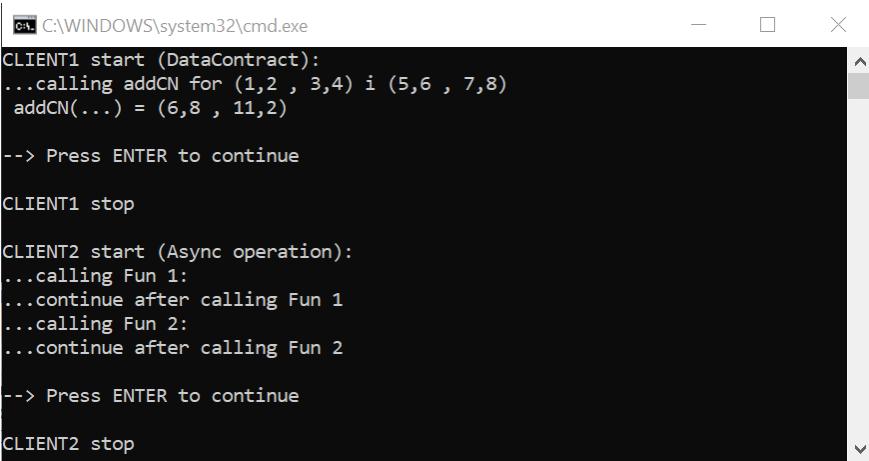
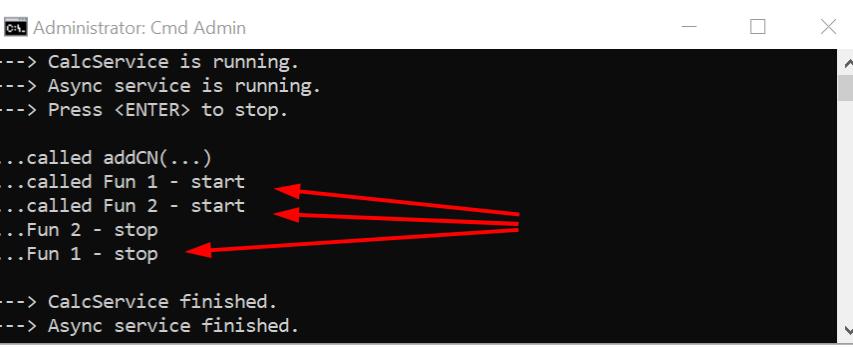
	<ul style="list-style-type: none"> Poniżej zdefiniuj kontrakt danych ([DataContract]) odpowiadający liczbie zespolonej ComplexNum (czyli złożony zestaw danych) przekazywanej i zwracanej przez metodę addCNum usługi, t.j: <ul style="list-style-type: none"> - klasę ComplexNum – ze specyfikatorem [DataContract], - elementy [DataMember] – są przekazywane w wiadomościach, - tu także wykorzystana możliwa opcja - konstruktor klasy ComplexNum. <pre>[DataContract] public class ComplexNum { string description = "Complex number"; [DataMember] public double real; [DataMember] public double imag; [DataMember] public string Desc { get { return description; } set { description = value; } } public ComplexNum(double r, double i) { this.real = r; this.imag = i; } }</pre>
3. Implementacja kontraktu usługi	<p>Zaimplementuj kontrakt – klasę MyComplexCalc (z wymaganymi metodami).</p> <ul style="list-style-type: none"> W pliku Service1.cs wpisz kod klasy MyComplexCalc implementującej interfejs IComplexCalc. <pre>public class MyComplexCalc : IComplexCalc { public ComplexNum addCNum(ComplexNum n1, ComplexNum n2) { Console.WriteLine("...called addCNum(...)"); return new ComplexNum(n1.real + n2.real, n1.imag + n2.imag); } }</pre> <ul style="list-style-type: none"> Dodaj odpowiednie deklaracje using ... stosownie do potrzeb (np. z menu kontekstowego prawego klawisza myszki).
4. Implementacja i konfiguracja hosta usługi.	<p>Utwórz aplikację konsolową hostującą usługę WCF.</p> <ul style="list-style-type: none"> W projekcie hosta w pliku konfiguracyjnym zdefiniuj podstawowe elementy serwisu – punkt końcowy, adres bazowy, binding, itp. jednym ze sposobów poznanych w poprzednim ćwiczeniu. Np.:

	<ul style="list-style-type: none"> ○ <u>Programując w kodzie C# hosta.</u> Otwórz plik Program.cs i wpisz kod realizujący następujące funkcje: <ul style="list-style-type: none"> ▪ Utworzenie URI z bazowym adresem serwisu. ▪ Utworzenie obiektu ServiceHost. ▪ Utworzenie (dodanie do hosta) punktu końcowego (endpoint). ▪ Skonfigurowanie hosta serwisu aby udostępniał metadane. ▪ Uruchomienie serwisu. (w miejsce kropek ... dopisz odpowiedni kod) <pre> Uri baseAddress1 = new Uri(...); ServiceHost myHost1 = new ServiceHost(...); ServiceEndpoint endpoint1 = myHost1.AddServiceEndpoint(...); // Metadata: ServiceMetadataBehavior smb = new ServiceMetadataBehavior(); mb.HttpGetEnabled = true; myHost1.Description.Behaviors.Add(smb); try { myHost1.Open(); Console.WriteLine("--> ComplexCalculator is running."); Console.WriteLine("--> Press <ENTER> to stop.\n"); Console.ReadLine(); //wait for stop myHost1.Close(); Console.WriteLine("--> ComplexCalculator finished"); } catch (CommunicationException ce) { Console.WriteLine("Exception occurred: {0}", ce.Message); myHost1.Abort(); } </pre> <p><u>albo częściowo skonfiguruj serwis w pliku App.config:</u></p> <ul style="list-style-type: none"> ○ Otwórz plik App.config i wpisz w węźle <configuration> (za węzłem <startup>) kod definiujący niezbędne elementy: ● Uruchom z konsoli serwis (host) i sprawdź działanie.
5. Tworzenie klienta proxy i implementacja klienta	<p><u>Utworzenie kodu klienta proxy (proxy client) z użyciem opcji Add Service Reference.</u></p> <ul style="list-style-type: none"> ● W projekcie klienta, dodaj referencję serwisową do zdefiniowanej usługi. Uwaga: pamiętaj o uruchomieniu najpierw serwisu! <p><u>Implementacja klienta (aplikacji):</u></p> <ul style="list-style-type: none"> ● W pliku Program.cs wpisz kod klienta realizujący następujące działania: <ul style="list-style-type: none"> ○ Utworzenie instancji klienta proxy. ○ Przygotowanie parametrów do wywołania operacji serwisu. ○ Wywołanie operacji usługi (z użyciem klienta proxy). ○ Zamknięcie klienta proxy.

	<pre> class Program { static void Main(string[] args) { ComplexCalcClient client1 = new ComplexCalcClient("WSHttpBinding_IComplexCalc"); ComplexNum cnum1 = new ComplexNum(); cnum1.real= 1.2; cnum1.imag= 3.4; ComplexNum cnum2 = new ComplexNum(); cnum2.real = 5.6; cnum2.imag = 7.8; Console.WriteLine("\nCLIENT1 - START"); Console.WriteLine("...calling addCNum(...)"); ComplexNum result1 = client1.addCNum(cnum1, cnum2); Console.WriteLine(" addCNum(...) = ({0},{1})", result1.real, result1.imag); Console.WriteLine("--> Press ENTER to continue"); Console.ReadLine(); client1.Close(); Console.WriteLine("CLIENT1 - STOP"); } } </pre> <ul style="list-style-type: none"> Utwórz odpowiednie referencje i zlikwiduj sygnalizowane błędy.
6. Testowanie działania aplikacji	<ul style="list-style-type: none"> Uruchom serwis (aplikację hostującą usługę WCF) i klienta w oddzielnych oknach konsoli i skontroluj działanie aplikacji. Zakończ działanie wszystkich aplikacji.
7. Serwis wielowątkowy i operacje asynchroniczne	<p>Zdefiniuj w dotychczasowym projekcie usługi, drugą usługę z operacją asynchroniczną.</p> <p>Note: Presented here asynchronous requests pattern is one of several available.</p> <ul style="list-style-type: none"> W pliku interfejsu, dopisz interfejs drugiej usługi zawierającej deklaracje dwóch metod: <ul style="list-style-type: none"> - operację Fun1 – operację typu asynchronicznego bez odpowiedzi (operację jednokierunkową) – typu OneWay - operację Fun2 standardową operację typu żądanie-odpowiedź. Obie funkcje nic nie zwracają. <pre> [ServiceContract] public interface IAsyncService { [OperationContract(IsOneWay = true)] void Fun1(String s1); [OperationContract] void Fun2(String s2); } </pre>

	<ul style="list-style-type: none"> Zaimplementuj w kontrakcie nową usługę i obie operacje. <p>W pliku Service1.cs dopisz kod klasy AsyncService: implementującej interfejs IAsyncService (są to nazwy własne).</p> <ul style="list-style-type: none"> Zadeklaruj zachowanie serwisu jako wielowątkowe. Dopisz kod operacji Fun1 i Fun2 Umieść w nich uśpienie procesu na kilka sekund w celu symulacji długiego czasu przetwarzania. <pre>[ServiceBehavior(ConcurrencyMode=ConcurrencyMode.Multiple)] public class AsyncService : IAsyncService { public void Fun1(String s1) { Console.WriteLine("...called Fun 1 - start"); Thread.Sleep(4*1000); // sleep for 4 sec. (4000 ms) Console.WriteLine("...Fun 1 - stop"); return; } public void Function2(String s2) { Console.WriteLine("...called Fun 2 - start "); Thread.Sleep(2*1000); // sleep for 2 sec. (2000 ms) Console.WriteLine("...Fun 2 - stop"); return; } }</pre> <p>Uwaga: zachowanie wielowątkowe jest niezbędne do realizacji żądań równolegle (natychmiastowe obsługiwanie kolejnych żądań). W szczególnych przypadkach w zależności od wiązania i serwisu, klient mimo to może być blokowany do czasu ukończenia poprzedniego żądania</p>
8. Rozbudowa hosta dla drugiej usługi	<p>Dopisz w kodzie aplikacji hostującej uruchomienie drugiego serwisu.</p> <ul style="list-style-type: none"> W pliku Program.cs dopisz w odpowiednich miejscach kod realizujący następujące funkcje: <ul style="list-style-type: none"> Utworzenie URI z bazowym adresem drugiego serwisu. Podaj inny niż wcześniej adres (port) dla usługi. Utworzenie obiektu hosta drugiego serwisu. Dodanie punktu końcowego z wiązaniem BasicHttpBinding. Skonfigurowanie hosta serwisu aby udostępniał metadane. Uruchomienie drugiego serwisu. <p>Uwaga: tu kropki [...] wskazują już istniejące fragmenty kodu.</p> <pre>static void Main(string[] args) { [...] Uri baseAddress2 = new Uri(...); ServiceHost myHost2 = new ServiceHost(...); ServiceEndpoint endpoint2 = myHost2.AddServiceEndpoint(...); myHost2.Description.Behaviors.Add(smb);</pre>

	<pre> try { [...] myHost2.Open(); Console.WriteLine("--> Async service is running"); [...] myHost2.Close(); Console.WriteLine("--> Async service finished"); [...] } catch (CommunicationException ce) { [...] myHost2.Abort(); } } </pre> <p>Uważaj aby NIE zatrzymać serwisu przed odesłaniem wyników !</p> <ul style="list-style-type: none"> • Skompiluj i uruchom z konsoli serwis (host) i sprawdź jego działanie.
9. Rozbudowa klienta dla drugiej usługi.	<p><u>Utworzenie kodu proxy</u> (z użyciem opcji Add Service Reference).</p> <ul style="list-style-type: none"> • W projekcie aplikacji konsolowej klienta, dodaj referencję serwisową do drugiej usługi. Pamiętaj o uruchomieniu najpierw serwisu ! <p><u>Implementacja/rozbudowa klienta (aplikacji)</u>:</p> <ul style="list-style-type: none"> • W pliku Program.cs dopisz kod realizujący następujące działania: <ul style="list-style-type: none"> ◦ Utworzenie instancji klienta proxy do drugiej usługi. ◦ Wywołanie operacji Fun1 i Fun2 (10 ms delays are added to properly display the order of service activities) ◦ Zamknięcie klienta proxy. <pre> static void Main(string[] args) { [HERE THE PREVIOUS CODE] Console.WriteLine("CLIENT2 - START (Async service)"); AsyncServiceClient client2 = new AsyncServiceClient("BasicHttpBinding_IAsyncService"); Console.WriteLine("...calling Fun 1"); client2.Fun1("Client2"); Thread.Sleep(10); Console.WriteLine("...continue after Fun 1 call"); Console.WriteLine("...calling Fun 2"); client2.Fun2("Client2"); Thread.Sleep(10); Console.WriteLine("...continue after Fun 2 call"); Console.WriteLine("--> Press ENTER to continue"); Console.ReadLine(); client2.Close(); Console.WriteLine("CLIENT2 - STOP"); } </pre>

<p>10. Testowanie aplikacji</p>	<ul style="list-style-type: none"> • Uruchom serwis (aplikację hostującą usługę WCF) w jednym oknie konsoli. • Uruchom klienta w drugim oknie konsoli. • Skontroluj wyniki działania. <ul style="list-style-type: none"> ○ Zwróć uwagę na momenty czasowe działania klienta i serwisu przy wywoływaniu metod Funkcja1 i Funkcja2 oraz kolejność komunikatów w serwisie. ○ Zwróć uwagę, że po wywołaniu Funkcja1 natychmiast realizowano kolejne działania (wywołanie funkcji 2). Po wywołaniu funkcji 2 trzeba czekać na jej zakończenie. • Wynik działania powinien być podobny do poniższego: <p>Okno klienta:</p>  <pre>C:\> C:\WINDOWS\system32\cmd.exe CLIENT1 start (DataContract): ...calling addCN for (1,2 , 3,4) i (5,6 , 7,8) addCN(...) = (6,8 , 11,2) --> Press ENTER to continue CLIENT1 stop CLIENT2 start (Async operation): ...calling Fun 1: ...continue after calling Fun 1 ...calling Fun 2: ...continue after calling Fun 2 --> Press ENTER to continue CLIENT2 stop</pre> <p>Okno hosta:</p>  <pre>--> CalcService is running. --> Async service is running. --> Press <ENTER> to stop. ...called addCN(...) ...called Fun 1 - start ...called Fun 2 - start ...Fun 2 - stop ...Fun 1 - stop --> CalcService finished. --> Async service finished.</pre> <ul style="list-style-type: none"> • Zakończ działanie wszystkich aplikacji.
<p>11. Wstępne przygotowanie projektu trzeciej usługi</p>	<p>Trzecia usługa będzie zwracać wyniki działań poprzez wywołania zwrotne do klienta (ang. callbacks), czyli asynchronicznie. To pozwala klientowi od razu realizować działania bez blokowania się w oczekiwaniu na wyniki wywołania operacji. Wyniki zostaną zwrócone przez serwis po wykonaniu operacji.</p> <p>W aplikacji ten sam host będzie uruchamiał dwie usługi – poprzednią i tą trzecią. Trzecia usługa będzie realizowana w oddzielnym projekcie.</p>

	<ul style="list-style-type: none"> • Dodaj do solucji projekt WCF Library trzeciej usługi (np. o nazwie CallbackService). • Dodaj w hoście referencję do tego drugiego pliku kontraktu.
12. Definiowanie kontraktu usługi z operacjami typu callback	<p>Zdefiniuj w projekcie kontraktu – w pliku opisującym interfejs usługi IService1.cs – kontrakt usługi typu Callback mającej dwie operacje (metody). W tym celu definiuje się:</p> <ul style="list-style-type: none"> - operacje typu OneWay, - zachowanie usługi typu CallbackContract specyfikujące typ interfejsu zwrotnego (tu: specyfikujemy go jako ISuperCalcCallback) – interfejs dla obsługi wywołań zwrotnych - ten interfejs ICallbackHandler musi być implementowany w klientcie, - zachowanie można zdefiniować jako atrybut kontraktu serwisu (w [ServiceContract]), - dodatkowo określmy wymaganie działania instancji serwisu w ramach sesji. <ul style="list-style-type: none"> • Zdefiniuj kod interfejsu kontraktu usługi ISuperCalc zawierający metody/operacje asynchroniczne (z callbackiem) Factorial (obliczanie silni) i DoSomething: (symulacja obliczania czegoś), definiując dodatkowo atrybut serwisu CallbackContract i wymaganie trybu działania w sesji: <pre>[ServiceContract (SessionMode = SessionMode.Required, CallbackContract=typeof(ISuperCalcCallback))] public interface ISuperCalc { [OperationContract(IsOneWay = true)] void Factorial(double n); [OperationContract(IsOneWay = true)] void DoSomething(int sec); }</pre> • Zdefiniuj tutaj również interfejs ISuperCalcCallback zawierający opis metod wywoływanych u klienta w celu przekazania rezultatów wykonania operacji Factorial i DoSomething – tj. zawierający metodę FactorialResult dla silni i metodę DoSomethingResult dla obliczeń czegoś. <p>Dopisz w tym samym pliku drugi interfejs:</p> <pre>public interface ISuperCalcCallback { [OperationContract(IsOneWay = true)] void FactorialResult(double result); [OperationContract(IsOneWay = true)] void DoSomethingResult(string result); }</pre>

13.Implementacja kontraktu usługi	<p>Zaimplementuj kontrakt – klasę implementującą każdą z wymaganych metod interfejsu ISuperCalc.</p> <ul style="list-style-type: none"> • W pliku Service1.cs. wpisz kod klasy MySuperCalc implementującej interfejs ISuperCalc. • Dla usługi definiuje się także zachowanie InstanceContextMode=PerSession oznaczające tworzenie instancji obiektu (instancji usługi) dla każdej sesji. • W konstruktorze pobierany jest uchwyt (handler) dla callback'u. <pre>[ServiceBehavior(InstanceContextMode = InstanceContextMode.PerSession, ConcurrencyMode = ConcurrencyMode.Multiple)] public class MySuperCalc : ISuperCalc { double result; ISuperCalcCallback callback = null; public MySuperCalc() { callback = OperationContext.Current.GetCallbackChannel <ISuperCalcCallback>(); } public void DoSomething(int sec) { Console.WriteLine("...called DoSomething({0})", sec); if (sec > 2 & sec < 10) Thread.Sleep(sec * 1000); else Thread.Sleep(3000); callback.DoSomethingResult("Calculatons lasted " + sec + " second(s)"); } public void Factorial(double n) { Console.WriteLine("...called Factorial({0})", n); Thread.Sleep(1000); result = 1; for (int i = 1; i <= n; i++) result *= i; callback.FactorialResult(result); } }</pre> <ul style="list-style-type: none"> • W obu metodach na końcu wywołujemy metody callback'owe w kliencie.
14.Rozbudowa hosta dla trzeciej usługi	<p>Dopisz w kodzie aplikacji hostującej uruchomienie trzeciego serwisu.</p> <ul style="list-style-type: none"> • W pliku Program.cs dopisz w odpowiednich miejscach kod realizujący następujące funkcje: <ul style="list-style-type: none"> ◦ Utworzenie URI z bazowym adresem trzeciego serwisu. ◦ Utworzenie obiektu hosta trzeciego serwisu. ◦ Dodanie punktu końcowego z wiązaniem WSDualHttpBinding.

	<ul style="list-style-type: none"> ○ Zdefiniowanie metadanych serwisu. ○ Uruchomienie trzeciego serwisu. <p><i>Uwaga:</i> tu kropki [...] oddzielają już istniejące fragmenty kodu.</p> <pre> static void Main(string[] args) { [...] Uri baseAddress3 = new Uri(...); ServiceHost myHost3 = new ServiceHost(typeof(MySuperCalc), baseAddress3); WSDualHttpBinding myBinding3 = new WSDualHttpBinding(); ServiceEndpoint endpoint3 = myHost3.AddServiceEndpoint(typeof(ISuperCalc), myBinding3, "ThirdService"); myHost3.Description.Behaviors.Add(smb); try { [...] myHost3.Open(); Console.WriteLine("--> Callback SuperCalc is running."); [...] myHost3.Close(); Console.WriteLine("--> Callback SuperCalc finished"); } catch (CommunicationException ce) { [...] myHost3.Abort(); } } </pre> <ul style="list-style-type: none"> • Uruchom z konsoli serwis (host) i sprawdź działanie.
15.Rozbudowa klienta dla wykorzystania trzeciej usługi	<ul style="list-style-type: none"> • Dodaj w kliencie referencję serwisową do trzeciej usługi: <i>Uwaga: pamiętaj o uruchomieniu najpierw aplikacji hostującej usługę!</i> • Dodaj do projektu klienta nową klasę (np. o nazwie SuperCalcCallback), w której zdefiniowane będą operacje wywoływane zwrotnie przez serwis, aby odesłać wyniki działania jego operacji usług. <pre> class SuperCalcCallback : ISuperCalcCallback { public void FactorialResult(double result) { //here the result is consumed Console.WriteLine(" Factorial = {0}", result); } public void DoSomethingResult(string info) { //here the result is consumed Console.WriteLine(" Calculations: {0}", info); } } </pre>

	<ul style="list-style-type: none"> Otwórz plik Program.cs i dopisz kod w funkcji main realizujący następujące działania: <ul style="list-style-type: none"> ○ Utworzenie obiektu uchwytu (handlera) z operacjami odbioru wyników od serwisu. ○ Utworzenie instancji klienta proxy (<i>proxy client</i>). ○ Wywołanie operacji usługi z klienta (proxy). ○ Zamknięcia klienta Odbiór i wypisywanie wyników będzie asynchroniczny – inicjowany przez serwis. <pre><code>static void Main(string[] args) { [HERE THE PREVIOUS CODE] Console.WriteLine("\nCLIENT3 - START (Callbacks):"); SuperCalcCallback myCbBHandler = new SuperCalcCallback (); InstanceContext instanceContext = new InstanceContext(myCbHandler); SuperCalcClient client3 = new SuperCalcClient(instanceContext); double value1 = 10; Console.WriteLine("...call of Factorial({0})...", value1); client3.Factorial(value1); int value2 = 5; Console.WriteLine("...call of DoSomething..."); client3.DoSomething(value2); value1 = 20; Console.WriteLine("...call of Factorial({0})...", value1); client3.Factorial(value1); Console.WriteLine("--> Client must wait for the results"); Console.WriteLine("--> Press ENTER after receiving ALL results"); Console.ReadLine(); client3.Close(); Console.WriteLine("CLIENT3 - STOP"); }</code></pre>
16. Testowanie działania aplikacji	<ul style="list-style-type: none"> Uruchom serwis (aplikację hostującą usługę) w jednym oknie konsoli. Uruchom klienta w drugim oknie konsoli. Skontroluj wyniki działania. Zwróć uwagę na momenty czasowe działania serwisu i klienta. Ostateczny wynik działania powinien być podobny do poniższego:

Okno klienta:

```
C:\ C:\WINDOWS\system32\cmd.exe
CLIENT1 start (DataContract):
...calling addCN for (1,2 , 3,4) i (5,6 , 7,8)
addCN(...) = (6,8 , 11,2)

--> Press ENTER to continue

CLIENT1 stop

CLIENT2 start (Async operation):
...calling Fun 1:
...continue after calling Fun 1
...calling Fun 2:
...continue after calling Fun 2

--> Press ENTER to continue

CLIENT2 stop

CLIENT3 start:
...calling Factorial(10)...
...calling DoSomething...
...calling Factorial(20)...
--> Client must wait to NOT finish before receiving results (callback)
--> Press ENTER after (!) you receive ALL results
Factorial = 3628800
Factorial = 2,43290200817664E+18
SomethingDone: Calculations took 5 seconds

CLIENT3 - stop
```

Okno hosta:

```
C:\ Administrator: Cmd Admin
--> CalcService is running.
--> Async service is running.
--> Callback service SuperCalc is running
--> Press <ENTER> to stop.

...called addCN(...)
...called Fun 1 - start
...called Fun 2 - start
...Fun 2 - stop
...Fun 1 - stop
...called Factorial(10)
...called DoSomething(5)
...called Factorial(20)
...Factorial(10) - finished
...Factorial(20) - finished
...DoSomething(5) - finished

--> CalcService finished.
--> Async service finished.
--> SuperCalc service finished.
```

- Zakończ działanie wszystkich aplikacji..

3 Zadanie – część II

Szczegółowe wymagania i wybór zadania (A, B lub inne) określa prowadzący grupy.

A. Przećwiczyć prezentowane w ćwiczeniu techniki.

1. Definiowanie, implementacja i wykorzystywanie kontraktów danych.
2. Operacje jednokierunkowe - typu OneWay.
3. Tworzenie serwisu i klienta z użyciem kontraktu serwisu typu Callback..

B. Przykładowe zadanie do wykonania

Napisać aplikację WCF - serwis i klienta spełniające następujące wymagania:

1. Klient jest aplikacją konsolową lub graficzną oraz:

- a. Wywoływanie operacji następuje po wybraniu akcji – np. po naciśnięciu odpowiedniego przycisku lub wybraniu opcji menu.
- b. Przesyłane dane są podawane przez użytkownika (są pobierane z kontrolek w aplikacji (dla aplikacji graficznej) lub wpisywane w konsoli przez użytkownika).
- c. Wyniki działania wywoływanych operacji są wypisywane na ekranie (w odpowiednim polu okna aplikacji (ewentualnie w kilku oddzielnych polach) lub w konsoli).

Uwaga: aplikacja graficzna może być troszkę wyżej punktowana!

2. Serwis (usługa) jest hostowany w aplikacji konsolowej oraz:

- a. Usługa utrzymuje (przechowuje) jakiś magazyn danych (zbiór/lista rekordów danych itp.), które można modyfikować (dodawać, usuwać,...) za pomocą operacji.
- b. Rekord danych powinien zawierać **dane różnego typu**.
- c. Usługa ma operację, która dodaje rekord danych. Ta operacja przyjmuje parametr określony za pomocą **kontraktu danych** (typ określający cały rekord).
- d. Usługa ma operację zwracającą wartości zestawu danych, oraz operacje zwracające wybrane pojedyncze dane.
- e. Usługa ma jakąś **operację typu callback (duplex)**
 - np. operację która coś wyszukuje lub sprawdza (np. czy w zestawie występuje jakaś podana wartość albo cecha albo podzbiór rekordów lub ile jest rekordów jakiegoś rodzaju itp.). Czyli:
 - Operacja po wywołaniu natychmiast zwraca sterowanie. Klient może w tym czasie wykonywać inne działania.
 - Po wykonaniu operacji informuje klienta o wyniku działania operacji (np. czy dane występują, wyświetla znalezione dane, lub podaje jakieś inne informacje, itp. – zależnie od tego jaka to operacja). Te działanie jest realizowane poprzez wywołanie zwrotne (wywołanie *callback* w kliencie).
 - Dodać kilkusekundową zwłokę w operacji dla zauważenia efektu działania, aby można było pokazać, jej asynchroniczność.

3. Zestaw danych (znaczenie, typy) należy sobie wybrać indywidualnie. Rekord zestawu danych powinien się składać z **danych różnego typu!**

Uwaga: klient nie może przechowywać listy danych głównie u siebie i jedynie kopiować je na serwer. To serwer przechowuje dane i w razie potrzeby udostępnia je klientowi.

Uwaga: Aby utrzymywać trwale dane zdefiniować zachowanie usługi jako Single, lub zdefiniować zmienną dla przechowywania danych jako statyczną.

Exercise 5

SOAP Web Services 2 (Java)

Author: Mariusz Fraś

1 Objectives of the exercise

The purpose of the exercise is:

1. Acquaintance with architecture of W3C SOAP Web Services.
2. Mastering the technique of creating services in Java with the use of XML Web Services API.

2 Development environment

The developing of applications can be performed using various platforms. Here is considered the following environment:

- Windows 7 (or higher) Operating System (eventually MacOS).
- Java Software Development Kit (JDK 11 or newer).
- Java application development platform (here: IntelliJ IDEA).

3 Exercise – Part I

Creating a W3C SOAP Web Service in Java

In the exercise will be implemented an application (SOAP W3C web service and client) using: XML-Based Web Services API (JAX-WS), in particular:

- Implementation of the W3C Web Service using the Bottom-Up method.
- Implementation of the W3C Web Service using the Top-Down method.
- Java client implementation using classes generated using Bottom-Up method.

The implementation of the Web Service using the Bottom-Up method consists in defining appropriate interfaces and classes for the service and then using appropriate tools to generate the target code.

The implementation of the service using the Top-Down method consists in defining the WSDL service description file (description of message formats, types, addresses, etc. defining all logical and technical aspects of the service) and on its basis generating the source code of interfaces and classes for building the application (both the service and client). Next implementing Service class with service operations code.

3.1 W3C Web Service (JAX-WS framework) - Bottom-Up method

3.1.1 Java Maven project and POM file

- Create the standard Maven project (here named jaxws1).
- Add in the POM file the section **dependences** and one **dependency** for using package **jaxws-rt** from **com.sun.xml.ws**. (containing the core Jakarta XML Web Services APIs).

This permits to use annotations to build necessary objects.

```
<?xml version="1.0" encoding="UTF-8"?>
<project ...>
  <groupId>org.example</groupId>
  <artifactId>jaxws1</artifactId>
  <version>1.0-SNAPSHOT</version>
  ...
  <dependencies>
    <dependency>
      <groupId>com.sun.xml.ws</groupId>
      <artifactId>jaxws-rt</artifactId>
      <version>3.0.2</version>
    </dependency>
  </dependencies>
</project>
```

Note:

*Originally, API components for implementing Web services were included in Java EE in **javax.xml.ws** package (jaxws-api artifact Id) – last ver. in 2018. Next it was moved*

*to Eclipse Foundation (EE4J – Eclipse Enterprise for Java) as a part of Eclipse Metro project – **jakarta.xml.ws** package. It is recommended to use reference implementation from sun (as in this exercise **jaxws-rt** runtime of **com.sun.xml.ws** package).*

- If necessary reload maven project and download used packages (in Maven tab on the right)

3.1.2 Interfaces and classes to support operation on data

Implement the following interfaces/classes (names are :

- Person – containing same personal data and getters/setters,
- PersonRepository – interface of data repository.
- PersonRepositoryImpl – data repository,
- PersonExistsEx – exception thrown when trying add person with the same ID,
- PersonNotFoundEx - exception thrown when trying get not existent person,
- Define the class for personal data:

```
public class Person {
    private int id;
    private String firstName;
    private int age;
    public Person() {
    }
    public Person(int id, String firstName, int age) {
        this.id = id;
        this.firstName = firstName;
        this.age = age;
    }
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    [...] //the rest of getters and setters (getFirstName(), getAge(), ...
}
```

- Define classes of exceptions. Note the classes are annotated with **@WebFault annotation** (imported from jakarta.xml.ws package).

```
import jakarta.xml.ws.WebFault;
@WebFault
public class PersonExistsEx extends Exception {
    public PersonExistsEx() {
        super("This person already exists");
    }
}
```

and:

```

import jakarta.xml.ws.WebFault;
@WebFault
public class PersonNotFoundEx extends Exception {
    public PersonNotFoundEx() {
        super("The specified person does not exist");
    }
}

```

- Define the data repository interface. The interface specifies typical CRUD operations.

```

public interface PersonRepository {
    List<Person> getAllPersons();
    Person getPerson(int id) throws PersonNotFoundEx;
    Person updatePerson(int id, String name, int age) throws
        PersonNotFoundEx;
    boolean deletePerson(int id) throws PersonNotFoundEx;
    Person addPerson(int id, String name, int age) throws PersonExistsEx;
    int countPersons();
}

```

Define the class implementing interface. Initialize the repository with some data.

```

public class PersonRepositoryImpl implements PersonRepository {
    private List<Person> personList;
    public PersonRepositoryImpl() {
        personList = new ArrayList<>();
        personList.add(new Person(1, "Mariusz", 9));
        personList.add(new Person(2, "Andrzej", 10));
        personList.add(new Person(3, "Tomasz", 11));
    }
    public List<Person> getAllPersons() {
        return personList;
    }
    public Person getPerson(int id) throws PersonNotFoundEx {
        for (Person thePerson : personList) {
            if (thePerson.getId() == id) {
                return thePerson;
            }
        }
        throw new PersonNotFoundEx();
    }
    public Person updatePerson(int id, String name, int age) throws
        PersonNotFoundEx {
        for (Person thePerson : personList) {
            if (thePerson.getId() == id) {
                thePerson.setFirstName(name);
                thePerson.setAge(age);
                return thePerson;
            }
        }
        throw new PersonNotFoundEx();
    }
}

```

```

public boolean deletePerson(int id) throws PersonNotFoundEx {
    for (Person thePerson : personList) {
        if (thePerson.getId() == id) {
            personList.remove(thePerson);
            return true;
        }
    }
    throw new PersonNotFoundEx();
}

public Person addPerson(int id, String name, int age) throws
    PersonExistsEx {
    for (Person thePerson : personList) {
        if (thePerson.getId() == id) {
            throw new PersonExistsEx();
        }
    }
    Person person = new Person(id, name, age);
    personList.add(person);
    return person;
}

public int countPersons() {
    return personList.size();
}
}

```

3.1.3 Web Service

Implement the interface and class of the web Service.

- Define the interface of the service.

Annotate the interface and methods with **@WebService** and **@WebMethod** annotations.

```

@WebService
public interface PersonService {
    @WebMethod
    Person getPerson(int id) throws PersonNotFoundEx;
    @WebMethod
    Person updatePerson(int id, String name, int age) throws
        PersonNotFoundEx;
    @WebMethod
    boolean deletePerson(int id) throws PersonNotFoundEx;
    @WebMethod
    Person addPerson(int id, String name, int age) throws PersonExistsEx;
    @WebMethod
    int countPersons();
    @WebMethod
    List<Person> getAllPersons();
}

```

- Define the class implementing the interface.
 - In the **@WebService** annotation include **serviceName** and **endpointInterface** attributes.
 - In the class include the initialized data repository.
 - Complete the methods with calls of repository methods and printing comments similarly as for first two methods.

```

@WebService(serviceName = "PersonService",
            endpointInterface = "org.example.jaxws.server.PersonService")
public class PersonServiceImpl implements PersonService {
    private PersonRepository dataRepository = new DataRepository();

    @WebMethod
    public Person getPerson(int id) throws PersonNotFoundEx {
        System.out.println("...called getPerson id=" + id);
        return dataRepository.getPerson(id);
    }

    @WebMethod
    public Person updatePerson(int id, String name, int age) throws
        PersonNotFoundEx {
        System.out.println("...called updatePerson");
        return dataRepository.updatePerson(id, name, age);
    }

    @WebMethod
    public boolean deletePerson(int id) throws PersonNotFoundEx {
        [...]
    }

    @WebMethod
    public Person addPerson(int id, String name, int age) throws
        PersonExistsEx {
        [...]
    }

    @WebMethod
    public int countPersons() {
        [...]
    }

    @WebMethod
    public List<Person> getAllPersons() {
        [...]
    }
}

```

3.1.4 Hosting the service (main class)

To run the service the service implementation object is created and it is “published” with use of **jakarta.xml.ws.Endpoint**.

- Define the main class of the project that hosts and run the Web Service (here named **ServiceHost**).
 - Use in the service endpoint (address) a TCP port and a name for this endpoint.

```

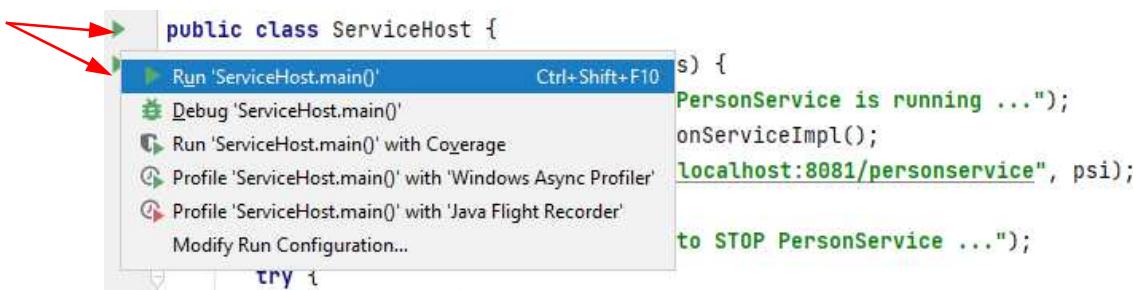
public class ServiceHost {
    public static void main(String[] args) {
        System.out.println("Web Service PersonService is running ...");
        PersonServiceImpl psi = new PersonServiceImpl();
        Endpoint.publish("http://localhost:8081/personservice", psi);
        System.out.println("Press ENTER to STOP PersonService ...");
        try {
            System.in.read();
        } catch (IOException e) {
            e.printStackTrace();
        }
        exit(0);
    }
}

```

3.1.5 Running and testing the service

- Compile and run the application.

The best way is to run it using context menu from vertical bar.



- Run the browser and enter in the address used for endpoint.
 - Check if in the browser the WSDL data are displayed – if yes it means the service was started properly.

Endpoint	Information
Service Name: {http://server.jaxws.example.org/}PersonService Port Name: {http://server.jaxws.example.org/}PersonServiceImplPort	Address: http://localhost:8081/personservice WSDL: http://localhost:8081/personservice?wsdl Implementation class: org.example.jaxws.server.PersonServiceImpl

- Open the link of the WSDL data (here: <http://localhost:8081/personservice?wsdl>) and verify the content.

Note: below in the figure most nodes are not unfolded.

```

<definitions targetNamespace="http://server.jaxws.example.org/" name="PersonService">
  <types>
    <xsd:schema>
      <xsd:import namespace="http://server.jaxws.example.org/" schemaLocation="http://localhost:8081/personservice?xsd=1"/>
    </xsd:schema>
  </types>
  <message name="getAllPersons"></message>
  <message name="getAllPersonsResponse"></message>
  <message name="countPersons"></message>
  <message name="countPersonsResponse"></message>
  <message name="deletePerson"></message>
  <message name="deletePersonResponse"></message>
  <message name="PersonNotFoundEx"></message>
  <message name="getPerson"></message>
  <message name="getPersonResponse"></message>
  <message name="updatePerson"></message>
  <message name="updatePersonResponse"></message>
  <message name="addPerson"></message>
  <message name="addPersonResponse"></message>
  <message name="PersonExistsEx"></message>
  <portType name="PersonService">
    <operation name="getAllPersons"></operation>
    <operation name="countPersons"></operation>
    <operation name="deletePerson"></operation>
    <operation name="getPerson"></operation>
    <operation name="updatePerson"></operation>
    <operation name="addPerson"></operation>
  </portType>
  <binding name="PersonServiceImplPortBinding" type="tns:PersonService">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
    <operation name="getAllPersons"></operation>
    <operation name="countPersons"></operation>
    <operation name="deletePerson"></operation>
    <operation name="getPerson"></operation>
    <operation name="updatePerson"></operation>
    <operation name="addPerson"></operation>
  </binding>
  <service name="PersonService">
    <port name="PersonServiceImplPort" binding="tns:PersonServiceImplPortBinding">
      <soap:address location="http://localhost:8081/personservice"/>
    </port>
  </service>

```

- Next open the link of the schema contained in the WSDL data (here: <http://localhost:8081/personservice?xsd=1>).

```

<xs:schema version="1.0" targetNamespace="http://server.jaxws.example.org/">
  <xs:element name="PersonExistsEx" type="tns:PersonExistsEx"/>
  <xs:element name="PersonNotFoundEx" type="tns:PersonNotFoundEx"/>
  <xs:element name="addPerson" type="tns:addPerson"/>
  <xs:element name="addPersonResponse" type="tns:addPersonResponse"/>
  <xs:element name="countPersons" type="tns:countPersons"/>
  <xs:element name="countPersonsResponse" type="tns:countPersonsResponse"/>
  <xs:element name="deletePerson" type="tns:deletePerson"/>
  <xs:element name="deletePersonResponse" type="tns:deletePersonResponse"/>
  <xs:element name="getAllPersons" type="tns:getAllPersons"/>
  <xs:element name="getAllPersonsResponse" type="tns:getAllPersonsResponse"/>
  <xs:element name="getPerson" type="tns:getPerson"/>
  <xs:element name="getPersonResponse" type="tns:getPersonResponse"/>
  <xs:element name="updatePerson" type="tns:updatePerson"/>
  <xs:element name="updatePersonResponse" type="tns:updatePersonResponse"/>
  +<xs:complexType name="deletePerson"></xs:complexType>
  +<xs:complexType name="deletePersonResponse"></xs:complexType>
  +<xs:complexType name="PersonNotFoundEx"></xs:complexType>
  -<xs:complexType name="getPerson">
    <xs:sequence>
      <xs:element name="arg0" type="xs:int"/>
    </xs:sequence>
  </xs:complexType>
  +<xs:complexType name="getPersonResponse"></xs:complexType>
  +<xs:complexType name="person"></xs:complexType>

```

*Note: Above in the figure only one type (**getPerson** – composed of one integer(input parameter for **getPerson** message/method)) is unfolded.*

- Run the Postman app (or SOAP UI) and compose the **POST message** with the following settings
 - The **address** of the service (as implemented for the endpoint).
 - The **Content-type** header equal **text/xml**.
 - The **SOAPAction** header – should be equal to SOAP Action attribute of the message in WSDL but in this case it not works so can be set to anything.

KEY	VALUE	Bulk Edit	Presets
<input checked="" type="checkbox"/> Content-Type	text/xml		
<input checked="" type="checkbox"/> SOAPAction	"http://server.jaxws.example.org/PersonService/getAllPersons"		
Key	Value		

- In the body enter the SOAP xml message to call **getAllPersons** method (the Header node can be omitted):

```
<?xml version="1.0" encoding="utf-8"?>
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
  <s:Body>
    <getAllPersons xmlns="http://server.jaxws.example.org/">
      </getAllPersons>
    </s:Body>
  </s:Envelope>
```

As the result you should receive the list of person data.

```
<?xml version='1.0' encoding='UTF-8'?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Body>
    <ns2:getAllPersonsResponse xmlns:ns2=...>
      <return>
        <age>9</age>
        <firstName>Mariusz</firstName>
        <id>1</id>
      </return>
      <return>
        <age>10</age>
        <firstName>Andrzej</firstName>
        <id>2</id>
      </return>
      <return>
        <age>11</age>
        <firstName>Tomasz</firstName>
        <id>3</id>
      </return>
    </ns2:getAllPersonsResponse>
  </S:Body>
</S:Envelope>
```

3.2 Web Service JAX-WS client

The first step of creating Web Service JAX-WS client is the first step creating Web Service using **Top-Down method**.

3.2.1 WSDL file

The WSDL file will be used to generate interfaces and classes used to build Web Service and Web Service client.

- Add to the project the file (here named **personservice.wsdl**) in which description of service will be included – e.g. in newly created *resource* folder in *src/main/java* folder.
- Run the service created in section 3.1. Open the browser and open the link of the WSDL data.
 - Copy the whole content (you can omit comments) to the personservice.wsdl file.
 - Open the XSD schema data – the link is in the beginning of WSDL data in the **<import .../>** element inside **<types>** element.
 - Copy the whole content of schema data (you can omit comments) in place of **<types>** section in personservice.wsdl file.
 - Remove all **wsam:Action** attributes (only these attributes and its values) inside **<portType>** section.
 - Correct the **<definitions>** tag to make all data consistent:
 - the Schema, SOAP, and general WSDL namespaces.

Here:

```
<definitions targetNamespace="http://server.jaxws.example.org/">
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:tns="http://server.jaxws.example.org/"
    xmlns="http://schemas.xmlsoap.org/wsdl/">
    <name>PersonService</name>
```

3.2.2 POM file

In the POM file the plugin that uses **wsimport** tool and generates necessary interfaces and classes must be added.

- Open POM file and enter behind **<dependences>** the **<build>** section and inside **com.sun.xml.ws/jaxws-maven-plugin** plugin with **wsimport goal** and proper configuration.

```
<build>
    <plugins>
        <plugin>
            <groupId>com.sun.xml.ws</groupId>
            <artifactId>jaxws-maven-plugin</artifactId>
            <version>3.0.2</version>
```

```

<executions>
  <execution>
    <goals>
      <goal>wsimport</goal>
    </goals>
  </execution>
</executions>
<configuration>
  <wsdlFiles>
    <wsdlFile>
      ${project.basedir}/src/main/resources/personservice.wsdl
    </wsdlFile>
  </wsdlFiles>
  <packageName>org.example.jaxws.server_topdown</packageName>
  <sourceDestDir>${project.basedir}/src/main/java/</sourceDestDir>
</configuration>
</plugin>
</plugins>
</build>

```

The classes will be generated in **org.example.jaxws.server_topdown** package on the basis of the WSDL file specified in **<wsdlFile>** element., in **<sourceDestDir>** folder.

- Compile the project and verify project structure and generated classes.

3.2.3 The Web Service Client

The client uses generated **ServiceName_Service** class (where **ServiceName** is the name of defined service interface) to get proxy client used to call service operations.

- Create the new class (here in the separate client package) for the client. In the code:
 - define URL of the service,
 - create PersonService_Service object,
 - get client proxy from PersonService_Service object,
 - use client proxy to call Web Service operations.

```

package org.example.jaxws.client;
import [...]
public class ESClient {
  public static void main(String[] args) throws MalformedURLException,
                                                PersonNotFoundException
  {
    int num = -1;
    URL addr = new URL("http://localhost:8081/personservice?wsdl");
    PersonService_Service pService = new PersonService_Service();
    PersonService pServiceProxy = pService.getPersonServiceImplPort();
    num = pServiceProxy.countPersons();
    System.out.println("Num of Persons = " + num);
  }
}

```

```
Person person = pServiceProxy.getPerson(2);
System.out.println("Person "+person.getFirstName()+"",
                   id = "+person.getId());
}
}
```

3.3 W3C Web service – Top-Down method

3.3.1 WSDL file

The first step of creating Web Service using **Top-Down method** is to create WSDL file. It is used to generate necessary interfaces and classes as described in previous section (client section).

Here the WSDL file already created for the client will be used.

3.3.2 POM file

The plugin necessary for building service interfaces and classes was already described in previous section (client section).

3.3.3 The service implementation

The generated classes defines only service interfaces, basic data structures, messages, etc. The service logic (implementation) must be defined similarly as for Bottom-Up method. With proper organization of packages in this exercise it could be the same classes. However, it was used separate **org.example.jaxws.server_topdown** package to present build the service in different way.

Here will be created in this package: Repository interface and class and service implementation class (here named **PersonService2Impl**).

- Copy **PersonRepository** class to the package. You may rename it to **PersonRepository2** (to distinct them).
 - Correct the code to use classes only from **org.example.jaxws.server_topdown** package.
 - Use **PersonNotFoundEx_Exception** and **PersonExistsEx_Exception** instead previous ones (because we use only classes generated from the WSDL file).
- Copy **PersonRepositoryImpl** class to the package. You may rename it to **PersonRepository2Impl** (to distinct them).
 - Do the same corrections as a moment before (to use classes from**server_topdown** package only).
 - You may remove annotations as they are not used now.

3.3.4 Service host

The service host (here named **ServiceHost2**) is almost the same as previously. The only difference is to use classes from **org.example.jaxws.server_topdown** package – precisely **PersonService2Impl**.

3.3.5 Running and testing the service

- Compile and run the application.

Be aware that if you have not defined other service address (use the same address as first service) you can't run both at the same time.

- Run the client to test operation.

As the client uses the same classes it may be used to test the service.

Be aware of service address if you used the other one than previously.

4 Exercise – Part II

The detailed and final requirements for Part II are set by the teacher.

- A. Develop or prepare to develop a program according to the teacher's instructions.

Ćwiczenie 6-A

Usługi typu REST

Autor: Mariusz Fraś

1 Cele ćwiczenia

Celem ćwiczenia jest:

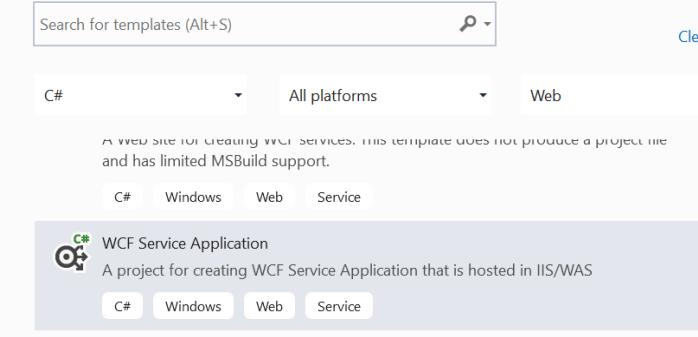
1. Opanowanie techniki realizacji serwisu hostowanego przez serwer www IIS.
2. Opanowanie techniki realizacji serwisów webowych typu REST.

Zadanie polega na:

- Zrealizowanie usługi typu REST hostowanej przez serwer WWW (w tym przypadku użyte będzie uproszczone oprogramowanie, **IIS Express** jako host usługi.).
 - usługa będzie obsługiwać żądania HTTP (metody GET, POST, ...) odwołujących się do zasobów z użyciem konkretnych URL-i.
 - w ćwiczeniu usługa operuje na danych w formatach XML i JSON.
- Zrealizowanie klienta konsolowego umożliwiającego wysłanie żądania odwołującego się do konkretnego zasobu.

2 Zadanie – część I

Usługa typu REST hostowana przez serwer www.

<p>1. Definiowanie usługi WCF w stylu REST</p>	<p>Utwórz aplikację hostowaną w serwerze IIS.</p> <ul style="list-style-type: none"> Utwórz nowąację i projekt aplikacji typu WCF Service Application nadając jej własną nazwę – opcja: Add... → New Project → Odpowiednia opcja. <p>Create a new project</p>  <p>Recent project templates</p> <ul style="list-style-type: none"> WCF Service Application C# Console App (.NET Framework) C# WCF Service Library C# <p>WCF Service Application A project for creating WCF Service Application that is hosted in IIS/WAS</p> <p>Solution Explorer</p> <ul style="list-style-type: none"> MyWebService <ul style="list-style-type: none"> Connected Services Properties References App_Data IService1.cs Service1.svc Web.config Web.Debug.config Web.Release.config <ul style="list-style-type: none"> Przejrzyj zawartość projektu. Zwróć uwagę na następujący element: <ul style="list-style-type: none"> Web.config – konfiguracja niezbędna do załadowania kontraktu poprzez host serwisu WCF (tu: serwis www). Service1.svc – węzeł z kodem implementacji usługi. Przejrzyj plik Web.config. Zwróć uwagę na brak węzła <Service> - jest on opcjonalny. W pliku IService1.cs i zdefiniuj interfejs (kontrakt) usługi IRestService oraz kontrakt danych (w tym celu można sfaktoryzować istniejący kod).
--	---

- Zdefiniuj kontrakt danych (*DataContract*) – klasę opisującą jakiś zestaw danych, zawierający co najmniej 3 pola, w tym identyfikator (np. ID elementu, nazwa elementu, jakaś wartość elementu, itp.):

```
[DataContract]
public class contract_type {
    [DataMember(Order = 0)]
    public int X1 { get; set; }
    [DataMember(Order = 1)]
    public string X2 { get; set; }
    [DataMember(Order = 2)]
    public double X3 { get; set; }
}
```

gdzie **contract_type** to Twoja nazwa klasy kontraktu (np. Item Book, Car, itp.), **X1**, **X2**,... to nazwy pól kontraktu danych (nadane w definicji kontraktu własne nazwy).

Dla tworzonego później konstruktora, pola muszą być **public**.

- Zdefiniuj kontrakt serwisu **IRestService** z metodami:

- metoda **getAllXml**
 - zwraca listę elementów (zdefiniowanych przez kontrakt)
 - ma atrybut typu **WebGet** – realizuje pobieranie danych przez klienta metodą **GET**,
 - nie ma specyfikacji formatu danych – używa domyślnego **xml**.
- metoda **getByIdXml**
 - zwraca jeden element o identyfikatorze **Id**
 - ma atrybut typu **WebGet** oraz
 - specyfikuje format zwracanych danych **xml**,
- metoda **addXml**
 - dodaje element o identyfikatorze **Id**
 - ma atrybut typu **WebInvoke** i specyfikowaną metodę **POST**,
 - specyfikuje przekazywanie danych w formacie **xml**,
 - zwraca string opisujący wynik operacji.
- metoda **deleteXml**
 - usuwa element o podanym identyfikatorze metodą **DELETE**.

```
public interface IRestService
{
    [OperationContract]
    [WebGet(UriTemplate = "/Xxx")]
    List<contract_type> getAllXml();

    [OperationContract]
    [WebGet(UriTemplate = "/Xxx/{id}",
        ResponseFormat = WebMessageFormat.Xml)]
    typ_kontraktu getByIdXml(string Id);

    [OperationContract]
    [WebInvoke(UriTemplate = "/Xxx",
        Method = "POST",
        RequestFormat = WebMessageFormat.Xml)]
    string addXml(contract_type item);
```

```

[OperationContract]
[WebInvoke(UriTemplate = "/Xxx{id}", Method = "DELETE")]
    string deleteXml(string Id);
}

```

gdzie: **contract_type** – to typ (klasa) kontraktu danych,
Xxx – to własna nazwa folderu (zasobnika danych) dla zasobów – np.
items, *books* dla książek, *cars* dla samochodów, itp. Ponadto:

- w metodach specyfikowany jest szablon **Uri** – relatywna (do adresu bazowego usługi) ścieżka URL do zasobu (jako string).
 - w ścieżce możliwa jest specyfikacja zmiennej części stringu (różna w różnych wywołaniach), która jest podawana w nawiasach {...}.
 - W pliku **Service1.svc.cs** wpisz kod klasy **MyRestService** implementującej interfejs **IRestService**:
- W klasie ma być:
- Specyfikacja jednej instancji serwisu dla wszystkich wywołań:
- ```
[ServiceBehavior(InstanceContextMode=InstanceContextMode.Single)]
```
- Zadeklarowana lista elementów – globalna zmienna dla klasy:
- ```
private static List<contract_type> Yyy;
```
- gdzie: **Yyy** to nazwa listy, **contract_type** to typ kontraktu danych.
- Można dodać konstruktor, a w nim dodać do listy trzy elementy:
- ```

Yyy = new List<contract_type>() {
 new contract_type {x1=...,x2=...,x3=...},
 new contract_type {x1=...,x2=...,x3=...},
 new contract_type {x1=...,x2=...,x3=...},
}

```
- gdzie **x1**, **x2** i **x3** to nazwy pól kontraktu danych (nadane w definicji kontraktu własne nazwy). W miejsce kropek wpisz odpowiednie wartości. Np. {Id=123, Price=9,2,...}. (o **x1** patrz *Uwagi* poniżej)
- Metoda **getAll()** , która zwraca wszystkie elementy listy **Yyy**.
- ```
public List<contract_type> getAllXml() {
    return Yyy;
}
```
- Metoda **getByIdXml(string Id)** , która zwraca jeden element typu **contract_type** o identyfikatorze **Id**.
- ```

public contract_type getByIdXml(string Id)
{
 int intId = int.Parse(Id);
 int idx = Yyy.FindIndex(b => b.Id == intId);
 if (idx == -1)
 throw new WebFaultException<string>("404: Not Found",
 HttpStatusCode.NotFound);
 return Yyy.ElementAt(idx);
}

```

- Metoda `addXml(contract_type item)`, która dodaje do listy element o identyfikatorze **Id** przekazywany w zmiennej **item**.  
*(Uwaga: tu zakładamy, że serwis przypisuje nowe ID).*  
 W metodzie należy (podobnie jak dla `getByIdXml(...)`):  
  - sprawdzić czy **item** nie jest null,
  - sprawdzić, czy nie istnieje już element o takim samym ID – jeśli nie to dodać dane, jeśli tak to zwrócić odpowiedni kod błędu,

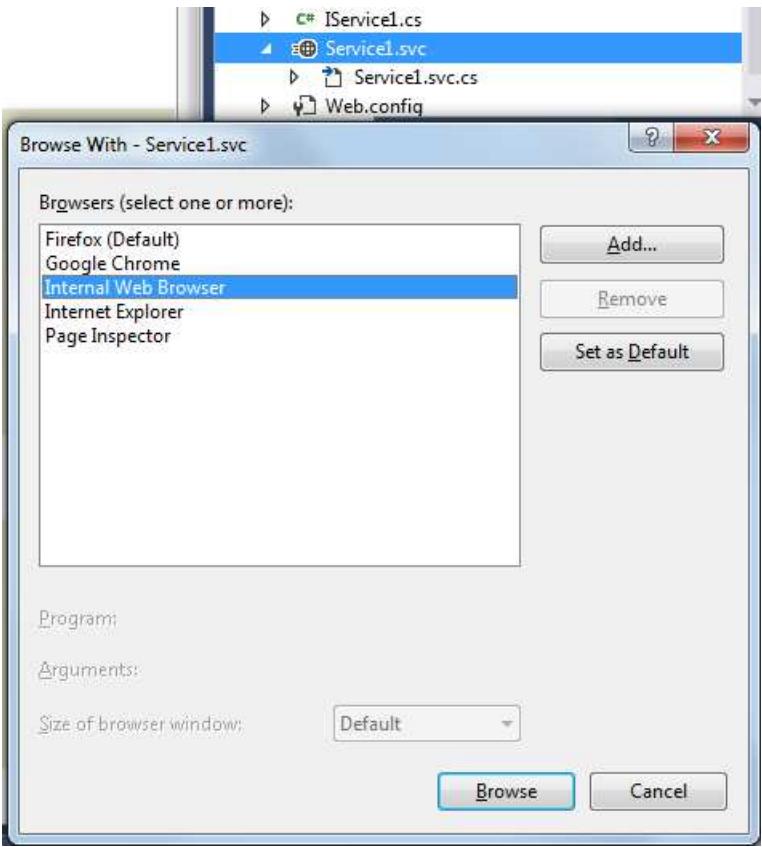
```
public string addXml(contract_type item) {
 if (element == null)
 throw new WebFaultException<string>("400: BadRequest",
 HttpStatusCode.BadRequest);
 [dobierz nowy indeks (identyfikator) - newIdx]
 int idx = Yyy.FindIndex(b => b.Id == newIdx);
 if (idx == -1) {
 item.Id = newIdx;
 Yyy.Add(item);
 return "Added item with ID=" + item.Id;
 } else
 throw new WebFaultException<string>("409: Conflict",
 HttpStatusCode.Conflict);
}
```
- Metoda `string deleteXml(string Id)`, która usuwa element o identyfikatorze **Id**.  
 Metoda jest niemal identyczna jak `getByIdXml(string Id)`  
 tylko:  
  - metoda zwraca `string`,
  - zamiast:  
`return Yyy.ElementAt(idx);`  
 należy wywołać:  
`Yyy.RemoveAt(idx);`  
`return "Removed item with ID=" + Id;`

### ***Uwagi:***

- parametry trzeba przekształtać na odpowiedni typ,
- tu założono, że pierwsze pole kontraktu danych **x1** to unikalny identyfikator – zamiast **x1** użyj własnej nazwy (tu użyto **Id**).
- instrukcja `Find(...)` wyszukuje dla danej listy element spełniający podany w nawiasach warunek (w takiej postaci jak w przykładzie)  
 – **b** oznacza odwołanie do elementu listy (można użyć dowolnego oznaczenia zamiast **b**) – tu szukamy elementu listy **Yyy** dla którego pierwsze pole ma wartość identyfikatora **Id**..
- tu: **Id** równe **-1** oznacza, że nie znaleziono elementu,

|  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|--|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|  | <p>Ustawianie i wysyłanie odpowiedzi z odpowiednimi nagłówkami.</p> <p>- skonfigurowanie nagłówków odpowiedzi (w tym kodu odpowiedzi) jest możliwe za pomocą obiektu reprezentującego odpowiedź, uzyskanego z kontekstu operacji:</p> <pre>OutgoingWebResponseContext response; //global variable response = WebOperationContext.Current.OutgoingResponse; response.StatusCode = HttpStatusCode.some_code;</pre> <p>gdzie <b>some_code</b> to ustawiany jakiś kod HTTP.</p> <p>Zmienna <b>response</b> powinna być globalna w klasie.</p> <p>- w przypadku wystąpienia błędu, który nie pozwala przekazać danych można użyć metody <b>WebFaultException&lt;T&gt;(...)</b>, która powoduje zwrot odpowiedniego kodu błędu HTTP oraz krótki opis błędu (podawanych w parametrach).</p> <ul style="list-style-type: none"><li>• Otwórz plik <b>Web.config</b> i zmodyfikuj w celu obsługi żądań REST:<ul style="list-style-type: none"><li>○ W sekcji <b>&lt;system.serviceModel&gt;</b> dopisz opis serwisu i endpointów:<pre>&lt;services&gt;   &lt;service name="MyWebSerwis.RestService"&gt;     &lt;endpoint address=""       binding="webHttpBinding"       contract="MyWebSerwis.IRestService"       behaviorConfiguration="myRESTEndpointBehavior"&gt;       &lt;/endpoint&gt;     &lt;/service&gt;   &lt;/services&gt;</pre>gdzie: <b>MyWebSerwis</b> jest nazwą użytą jako przestrzeń nazw (<i>namespace</i>) aplikacji.<ul style="list-style-type: none"><li>● Wiązanie musi być typu <b>webHttpBinding</b>.</li><li>● Serwis i endpoint jest opisany w zachowaniach.</li><li>● Nazwa konfiguracji zachowania <b>behaviorConfiguration</b> jest opcjonalna – jeśli ma być użyta to musi być zdefiniowana.</li></ul></li><li>○ W sekcji <b>&lt;behaviors&gt;</b> dopisz zachowanie endpointu <b>webHttp</b> pozwalające pobrać opis operacji w przeglądarce:<pre>&lt;endpointBehaviors&gt;   &lt;behavior name="myRESTEndpointBehavior"&gt;     &lt;webHttp helpEnabled="true" /&gt;   &lt;/behavior&gt; &lt;/endpointBehaviors&gt;</pre></li><li>○ Upewnij się że jest ustawione generowanie metadanych o usłudze (w sekcji <b>&lt;behaviors&gt;</b> w <b>&lt;serviceBehaviors&gt;</b>):<br/><b>serviceMetadata httpGetEnabled="true"</b></li></ul></li></ul> |
|--|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

|                                                                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|--------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p>2. Supporting CORS mechanism<br/><i>(tymczasowo po angielsku)</i></p> | <p>In order to be able to execute operations by sending requests from Web browsers (AJAX approach), the service must support the CORS mechanism (for <i>preflighted requests</i>), since most AJAX requests by definition are not allowed in the browser without the control of this mechanism.</p> <p>In WCF applications, support for preflighted requests can be done by adding an additional Global.asax class to the project - a class that contains the code of methods called when handling requests to the service.</p> <ul style="list-style-type: none"> <li>• Add a class with the option:<br/><i>Add → New Item → Web\General\Global application class</i></li> <li>• Add the code for the <b>Application_BeginRequest</b> method, in which we allow requests for appropriate HTTP methods and headers from each location (e.g. for a period of 2 hours):</li> </ul> <pre><code>public class Global : System.Web.HttpApplication {     protected void Application_Start(object sender, EventArgs e) {}     protected void Session_Start(object sender, EventArgs e) {}     protected void Application_BeginRequest(object sender,                                             EventArgs e) {         HttpContext.Current.Response.AddHeader(             "Access-Control-Allow-Origin",             "*");         if (HttpContext.Current.Request.HttpMethod == "OPTIONS") {             HttpContext.Current.Response.AddHeader(                 "Access-Control-Allow-Methods",                 "POST,                 PUT, DELETE");             HttpContext.Current.Response.AddHeader(                 "Access-Control-Allow-Headers",                 "Content-Type,                 Accept");             HttpContext.Current.Response.AddHeader(                 "Access-Control-Max-Age", "7200");             HttpContext.Current.Response.End();         }     }     protected void Application_AuthenticateRequest(object sender,                                                 EventArgs e) {}     protected void Application_Error(object sender, EventArgs e) {}     protected void Session_End(object sender, EventArgs e) {}     protected void Application_End(object sender, EventArgs e) {} }</code></pre> |
|--------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

|                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|-------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p>3. Uruchomienie usługi</p> | <ul style="list-style-type: none"> <li>Skompiluj projekt.</li> <li>Kliknij w Solution Explorer prawym klawiszem węzeł z kodem usługi (np. <b>Service1.svc.cs</b>) i wybierz opcję <b>Browse with....</b> i wybierz jedną z możliwych opcji uruchomienia (przeglądarkę).</li> <li></li> </ul>  <ul style="list-style-type: none"> <li>Poprzez ikonę IIS Express na pasku zadań (<i>Trylcon</i>) sprawdź uruchomienie usługi i informacje o niej       <ul style="list-style-type: none"> <li>Zapamiętaj adres URL do usługi.</li> </ul> </li> </ul> |
| <p>4. Testowanie usługi</p>   | <p>Sprawdź działanie usługi za pomocą uruchomionej przeglądarki</p> <ul style="list-style-type: none"> <li>Wyświetl opis dostępnych operacji serwisu: w pasku adresu przeglądarki do adresu opisu usługi <b>Service1.svc</b> dopisz element ścieżki: <b>/help</b>.<br/>Zapoznaj się z podanymi informacjami.</li> <li>Wywołaj operację wyświetlenia wszystkich elementów: podaj adres operacji <b>getAllXml</b> (tu: dopisz w pasku adresu przeglądarki do adresu usługi <b>Service1.svc</b> element ścieżki: <b>/XXX</b>).<br/>Zwróć uwagę na format wyświetlanych danych.</li> </ul>                                                |

|                                                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                                                | <ul style="list-style-type: none"> <li>Wywołaj operację wyświetlenia jednego elementu:<br/>podaj adres operacji <b>getByIdXml</b> (tu: dopisz w pasku adresu przeglądarki do adresu usługi <b>Service1.svc</b> element ścieżki: <b>/Xxx/num</b>, gdzie <b>num</b> – wartość identyfikatora danego elementu).<br/>Zwróć uwagę na format wyświetlanych danych.</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 5. Dopisanie operacji używających formatu json | <p><b>Uwaga:</b> teraz metody będą dla dwóch rodzajów reprezentacji danych (<b>xml</b> i nowe dla <b>json</b>) więc adresy operacji usługi dodatkowo będą miały część specyfikującą te reprezentacje (yu człon <b>json</b>). <u>Jeśli stosowany jest jeden format taki dodatkowy człon w adresie jest niepotrzebny!</u></p> <ul style="list-style-type: none"> <li>Otwórz plik <b>IService1.cs</b> i dopisz w interfejsie usługi metody operacji na zasobach dla json: <ul style="list-style-type: none"> <li>metoda <b>getAllJson</b><br/>- identycznie jak dla <b>getAllXml</b> ale w kontrakcie wyspecyfikować adres (Uri) "<b>/json/Xxx</b>" zamiast "<b>/Xxx</b>", oraz format danych Json:<br/><code>... = WebMessageFormat.Json</code></li> <li>metoda <b>getByIdJson</b><br/>- identycznie jak dla <b>getByIdXml</b> ale w kontrakcie wyspecyfikować format Json i adres (Uri) "<b>/json/Xxx/{id}</b>" zamiast "<b>/Xxx/{id}</b>".</li> <li>metoda <b>addJson</b><br/>- identycznie jak dla <b>addXml</b> ale w kontrakcie wyspecyfikować format Json i adres (Uri) "<b>/json/Xxx</b>" zamiast "<b>/Xxx</b>".</li> <li>metoda <b>deleteJson</b><br/>- analogiczne zmiany jak powyżej.</li> </ul> </li> <li>Otwórz plik <b>Service1.svc.cs</b>. Dopusz kod metod dla formatu json: <ul style="list-style-type: none"> <li>Metody są identyczne jak dla xml – mają tylko inne nazwy:<br/><code>getAllJson(),<br/>getByIdJson(string Id),<br/>addJson(contract_type item),<br/>deleteJson(string Id).</code></li> </ul> Można też po prostu wywołać z nich odpowiednie metody dla xml.</li> </ul> |
| 6. Uruchomienie i testowanie usługi            | <ul style="list-style-type: none"> <li>Skompiluj projekt.</li> <li>Uruchom usługę za pomocą opcji <b>Browse with....</b></li> <li>Sprawdź działanie metod <b>get...</b> za pomocą przeglądarki.</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 7. Testowanie działania aplikacji              | <p><b>Testowanie działania aplikacji za pomocą narzędzia Postman</b></p> <ul style="list-style-type: none"> <li>Jeśli w systemie nie ma narzędzia Postman to ściągnij je i zainstaluj. Adres witryny: <a href="https://www.getpostman.com">https://www.getpostman.com</a></li> <li>Przygotuj odpowiednie dane xml i json dla operacji addXml i addJson. Można kierować się informacją zawartą na stronie help opisu operacji serwisu.</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |

|                                                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                                                            | <ul style="list-style-type: none"> <li>Przetestuj działanie aplikacji wysyłając żądania GET, POST, DELETE.           <ul style="list-style-type: none"> <li>Żądania w aplikacji tworzy się opcją New/Request</li> <li>W żądaniach wypełnij odpowiednio pole adresu.</li> <li>Dla żądań wysyłających dane (np. PUT i POST) trzeba także wyspecyfikować:               <ul style="list-style-type: none"> <li>w nagłówkach (<b>Headers</b>), nagłówek: <b>Content-type</b> o odpowiedniej wartości dla json (application/json) i dla xml (text/xml),</li> <li>dane należy umieścić w <b>body</b> żądania; najlepiej wybrać zakładkę Body typu <b>raw</b> i adekwatnie ustawić dane typu JSON albo XML; poprawną zawartość body dla XML wpisać wg wskazówek na stronie help operacji serwisu.</li> </ul> </li> <li>Pozostałe pola można zostawić nietknięte.</li> </ul> </li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <i>Poniżej opcjonalna wersja klienta (nieobowiązkowa):</i> |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 8. Tworzenie konsolowej aplikacji klienckiej               | <p>Utworzenie klienta wywołującego zaimplementowane usługi.</p> <ul style="list-style-type: none"> <li>Dodaj do solucji nowy projekt aplikacji klienta z szablonu <b>Visual C# Console Application</b> nadając jej własną nazwę.</li> <li><b>Uwaga:</b> w tym przypadku nie trzeba dodawać referencji serwisowej.</li> <li>Otwórz plik <b>Program.cs</b> i dopisz następujący kod:</li> </ul> <pre>static void Main(string[] args) {     do {         try {             Console.WriteLine("Podaj format (xml lub json):");             string format = Console.ReadLine();             Console.WriteLine("Podaj metode (GET lub POST lub ...):");             string method = Console.ReadLine();             Console.WriteLine("Podaj URI:");             string uri = Console.ReadLine();             HttpWebRequest req = WebRequest.Create(uri) as                 HttpWebRequest;             req.KeepAlive = false;             req.Method = method.ToUpper();             if (format == "xml")                 req.ContentType = "text/xml";             else if (format == "json")                 req.ContentType = "application/json";             else {                 Console.WriteLine("Podales zle dane!");                 return;             }             switch(method.ToUpper()) {                 case "GET":                     break;                 case "POST":                     break;                 case "PUT":                     break;                 case "DELETE":                     break;                 default:                     Console.WriteLine("Metoda nieznana!");                     return;             }             if (req != null)                 req.GetResponse();         } catch (Exception ex) {             Console.WriteLine(ex.Message);         }     } }</pre> |

|  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|--|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|  | <pre> // cont. on the next page case "POST":     Console.WriteLine("Wklej zawartosc XML-a lub                       JSON-a (w jednej linii !)");     string dane = Console.ReadLine();     //przekodowanie tekstu wiadomosci:     byte[] bufor = Encoding.UTF8.GetBytes(dane);     req.ContentLength = bufor.Length;     Stream postData = req.GetRequestStream();     postData.Write(bufor, 0, bufor.Length);     postData.Close();     break; //tu ewentualnie kolejne opcje default:     break; } HttpWebResponse resp = req.GetResponse()                         as HttpWebResponse; //przekodowanie tekstu odpowiedzi: Encoding enc = System.Text.Encoding.GetEncoding(1252); StreamReader responseStream =     new StreamReader(resp.GetResponseStream(), enc); string responseString = responseStream.ReadToEnd(); responseStream.Close(); resp.Close(); Console.WriteLine(responseString); } catch (Exception ex) {     Console.WriteLine(ex.Message.ToString()); } Console.WriteLine(); Console.WriteLine("Do you want to continue?"); } while (Console.ReadLine().ToUpper() == "Y"); }  Uwagi: - obiekty typu <code>HttpWebRequest</code> i <code>HttpWebResponse</code> służą do wysyłania i odbierania żądań HTTP. - operacje na strumieniach (i buforach) służą do ustawienia odpowiedniego kodowania (niestety Windows używa inny, własny standard). </pre> |
|--|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

|                                   |                                                                                                                                                                                                                                                                                                                                               |
|-----------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 9. Testowanie działania aplikacji | <ul style="list-style-type: none"> <li>Skompiluj aplikację kliencką</li> <li>Przygotuj odpowiednie dane xml i json dla operacji addXml i addJson. Można kierować informacją zawartą na stronie help opisu operacji serwisu.</li> <li>Uruchom klienta w oknie konsoli i przetestuj działanie klienta dla kilku dostępnych operacji.</li> </ul> |
|-----------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

### 3 Zadanie – część II (ta część może być jeszcze uzupełniona)

**Szczegółowe wymagania i wybór zadania (A, B lub inne) określa prowadzący grupy.**

- A. Przećwiczyć prezentowane w ćwiczeniu techniki.
- B. Przygotować się do napisania aplikacji o podobnych funkcjonalnościach według wskazówek prowadzącego  
lub
- C. zrealizować zadany przez prowadzącego program.

Przykładowy program do przygotowania na zaliczenie:

Opracować aplikację WCF (serwis i klienta) spełniające następujące wymagania:

Serwis:

1. Serwis jest usługą webową typu REST.
2. Serwis realizuje funkcjonalność typu magazyn danych, przy czym:
  - Obsługuje wszystkie operacje CRUD (Create, Read, Update, Delete), czyli:
    - odczyt pojedynczego rekordu oraz całej listy rekordów,
    - dodawanie rekordów danych,
    - usuwanie rekordów danych,
    - modyfikowanie wskazanego rekordu,
  - W przypadku próby wykonania niedozwolonej operacji (np. próba usunięcia rekordu o identyfikatorze którego nie ma) każda operacja powinna odpowiednio reagować.
  - Dane przechowywane są na serwerze (w dowolny sposób – np. w postaci listy).
3. Usługa używa formaty JSON i XML (każda operacja jest dla obu formatów).

Klient:

Klientem do testowania aplikacji może być narzędzie **Postman** (<https://www.getpostman.com>), które pozwala wykonywać różne żądania http.

*W takim wypadku trzeba ściągnąć, zainstalować i opanować te narzędzie.*

Klientem może być też własna (np. konsolowa) aplikacja, która pozwala korzystać z serwisu.

- Klient pozwala korzystać z wszystkich operacji serwisu.
- Wyniki operacji są wizualizowane – wyświetlane na ekranie.
- Wprowadzanie danych dla własnej aplikacji powinno być wygodne. Dobrym podejściem jest wprowadzanie kolejnych danych przez użytkownika:
  - podając tylko wartości w konsoli dla kolejnych pól rekordów,
  - lub w polach aplikacji graficznej.

## Exercise 7

### AJAX Application (with REST service)

*Author: Mariusz Fraś*

#### Objectives of the exercise

The purpose of the exercise is:

1. Acquaintance with architecture of AJAX based Web applications.
2. Mastering the technique of building REST service with AJAX Client.

#### Development environment

The developing of applications can be performed using various platforms. Here is considered the following environment:

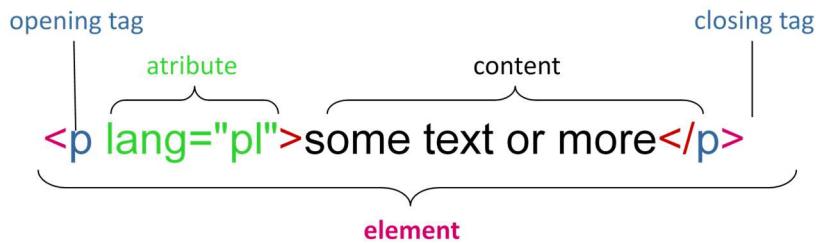
- Windows 7 (or higher) Operating System (eventually MacOS).
- Java Software Development Kit (JDK 11 or newer) and Java application development platform (here: IntelliJ IDEA).  
or
- Visual Studio 2017/2019/2021
- Any editor for HTML doc and Web Browser

## 1 Basic elements of creating a web application (for the browser)

The chapter briefly presents the basic programming elements important for AJAX technique based application client.

### 1.1 Elements of HTML language essential for the implementation of AJAX mechanisms

HTML (HyperText Markup Language) is a language for describing hypermedia content / web documents. The basic components of HTML documents are **HTML elements** defined by tags:



Elements can be nested and tags can contain attributes that define various properties of the elements (e.g. colors, identifiers, etc.).

For the implementation of operations as a result of AJAX requests, the selection of HTML elements is important. The most common uses are:

- Selection based on element ID. One of the attributes can be a unique identifier **id** (in the example for a paragraph of the text (specified by the `<p>` tag)):

```
<p id="my-id"> here is some text </p>
```

- Selection based on the class to which many elements can be assigned. The **class** attribute can be assigned to multiple elements and multiple classes can be defined (in the example for a text paragraph and 2nd-order heading):

```
<p id="my-id" class="my-first-class"> ... </p>
```

```
<h2 id="other-id" class="my-first-class my-second-class"> ... </h2>
```

- Selection by element type (tag name).

Some of the most basic tags for defining the structure and content of HTML docs are:

Tag	Description	Tag	Description
<code>&lt;html&gt;</code>	Defines an HTML document	<code>&lt;ul&gt;</code>	Defines an unordered list
<code>&lt;head&gt;</code>	Defines information about the document	<code>&lt;ol&gt;</code>	Defines an ordered list
<code>&lt;body&gt;</code>	Defines the document's body	<code>&lt;li&gt;</code>	Defines a list item
<code>&lt;p&gt;</code>	Defines a paragraph	<code>&lt;table&gt;</code>	Defines a table
<code>&lt;h1&gt; to &lt;h6&gt;</code>	Defines HTML headings	<code>&lt;tr&gt;</code>	Defines a row in a table
<code>&lt;a&gt;</code>	Defines a hyperlink	<code>&lt;td&gt;</code>	Defines a cell in a table

<code>&lt;img&gt;</code>	Defines an image	<code>&lt;input&gt;</code>	Defines an input element to enter data
<code>&lt;button&gt;</code>	Defines a clickable button	<code>&lt;div&gt;</code>	Defines a section in a document

The `<html>`, `<head>`, and `<body>` tags define the required basic document structure:

```

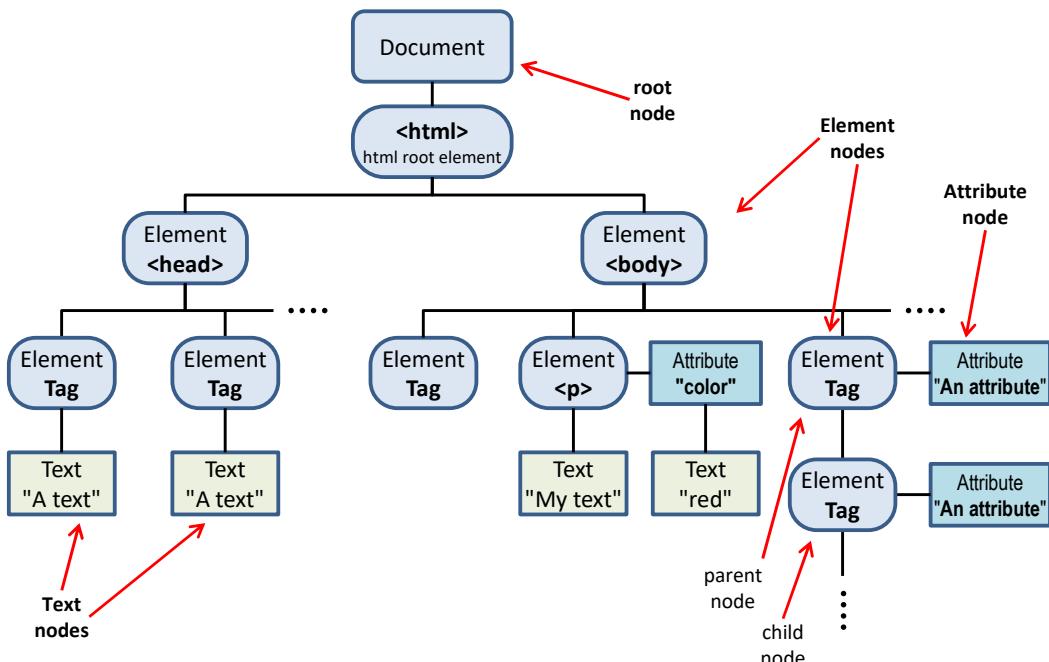
<!DOCTYPE html>
<html>
 <head>
 <title>Web Page Title</title>
 </head>
 <body bgcolor="blue">
 <div id="section-1">
 <h2 id="id1">Section title</h2>
 <p id="id2">My text on the page</p>
 </div>

 <p>Next line here</p>
 <!-- the comment -->
 </body>
</html>

```

## 1.2 DOM and JavaScript

Web browsers display the content on the screen (browser window) based on the DOM tree (including HTML DOM). The DOM (Document Object Model) tree is a tree of objects representing the elements and structure of the HTML document to display, built by the browser based from HTML code and CSS styles.



DOM tree objects are accessed through the DOM API. The ability to modify the DOM tree allows you to directly impact what is displayed in the browser (i.e. on the client side).

The API for JavaScript provides objects and methods that allow you to operate on the DOM tree. Many of the actions performed are initiated using the **document** object. The objects and methods available include, for example:

- searching for HTML elements - for example:

Method	Description
<b>document.getElementById(id)</b>	Find an element by element id
<b>document.getElementsByTagName(name)</b>	Find elements by tag name
<b>document.getElementsByClassName(name)</b>	Find elements by class name

- adding/removing elements - e.g.:

Method	Description
<b>document.createElement(element)</b>	Create an HTML element
<b>document.removeChild(element)</b>	Remove an HTML element
<b>document.appendChild(element)</b>	Add an HTML element
<b>document.replaceChild(new, old)</b>	Replace an HTML element
<b>document.write(text)</b>	Write into the HTML output stream

- change of HTML elements (attributes) and CSS styles - for example:

Property	Description
<b>element.innerHTML = new html content</b>	Change the inner HTML of an element
<b>element.attribute = new value</b>	Change the attribute value of an HTML element
<b>element.style.property = new style</b>	Change the style of an HTML element
Method	Description
<b>element.setAttribute(attribute, value)</b>	Change the attribute value of an HTML element

where *element* is a selected specific html element (e.g. a paragraph of text), *attribute* is a specific attribute of the element, *style.property* is a specific style of a given element.

### Examples:

For HTML document::

```
<html>
 <body bgcolor="blue">
 <p id="id1">First text</p>
 <p id="id2" class="c2">Second text</p>
 <p id="id3">Third text</p>
 </body>
</html>
```

The Javascript code:

```
<script type="text/javascript">
 document.getElementById("id1").innerHTML = "New Text";
 document.getElementsByClassName("c2").innerHTML = "New Text 2";
 document.getElementsByTagName("p").innerHTML = "New Text 3";
 document.getElementById("id1").align = "center";
 document.getElementById("id2").style.color = "red";
 document.getElementById("id3").setAttribute("align", "center");
</script>
```

The given JavaScript code (e.g. defined in a function) is executed as a result of events, including interface **input events** (user events) - e.g. the **onClick** event - clicking a given element. JavaScript allows you to define event handling statically and dynamically.

The assignment of code execution to a given event is defined statically by specifying an attribute of the element with **name of event** and assigning it a code to be executed (e.g. assigning the name of a function defined in the script):

```
<html>
 <head>
 <script type="text/javascript">
 function myFunction() {
 ...
 }
 </script>
 </head>
 <body>
 <button id="id1" onClick = "myFunction()">...</button>
 ...
 </body>
</html>
```

### 1.3 JSON methods

Among many other JavaScript objects and functions worth to know are those for handling AJAX requests that are useful for converting Json objects to text and vice versa:

```
JSON.parse(text)
JSON.stringify(object)
```

## 2 Creating an AJAX based web application

In the exercise will be implemented an AJAX based application i.e.:

- REST service with CRUD operations
- Web client built with AJAX technology.

## 2.1 HTTP-based REST service

### Basic HTTP-based REST service characteristics

- The HTTP REST service should adhere among the others to the following rules:
  - **URIs/URLs** are used to locate the resource,
  - the **application/json** or/and **application/xml** (or **text/xml**) data type (or similar open standard) should be used for transferred data,
  - the standard HTTP methods GET, POST, PUT (or PATCH), and DELETE should be used to perform CRUD operations.
  - standard HTTP status codes (response status) should be used in responses,
  - to support AJAX Web clients the CORS mechanism should be used.
- To implement CRUD operations the following scheme (a combination of URLs and HTTP methods) should be used:
  - GET** method to **Retrieve** resource (data),
  - POST** method to **Create** resource (add data),
  - PUT** (or **PATCH**) method to **Update** resource (modify data),
  - DELETE** method to **Delete** resource (remove data).

Usually PUT replaces whole resource and PATCH only a part.

- The scheme for operation on items in the collection (set, list, etc.) of items is:

URL	Method	Operation
http://example.com/items	GET	Get list (collection) of items
http://example.com/items/{id}	GET	Get item data of item with id={id}
http://example.com/items	POST	Adding a new item in collection of items
http://example.com/items/{id}	PUT	Updating item data of item with id={id} (or adding an item – not a standard)
http://example.com/items	DELETE	Removal of the whole collection of items
http://example.com/items/{id}	DELETE	Removal of item with id={id}

Where:

- **http://example.com/items** is base URL to the service (resource data) – the name items can be different of course,
- **{id}** is unique identifier (any unique string) for the resource.

The other (not CRUD) operations should be specified in the URL **after** the resource it apply.

The scheme takes one more level of path if the resource set contains subsets of items.

- The result status of operations should follow the following rules:

HTTP method	CRUD	Entire Collection ( e.g. /items )	Specific Item ( e.g. /items/{id} )
<b>POST</b>	Create	201 (Created), 'Location' header with link to /items/{id} containing new ID.	404 (Not Found), 409 (Conflict) if resource already exists..
<b>GET</b>	Read	200 (OK), list of items.	200 (OK), single item. 404 (Not Found), if ID not found or invalid.
<b>PUT</b>	Update / Replace	405 (Method Not Allowed), unless you want to update / replace every resource in the entire collection.	200 (OK) ... or 204 (No Content). 404 (Not Found), if ID not found or invalid.
<b>PATCH</b>	Update / Modify	405 (Method Not Allowed), unless you want to modify the collection itself.	200 (OK) ... or 204 (No Content). 404 (Not Found), if ID not found or invalid.
<b>DELETE</b>	Delete	405 (Method Not Allowed), unless you want to delete the whole collection.	200 (OK). 404 (Not Found), if ID not found or invalid.

## 2.2 AJAX Web Client

The Web client of HTTP-based REST service in the form of Web page with appropriate event handling, can be implemented with pure **JavaScript** code, any ready to use library like **jQuery**, or a **framework** build on basic mechanisms.

### 2.2.1 Using pure JavaScript

The JavaScript code uses **XMLHttpRequest** (XHR for short) object to define/prepare and send Ajax request, and define handling of asynchronous response.

- The basic **XHR methods** to perform basic operations are:

```
open(method, url) – to create a query/request (asynchronous by default), or:
open(method, url, async)
open(method, url, async, username, password)
```

where: *method* – is one of HTTP method, *url* – is URL to send the request to, *async* – the Boolean parameter indicating whether or not to perform the operation asynchronously.

*send()* – sends the request to the server,

*send(body)* – method accepts an optional parameter which lets to specify the request's body

where: *body* – can be a *Blob*, *BufferSource*, *FormData*, *URLSearchParams*, or string object.

`abort()` – aborts the request if it has already been sent.

- The basic **object properties** used to support operations are:

- `readystate` – the status of request execution

The following states are possible:

Value	State	Description
0	UNSENT	Client has been created. <code>open()</code> not called yet.
1	OPENED	<code>open()</code> has been called.
2	HEADERS_RECEIVED	<code>send()</code> has been called, and headers and status are available.
3	LOADING	Downloading; <code>responseText</code> holds partial data.
4	DONE	The operation is complete.

- `responseType` – specifies the type of the response.

- `response`

`responseText`  
`responseXML` – the response of given type: (1) an *ArrayBuffer*, a *Blob*, a *Document* (XML, HTML,...), a JavaScript object, or a string, depending on the value of `responseType`, or (2) *plain text*, or (3) a *Document* containing the HTML or XML

For `responseXML` usually, the response is parsed as "text/xml". If the `responseType` is set to "document" and the request was made asynchronously, instead the response is parsed as "text/html".

- `status` – HTTP response code (e.g. 200, 404, etc.)

- `statusText` – contains status as a string (e.g. "OK" or "Not Found")

- The basic **object events** used to support operations are:

- `readystatechange` – fired whenever the `readyState` property changes. Also available via the `onreadystatechange` event handler property.

- `load` – fired when an XHR transaction completes successfully. Also available via the `onload` event handler property,

- The basic **object events handlers** used to support operations are:

- `onreadystatechange` – reference to the function invoked by the `readyState` change.

- `onload` – reference to the function invoked when request is completed.

### Code examples

- 1) A simple example (a pattern) to implement the GET request (enclosed in **doRequest** function) and reaction to completion of the request with execution of **doJob** function:

```
function doRequest()
{
 var xhr = new XMLHttpRequest();
 xhr.onreadystatechange = doJob; // callback
 xhr.open("GET", url, true);
 xhr.send();
}

function doJob()
{
 if (xhr.readyState==4) {
 if (xhr.status==200) {
 [...] // process received data and/or do anything you want
 }
 else { alert("Data querry/response error"); }
 }
}
```

- 2) A snippet with using **onload** property:

```
function doRequest()
{
 var xhr = new XMLHttpRequest();
 xhr.onload = function doJob() { // callback
 [...] //process received data and/or do anything you want
 };
 xhr.open("GET", url, true);
 xhr.send();
}
```

- 3) An example with POST method:

```
var xhr = new XMLHttpRequest();
xhr.open("POST", url, true);

//Send the proper header information along with the request
xhr.setRequestHeader("Content-Type", "application/json");

xhr.onreadystatechange = function() { // when the state changes.

 if (this.readyState === XMLHttpRequest.DONE &&
 this.status === 200) {
 [...] //process received data and/or do anything you want.
 }
}

xhr.send("{foo:bar , lorem:ipsum}");
```

## 2.2.2 Using Fetch API

*Not described yet – sorry 😞*

## 2.2.3 Using jQuery

The jQuery library supports AJAX requests with **\$.ajax()** method.

### 2.2.3.1 \$.ajax() method

The scheme of using **\$.ajax()** function is the following:

```
$.ajax({
 url : "http://host.domain.org/service",
 method : "POST",
 dataType : "json",
 contentType : "application/json",
 data : {
 name : "Mariusz",
 country : "Polska"
 }
})
.done(function(data) {
 [...] //process received data
})
.fail(function(xhr, status, error) {
 [...] //handle the error
})
.always(function() {
 [...] //perform some processing
});
```

Where properties (method parameters) are:

url – targeted address (where to send request),  
 method – connection type/HTTP method (GET by default),  
 dataType – the MIME type of data expected in response,  
 contentType – the type of the sent data,  
 data – data to be sent (e.g. JSON object),

The defined methods called in the effect of the request are:

- .done – called when the request is successfully completed,
- .fail – called after an error,
- .always – code called in both cases.

Unused method can be omitted.

The other useful methods (some selection) are:

`$.load()` – loads data from a server and puts the returned data into the selected element

`$.get()` – loads data from a server using an AJAX HTTP GET request

`$.getJSON()` – loads JSON-encoded data from a server using a HTTP GET request

`$.getScript()` – loads (and executes) a JavaScript from a server using an AJAX HTTP GET request

`$.post()` – loads data from a server using an AJAX HTTP POST request

...and a few more.

### 2.2.3.2 Examples of implementing Ajax Web client using jQuery

- Let's assume the REST service that:
  - performs CRUD operations on the list of the following book data:  
**Id** – the book identifier,  
**Title** – the book title,  
**Price** – the book price.
  - the base address of resources is **http://example.com/books**, (for local tests e.g.:  
**http://localhost:8080/books**).
  - It uses JSON format for transferred data.

In JavaScript code variables for this data (string and JSON object respectively) can be:

```
<script type="text/javascript">
 var myUrl = "http://example.com/books"
 var Book = {
 "Id":0,
 "Title":"",
 "Price":0
 }
 ...
</script>
```

- The HTML code of the web page that support operation with the service and presents results of operation contains:

- Input type elements that allow to enter some data:

```
<input id="bid" type="text" />
<input id="btitle" type="text" />
<input id="bprice" type="text" />
```

For each field the unique identifier attribute is set.

These elements will be used to enter data to be sent or data read from server.

- The buttons that after clicking call a Javascript function that send ajax request:

```
<button type="button" id="read" onclick="getBook()">Read</button>
<button type="button" id="add" onclick="addBook()">Add</button>
```

- The table where the read data list is presented:

```
<table id="booktable" >
</table>
```

At the beginning the table is empty.

- The code to read and present the book data with Ajax request can be the following:

```
function getBook() {
 var id = $("#bid").val();

 $.ajax({
 type: "GET",
 url: myUrl + "/" + id,
 contentType: "application/json; charset=utf-8",
 dataType: "json",
 })
 .done(function(msg) {
 document.getElementById("btitle").value=msg.Title;
 document.getElementById("bprice").value=msg.Price;
 })
 .fail(function(errMsg) {
 alert(JSON.stringify(errMsg));
 });
}
```

Here:

- the book **id** is read from input element of id=bid,
- next Ajax GET request is sent to **http://example.com/books/id** (where **id** is entered id in input element),
- on request completion the received data in the form of JSON (in **msg** parameter) is put into the rest of input elements (of id=btitle and id=bprice).

Similarly the data may be inserted in other HTML element – e.g. text paragraph

`<p id=pid1></p>` – with the instruction:

```
document.getElementById("pid1").innerHTML = msg.Title;
```

- The example code to add a book data with Ajax request can be the following:

```
function addBook() {
 var book = Book;

 book.Id = $("#bid").val();
 book.Title = $("#title").val();
 book.Price = $("#price").val();
```

```

var id = $("#bid").val();
$.ajax({
 type: "POST",
 url: myUrl + "/", // /+id,
 data: JSON.stringify(book),
 contentType: "application/json; charset=utf-8",
 dataType: "json",
})
.done(function(msg) {
 [...] // here may be some action to show that data are stored
 // e.g.: alert('the book is stored');
 // or reloading data list, etc.
})
.fail(function(errMsg) {
 alert(JSON.stringify(errMsg));
});
}

```

Here:

- the data are read from HTML <input> elements and saved in JSON *book* variable,
- the Ajax POST request is sent,
- with data stringified to text.
- The example code to read whole data and present it with use of iterating JSON data can be the following:
  - The Javascript function sending ajax request is almost the same as for reading single data item except for URL and the call of function displaying data on completion of the request:

```

function getAllBooks()
{
 [...]
 url: myUrl,
 [...]
 .done(function(msg) {
 showBooks(msg)
 });
}

```

- The function presenting read data in <table> element (in browser window) will:
  - remove previous content of table (tbody),
  - iterate received data – all elements of received JSON data containing book list with use of **\$.each** method,
  - add table rows with data taken from JSON objects into table <tbody> element.

```
function showBooks(msg)
{
 $("#booktable tbody").remove(); //remove previous data
 $.each(msg, function (id, book) { // iterate by all JSON elements
 // (books)

 // First check if a <tbody> tag exists and add one if not
 if ($("#booktable tbody").length == 0) {
 $("#booktable").append("<tbody></tbody>");
 }

 // Append table row to <table> (in <tbody> part)
 $("#booktable tbody").append("<tr>" +
 "<td>" + book.Id + "</td>" +
 "<td>" + book.Title + "</td>" +
 "<td>" + book.Price + "</td>" +
 "</td>" +
 "</tr>");

 });
}
```

The rest of CRUD operations and presentation of results in browser window is implemented similarly.

### 3 Exercise – Part II

*The detailed and final requirements for Part II are set by the teacher.*

- A. Develop or prepare to develop a program according to the teacher's instructions.