Programowanie Aplikacji Internetowych

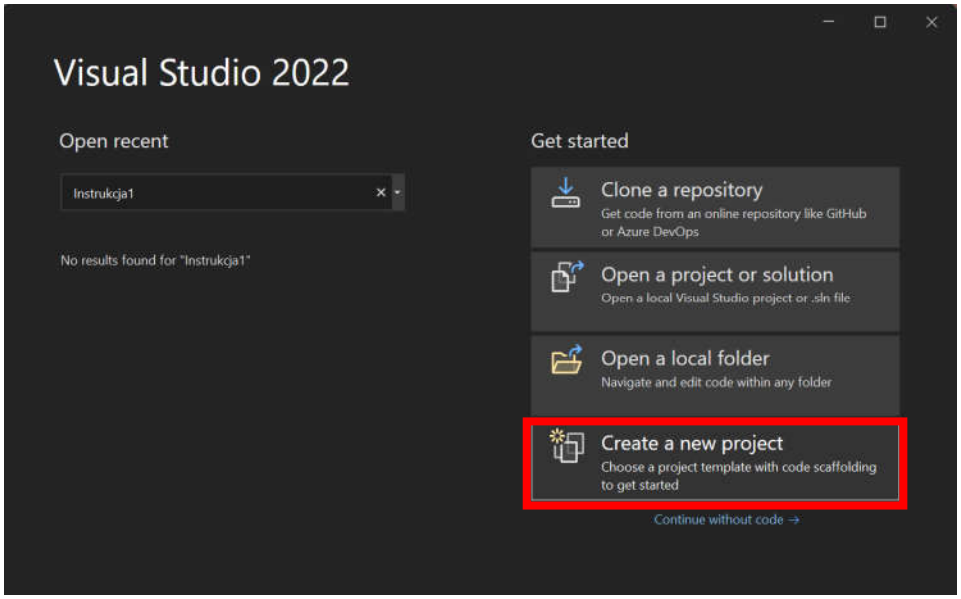
Instrukcja 1
Podstawy programowania obiektowego

Czas na realizację: 4 godziny (dwa laboratoria)

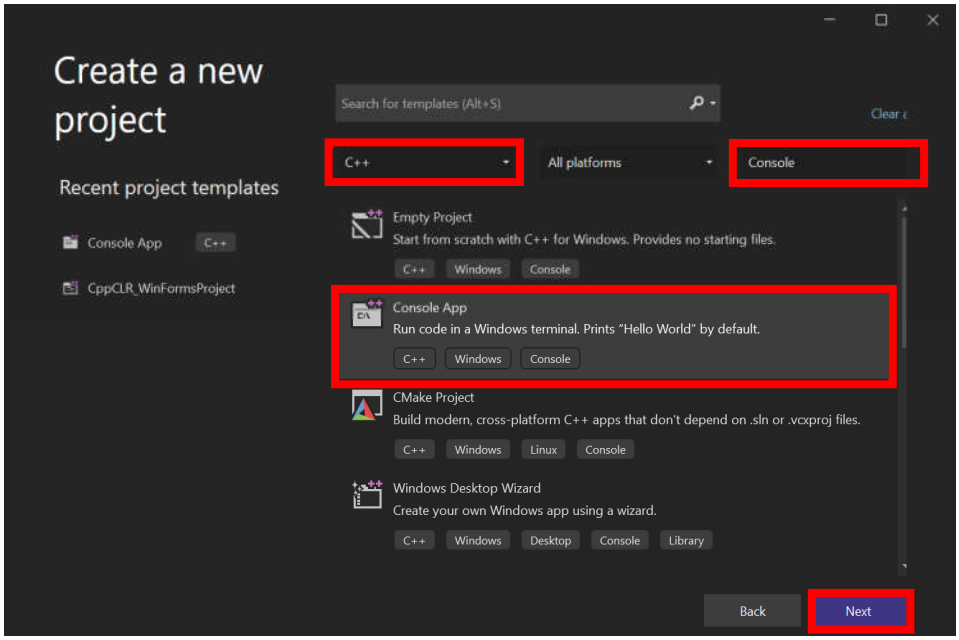
Instrukcja została podzielona na dwie części – zapoznanie się z środowiskiem Microsoft Visual Studio C++ oraz podstawy programowania obiektowego. W ramach realizacji instrukcji należy wykonać sprawozdanie (wzór na ostatniej stronie).

1. Środowisko Microsoft Visual Studio i programowanie w języku C++ (2 godziny)

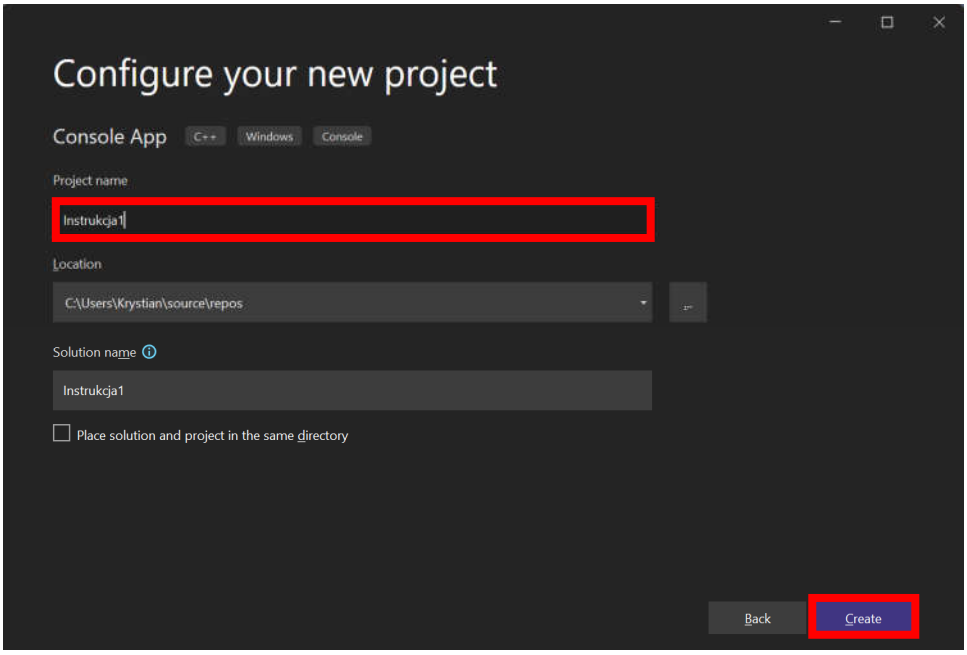
W celu przetestowania środowiska *Microsoft Visual Studio* w ramach tej instrukcji zostanie wykonany projekt w konsoli. Aby to zrobić należy po uruchomieniu środowiska wybrać opcję Create a new project:



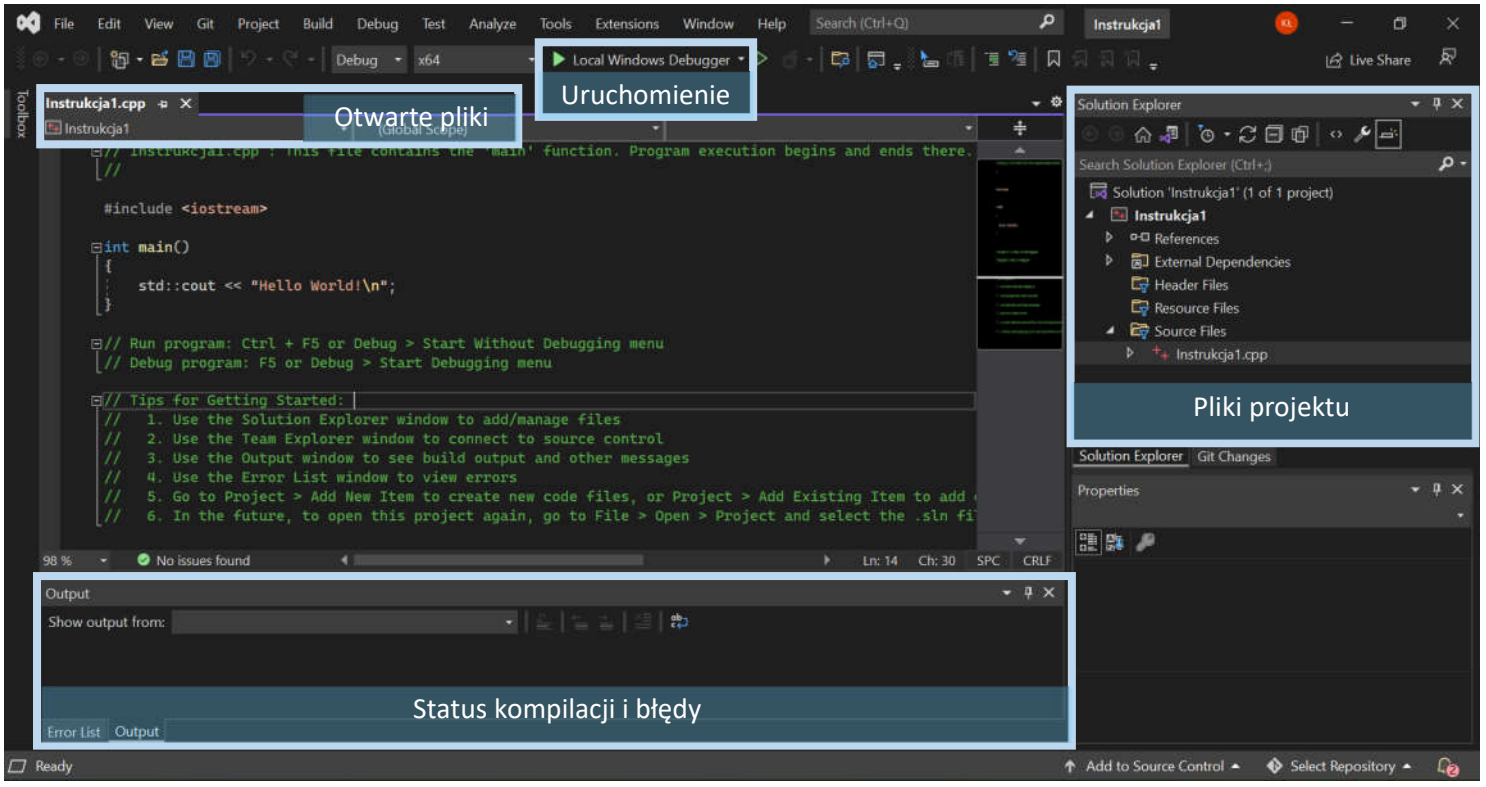
Następnie należy albo wpisać Console App (Aplikacja konsoli) **lub** wybrać język C++ i tryb Console (Konsola):



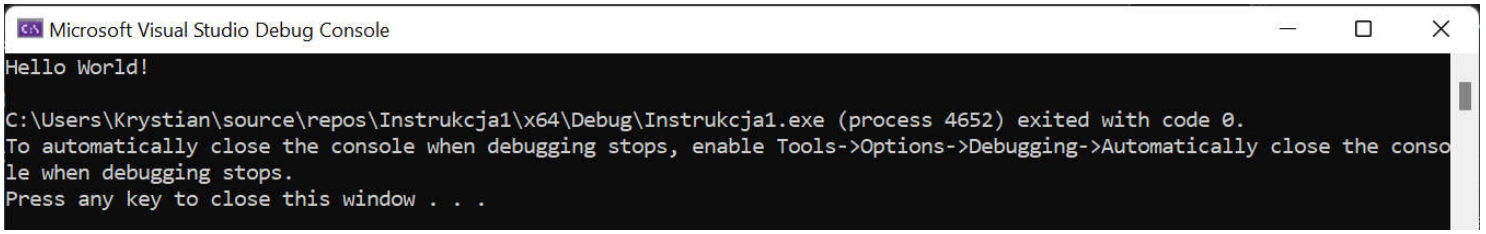
Po przejściu dalej można wpisać nazwę projektu (np. Instrukcja1) i utworzyć projekt:



Okno programu prezentuje się następująco:



Domyślny program wyświetla w konsoli tekst "Hellow World!". Po uruchomieniu programu otrzyma się następujące okno:



Zadanie 1

W funkcji **main** wypisać w konsoli tekst "Podaj dwie liczby: ". Następnie pobrać od użytkownika **dwie** liczby **całkowite**. Służy do tego następujący zapis:

```
int liczba_a; // utworzenie zmiennej
std::cin >> liczba_a; // pobranie zmiennej od użytkownika (zatwierdzenie klawiszem Enter)
```

Po pobraniu dwóch liczb od użytkownika obliczyć ich **sumę** i wyświetlić na ekranie. Wyświetlanie danych za pomocą strumienia std::cout umożliwia wyświetlanie kilku wartości jednocześnie, np.:

```
std::cout << "Wynik: " << wynik << std::endl; // wyświetli: "Wynik: X\n", gdzie X jest liczbą
```

W powyższym zapisie std::endl oznacza znak nowej linii. W celu pominięcia wykorzystania przestrzeni nazw std (czyli zapisu std::) można opcjonalnie przed funkcją main umieścić zapis domyślnie wykorzystujący tą przestrzeń:

```
using namespace std; // od teraz można pisać cout zamiast std::cout (analogicznie cin i endl)
```

Przetestować utworzony program podając jako liczby:

- dwie wartości całkowite
- dwie wartości zmiennoprzecinkowe
- dwa wyrazy

Krótkie wnioski oraz kod źródłowy zamieścić w sprawozdaniu **(2 pkt)**.

Opcjonalnie napisać program w taki sposób, aby po podaniu nieprawidłowych danych (np. tekstu), program ponownie prosił o podanie danej liczby. Utworzony kod umieścić w sprawozdaniu **(1 pkt)**.

Zadanie 2

Celem zadania jest porównanie czasu obliczeń **dzielenia** oraz **mnożenia** liczb podwójnej precyzji (**double**). Do zmierzenia czasu można wykorzystać np. bibliotekę **chrono**. Aby to zrobić należy załączyć plik blbioteki chrono:

```
#include <chrono>
```

Następnie czas obliczeń można zmierzyć następująco:

```
auto start = std::chrono::system_clock::now();
// miejsce wstawienia kodu obliczeń
auto koniec = std::chrono::system_clock::now();
auto roznica = std::chrono::duration_cast<std::chrono::milliseconds>(koniec - start).count();
std::cout << "Czas obliczen: " << roznica << "ms\n";
```

W ramach obliczeń wykonać za pomocą pętli 100000000 operacji mnożenia typu double, np.:

```
for (int i = 0; i < 100000000; i++) {
    double a = 10.0 * i;
}
```

Sprawdzić czas obliczeń dla powyższego kodu (**mnożenie**) oraz zrobić to samo dla **dzielenia**.

Porównać wyniki, otrzymane czasy i krótkie wnioski umieścić w sprawozdaniu **(2 pkt)**.

Opcjonalnie sprawdzić czas wykonania pustej pętli (bez żadnej operacji w środku) i odpowiednio uwzględnić ten czas w wynikach **(1 pkt).**

Zadanie 3

Utworzyć funkcję o nazwie **min3**, przyjmującą trzy liczby całkowite jako parametry i zwracającą najmniejszy z podanych parametrów (najmniejszą liczbę). Wynik dla wybranych przez siebie liczb wyświetlić w konsoli.

Kod źródłowy zamieścić w sprawozdaniu **(2 pkt)**.

Zadanie 4

Celem zadania jest przetestowanie debugowania w środowisko Microsoft Visual Studio.

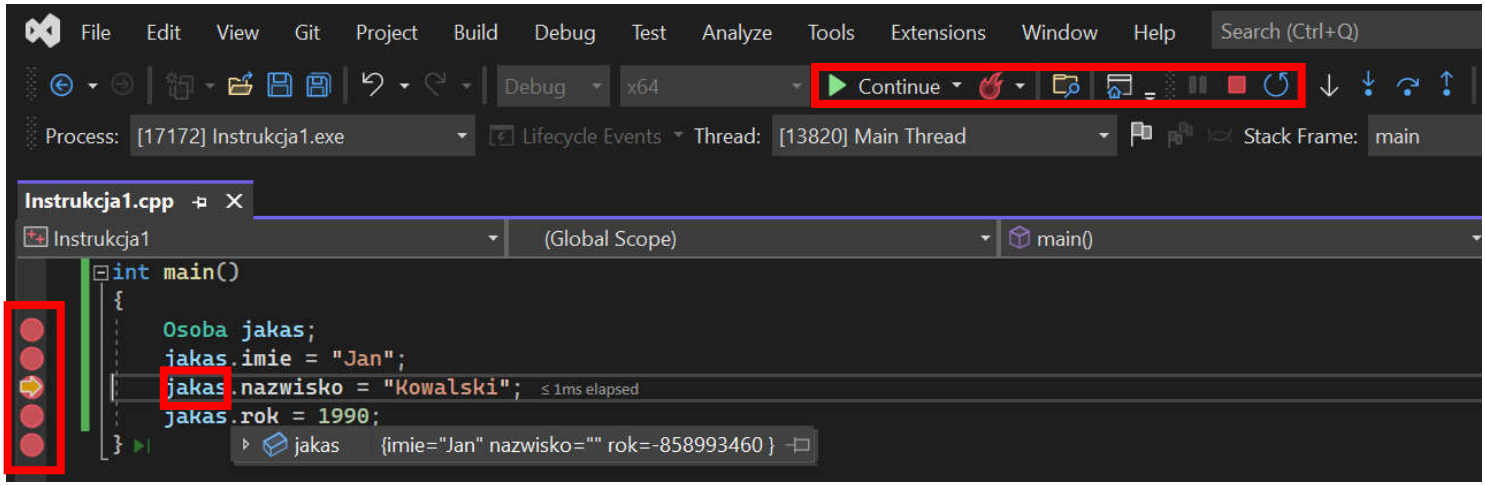
W ramach zadania należy utworzyć strukturę **Osoba** zawierającą następujące zmienne: string imie, string nazwisko, int rok. Strukturę można zdefiniować bezpośrednio nad funkcją **main**. Aby korzystać w taki sposób ze zmiennych string należy załączyć bibliotekę string i użyć przestrzeni nazw std:

```
#include <iostream>
#include <string>

using namespace std;
```

W środowisku Microsoft Visual Studio można ustawiać pułapki (*breakpointy*), czyli miejsca w których zatrzyma się program podczas wykonywania, jeżeli został uruchomiony w trybie debugowania (czyli domyślnie).

Aby ustawić *breakpoint* wystarczy kliknąć obok wybranej linii kodu z lewej strony. **Uwaga:** program zatrzymuje się domyślnie przed zaznaczonym miejscem (kod z zaznaczonej linii nie zostanie wykonany). Po zatrzymaniu programu można najechać kursorem na zmienne w programie i podejrzeć ich zawartość w danej chwili. Aby program kontynuował działanie należy wybrać opcję "Continue". Pokazuje to następujący przykład:



Celem zadania jest przetestowanie powyższego kodu i ustawienie 5 *breakpointów* jak na zrzucie ekranu powyżej. Należy sprawdzić jakie są wartości zmiennych struktury osoba w przypadku **pierwszego** i **piątego** *breakpointa*. Do przejścia do kolejnego *breakpointa* należy wykorzystać przycisk Continue.

Krótkie wnioski zamieścić w sprawozdaniu **(2 pkt)**.

2. Podstawy programowania obiektowego (2 godziny)

Tworząc struktury w każdym miejscu kodu można zmodyfikować ich zawartość, a funkcje na nich operujące **nie są** ściśle z nimi powiązane. W **programowaniu obiektowym** tworzy się **klasy**, które definiują nie tylko **dane** (tak jak w strukturach), ale także **poziom dostępu** do tych danych (publiczny lub prywatny) i **metody** związane z tymi danymi (metody, czyli funkcje związane z daną klasą także mają przypisany poziom dostępu). Takie podejście zapewnia **hermetyzację** (enkapsulację), czyli ukrywanie pewnych składowych, zapewniające m.in. bezpieczeństwo, wyodrębnienie funkcjonalności, czy uodpornienie na błędy i lepsze odzwierciedlenie rzeczywistości.

Porównanie obu podejść:

Programowanie funkcyjne (struktury)	**Programowanie obiektowe (klasy)**
plik osoba.h (nagłówkowy)	plik osoba.h (nagłówkowy)

`#ifndef OSOBA_H`
`#define OSOBA_H`
 	
`#include <string>`
 	
`using namespace std;`
 	
`struct Osoba`
`{`
`  string imie;`
`  string nazwisko;`
`  int rok;`
`};`
 	
`void OsobaInicjalizacja(Osoba& osoba, string imie string nazwisko);`
 	
`void OsobaInformacje(Osoba& osoba);`
 	
`#endif`
	`#ifndef OSOBA_H`
	`#define OSOBA_H`
	
	`#include <string>`
	
	`using namespace std;`
	
	`class Osoba`
	`{`
	`  private:`
	`  string imie;`
	`  string nazwisko;`
	`  int rok;`
	`  public:`
	`  void Inicjalizacja(string imie,`
	`  string nazwisko, int rok);`
	`  void Informacje();`
	`};`
	
	`#endif`

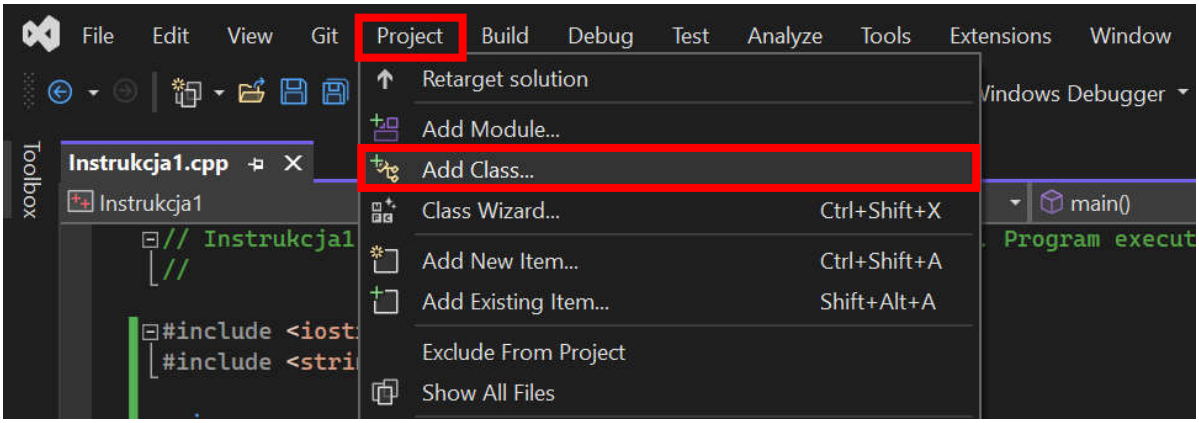
plik osoba.cpp (źródłowy)	plik osoba.cpp (źródłowy)

`#include <iostream>`
`#include <string>`
`#include "osoba.h"`
 	
`void OsobaInicjalizacja(Osoba& osoba, string imie string nazwisko, int rok)`
`{`
`  osoba.imie = imie;`
`  osoba.nazwisko = nazwisko;`
`  osoba.rok = rok;`
`}`
 	
`void OsobaInformacje(Osoba& osoba)`
`{`
`  cout << osoba.imie << " ";`
`  cout << osoba.nazwisko.substr(0, 1);`
`  cout << ". " << osoba.rok << endl;`
`}`
	`#include <iostream>`
	`#include <string>`
	`#include "osoba.h"`
	
	`void Osoba::Inicjalizacja(string imie, string nazwisko, int rok)`
	`{`
	`  this->imie = imie;`
	`  this->nazwisko = nazwisko;`
	`  this->rok = rok;`
	`}`
	
	`void Osoba::Informacje()`
	`{`
	`  cout << this->imie << " ";`
	`  cout << this->nazwisko.substr(0, 1);`
	`  cout << ". " << this->rok << endl;`
	`}`

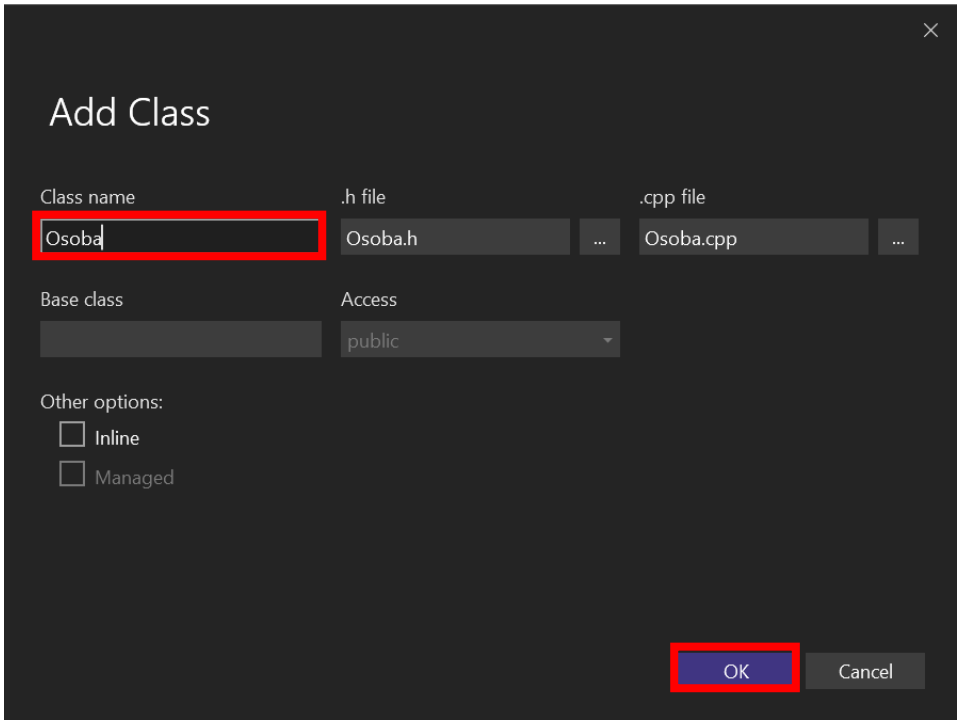
W przypadku **programowania obiektowego** warto zauważyć:

- Wydzielenie bloku **private** (zmienne i metody dostępne tylko w ramach metod klasy) i **public** (zmienne i metody dostępne publicznie) – słowa kluczowe private i public nazywane są **modyfikatorami dostępu**
- Poprzedzenie nazw metod w pliki osoba.cpp nazwą klasy i podwójnym dwukropkiem (Osoba::)
- Wykorzystanie słowa kluczowego **this** do odwołania się do danych klasy

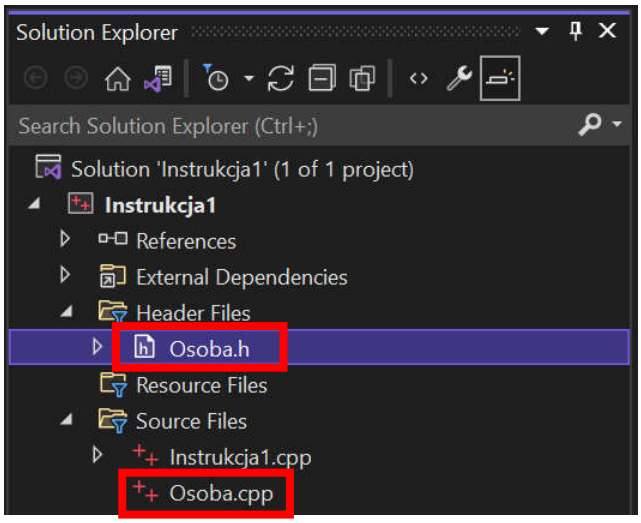
Aby dodać pliki .h oraz .cpp związane z tworzoną klasą należy wybrać opcję Project → Add class…:



Następnie można wpisać nazwę klasy i ją utworzyć:



W Solution Explorer umieszczane są automatycznie dwa pliki związane z klasą. Nazwa.cpp, oraz Nazwa.h (w tym pliku zamiast instrukcji #ifndef itd. wykorzystany jest zapis #pragma once**,** informujący że dany plik ma być tylko jeden raz przetwarzany przez kompilator.



W ramach przykładu zmodyfikujmy plik Osoba.h następująco:

```
#pragma once

#include <string>

using namespace std;

class Osoba
{
    private:
        string imie;
        string nazwisko;
        int rok;
        int* dane;
    public:
        Osoba(string imie, string nazwisko, int rok);
        ~Osoba();
        void Informacje();
};
```

Natomiast Osoba.cpp następująco:

```
#include <iostream>
#include <string>
#include "Osoba.h"

Osoba::Osoba(string imie, string nazwisko, int rok)
{
    this->imie = imie;
    this->nazwisko = nazwisko;
    this->rok = rok;
    this->dane = new int[10];
}

Osoba::~Osoba()
{
    delete[] dane;
}

void Osoba::Informacje()
{
    cout << this->imie << " ";
    cout << this->nazwisko.substr(0, 1);
    cout << ". " << this->rok << endl;
}
```

W powyższym przykładzie umieszczono dwie nowe metody (funkcje związane z klasą) o nazwie **Osoba** oraz **~Osoba** nie zwracające żadnego typu. Jest to odpowiednio **konstruktor** (metoda wywoływana przy tworzeniu obiektu) oraz **destruktor** (metoda wywoływana przy niszczeniu obiektu, może ona zwalniać wcześniej przydzieloną pamięć).

Przykład wykorzystania powyższej klasy (plik Instrukcja.cpp):

```
#include <iostream>
#include "Osoba.h"

int main()
{
    Osoba jakas("Jan", "Kowalski", 1990); // wywołanie konstruktora i utworzenie obiektu 'jakas'
    Osoba druga("Anna", "Nowak", 1992); // wywołanie konstruktora i utworzenie obiektu 'druga'
    jakas.Informacje(); // wywołanie metody Osoba::Informacje dla obiektu 'jakas'
    druga.Informacje(); // wywołanie metody Osoba::Informacje dla obiektu 'druga'
} // w tym momencie wywoływane są destruktory wszystkich obiektów
```

Zadanie 5

Przetestować przykład z klasą Osoba (szósta i siódma strona tej instrukcji).

Wynik działania programu umieścić w sprawozdaniu **(2 pkt).**

Rozszerzyć funkcjonalność klasy Osoba dodając **metodę - void Losuj()**, która wylosuje **10 liczb** i umieści je w tablicy **dane** (można zastosować pętle). Do wylosowania liczby wykorzystać zapis: `rand() % 50`. Dodać metodę wyświetlającą liczby z tablicy **dane - void WypiszWylosowane().** Zrobić to w taki sposób, aby była to metoda **prywatna**. Wywołać tą metodę wewnątrz metody **Informacje()**.

Przetestować działanie programu i nowych metod. Kod źródłowy umieścić w sprawozdaniu **(3 pkt).**

Spróbować wyświetlić nazwisko jednej z osób w funkcji main (std::cout << jakas.nazwisko;). Wynik i wnioski dotyczące potencjalnego rozwiązania umieścić w sprawozdaniu **(1 pkt).**

Opcjonalnie dodać drugi konstruktor, który przyjmie dodatkowo parametr typu int, określający ile liczb ma zawierać tablica **dane**. W razie konieczności dodać w klasie dodatkowe zmienne. **(2 pkt)**

Opcjonalnie zastanowić się nad umożliwieniem odczytaniem nazwiska osoby, ale bez możliwości jego modyfikacji. **(2 pkt)**

Programowanie Aplikacji Internetowych - Sprawozdanie

Instrukcja 1

Podstawy programowania obiektowego

Imię I nazwisko, numer indeksu

Zadanie 1

….. (wnioski, kody źródłowe, wyniki działania, zadania opcjonalne)

Zadanie 2

….. (wnioski, kody źródłowe, wyniki działania, zadania opcjonalne)

Zadanie 3

….. (wnioski, kody źródłowe, wyniki działania, zadania opcjonalne)

Zadanie 4

….. (wnioski, kody źródłowe, wyniki działania, zadania opcjonalne)

Zadanie 5

….. (wnioski, kody źródłowe, wyniki działania, zadania opcjonalne)