giganci programowania

Materiały powtórzeniowe

Lekcja 8. Dobre praktyki programistyczne

Witaj w materiałach powtórzeniowych przygotowanych specjalnie dla Ciebie. Znajdziesz tutaj powtórkę materiału z ostatniej lekcji oraz zadania do wykonania dla chętnych. Z pewnością sobie poradzisz. Powodzenia!

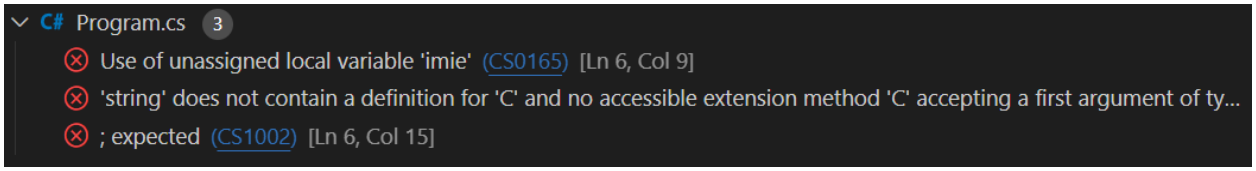
Cel lekcji

Celem zajęć jest nauka testowania i naprawiania swojego kodu oraz efektywne wykorzystywanie IDE (Zintegrowanego środowiska programistycznego) w celu ułatwienia pracy.

Debugowanie kodu

Proces analizowania kodu w poszukiwaniu błędów nazywamy **debugowaniem**, od angielskiego słowa **debug** i czynności **debugging**, oznaczającej dosłownie "**odpluskwianie**".

Błędy najczęściej napotkamy na **Problems**, w formie czerwonych komunikatów uniemożliwiających kompilowanie programu.



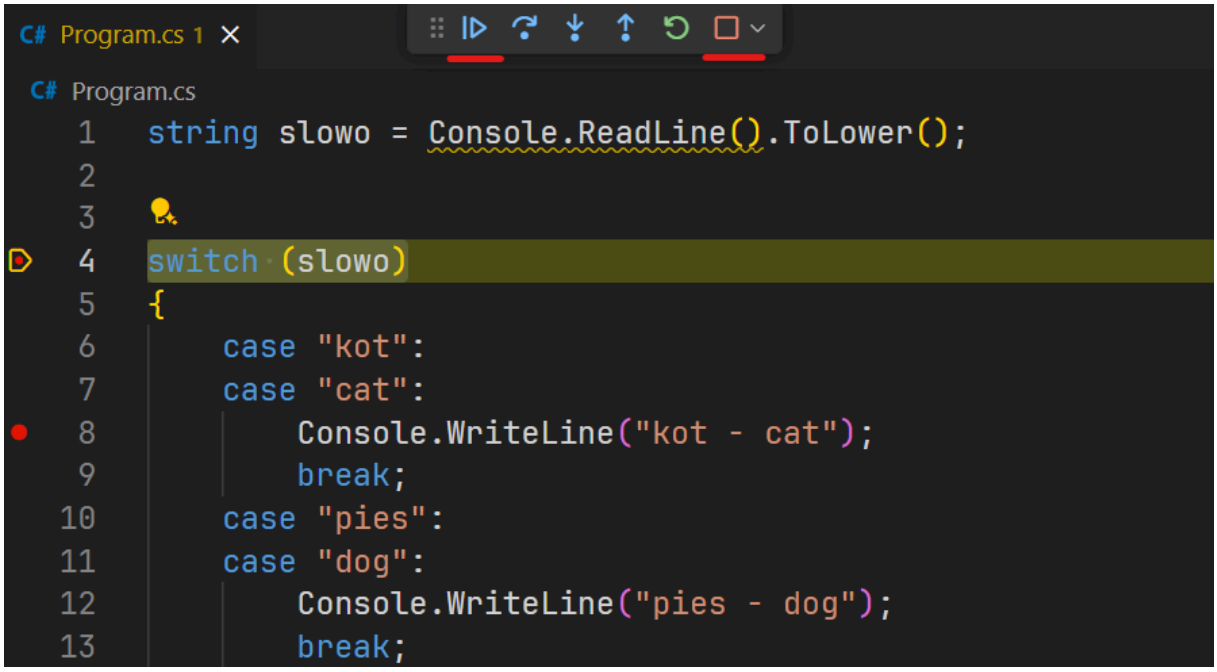
Najbardziej podstawową metodą szukania błędów, oprócz reagowania na czerwone komunikaty wyświetlane przez Error List jest stosowanie **Breakpointów**.

Breakpointy, znane także jako punkty przerwania, są narzędziem używanym w debugowaniu kodu w celu zatrzymania wykonania programu w określonym miejscu. Ze względu na to, że kod programu wykonywany jest linijka po linijce, to punkty przerwania, umożliwiają programistom dokładne przeanalizowanie stanu programu w momencie zatrzymania.

Dodajemy je **do konkretnej linii**, klikając na jasną belkę na lewo od numerów linii. Breakpoint zatrzyma program **PRZED** wykonaniem jej. Punktów takich możemy dodać nieskończenie wiele.

```
C# Program.cs
1   string slowo = Console.ReadLine().ToLower();
2
3
4   switch (slowo)
5   {
6       case "kot":
7       case "cat":
8           Console.WriteLine("kot - cat");
9           break;
10      case "pies":
11      case "dog":
```

Debugowanie włączy się automatycznie po standardowym włączeniu programu **przyciskiem Start lub skrótem F5**. Gdy kod dojdzie do linii z breakpoint to program się na niej zatrzyma, a my będziemy mogli analizować wykonany dotychczas kod.

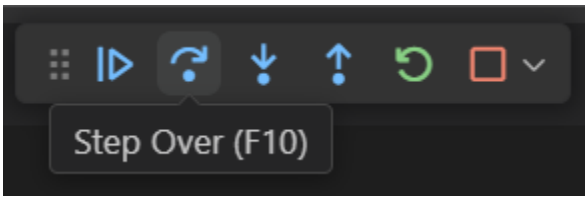


Aby kontynuować działanie programu i przejść do następnego breakpointa **naciskamy kolejny raz F5 lub przycisk Start, który teraz zamienił się na Continue**. Aby zakończyć działanie programu musimy skorzystać z przycisku przerwania oznaczonego czerwonym kwadratem.

Jeżeli w pamięci komputera zostały zadeklarowane już jakieś zmienne i wykonane obliczenia, to będziemy mogli prześledzić ich rezultat lub przypisane dane **najeżdżając na nie myszą.**

```
C# Program.cs
1   string slowo = Console.ReadLine().ToLower();
2
3           "mother"
4   switch (slowo)
5   {
6       case "kot":
7       case "cat":
8           Console.WriteLine("kot - cat");
9           break;
```

Istnieje sposób na analizowanie kodu, bez oznaczania kolejnych breakpointów. przykładowo, gdy, po wykonaniu warunku nie wiemy, do której linii program przeskoczy, możemy skorzystać **z przycisku przejścia dalej do następnej linii Step Over, lub skrótu F10 .**



```
26      case "jabłko":
27      case "apple":
28          Console.WriteLine("jabłko - apple");
29          break;
30      default:
31          Console.WriteLine("Słowo nieznane w słowniku.");
32          break;
33  }
34
```

Po ponownym naciśnięciu F5 lub Continue, program zacznie poszukiwać dalszych breakpointów, a gdy ich nie znajdzie to po prostu wznowi pracę. Możemy też kontynuować analizę linijka po linijce, skrótem F10, Step Over.

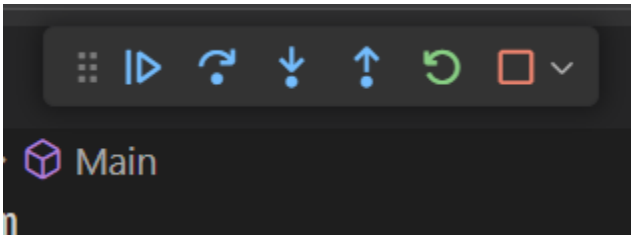
Dodatkowo

Możemy również wykorzystać **kursor,** (żółta strzałka)  , umożliwia zmianę punktu wykonania programu. Aby to zrobić, kliknij prawym przyciskiem myszy na wybraną linijkę kodu i wybierz opcję **„Jump to Cursor”** (lub **„Przeskocz do kursora”**).

```
1   string slowo = Console.ReadLine().ToLower();
2
3   switch (slowo)
4   {
5       case "kot":
6       case "cat":
7           Console.WriteLine("kot - cat");
8           break;
```

Po zatrzymaniu programu na break poincie, możesz przesunąć **kursor** np. na linię z odczytem zmiennej **slowo**, a następnie wcisnąć F5 (lub „Continue”), aby program wrócił do tego miejsca. Dzięki temu możesz wprowadzić nowe dane w konsoli i ponownie je przetworzyć.

Gdy przyjrzymy się opcjom górnego paska, dostrzeżemy, że skakanie pomiędzy breakpointami i do następnej wykonywanej linii, to nie jedyne operacje, które możemy wykonywać. Przyjrzyjmy się przyciskom:



- czerwony kwadrat - przerwanie programu
- niebieska zapętlająca się strzała - (restart) ponowne włączenie programu
- szara strzałka w prawo - przeskoczenie widoku edytora do aktualnie zatrzymanej linii programu
- niebieska strzałka z kropką skierowana w dół (F11) - wejście do środka wykonywanej operacji / metody
- niebieska strzałka wygięta w łuk z kropką pośrodku (F10) - przeskoczenie do następnej linii
- niebieska strzałka z kropką skierowana w górę (Shift+F11) - wyjście ze środka wykonywanej operacji funkcji

Dzięki przyciskowi F11 możemy wejść do środka wykonywanej metody stworzonej przez nas (nie zadziała to na Math czy Console). Oznacza to, że opcja **Step Into** dosłownie wchodzi wewnątrz operacji i pozwala na analizowanie jej krok po kroku, aż do wyjścia i powrotu programu znów do case. Gdybyśmy chcieli szybciej wyskoczyć z metody, możemy skorzystać z opcji **Shift+F11 Step Out**.

Popularne błędy składniowe

- Oczekiwanie znaku np. średnika lub dwukropka

giganci programowania

```
0 references
class Program
{
    0 references
    static void Main()
    {
        string slowo = Console.ReadLine().ToLower

        switch (slowo)
        {
            case "kot"
            case "cat":
                Console.WriteLine("kot - cat");
                break;
```

PROBLEMS 6 OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS Filter (

C# Program.cs 6

Syntax error, ':' expected (CS1003) [Ln 9, Col 23]

Syntax error, ':' expected (CS1003) [Ln 10, Col 23]

- **Niespodziewany znak klamry**

```
38
39          Console.ReadKey();
40
41      }
42  }
```

PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL PORTS COMME

C# Program.cs 2

Dereference of a possibly null reference. (CS8602) [Ln 5, Col 24]

Unreachable code detected (CS0162) [Ln 39, Col 9]

- **Nie istnienie zmiennej** - literówki i wyjście poza zasięg

```
38
39          if (liczba > 10)
40          {
41
42          } esle {
43
44          }
45
46      }
```

PROBLEMS 3 OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS Filte

C# Program.cs 3

The name 'esle' does not exist in the current context (CS0103) [Ln 42, Col 11]

; expected (CS1002) [Ln 42, Col 16]

Dereference of a possibly null reference. (CS8602) [Ln 5, Col 24]

```
39          if (liczba > 10)
40          {
41              int wynik = liczba * 2;
42          }
43
44          Console.WriteLine(wynik);
45
46      }
47  }
```

PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS Filter

C# Program.cs 2

The name 'wynik' does not exist in the current context (CS0103) [Ln 44, Col 27]

Dereference of a possibly null reference. (CS8602) [Ln 5, Col 24]

- Nie można rzutować / przekonwertować…

```
44      bool zmienna = 12;
45
46  }
47  }
```

PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL PORTS COMME

C# Program.cs 2

⊗ Cannot implicitly convert type 'int' to 'bool' (CS0029) [Ln 44, Col 24]

⚠ Dereference of a possibly null reference. (CS8602) [Ln 5, Col 24]

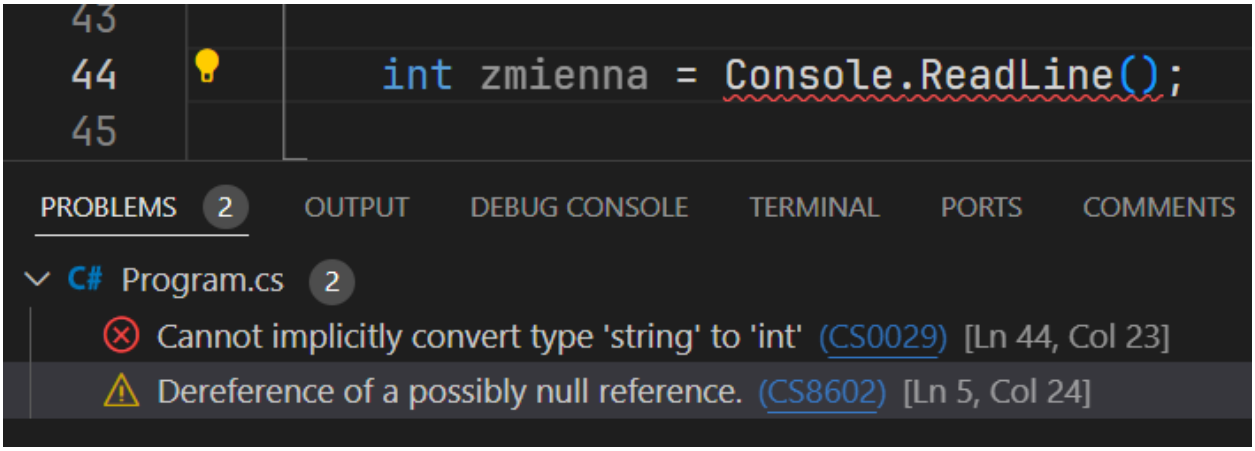
```
44      bool zmienna = true;
45      if (zmienna == "true")
```

PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS Filter (e.g. text, **/*.ts,

C# Program.cs 2

⊗ Operator '==' cannot be applied to operands of type 'bool' and 'string' (CS0019) [Ln 45, Col 13]

⚠ Dereference of a possibly null reference. (CS8602) [Ln 5, Col 24]


43
44
45
int zmienna = Console.ReadLine();
PROBLEMS 2
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
COMMENTS
C# Program.cs 2
Cannot implicitly convert type 'string' to 'int' (CS0029) [Ln 44, Col 23]
Dereference of a possibly null reference. (CS8602) [Ln 5, Col 24]


HTML
JS
C#