LEKCJA 2 – Zmienne i stałe

Zaczynamy tworzenie naszej aplikacji konsolowej.

Utworzenie nowego projektu

Tworzymy nowy projekt (*Create a new project*) aplikacji konsolowej .NET Core (*Console App (.NET Core)*). Aby zawęzić opcje wyszukiwania odpowiedniego typu aplikacji możemy ustawić język wyszukiwania (rozwijane menu z napisem *All languages*) wybierając opcję C#. Następnie możemy ustawić platformę, na której ma działać nasz program (rozwijane menu z napisem *All platforms*) jako Windows, oraz wybrać typ projektu (rozwijane menu z napisem *All project types*) – aplikacja konsolowa (*Console*). Przy wyborze typu projektu należy zwrócić uwagę, aby wybrać opcję z językiem C#, a nie np. Visual Basic (VB). Następnie nadajemy nazwę naszemu projektowi. Przypominamy, że nazwa powinna być w formacie PascalCase (wyrazy pisane łącznie, pierwsza litera każdego wyrazu wielka, nazwa najlepiej w języku angielskim). Jeżeli chcemy możemy zmienić lokalizację projektu (*Location*), aby później łatwo było nam go znaleźć. Tworzymy nasz projekt. Możemy go od razu umieścić na naszym koncie na githubie, tworząc nowe repozytorium.

Umieszczenie projektu na naszym koncie na githubie

Za pośrednictwem Visual Studio

Można to zrobić np. poprzez kliknięcie zakładki *Git* z menu głównego i wybranie opcji *Create Git Repository*. Następnie uzupełniamy dane w oknie *Create a Git Repository*, które nam się pojawi. Zmieniamy w nim typ repozytorium z prywatnego (odznaczamy opcję *Private repository*, jeżeli jest zaznaczona) na publiczny (dostępny dla wszystkich).

Za pośrednictwem strony internetowej

Alternatywnie możemy utworzyć nowe repozytorium na swoim koncie przez stronę internetową, a następnie umieścić w nim nasz projekt.

1. Zaloguj się na swoim koncie github.
2. Kliknij przycisk *New* przy *Repositories*.
3. Wpisz nazwę repozytorium (*Repository name*), najlepiej taką samą jak nazwa naszego projektu. Jeśli chcemy możemy dodać opis projektu (*Description*). Pozostawiamy wybraną opcję *Public* (tworzymy repozytorium publiczne – dostępne dla wszystkich).
4. Można również dodać plik *README*, zaznaczając opcję *Add a README file*, w którym możemy opisać nasz projekt.
5. Od razu możemy dodać też plik *.gitignore* zawierający pliki będące częścią naszego projektu, które nie chcemy jednak, aby zostały umieszczone w naszym repozytorium. Możemy wybrać gotowy szablon pliku (*.gitignore template*). Nie ma na razie szablonu dla platformy .NET Core. Możemy więc wybrać szablon *Visual Studio* lub utworzyć pusty plik (typ szablonu *None*). Utworzony plik można później zmodyfikować do naszych potrzeb.

Przy pomocy konsoli Windows (*cmd* – ang. *Command Line* - Wiersz poleceń)

Można również wykonać te czynności za pośrednictwem konsoli Windows, poprzez użycie odpowiednich poleceń (pogrubiony tekst należy zastąpić właściwymi danymi):

```
cd ścieżka\do\folderu\z\projektem
git init
dotnet new gitignore
git add -A
git commit -m "Initialize project"
```

Jeżeli nasza konsola nie obsługuje poleceń *git*, to być może nie mamy go jeszcze zainstalowanego w naszym systemie. Należy go wówczas pobrać i zainstalować zgodnie ze wskazówkami ze strony. Dotneta zainstalowaliśmy razem z programem Visual Studio, ale jeżeli nasza konsola nie obsługuje polecenia *dotnet*, być może musimy dodać jego lokalizację do *Path* w zmiennych środowiskowych (Klikamy ikonkę Windows i wyszukujemy *Edytuj zmienne środowiskowe systemu*. -> Po kliknięciu na wynik wyszukiwania pojawi nam się okno *Właściwości* systemu, zakładka *Zaawansowane*. -> Klikamy *Zmienne środowiskowe...* -> W *Zmienne systemowe* wybieramy zmienną *Path*, *path* lub *PATH* i klikamy *Edytuj...* -> Klikamy *Nowy*. -> Wpisujemy ścieżkę dostępu do folderu zawierającego plik *dotnet.exe*, np. *C:\Program Files\dotnet*. - > Na zakończanie klikamy *OK*.). Jeżeli nasza konsola nie obsługuje poleceń *git*, pomimo, że jest on zainstalowany w systemie to być może również trzeba dodać go do *Path* w zmiennych środowiskowych, analogicznie jak dotnet. Kiedy już wszystko zainstalujemy i wykonamy powyższe polecenia, to znaczy, że utworzyliśmy już lokalne repozytorium gitowe i umieściliśmy w nim nasz projekt. Teraz musimy wejść na nasze konto github na stronie internetowej i utworzyć nowe puste repozytorium, najlepiej o tej samej nazwie co nasz projekt. Następnie możemy wrócić do naszej konsoli i wykonać kolejne polecenia:

```
git remote add origin url/utworzonego/przed/chwilą/ repozytorium/na/naszym/koncie/github
git push --set-upstream origin master
```

Po ich wykonaniu aktualna wersja naszego projektu została umieszczona w repozytorium na naszym koncie github. Być może będziemy musieli ustawić opcje w globalnej konfiguracji git, takie jak nazwa użytkownika (autora), czy e-mail:

```
git config --global user.name NazwaUżytkownika(naszPodpis)
git config --global user.email NaszAdresEmail
```

Powyższe polecenia ustawią nazwę użytkownika i email, które będą stosowane dla wszystkich repozytoriów utworzonych przez aktualnie zalogowanego użytkownika systemu. Można również ustawić te właściwości tylko dla danego repozytorium, wówczas zamiast *--global*, piszemy *--local*.

Alternatywnie możemy utworzyć nowe repozytorium i umieścić w nim nasz projekt przy pomocy przeznaczonej do tego aplikacji, takiej jak np. *GitHub Desktop* czy *Git GUI*.

Kod projektu

Sekcja „using” – używanie aliasów namespace’ów

W górnej części pliku .cs, jak już wiemy, mamy sekcję „using”. Znajdują się w niej aliasy używanych przez nas bibliotek. Np. polecenia C*onsole.WriteLine* (służące do wypisania linijki tekstu w konsoli), czy *Console.ReadLine* (służące do pobierania z konsoli wpisanej przez użytkownika linijki tekstu) znajdują się w bibliotece *System*. Możemy z nich skorzystać zapisując całą ścieżkę dostępu (*System.Console.WriteLine*, *System.Console.ReadLine*) lub używając właśnie aliasu. Drugi sposób oznacza, że na górze pliku w którym korzystać będziemy z tej/tych funkcji zapisujemy „*using System;*”. Wówczas możemy używać skróconego zapisu nazw funkcji (*Console.WriteLine*, *Console.ReadLine*). Alias oznacza, że do naszej przestrzeni nazw (*namespace*) ładujemy wszystkie klasy, metody itd. znajdujące się pod podaną ścieżką dostępu. Dzięki temu możemy z nich korzystać tak, jakby umieszczone były w naszym namespace’ie. Metody tej warto użyć, jeżeli w danym pliku wielokrotnie używamy jakiejś klasy/metody lub wielu elementów z danej przestrzeni nazw, aby skrócić ich zapis. Jeżeli natomiast takie użycia są jednostkowe, lepiej korzystać z pełnych ścieżek dostępów.

Metoda *Main*

Jak już wiemy jest to główna metoda programu. Od niej zaczyna on swoją pracę i wykonuje kolejne znajdujące się w niej polecenia. Jeżeli więc na jej początku umieścimy kilka poleceń *Console.WriteLine(„tekst”)*, to właśnie umieszczony w nich tekst wyświetli się jako pierwszy w konsoli.

Polecenia Console.WriteLine, Console.ReadLine

Tych metod można użyć do stworzenia prostego menu:

```
Console.WriteLine(„Witamy w aplikacji Zwierzak”);

Console.WriteLine();

Console.WriteLine(„Co chcesz zrobić ze swoim zwierzakiem?”);

Console.WriteLine(„\t1. Nakarm Zwierzaka.“);

Console.WriteLine(„\t2. Baw się ze Zwierzakiem.”);

Console.WriteLine(„\t3. Zabierz Zwierzaka na spacer.”);

Console.WriteLine(„\t4. Połóż Zwierzaka spać.”);

Console.WriteLine();

Console.Write(„Wybierz co chcesz zrobić wpisując numer
czynności (1, 2, 3, 4): „);

string action = Console.ReadLine();

int actionNo;

Int32.TryParse(action, out actionNo);
```

Powyższy kod spowoduje wyświetlenie się w konsoli:

```
Witamy w aplikacji Zwierzak

Co chcesz zrobić ze swoim zwierzakiem?
        1. Nakarm Zwierzaka.
        2. Baw się ze Zwierzakiem.
        3. Zabierz Zwierzaka na spacer.
        4. Połóż Zwierzaka spać.

Wybierz co chcesz zrobić wpisując numer czynności (1, 2, 3, 4):
```

Metoda *Console.Write* spowoduje wyświetlenie w konsoli podanego jej tekstu, bez przejścia do nowej linii.

Po wyświetleniu tekstu program czeka na wpisanie przez użytkownika w konsoli dowolnego tekstu i wciśnięciu klawisza Enter (metoda *Console.ReadLine*). Wpisaną przez użytkownika wartość zapisujemy w zmiennej *action* (*string action = Console.ReadLine();*). Jest to zmienna typu *string*, czyli typu tekstowego (przechowująca tekst). Następnie tworzymy zmienną typu *int*, czyli mogącą przechowywać liczby całkowite z pewnego zakresu. Na koniec przy pomocy metody *Int32.TryParse(string number, out int no)* próbujemy dokonać konwersji z typu *string* na *int* (liczbę zapisaną w formie tekstu przekształcamy w wartość liczbową), a wynik tej operacji zapisujemy w zmiennej wyjściowej (*out*).

Stałe

Stała jest to taki element aplikacji, który raz zdefiniowany nie może zostać zmieniony. Tworzymy ją podając kolejno słowo kluczowe *const*, typ stałej (np. *string* lub *int*) i jej nazwę. Następnie po znaku „=" możemy zadeklarować tą stałą konkretną wartością.

```
const string APPLICATION_NAME = "Zwierzak";

const int NUMBER_OF_MEALS_PER_DAY = 2;
```

Jest to jedyne miejsce w kodzie w którym stałej możemy przypisać wartość. Wartość ta jest niezmienna. Gdybyśmy w dalszej części programu próbowali stałej przypisać nową wartość, to otrzymamy błąd kompilacji, nawet jeśli ta nowa wartość będzie taka sama jak obecna. Przypominamy, że nazwy stałych tworzymy najczęściej używając konwencji nazewnictwa UPPER_CASE. Stałe często są w pewien sposób globalne dla całej klasy lub dla większej części projektu. W związku z tym można jej definicję umieścić poza metodą *Main* i w jej zapisie typ stałej poprzedzić modyfikatorem dostępu. Jeżeli stała będzie globalna, oznaczać to będzie, że zarezerwowane zostanie dla niej miejsce w pamięci, na cały okres działania programu. W przeciwnym razie przestanie ona istnieć w momencie gdy program wyjdzie poza zakres w którym została utworzona (najczęściej poza nawiasy klamrowe między którymi znajduje się jej definicja).

Zmienne

Podobny do stałej element aplikacji, umożliwiający przechowywanie wartości w znanym miejscu pamięci aplikacji, jednak w odróżnieniu od stałej przechowywana tam wartość może się wielokrotnie zmieniać. Aby używać zmiennej musimy ją najpierw zadeklarować. W tym celu podajemy typ danych jakie przechowywać będzie nasza zmienna i jej nazwę (zapisaną najczęściej w konwencji camelCase). Na początku przed typem można jeszcze podać modyfikator dostępu. Utworzoną zmienną można teraz zainicjalizować konkretną wartością. Można to zrobić od razu z deklaracją, czyli analogicznie jak przy stałej, po nazwie zmiennej wstawić znak przypisania („=") i podać wartość jaką chcemy przypisać do zmiennej, zakańczając jak zawsze linię średnikiem. Można również po nazwie zmiennej wstawić średnik, a inicjalizacji dokonać w dalszej części programu. Kiedy do zdefiniowanej już zmiennej będziemy chcieli przypisać jakąś wartość (wszystko jedno czy po raz pierwszy, czy kolejny) wystarczy podać nazwę zmiennej, znak przypisania („="), wybraną wartość i średnik. Np.:

```
string petName;

int actionNo = 0;

…

petName = "Rex";

…

actionNo = 1;

…

actionNo = 2;
```

Zmienna przestaje istnieć w momencie gdy program wyjdzie poza zakres w którym została utworzona (najczęściej poza nawiasy klamrowe między którymi znajduje się jej definicja). Wyjątek stanowią zmienne globalne, które istnieją przez cały czas trwania programu.