Warsztaty C#, Dzień 5

v3.0

Plan

1. Warsztaty

Warsztaty

Cel Warsztatów

Cel warsztatów to stworzenie konsolowego menadżera zadań.

Aplikacja musi posiadać możliwość wpisywania komend i wykonywania odpowiednich operacji i zależności od komendy, która została wpisana.

Aplikacja musi posiadać następujące funkcje:

- Wyjście z aplikacji na polecenie `exit`.
- Dodanie i usuwanie zadania.
- Wyświetlenie wszystkich zadań.
- Zapisywanie i wczytywania zadań z pliku.

Model Danych

Model danych musi się składać z encji o nazwie `TaskModel`.

Encja `TaskModel` musi zawierać następujące atrybuty.

- Opis - Wymagany.
- Datę Rozpoczęcia - Wymagana.
- Datę Zakończenia - Niewymagana, jeśli zadanie jest całodniowe.
- Flaga Zadanie Całodniowe - Niewymagana, domyślnie zadanie nie jest całodniowe.
- Flagę Zadanie Ważne - Niewymagana, domyślnie zadanie nie jest ważne.

Jeśli flaga zadanie całodniowe jest ustawiona, data zakończenia nie jest wymagana.

Flaga

Flaga to zmienna, która przechowuje jedną z kilku oczekiwanych wartości.

Flaga może być różnego typu.

Jeśli ma przechowywać jedną z dwóch, zmienna może być typu `bool`.

Jeśli flaga ma przechowywać jedną z kilku możliwych wartości, najlepszym rozwiązaniem będzie użycie typu `enum`.

Github

Zanim rozpoczniesz realizację warsztatów, utwórz repozytorium, w którym będziesz przechowywał kod źródłowy aplikacji.

Wejdź na swoje konto Github i za pomocą przycisku `New Repository` utwórz nowe repozytorium.

Wpisz nazwę repozytorium oraz wybierz plik `.gitignore` dla programu Visual Studio.

Nazwa repozytorium może być dowolna.

Pętla

W metodzie `Main` pobierz polecenie z konsoli.

Utwórz pętlę `do while`, która zakończy się, jeśli wpiszemy polecenie `exit`.

Pobranie polecenia musi być poprzedzone odpowiednimi komunikatami.

Console Color

Ponieważ cała aplikacja opiera się na danych wyświetlanych w konsoli, stwórz klasę, która pozwoli w prosty sposób wyświetlać tekst w różnych kolorach.

Utwórz klasę `ConsoleEx` wraz z metodami `Write` i `WriteLine`, które odpowiednio wyświetlą tekst bez i ze znakiem nowego wiersza podany w parametrze.

Obie metody muszą przyjmować drugi parametr typu `ConsoleColor`, który ustawi kolor wyświetlanego tekstu.

Zarówno klasa, jak i metody muszą być statyczne.

Klasa TaskModel

Klasa `TaskModel` będzie reprezentować model zadania.

Klasa musi zawierać właściwości niezbędne do przechowywania danych wymaganych przez cele projektu.

Dobierz odpowiednie typy do poszczególnych właściwości.

Dla właściwości niewymaganych możesz skorzystać z typów `nullable`.

Właściwości wymagane muszą zostać uzupełnione w konstruktorze.

Release

Kiedy przygotujemy szkielet aplikacji, serwis Github umożliwia stworzenie wydania.

Release to działające wersja aplikacji, posiadająca wersję.

Aby wydać Release, trzeba przejść do zakładki Release na stronie Github, wpisać tytuł i numer wersji.

W tym przypadku może to być na przykład "Szkielet Aplikacji".

Lista na Wpisy

W klasie `Program` musimy utworzyć pole statyczne typu `List` z parametrem generycznym `TaskModel`.

Lista będzie przechowywać zadania dodawane w aplikacji.

Polecenie AddTask

Utwórz metodę `AddTask`.

Metoda musi pobrać informacje niezbędne do utworzenia obiektu zadania.

Pobranie informacji musi zostać poprzedzone odpowiednimi komunikatami.

Pobrane dane muszą zostać rzutowane na odpowiedni typy, tak, aby na ich postawie utworzyć nowy obiekt typu `TaskModel`.

Utworzony obiekt powinien zostać dodany do listy.

W pętli `do while` utwórz instrukcję `if`, która wykona się na polecenie `add`.

W instrukcji `if` uruchom metodę `AddTask`.

Pobranie Danych

Pobranie danych z konsoli może być zrealizowane na kilka sposób, do Ciebie należy wybór jednego z nich.

Możesz pobierać każdą potrzebną informację osobno, poprzedzając ją stosownym komunikatem.

Możesz zrealizować pobieranie wszystkich informacji w jednym wierszu przedstawiając schemat.

W naszym przypadku istnieją dwie możliwości utworzenia nowego zadania - zadania zwykłe z datą początku i końca oraz zadania całodniowe.

Schematy mogą być następujące:

- opis;data_rozpoczęcia;[ważność-opcjonalne] - dla zdarzenia całodniowego
- opis;data_rozpoczęcia;data_zakończenia;[ważność-opcjonalne] - dla zdarzenia z konkretnymi datami.

Po pobraniu danych trzeba sprawdzić, do którego schematu pasują oraz czy wszystkie dane są prawidłowe.

Polecenie Remove Task

Utwórz metodę `RemoveTask`.

Metoda musi pobrać informacje niezbędne do usunięcia zadania.

Pobranie informacji musi zostać poprzedzone odpowiednimi komunikatami.

Decyzja w jaki sposób wskazywać zadanie do usunięcia należy do Ciebie.

Jednym ze sposobów jest wyświetlanie list zadań z indeksem w jakim znajdują się w kolekcji oraz prośba o podanie indeksu zadania do usunięcia.

Wybrany obiekt powinien zostać usunięty z listy.

W pętli `do while` utwórz instrukcję `if`, która wykona się na polecenie `remove`.

W instrukcji `if` uruchom metodę `RemoveTask`.

Wyświetlenie Zadań

Utwórz metodę `ShowTasks`.

Metoda musi wyświetlać wszystkie zadania w tabeli.

Tabela powinna mieć nagłówki, a kolumny muszą być wyrównane.

Zadania powinny się wyświetlać posortowane według daty rozpoczęcia. Zadania oznaczone flagą ważne powinny zostać wyjątkowo wyświetlone na początku.

W pętli `do while` utwórz instrukcję `if`, która wykona się na polecenie `show`.

W instrukcji `if` uruchom metodę `ShowTasks`.

Zapisywanie do Pliku

Utwórz metodę `SaveTasks`.

Metoda musi zapisywać wszystkie zadania z listy do pliku.

Żeby zapisać dane do pliku trzeba opracować format zapisu.

Zapis musi uwzględniać wszystkie dane i odpowiednio je reprezentować.

Dane powinny zostać zapisane w pliku `Data.csv` w katalogu roboczym aplikacji.

Możemy skorzystać z formatu `csv` czyli `comma separated values`.

W tym formacie każdy obiekt przechowywany jest w osobnym wierszu, a poszczególne wartości oddzielone są przecinkiem.

W pętli `do while` utwórz instrukcję `if`, która wykona się na polecenie `save`.

W instrukcji `if` uruchom metodę `SaveTasks`.

CSV

Nasz pojedynczy wiersz, będzie zawierał 5 pól, oddzielonych przecinkiem.

Wzór będzie następujący:

```
opis,data_roz,data_zak,całodniowe,ważne
```

Dane zapisane w pliku dla zwykłego zadania będą wyglądały następująco:

```
opis,2018-03-09,2018-03-09,false,false
```

Dla uproszczenia pominięto godziny w datach, które w programie oczywiście będą.

W pliku `csv`, dane muszą być zawsze umieszczone w odpowiednich kolumnach. Kiedy brakuje nam wartości w kolumnie pozostawiamy ją pustą.

Zadanie całodniowe zapisane w pliku będzie wyglądało następująco:

```
opis,2018-03-09,,true,false
```

Zwróć uwagę na brak wartości w miejscu daty zakończenia zadania.

Konwersja Obiektu

Potrzebny dla zapisu do pliku format możemy zaimplementować w wielu miejscach.

Możemy w obiekcie `TaskModel` utworzyć metodę `Export`, która zwróci obiekt string z danymi obiektu, na którym zostanie wywołana.

W metodzie `SaveTasks` będziemy wtedy wywoływać metodę `Export` na wszystkich obiektach listy, a zwrócone ciągi znaków zapisywać do pliku.

Do łączenia dużych ilości łańcuchów znaków należy użyć klasy `StringBuilder`.

Możemy również zając się konwersją bezpośrednio w metodzie `SaveTasks`.

Wczytywanie z Pliku

Utwórz metodę `LoadTasks`.

Wczytywanie z pliku powinno odbywać się analogicznie do zapisywania.

Sprawdź czy na dysku istnieje odpowiedni plik, jeśli tak pobierz dane wiersz po wierszu.

Na podstawie danych utwórz obiekty `TaskModel`, które powinny zostać dodane do kolekcji.

W pętli `do while` utwórz instrukcję `if`, która wykona się na polecenie `load`.

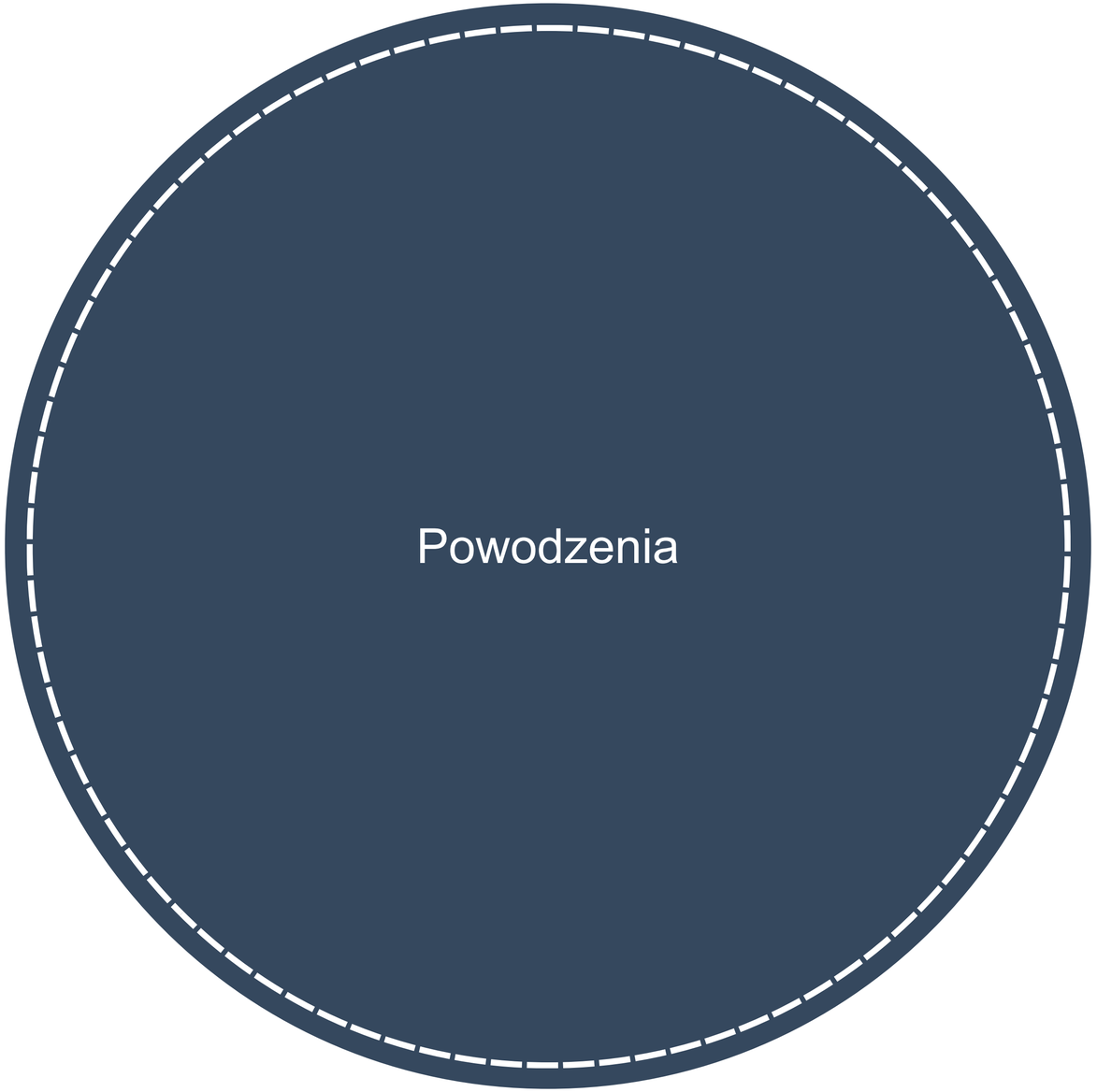
W instrukcji `if` uruchom metodę `LoadTasks`.

Kolejne Funkcjonalności

Aplikację można rozbudować o kolejne funkcjonalności takie jak:

- Wyświetlanie zbliżających się zadań.
- Filtrowanie zadań.
- Otwieranie innych plików z zadaniami.

Możesz wprowadzać nowe funkcjonalności w celu dalszego ulepszania aplikacji.


Powodzenia