CommunityToolkit.Mvvm — ToDo

Czas: 2 × 45 min | praca indywidualna | Wykład 2.06

Cel: Przepiszesz listę zadań z zadania 2 na MVVM z pakietem CommunityToolkit.Mvvm: właściwości i komendy wygeneruje generator kodu, a okno nie będzie miało żadnej metody obsługi zdarzeń.

Po zajęciach potrafisz:

- instaluje i stosuje pakiet CommunityToolkit.Mvvm
- używa [ObservableProperty], [RelayCommand] i atrybutów powiadomień
- przekazuje element listy do komendy przez CommandParameter
- porównuje objętość kodu ręcznego i generowanego

Sprzęt i materiały

- komputer z Windows 11 i dostępem do internetu
- pendrive lub konto w repozytorium do zapisania projektu

Oprogramowanie

- **Visual Studio Community 2026 (obciążenie „Programowanie aplikacji klasycznych platformy .NET”)** — darmowe dla uczniów i nauczycieli; w pracowni zainstalowane visualstudio.microsoft.com/pl/downloads
- **.NET 10 SDK (LTS)** — instalowany razem z Visual Studio; sprawdzenie: `dotnet --version` dotnet.microsoft.com/pl-pl/download/dotnet/10.0
- **Pakiet NuGet CommunityToolkit.Mvvm 8.4** — instalacja: Zarządzaj pakietami NuGet nuget.org/packages/CommunityToolkit.Mvvm

Przygotowanie przed zajęciami

- Projekt ToDo z zadania 2 (kopia w nowym folderze lub nowa gałąź Git).

Bezpieczeństwo

- Zapisuj projekt w swoim folderze (np. `C:\Users\<login>\source\repos`), a na koniec zajęć skopiuj go na pendrive lub wyślij do repozytorium.
- Nie uruchamiaj programów i pakietów z nieznanych źródeł; pakiety NuGet instaluj tylko te, które wskazuje zadanie.
- Rób przerwy dla oczu (zasada 20-20-20) — praca z projektantem okien wymaga długiego patrzenia w ekran.

Przebieg zadania

1 Pakiet i ViewModele

Zainstaluj CommunityToolkit.Mvvm.

```
[ObservableProperty]
[NotifyCanExecuteChangedFor(nameof(DodajCommand))]
private string _nowyTytul = "";

[RelayCommand(CanExecute = nameof(MoznaDodac))]
private void Dodaj()
{
    Zadania.Add(new ZadanieViewModel { Tytul = NowyTytul.Trim(), Termin = NowyTermin });
    NowyTytul = "";
}
```

1. `ZadanieViewModel` (partial, ObservableObject): pola `_tytul`, `_termin`, `_wykonane` z [ObservableProperty].
2. Właściwość `PoTerminie` (niewykonane z terminem w przeszłości) odświeżana w `OnWykonaneChanged` i `OnTerminChanged`.
3. `ListaViewModel`: kolekcja, `_nowyTytul` z `[NotifyCanExecuteChangedFor(nameof(DodajCommand))]`, `_nowyTermin`.
4. Komendy: `Dodaj` (CanExecute: niepusty tytuł), `Usun(ZadanieViewModel?)`, `UsunWykonane` (aktywna, gdy są wykonane).

2 Widok bez zdarzeń

Przerób okno.

1. DataContext w XAML; przyciski z `Command`; usuń metody z code-behind.
2. Przycisk ✕ w szablonie każdego zadania: komenda okna przez `RelativeSource AncestorType=ListBox` i `CommandParameter="{Binding}"`.
3. Klawisz Delete usuwa zaznaczone zadanie (`KeyBinding` z `CommandParameter` = zaznaczony element).
4. Termin po terminie na czerwono — `DataTrigger` na `PoTerminie` (kolor domyślny w stylu, nie na kontrolce).

3 Porównanie

Oceń zmianę.

1. Policz linie kodu C# w wersji z zadania 2 i w nowej wersji.
2. Podejrzyj wygenerowany kod: Eksplorator rozwiązań → Zależności → Analizatory → CommunityToolkit.Mvvm.SourceGenerators.
3. Zapisz, jaką właściwość i komendę wygenerowało pole `_nowyTytul` i metoda `UsunWykonane`.
4. Zatwierdź pracę w Git.

W sprawozdaniu umieść

- ☐ kod ViewModeli
- ☐ XAML okna
- ☐ porównanie liczby linii
- ☐ fragment wygenerowanego kodu z opisem

Pytania kontrolne

1. Dlaczego klasa z [ObservableProperty] musi być partial?
2. Jaką nazwę ma komenda wygenerowana z metody UsunWykonane?

3. Dlaczego ustawienie Foreground bezpośrednio na TextBlock blokuje DataTrigger ze stylu?
4. Jak przekazać do komendy element listy?

Kryteria oceny

Kryterium	Punkty
ViewModele z [ObservableProperty] i [RelayCommand]	7
CanExecute i powiadomienia	4
Widok bez zdarzeń, przycisk w szablonie, Delete	5
Porównanie i pytania	4
Razem	**20**

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).