Operacje asynchroniczne

Czas: 2 × 45 min | praca indywidualna | Wykład 2.11

Cel: Zbudujesz aplikację liczącą liczby pierwsze w dużym zakresie bez zawieszania interfejsu: z paskiem postępu, anulowaniem i pomiarem czasu.

Po zajęciach potrafisz:

- przenosi obliczenia do wątku roboczego Task.Run i czeka na wynik await
- raportuje postęp IProgress<T>
- anuluje operację CancellationToken
- unika typowych błędów: .Result, async void, dostęp do kontrolek z innego wątku

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

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 Obliczenia

Projekt WPF `Pierwsze`; klasa statyczna `Obliczenia` bez WPF.

1. `CzyPierwsza(int n)` z dzieleniem tylko przez nieparzyste dzielniki do √n.
2. `LiczPierwsze(od, do, IProgress<int>?, CancellationToken)` — raportuje zmianę procentu, co 1024 liczby sprawdza token.
3. Sprawdź: do 100 → 25, do 1 000 000 → 78 498.

2 Wersja blokująca

Najpierw zobacz problem.

1. Napisz komendę synchroniczną liczącą do 20 000 000 bez Task.Run.
2. Uruchom i spróbuj przesunąć okno — zapisz, co się dzieje i ile trwa obliczenie.

3 Wersja asynchroniczna

`PierwszeViewModel` z CommunityToolkit.Mvvm.

```
var postep = new Progress<int>(p => Postep = p);
int od = Od, doL = Do;
int ile = await Task.Run(() =>
    Obliczenia.LiczPierwsze(od, doL, postep, _cts.Token));
```

1. Komenda `LiczAsync`: `await Task.Run(() => Obliczenia.LiczPierwsze(...))` — wartości Od i Do skopiuj do zmiennych lokalnych.
2. `new Progress<int>(p => Postep = p)` → ProgressBar.
3. Komenda `Anuluj` (aktywna tylko w trakcie) i obsługa `OperationCanceledException`.
4. Czas obliczeń ze `Stopwatch` w komunikacie.

4 Pułapki

Sprawdź celowo błędy (i cofnij je).

1. Zamień `await` na `.Result` — co się dzieje z oknem?
2. W Task.Run spróbuj ustawić właściwość ProgressBar bezpośrednio — zapisz wyjątek.
3. Zatwierdź pracę w Git.

W sprawozdaniu umieść

- ☐ kod Obliczenia i ViewModelu
- ☐ porównanie wersji blokującej i asynchronicznej (czas, zachowanie okna)
- ☐ opis skutków pułapek
- ☐ zrzut w trakcie liczenia

Pytania kontrolne

1. Dlaczego obliczenie w wątku interfejsu zawiesza okno?
2. Po co kopiować Od i Do do zmiennych lokalnych przed Task.Run?
3. W którym wątku wykonuje się lambda przekazana do Progress<int>?
4. Dlaczego .Result może zakleszczyć aplikację?

Kryteria oceny

Kryterium	Punkty
Klasa Obliczenia z postępem i anulowaniem	5
Wersja blokująca — obserwacje	2
ViewModel asynchroniczny	6
Pułapki	4
Pytania	3
Razem	**20**

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