Programowanie asynchroniczne

Wprowadzenie

- Model programowania asynchronicznego w .Net bazuje na:
  - Promise Model of Concurrency (en.wikipedia.org/wiki/Futures_and_promises)
- Realizuje to za pomocą wzorca:
  - Task-based Asynchronous Pattern (TAP)
- Operacje asynchroniczne są modelowane przez klasy:
  - **`Task`** : operacja niezwracająca wartości
  - **`Task<T>`** : operacja zwracająca wartość typu **`T`**
- Słowo kluczowe **`await`** pozwala oddać sterowanie programem do funkcji wywołującej funkcję wykonującą **`await`**

Przykład obsługi wejścia/wyjścia

- W tego typu zastosowaniach zwykle wywołujemy metodę I/O z **`await`**, która oddaje wynik typu **`Task`** lub **`Task<T>`** wewnątrz metody **`async`**

```
private readonly HttpClient _Client = new HttpClient();

downloadButton.Clicked +=
   async (o, e) =>
   {
      var data = await _Client.GetStringAsync(URL);
      DoSomethingWithData(data);
   };
```

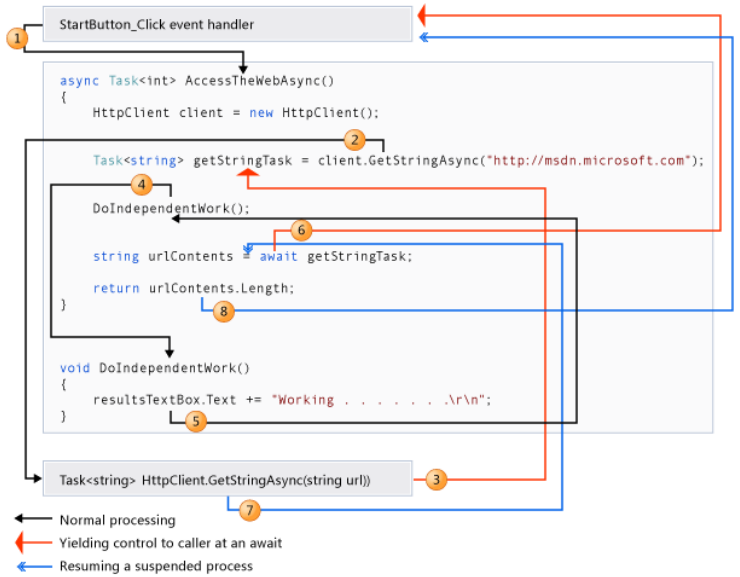
Przykład obsługi czasochłonnych obliczeń

- W tego typu zastosowaniach zwykle wywołujemy czasochłonną metodę z **`await`**, którą startujemy w tle za pomocą **`Task.Run`**

```
private DamageResult CalculateDamageDone()
{
    // expensive calculation
}

calculateButton.Clicked +=
      async (o, e) =>
      {
         var damageResult = await Task.Run(
             () => CalculateDamageDone()
         );
         DisplayDamage(damageResult);
      };
```

Jak to działa ?



Zapamiętaj, że ...

- Metody **`async`** powinny posiadać wywołanie słowa kluczowego **`await`** w swoim ciele, inaczej nie będzie kończył rozpoczętych zadań
- Powinniśmu dodawać przyrostek Async do wszystkich tworzonych metod asynchronicznych
- Typ **`async void`** powinien być stosowany tylko do zdarzeń:
  - Trudne w testowaniu
  - Zdarzenia nie posiadają typu wynikowego
- Ze względu na mechanizm odroczonego wykonania ostrożnie stosujemy metody asynchroniczne w LINQ