

Wykład 11

Wątki, Timer

Wątki – programowanie współbieżne

2 cores





Wątki

Wątki – klasa **BackgroundWorker** – umożliwiają delegowanie pewnych operacji do wątków pracujących współbieżnie (równolegle) z główną aplikacją.

Za ich pomocą pewne operacje mogą być wykonywane w tle,

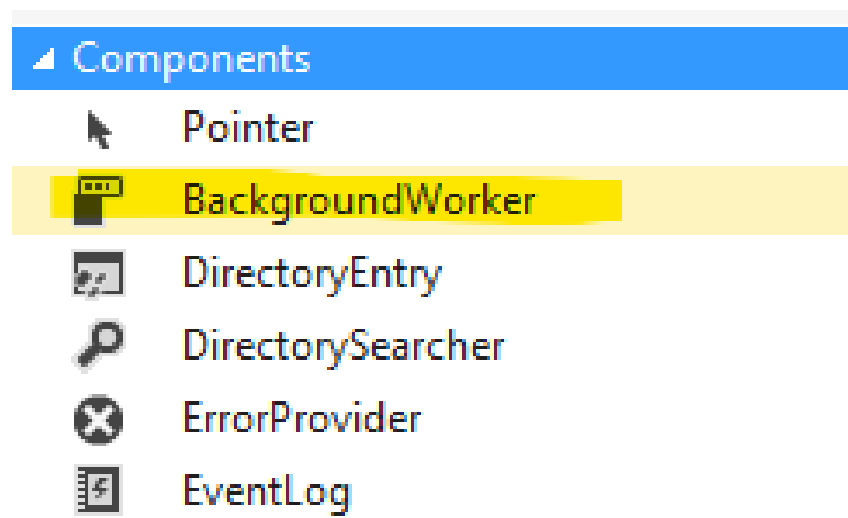
Uruchomienie wątku:

```
backgroundWorker1.WorkerReportsProgress = true;  
backgroundWorker1.WorkerSupportsCancellation = true;  
backgroundWorker1.RunWorkerAsync();
```

Wątki

Należy pamiętać o dodaniu komponentu **BackgroundWorker** do projektu.

Utworzony zostanie instancja tej klasy – w naszym przykładzie - obiekt `backgroundWorker1`

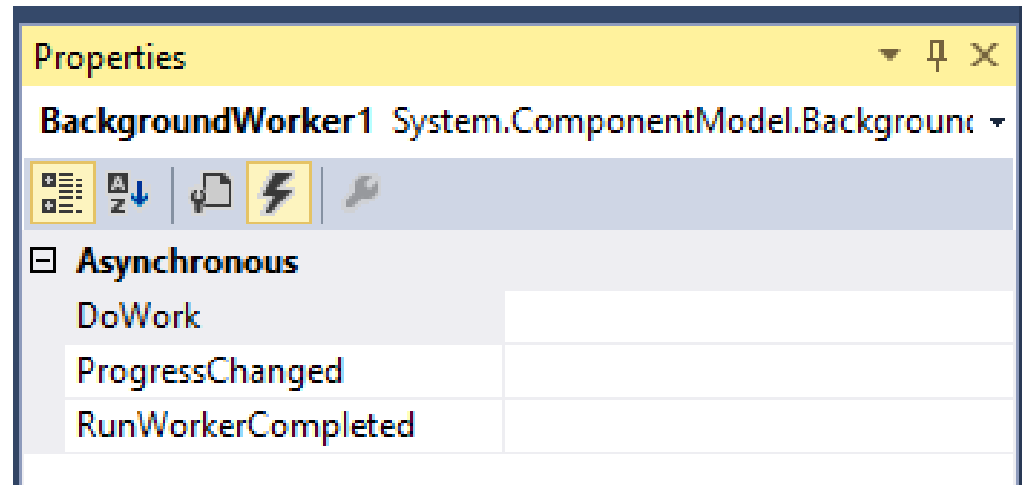




Wątki

Klasa **BackgroundWorker** posiada zdarzenia:

- ✓ **DoWork** – w którym umieszczamy operacje do wykonania w tle
- ✓ **RunWorkerCompleted** – które wywoływane jest po zakończeniu pracy wątku.
- ✓ **ProgressChanged** – który wykorzystać można do raportowania postępów wątku do programu głównego (pasek postępu)





Wątki

DoWork – w którym umieszczamy operacje do wykonania w tle.

Uwaga: z wnętrza metody obsługi zdarzenia DuWork nie mamy możliwości sięgnięcia do kontrolek okna głównego programu.

Metoda DoWork zwraca wartość za pośrednictwem zmiennej „e”

Np.: `e.Result = wynik_obliczen;`



Wątki

Metoda **DuWork** może otrzymać parametry „zapakowane” w argument „e” typu DoWorkEventArgs.

Np.:

Wywołanie wątku z argumentem:

```
backgroundWorker1.RunWorkerAsync(50);
```

Odebranie argumentu w metodzie DoWork:

```
int a = (int)e.Argument;
```



Wątki

ProgressChanged – metoda, którą wykorzystać można do raportowania postępów wątku do programu głównego (pasek postępu)

Metoda **ProgressChanged** musi być wywoływana cyklicznie wewnątrz metody **DoWork()** z parametrem mówiącym o postępie wątku.

```
backgroundWorker1.ReportProgress (i);
```

Z wewnątrz metody **ProgressChanged** możemy sięgnąć do kontrolek procesu głównego.

Np.: `progressBar1.PerformStep();`

lub

```
progressBar1.Value = e.ProgressPercentage;
```




Wątki

Wstrzymanie pracy wątku:

`Thread.Sleep (czas);`

czas – podany w milisekundach

Dodać należy przestrzeń nazw: `using System.Threading;`



Wątki

Przerywanie wątku

Do przerywania pracy wątku służy metoda **CancelAsync()**

Np.: `backgroundWorker1.CancelAsync();`

UWAGA: nie wymusza ona bezwarunkowego przerywania wątku – stanowi tylko informację, że wątek powinien zakończyć pracę. Sposób jego zakończenia należy zdefiniować wewnątrz wątku.



Wątki

Oprogramowanie przerwania wątku:

W metodzie `DoWork` dodajemy reakcję na własność **`CancellationPending`** (`true` oznacza żądanie przerwania wątku)

```
if (backgroundWorker1.CancellationPending == true) {  
    //akcja wątku  
} else {  
    e.Cancel = true;  
    break;  
}
```

W takim przypadku ustawiamy pole **`Cancel`** zdarzenia „e” na **`true`** (potwierdzamy zamknięcie) i przerywamy pracę metody **`DoWork`** (polecenie `break`)



Wątki

Oprogramowanie przerywania wątku:

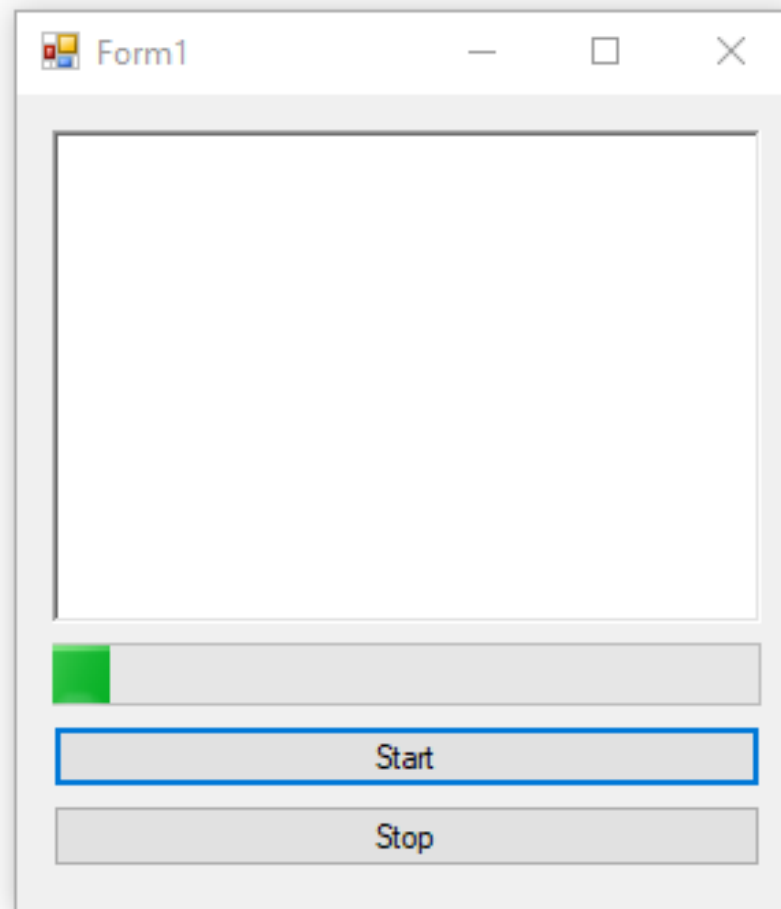
W metodzie **RunWorkerCompleted**, która kończy pracę wątku.

```
if (e.Cancelled == true) {  
    richTextBox1.Text = "Błąd: wątek został zatrzymany";  
} else {  
    // normalne zakończenie pracy wątku  
}
```

Jeżeli pole **Canceled** zdarzenia „e” jest równe **true** (wątek został zamknięty) reagujemy na ten fakt – np. wypisując komunikat o błędzie. W przeciwnym razie kończymy wątek normalnie

PRZYKŁAD:

Generowanie tablicy stringów w osobnym wątku (sztucznie spowolnione)





Wątki

Uruchamianie wątku – po kliknięciu przycisku „Start”

```
private void button1_Click(object sender, EventArgs e)
{
    backgroundWorker1.WorkerReportsProgress = true;
    backgroundWorker1.WorkerSupportsCancellation = true;
    backgroundWorker1.RunWorkerAsync(50);
}
```

Wątki

```
private void backgroundWorker1_DoWork(object sender, DoWorkEventArgs e)
{
    string[] tab = new String[100];
    int sztuczneOpuznienie = (int)e.Argument;
    for (int i = 0; i < 100; i++)
    {
        tab[i] = "Wiersz: " + i;
        Thread.Sleep(sztuczneOpuznienie);
        backgroundWorker1.ReportProgress(i + 1);
    }
    e.Result = tab;
}
```

Zdarzenie DoWork – wykonywane w trakcie pracy wątku – jego główne zadania

Wersja pierwsza – bez możliwości przerwania wątku



Wątki

```
private void backgroundWorker1_RunWorkerCompleted(object sender,
                                                    RunWorkerCompletedEventArgs e)
{
    richTextBox1.Lines = (string[])e.Result;
}
```

Zdarzenie RunWorkerCompleted – wykonywane po zakończeniu wątku – zwraca wygenerowaną tablicę do pola tekstowego

Wersja pierwsza – bez możliwości przerwania wątku



Wątki

Zdarzenie ProgressChanged – sterowanie paskiem postępu

```
private void backgroundWorker1_ProgressChanged(object sender,
                                                ProgressChangedEventArgs e)
{
    //progressBar1.PerformStep();
    progressBar1.Value = e.ProgressPercentage;
}
```

Zdarzenie to wywoływane jest w DoWork:
backgroundWorker1.ReportProgress(i + 1);



Wątki

Przerwanie wątku

```
private void button2_Click(object sender, EventArgs e)
{
    backgroundWorker1.CancelAsync();
}
```

Wymaga zmian w DoWork i
RunWorkerCompleted

Patrz poniżej



Wątki

```
private void backgroundWorker1_DoWork(object sender, DoWorkEventArgs e)
{
    string[] tab = new String[100];
    int sztuczneOpuznienie = (int)e.Argument;
    for (int i = 0; i < 100; i++)
    {
        if (backgroundWorker1.CancellationPending == false)
        {
            tab[i] = "Wiersz: " + i;
            Thread.Sleep(sztuczneOpuznienie);
            backgroundWorker1.ReportProgress(i + 1);
        }
        else
        {
            e.Cancel = true;
            break;
        }
    }
    e.Result = tab;
}
```

Zdarzenie DoWork –
wykonywane w trakcie pracy
wątku – jego główne zadania

Wersja druga – z możliwością przerwania wątku



Wątki

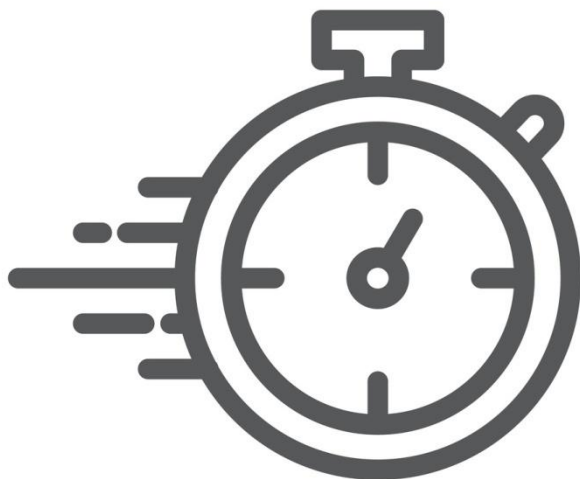
```
private void backgroundWorker1_RunWorkerCompleted(object sender,
                                                    RunWorkerCompletedEventArgs e)
{
    if (e.Cancelled == true)
    {
        richTextBox1.Text = "Błąd: wątek został zatrzymany";
    }
    else
    {
        richTextBox1.Lines = (string[])e.Result;
    }
}
```

Zdarzenie RunWorkerCompleted – wykonywane po zakończeniu wątku – zwraca wygenerowaną tablicę do pola tekstowego

Wersja druga – z możliwością przerwania wątku

Timer

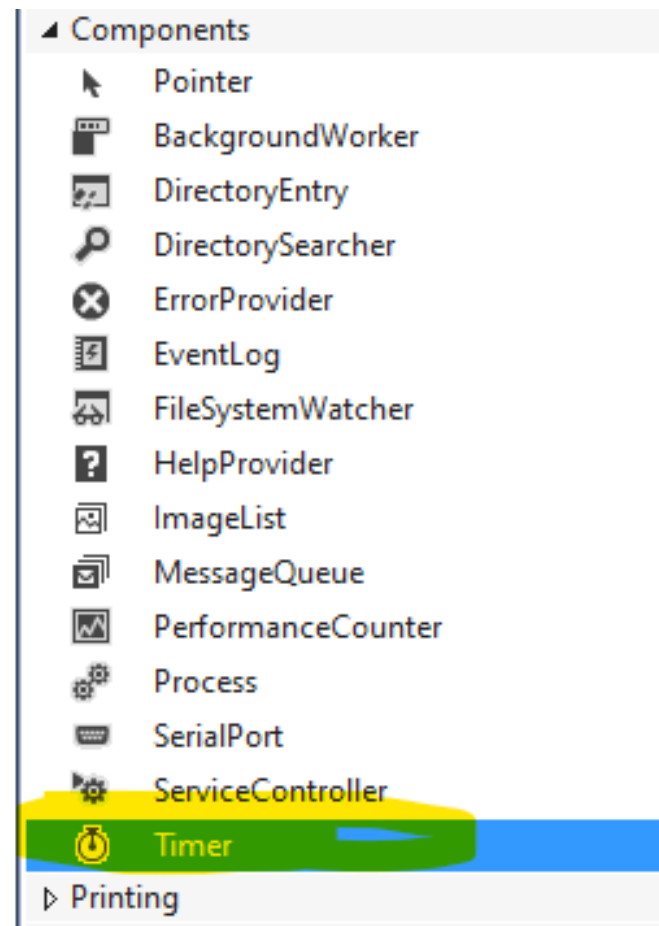
Timer jest wątkiem pracującym w tle który co pewien czas wywołuje jakąś akcję



Timer

Rozpocząć należy od dodania komponentu **Timer** do projektu.

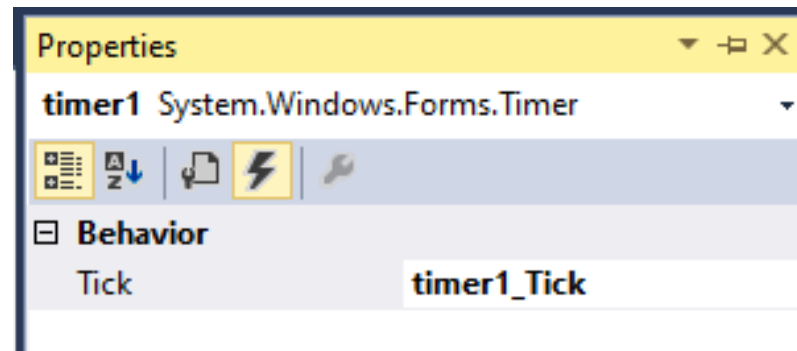
Utworzony zostanie
instancja tej klasy –
w naszym
przykładzie - obiekt
timer1



Timer

Timer posiada metodę **Tick** która wywoływana jest co określony interwał czasu.

Oprogramowanie timera to właściwie oprogramowanie tej metody



```
private void timer1_Tick(object sender, EventArgs e)
{
    //to co mamy cyklicznie wykonać
}
```

Timer

Ustawienia timera:

- **Enabled** – timer włączony / zatrzymany
- **Interval** – odstęp pomiędzy wywołaniami metody Tick – w milisekundach)

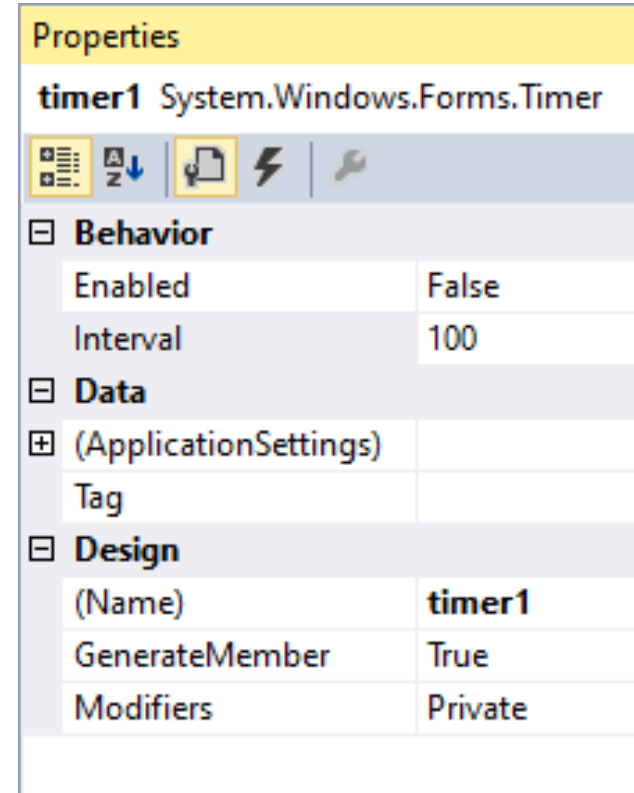
Z poziomu kodu:

- Uruchomienie timera

```
timer1.Enabled = true;
```

- Zmiana interwału

```
timer1.Interval = 100;
```

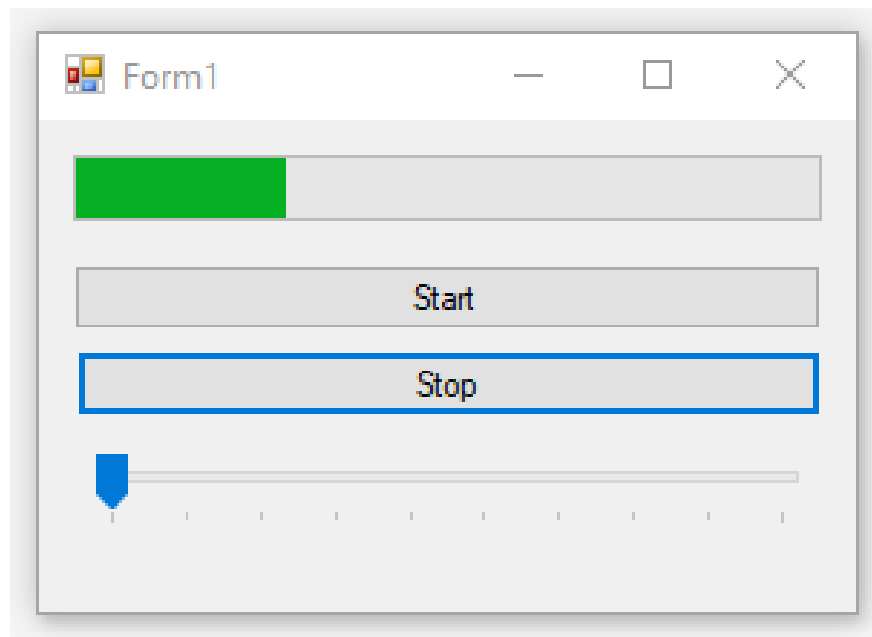


Properties	
timer1 System.Windows.Forms.Timer	
[Zbiór ikon]	
[-] Behavior	
Enabled	False
Interval	100
[-] Data	
[+] (ApplicationSettings)	
Tag	
[-] Design	
(Name)	timer1
GenerateMember	True
Modifiers	Private

Timer

PRZYKŁAD:

Sterowanie paskiem postępu za pomocą timera, z regulacją prędkości.



Timer

```
private void button1_Click(object sender, EventArgs e)
{
    timer1.Enabled = true;
}
```

```
private void button2_Click(object sender, EventArgs e)
{
    timer1.Enabled = false;
}
```

Uruchomienie i zatrzymanie Timera



Timer

```
private void timer1_Tick(object sender, EventArgs e)
{
    if (progressBar1.Value >= progressBar1.Maximum)
    {
        progressBar1.Value = 0;
    }
    progressBar1.Value = progressBar1.Value + 1;
}
```

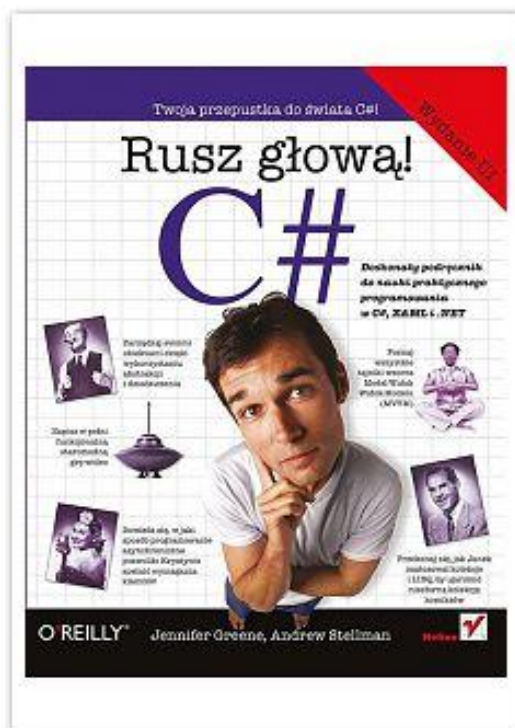
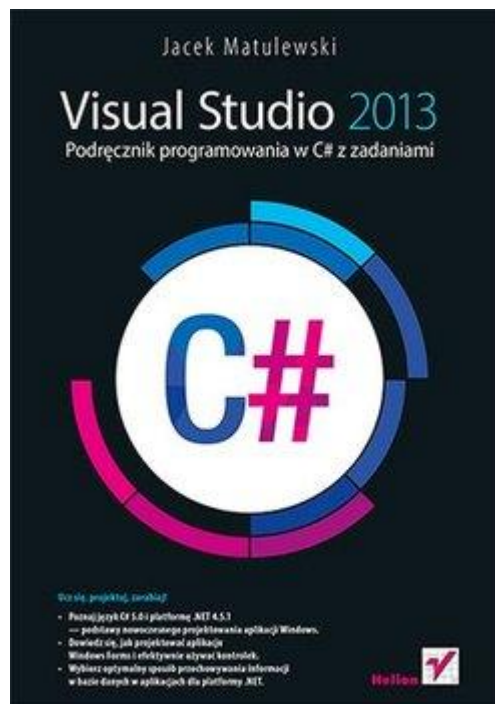
Metoda **Tick** – zwiększa wartość kontrolki progressBar
(pilnuje też, żeby nie nastąpiło jej przepełnienie)



Timer

```
private void trackBar1_Scroll(object sender, EventArgs e)
{
    timer1.Interval = 1000 - trackBar1.Value *100;
}
```

Sterowanie „prędkością” timera – czyli
ostępami pomiędzy wykonaniami metody
Tick



Użyte w tej prezentacji tabelki pochodzą z książki: Visual Studio 2013. Podręcznik programowania w C# z zadaniami Autor: Matulewski Jacek, Helion