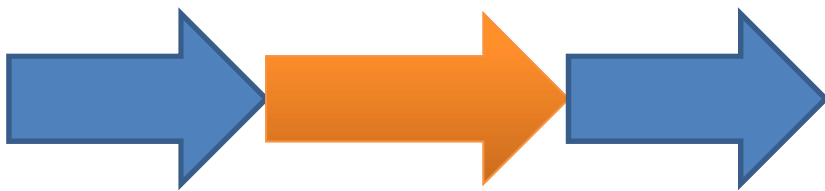


# PROGRAMOWANIE ASYNCHRONICZNE

# Programowanie synchroniczne

---

Programowanie synchroniczne polega na wykonywaniu poleceń w kolejności jedno po drugim



# Programowanie synchroniczne

---

Przykład 1:

```
private void Komunikat(string wiadomosc) {  
    MessageBox.Show($"{wiadomosc}");  
}  
  
private void button1_Click(object sender, EventArgs e)  
{  
    Stopwatch sw = new Stopwatch();  
    Komunikat("Button1_click: Początek");  
    sw.Start();  
    var wynik = PobierzDaneOPogodzie("synchronicznie");  
    sw.Stop();  
    Komunikat($"Wynik: {wynik} po: {sw.ElapsedMilliseconds}");  
    Komunikat("Button1_click: Koniec");  
}
```

# Programowanie synchroniczne

---

Przykład 1:

```
private string PobierzDaneOPogodzie(object argument)
{
    Thread.Sleep(5000);
    return "Słonecznie";
}
```

# Programowanie synchroniczne

---

Przykład 2:

```
private string PobierzDaneOWalutach(object argument)
{
    Thread.Sleep(10000);
    return "Dolar bez zmian";
}
```

```
private string PobierzWiadomosci(object argument)
{
    Thread.Sleep(1000);
    return "Wiadomości są dobre";
}
```

# Programowanie synchroniczne

---

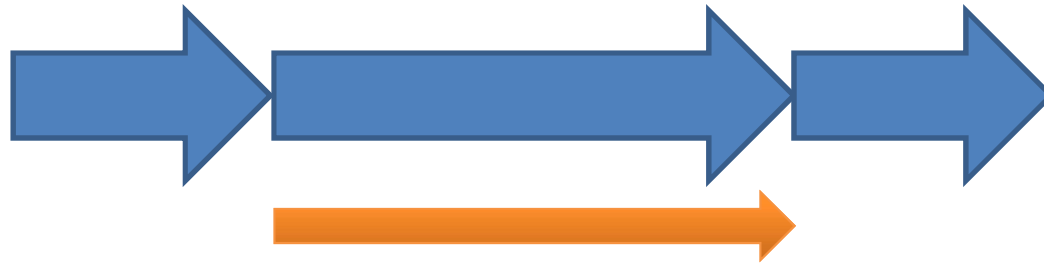
Przykład 2:

```
private void button2_Click(object sender, EventArgs e)
{
    Stopwatch sw = new Stopwatch();
    Komunikat("Button2_click: Początek");
    sw.Start();
    var wynik1 = PobierzDaneOPogodzie("synchronicznie");
    var wynik2 = PobierzDaneOWalutach("synchronicznie");
    var wynik3 = PobierzWiadomosci("synchronicznie");
    sw.Stop();
    Komunikat($"Wyniki po: {sw.ElapsedMilliseconds}");
    Komunikat("Button2_click: Koniec");
}
```

# Programowanie asynchroniczne

---

Zadania są wykonywane niezależnie (całość nie jest blokowana), przy czym poszczególne zadania mogą być (choć nie muszą) wykonywane w różnych wątkach.



# Programowanie asynchroniczne

---

W celu ułatwienia tworzenia aplikacji asynchronicznych w języku C# dostępne są dwie klasy

Task

Task<TResult>



# Klasa Task

---

Klasa Task reprezentuje operację asynchroniczną, która nie zwraca wartości

Metoda	Opis
Start	Rozpoczyna wykonywanie zadania
Wait	Czeka aż wątek zakończy wykonywanie swojego działania
WaitAll(Task[])	Czeka na zakończenie wszystkich działań
WaitAny(Task[])	Czeka na zakończenie dowolnego działania
Delay(int32)	Tworzy zadanie, które zakończy się po określonym czasie

# Klasa Task

---

Klasa Task reprezentuje operację asynchroniczną, która nie zwraca wartości

Metoda	Opis
ContinueWith(Action<Task>)	Tworzy kontynuację, która będzie uruchomiona w momencie zakończenia zadania
WhenAll(Task[])	Tworzy zadanie, które zakończy się jeżeli wszystkie zadania w tablicy się zakończą
WhenAny(Task[])	Tworzy zadanie, które zakończy się jeżeli jakiegokolwiek zadanie z tablicy się zakończy

## Klasa Task<TResult>

---

Klasa Task<TResult> reprezentuje operację asynchroniczną, która zwraca wartość typu TResult

Posiada ona podobne metody, co klasa Task, a dodatkowo zawiera właściwość:

**TResult Result**

która pozwala odczytać rezultat operacji asynchronicznej

# Programowanie asynchroniczne

---

Przykład 3:

```
private void Komunikat(string wiadomosc)
{
    string taskId = Task.CurrentId.HasValue ?
                    Task.CurrentId.ToString() : "UI";

    int threadId = Thread.CurrentThread.ManagedThreadId;

    MessageBox.Show($"{wiadomosc} ({taskId}) {threadId}");
}
```

# Programowanie asynchroniczne

---

Przykład 3:

```
private void button1_Click(object sender, EventArgs e) {  
    Komunikat("Button1_click: Początek");  
  
    var zadanie = new Task<string>(PobierzDaneOPogodzie, "zadanie");  
    zadanie.Start();  
  
    Komunikat("Akcja została uruchomiona");  
  
    if (zadanie.Status != TaskStatus.Running &&  
        zadanie.Status != TaskStatus.RanToCompletion)  
        Komunikat("Zadanie nie zostało uruchomione");  
    else  
        Komunikat($"Wynik: " + zadanie.Result);  
    Komunikat("Button1_click: koniec");  
}
```



# Programowanie asynchroniczne

---

Przykład 4:

```
private void button2_Click(object sender, EventArgs e) {  
    Komunikat("Button2_click: Początek");  
    Task<string> zadanie =  
        new Task<string>(PobierzDaneOPogodzie, "zadanie");  
  
    zadanie.ContinueWith(poprzednieZadanie => {  
        Komunikat($"Wynik: " + poprzednieZadanie.Result);  
    });  
  
    zadanie.Start();  
    Komunikat("Button3_click: koniec");  
}
```

# Programowanie synchroniczne

---

Przykład 5:

```
private void Kontynuacja(Task<string> zadanie) {  
    Komunikat($"Wynik: " + zadanie.Result);  
}  
  
private void button3_Click(object sender, EventArgs e) {  
    Komunikat("Button3_click: Początek");  
    Task<string> zadanie =  
        new Task<string>(PobierzDaneOPogodzie, "zadanie");  
  
    zadanie.ContinueWith(Kontynuacja);  
    zadanie.Start();  
}
```



# Programowanie asynchroniczne

---

Przykład 6:

```
private Task<string> OperacjaAsync() {  
    Task<string> zadanie =  
        new Task<string>(PobierzDaneOPogodzie, "zadanie");  
  
    zadanie.Start();  
    return zadanie;  
}  
  
private void button4_Click(object sender, EventArgs e) {  
    Komunikat("Button4_click: Początek");  
    var zadanie = OperacjaAsync();  
  
    zadanie.ContinueWith(Kontynuacja);  
}
```

## async/await

---

async / await są operatorami języka C#, które ułatwiają tworzenie operacji asynchronicznych.

Dzięki nim możliwe jest tworzenie programów asynchronicznych w sposób jak najbardziej zbliżony do programów synchronicznych

## async/await

---

Przykład 7:

```
private async void button5_Click(object sender, EventArgs e)
{
    Komunikat("Button5_click: Początek");
    string wynik = await OperacjaAsync();
    Komunikat($"Wynik: " + wynik);
    Komunikat("Button5_click: koniec");
}
```

# programowanie współbieżne

---

W programowaniu współbieżnym kilka zadań jest wykonywanych równocześnie.



# programowanie współbieżne

---

```
private Task<string> PobierzDaneOPogodzieAsync() {  
    var zadanie = new Task<string>(PobierzDaneOPogodzie, "Pogoda");  
    zadanie.Start();  
    return zadanie;  
}  
private Task<string> PobierzDaneOWalutachAsync() {  
    var zadanie = new Task<string>(PobierzDaneOWalutach, "Waluty");  
    zadanie.Start();  
    return zadanie;  
}  
private Task<string> PobierzDaneWiadomosciAsync() {  
    var zadanie = new Task<string>(PobierzWiadomosci, "Wiadomości");  
    zadanie.Start();  
    return zadanie;  
}
```

# programowanie współbieżne

---

```
private void button6_Click(object sender, EventArgs e) {  
  
    Komunikat("button6_click: Początek");  
    Stopwatch sw = new Stopwatch();  
    sw.Start();  
    var zadanie1 = PobierzDaneOWalutachAsync();  
    var zadanie2 = PobierzDaneOPogodzieAsync();  
    var zadanie3 = PobierzDaneWiadomosciAsync();  
  
    Task.WaitAll(zadanie1, zadanie2, zadanie3);  
    sw.Stop();  
    Komunikat("button6_click: Koniec " + sw.ElapsedMilliseconds);  
  
}
```

# programowanie współbieżne

---

```
private Task Zadanie()
{
    var zadanie = new Task(() =>
    {
        var zadanie1 = PobierzDaneOWalutachAsync();
        var zadanie2 = PobierzDaneOPogodzieAsync();
        var zadanie3 = PobierzDaneWiadomosciAsync();

        Task.WaitAll(zadanie1, zadanie2, zadanie3);
    });

    zadanie.Start();
    return zadanie;
}
```

# programowanie współbieżne

---

```
private Task Zadanie()
{
    var zadanie1 = PobierzDaneOWalutachAsync();
    var zadanie2 = PobierzDaneOPogodzieAsync();
    var zadanie3 = PobierzDaneWiadomosciAsync();
    return Task.WhenAll(zadanie1, zadanie2, zadanie3);
}
```



# programowanie współbieżne

---

```
private async void button7_Click(object sender, EventArgs e)
{
    Komunikat("button7_click: Początek");
    Stopwatch sw = new Stopwatch();
    sw.Start();
    await Zadanie();
    sw.Stop();
    Komunikat("button7_click: Koniec "+sw.ElapsedMilliseconds);
}
```

# programowanie współbieżne ze stanem

---

```
private void Wykonaj(int od, int @do) {  
    for (int i = od; i <= @do; i++) wynik++;  
    System.Threading.Thread.Sleep(1000);  
}  
private void button8_Click(object sender, EventArgs e) {  
    wynik = 0;  
    Stopwatch sw = new Stopwatch();  
    sw.Start();  
    Wykonaj(1, 250000);  
    Wykonaj(250001, 500000);  
    Wykonaj(500001, 750000);  
    Wykonaj(750001, 1000000);  
    sw.Stop();  
    WynikLabel.Text = wynik.ToString();  
    CzasLabel.Text = sw.ElapsedMilliseconds.ToString(); }  
}
```

# programowanie współbieżne ze stanem

---

```
private int wynik;
private void Wykonaj(int od, int @do) {
    for (int i = od; i <= @do; i++) wynik++;
    System.Threading.Thread.Sleep(1000);
}
private void button8_Click(object sender, EventArgs e) {
    wynik = 0;
    Stopwatch sw = new Stopwatch();
    sw.Start();
    Wykonaj(1, 250000);
    Wykonaj(250001, 500000);
    Wykonaj(500001, 750000);
    Wykonaj(750001, 1000000);
    sw.Stop();
    WynikLabel.Text = wynik.ToString();
    CzasLabel.Text = sw.ElapsedMilliseconds.ToString(); }
```

# programowanie współbieżne ze stanem

---

```
private void button9_Click(object sender, EventArgs e) {
    wynik = 0;
    Stopwatch sw = new Stopwatch();
    Task[] t = new[] { new Task(()=>Wykonaj(    1,  250000)),
                      new Task(()=>Wykonaj(250001,  500000)),
                      new Task(()=>Wykonaj(500001,  750000)),
                      new Task(()=>Wykonaj(750001, 1000000))};

    sw.Start();
    for (int i = 0; i < t.Length; i++) t[i].Start();
    Task.WaitAll(t);
    sw.Stop();
    label1.Text = wynik.ToString();
    label2.Text = sw.ElapsedMilliseconds.ToString();
}
```

# programowanie współbieżne ze stanem

---

```
object locker = new object();

void Wykonaj(int od, int @do)
{
    for (int i = od; i <= @do; i++)
        lock (locker)
        {
            wynik++;
        }
    System.Threading.Thread.Sleep(1000);
}
```

# programowanie współbieżne bez stanu

---

```
int WykonajBezStanu(int od, int @do) {  
    int wynik = 0;  
    for (int i = od; i <= @do; i++) wynik++;  
    System.Threading.Thread.Sleep(1000);  
    return wynik;  
}
```

# programowanie współbieżne bez stanu

---

```
private void button11_Click(object sender, EventArgs e) {  
    Stopwatch sw = new Stopwatch();  
    var ts = new[] {new Task<int>(()=>WykonajBezStanu(1, 250000)),  
                    new Task<int>(()=>WykonajBezStanu(250001, 500000)),  
                    new Task<int>(()=>WykonajBezStanu(500001, 750000)),  
                    new Task<int>(()=>WykonajBezStanu(750001, 1000000))};  
  
    sw.Start();  
    for (int i = 0; i < ts.Length; i++) ts[i].Start();  
    Task.WaitAll(ts);  
  
    wynik = ts.Sum(t=>t.Result);  
  
    sw.Stop();  
    label1.Text = wynik.ToString();  
    label2.Text = sw.ElapsedMilliseconds.ToString(); }  
}
```