Programowanie Współbieżne

C#

Delegaty i zdarzenia

Obiektowy odpowiednik wskaźnika do funkcji z c/c++

klasa pochodną z klasy System.Delegate

Deklaracja wygląda jak deklaracja funkcji:

```
delegate void PrzykladowyDelegat(); //deklaracja delegatu
  która jest równoważna z deklaracją klasy
```

By wykorzystać delegat musimy stworzyć nowy obiekt tej klasy.

```
delegate ddd = new PrzykladowyDelegat(jakasFunkcja);
```

jakasFunkcja musi być tego samego typu co delegat (mieć tę samą sygnaturę).

Funkcjonalnie zachowuje się jak klasy wewnętrzne w javie z tym, że w javie trzeba było tworzyć całą klasę, tu tylko metodę.

Delegaty i zdarzenia

//DelegateExample

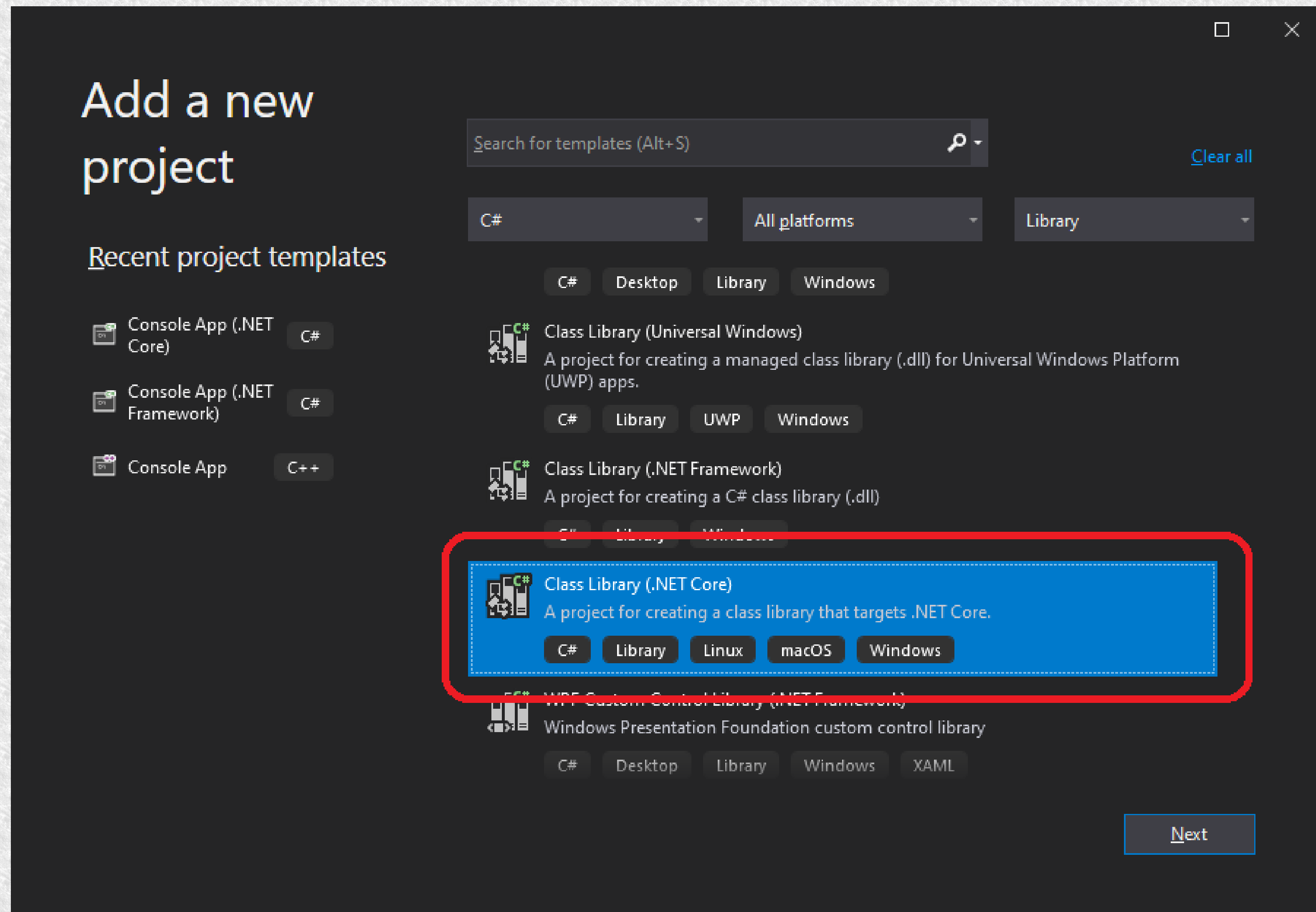
```
class Program
    {
    delegate void SomeDelegate();
     // method consistent with delegate declaration
     static void SomeMethod()
        {
        System.Console.WriteLine("Some Method? After all, it was a
delegate!");
        }
    static void Main(string[] args)
        {
        SomeDelegate ddd = new SomeDelegate(SomeMethod);
        // here we delegate the call to the SomeMethod ()
        ddd();
        Console.ReadLine();
        }
    }
```

Delegaty i zdarzenia

- Zdarzenia (events) są formą komunikacji informowania innych, że wystąpiła jakaś sytuacja.
- Realizowane są przy pomocy delegatów.
- Utwórzmy nowy projekt typu biblioteka

Delegaty i zdarzenia

//TickTockLib



Delegaty i zdarzenia

```
public class TickTock
{
    public delegate void TickTockEventHandler(object sender, TickTockEventArgs e);
    public event TickTockEventHandler TickEvent; //event of tick
    public class TickTockEventArgs : EventArgs
    {
        public int TickCounter { get; set; }
        public bool IsEndless;
    }
    private int count = 10;

    public Thread TickTockThread { get; }

    public TickTock(int count)
    {
        this.count = count;
        TickTockThread = new Thread(Ticker);
    }

    public void Join()
    {
        TickTockThread.Join();
    }

    public void Start()
    {
    TickTockThread.Start(count);
    }
```

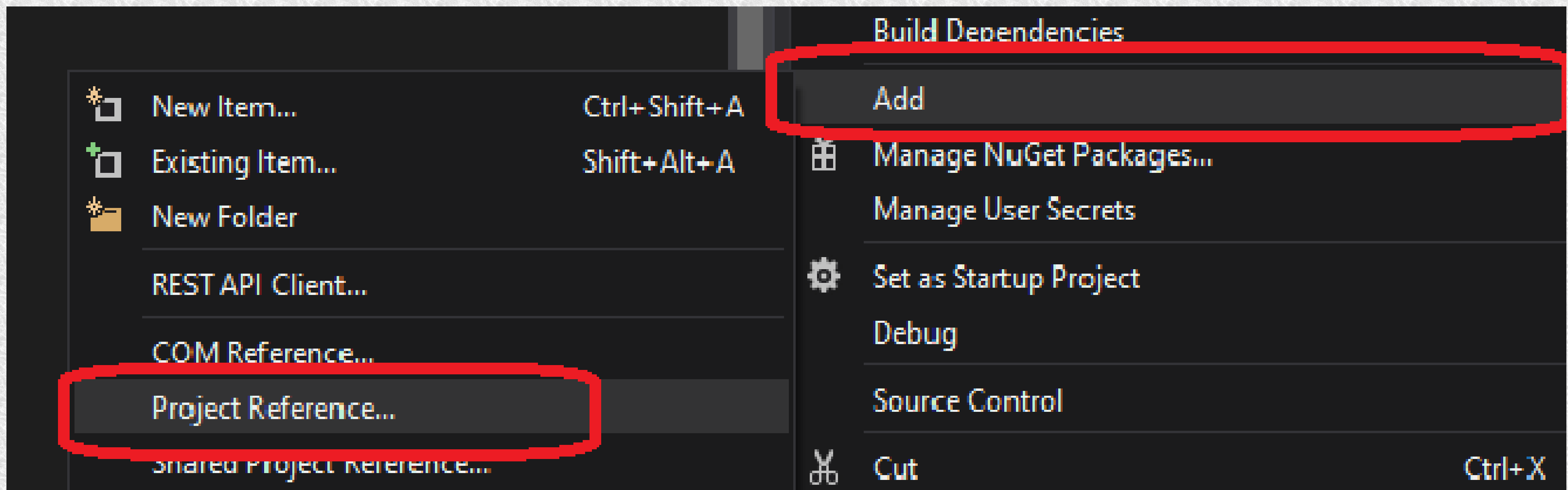
Delegaty i zdarzenia

```
    private void Ticker(object o)
    {
        int howMany = 0;
        if (o != null) howMany = (int)o;
        TickTockEventArgs tta = new TickTockEventArgs();
        tta.IsEndless = true;
        if (howMany == 0)
        {
            tta.IsEndless = true;
            int ii = 0;
            while (true)
            {
                tta.TickCounter = ii++;
                if (TickEvent != null) TickEvent(this, tta);
                Thread.Sleep(1000);
            }
        }
        else
        {
            tta.IsEndless = false;
            for (int ii = 0; ii < howMany; ii++)
            {
                tta.TickCounter = ii;
                if (TickEvent != null) TickEvent(this, tta);
                Thread.Sleep(1000);
            }
        }
    }
}
```

Delegaty i zdarzenia

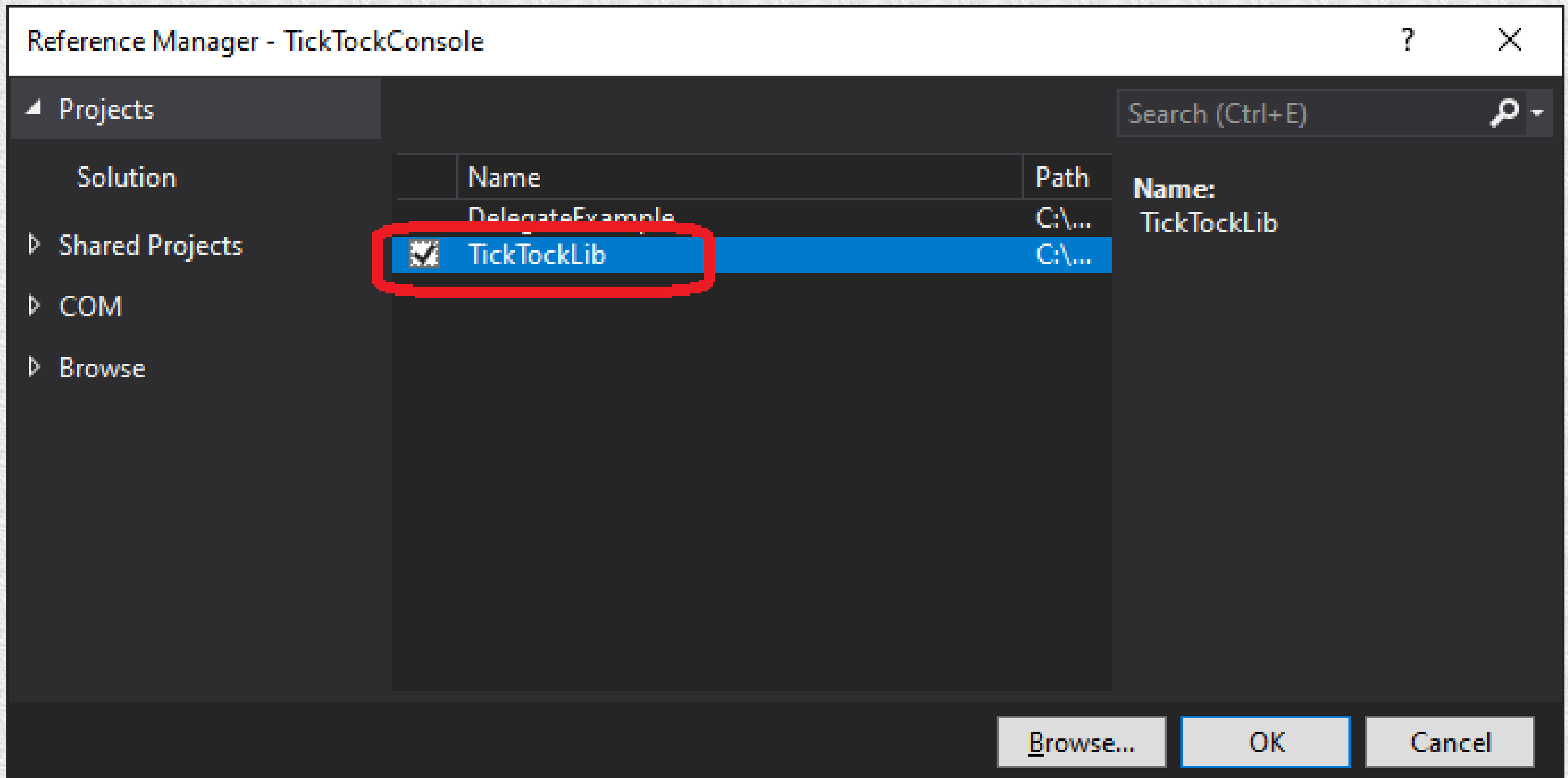
`TickTockConsole`

- Dodajemy nowy projekt typu konsola
- Dodajemy referencje do naszego projektu typu biblioteka



Delegaty i zdarzenia

TickTockConsole



Delegaty i zdarzenia

TickTockConsole

```
  static void Main(string[] args)
        {

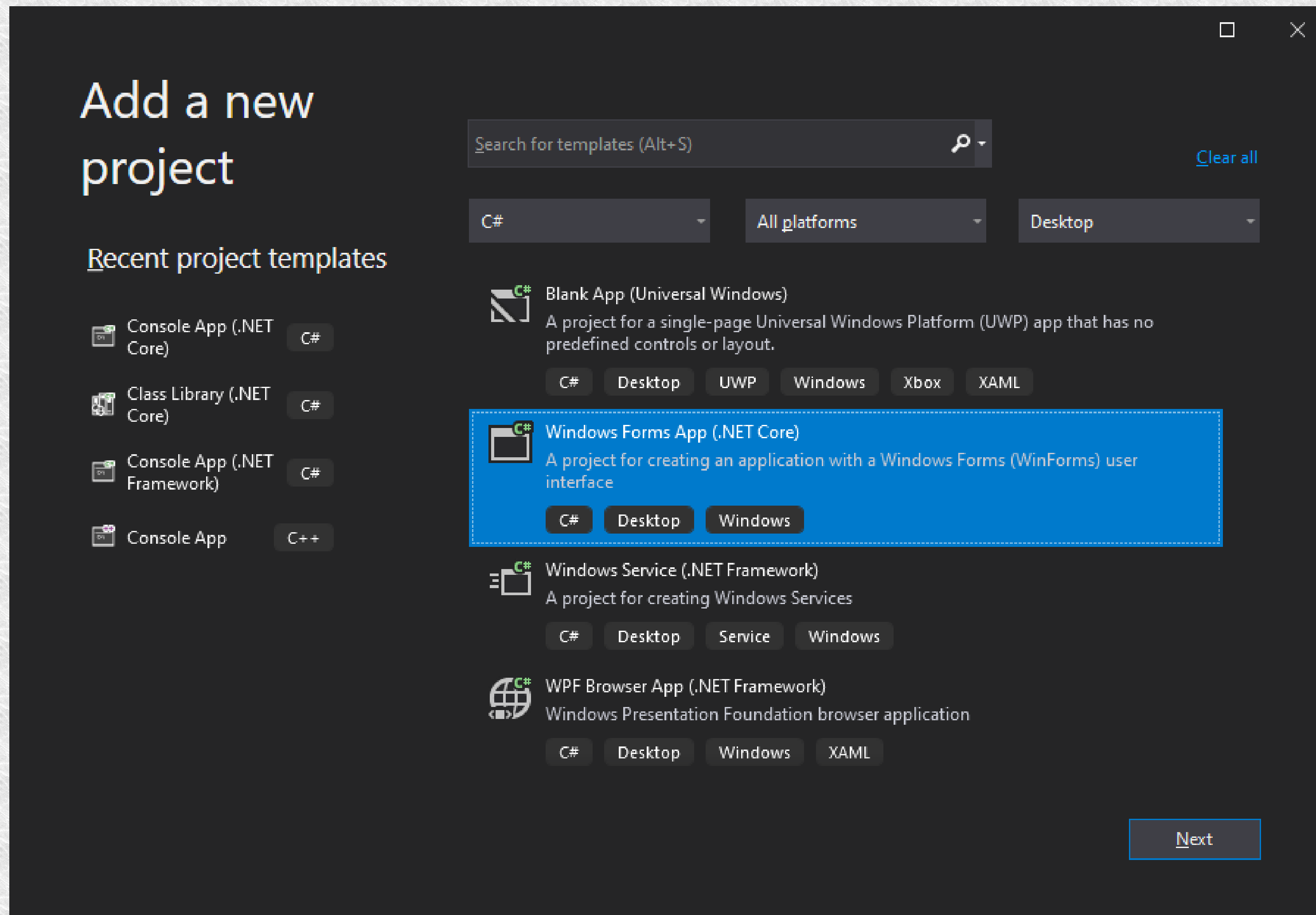
            TickTock tc = new TickTock(20);
            tc.TickEvent += Tc_TickEvent;
            tc.Start();
            Console.WriteLine("Tick Tock start ticking...");
            tc.Join();
            Console.WriteLine("Tick Tock stop ticking");
            Console.ReadLine();
        }

        private static void Tc_TickEvent(object sender,
TickTock.TickTockEventArgs e)
        {
            Console.WriteLine("We heard tick no: " + e.TickCounter);
        }
```

WinForms i wątki

błąd Cross-Thread

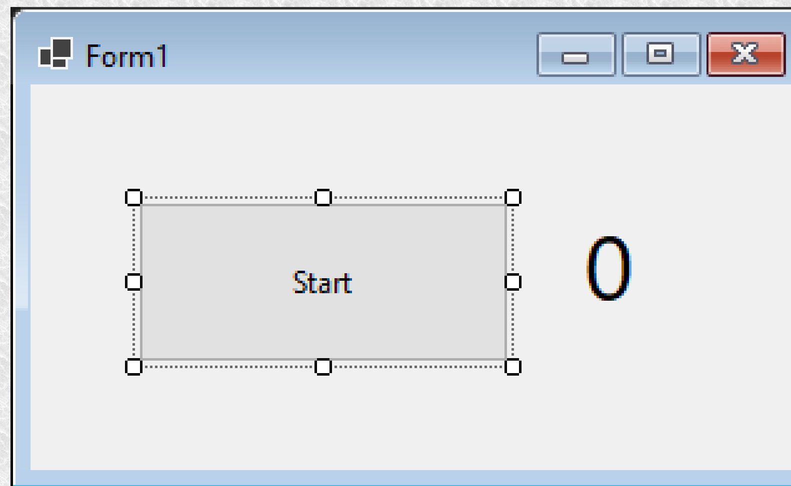
Dodajmy nowy projekt typu WinForms



Example: przykladCrossThread

WinForms i wątki
błąd Cross-Thread

Tworzymy GUI



Example: TickTockWinForms

WinForms i wątki

błąd Cross-Thread

Dodajemy referencje do TickTockLib

Dwókrotnie klikamy guzik aby podpiąć się pod zdarzenie kliknięcia (tak, *click* jest zdarzeniem które możemy obsłużyć. Visual Studio automatycznie wygeneruje nam szkielet metody w Form1.Designer.cs).

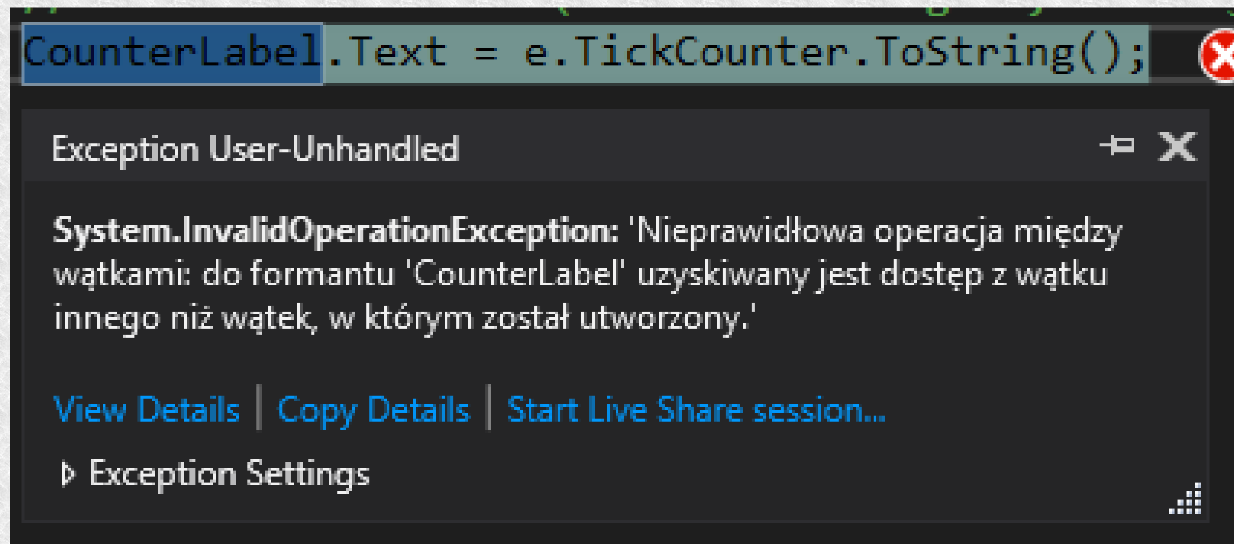
```
private void Form1_Load(object sender, EventArgs e)
        {
            tickTock = new TickTock(20);
        }

        private void StartButton_Click(object sender, EventArgs e)
        {
            CounterLabel.Text = "0";
            tickTock = new TickTock(20);
            tickTock.TickEvent += TickTock_TickEvent;
            tickTock.Start();
            MessageBox.Show("Tick Tock started....");
        }
        private void TickTock_TickEvent(object sender, TickTock.TickTockEventArgs e)
        {
                  CounterLabel.Text = e.TickCounter.ToString();
        }
```


Example: TickTockWinForms

WinForms i wątki
błąd Cross-Thread

Uruchamiamy program i....



Kiedy chcemy zapisać coś na formatce z innego wątku niż ten, który go utworzył, dostaniemy:

Cross-thread operation not valid: Control 'CounterLabel' accessed from a thread other than the thread it was created on

Dokumentacja MS podpowiada pewien trik:

Example: TickTockWinForms

WinForms i wątki

WinForms komponent zawiera własność **Control.InvokeRequired**, która oznacza, że dane przypisanie wartości musi być delegowane do wątka, który utworzył kontrolkę.

Żeby to zrobić należy użyć metody **Control.Invoke** wskazując delegata z odpowiednią metodą.

Example: TickTockWinForms

WinForms i wątki

```
delegate void SetCounterDelegate(string txt);
        private void SetCounterMethod(string txt)
        {
            CounterLabel.Text = txt;
        }

private void TickTock_TickEvent(object sender, TickTock.TickTockEventArgs e)
        {
        SetCounterDelegate setCounterDelegate = new
SetCounterDelegate(SetCounterMethod);
        if (this.CounterLabel.InvokeRequired) //check if invoke is unsafe
            {//when yes call delegate
            CounterLabel.Invoke(setCounterDelegate, new object[]
{ e.TickCounter.ToString() }) ;
            }
        else
            {//when it is safe write string directly to the controll
            CounterLabel.Text = e.TickCounter.ToString();
            }
        }
```

Example: TickTockWinForms

WinForms i wątki

Bardziej eleganckim sposobem jest utworzenie klasy z odpowiednimi "bezpiecnymi" metodami:

```
private delegate void SetLabelDelegate(Label label, string tekst);
public static void SetLabel(Label label, string txt)
    {
    //check if involving is not safe
    //(out of the thread which create the contrloll)
    if (label.InvokeRequired)
        {
        // it is in another thread so we need call delegate
        // w want changes in the owner thread sa we tell which
        label.Invoke(new SetLabelDelegate(SetLabel), new object[] { label, 
txt });
        }
    else
        {
        // when it is safe (it is in current thread)
        label.Text = txt;
        }
    }
```


Example: TickTockWinForms

WinForms i wątki

Ustawianie etykietki (Label) będzie prostrze i czytelniejsze.

```
private void TickTock_TickEvent(object sender, TickTock.TickTockEventArgs e)
    {
     ThreadSafeCalls.SetLabel(CounterLabel,
e.TickCounter.ToString());
     }
```

Example: TickTockWinForms

WinForms i wątki

Innym sposobem jest napisanie zmodyfikowanej kontrolki

```
public partial class LabelSafe : Label
    {
        public LabelSafe()
        {
            InitializeComponent();
        }
        public new string Text
        {
            get
            {
                return base.Text;
            }
            set
            {
                if (this.InvokeRequired)
                {
                    this.Invoke(new Action(() => { this.Text = value; }));
                }
                else
                {
                    base.Text = value;
                }
            }
        }
    }
```

Example: TickTockWinForms

WinForms i wątki

Możemy umieścić naszą nową kontrolkę na formatce i normalnie używać. Należy podmienić klasę Label na klasę LabelSafe

```
private System.Windows.Forms.Button StartButton;
//private System.Windows.Forms.Label CounterLabel;
private LabelSafe CounterLabel; // --------- here
```

```
this.StartButton = new System.Windows.Forms.Button();
//this.CounterLabel = new System.Windows.Forms.Label();
this.CounterLabel = new LabelSafe(); //---------- and here
this.SuspendLayout();
```

```
private void TickTock_TickEvent(object sender,
TickTock.TickTockEventArgs e)
    {
    //ThreadSafeCalls.SetLabel(CounterLabel,
e.TickCounter.ToString());
    CounterLabel.Text = e.TickCounter.ToString(); //------- and here
    }
```

WinForms i wątki
Kontekst synchronizacji

- W .Net, wątki moga posiadać konteksty synchronizacji. są to obiekty klasy SynchronizationContext. Mogą być odczytane za pomocą statycznej własniści SynchronizationContext.Current.
- Aplikacje desktopowe mają automatycznie tworzony konteks dla wątków UI.
    - *WindowsFormsSynchronizationContext (WinForms),*
    - *DispatcherSynchronizationContext (WPF),*
    - *AspNetSynchronizationContext (ASP.NET)*
- Jest możliwość przekazania kontekstu synchronizacji do innego wątka poprzez referencje.
- Wątek, który otrzymał kontekst synchronizacji, może wywołać metody bądź wyrażenia lambda na wątku, który jest właścicielem danego kontekstu

WinForms i wątki
Synchronization Context

- Ten mechanizm też może być użyty do zmiany wartości pól formatki , które były utworzone przez innych wątek niż ten, których chce zmienić.
- Context posiada dwie metody do wywoływania kodu po stronie właściciela kontekstu.
  - *Send* – Podobne do Invoke (blokujące)
  - *Post* – Asynchroniczna metoda podobna do BeginInvoke / EndInvoke
- Powyższa asynchroniczna metoda nie przekazuje wyjątków na zewnątrz,
- W przypadku WPF także Send nie przekazuje wyjątków
- Używając metody BeginInvoke methods możemy odczytać status wątku, czy się zakończył czy nie, co nie jest możliwe w przypadku metody Post.

WinForms i wątki

Synchronization Context

```
public SynchronizationContext Context { get; set; }
      public class TickTockEventArgs : EventArgs
      {
          public int TickCounter { get; set; }
          public bool IsEndless;
          public SynchronizationContext Context { get; set; }
      }
```

```
public TickTock(int count, SynchronizationContext context) //new
constructor with sychnronization context
        {
            this.count = count;
            TickTockThread = new Thread(Ticker);
            Context = context;
        }
```

WinForms i wątki

Synchronization Context

```
private void Ticker(object o)
        {
            int howMany = 0;
            if (o != null) howMany = (int)o;
            TickTockEventArgs tta = new TickTockEventArgs();
            tta.IsEndless = true;
            tta.Context = Context;
           ...


private void TickTock_TickEvent(object sender, 
TickTock.TickTockEventArgs e)
        {
            e.Context.Send(updateUI, e.TickCounter.ToString());
        }
```

Background Worker

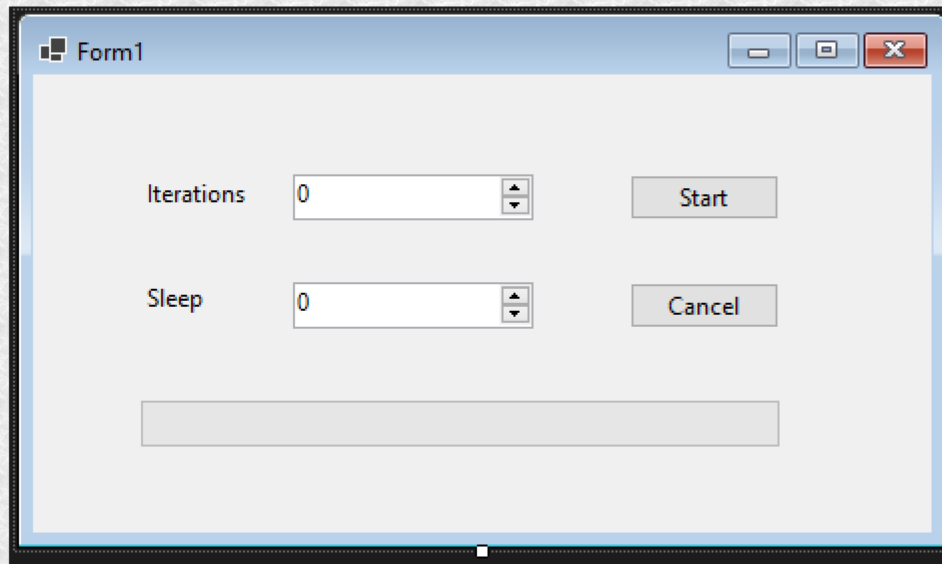
- Jest to pomocna klasa z System.ComponentModel, która dostarcza nam następującej funkcjonalności:
  - *Flaga Cancel* flag do anulowania, zamiast Abort
  - Standardowy protokół raportowania postępu, zakończenia czy przerwania pracy
  - Implementacja interfejsu IComponent która pozwala zamieścić kontrolkę w VS Designer.
  - Łapanie wyjątków w wątku workera
  - Możliwość zapisywania postępu bespośrednio na formatce stworzonej przez inny wątek (nie ma problemów z błędem Cross-Thread) i nie musimy wołać Control.Invoke.

Background Worker

- Model tego typu wykorzystuje identyczną składnię, jak asynchroniczne delegaty
- Aby użyć *BacgroundWorkera* wystarczy poinformować go obsługując zdarzenie *DoWork* jaka metoda ma być wykonana w tle i wywołać RunWorkerAsync()
- Wątek główny kontynuuje działanie, a w tle wykonywana jest funkcja zgłoszona do *BackgroundWorkera*.
- *BW* sygnalizuje postęp prac za pomocą zdarzenia *ProgressChanged* – w jego obsłudze można aktualizować np. *ProgressBar*
- Gdy *BW* skończy sygnalizuje to zdarzeniem *RunWorkerCompleted*

Background Worker

- Tworzymy nowy projekt typu Windows Form nadajemy tytuł: BackgroundWorkerExample
- Zróbmy sobie formatkę jak niżej:



- Mam dwa pola Iterations i Sleep Background Worker będzie wykonywał zadaną liczbę iteracji spanie przez określoną liczbę milisekund.

Background Worker

- Definiujemy BW i inicjujemy go

```
private BackgroundWorker bw;
public Form1()
    {
     InitializeComponent();
     bw = new BackgroundWorker();
     bw.DoWork += Bw_DoWork;
     bw.RunWorkerCompleted += Bw_RunWorkerCompleted;
     bw.ProgressChanged += Bw_ProgressChanged;
     //Make BackgroundWorker to report progress
     bw.WorkerReportsProgress = true;
     //Allow cancelation
     bw.WorkerSupportsCancellation = true;
     }
```

- Handle Events DoWork,RunWorkerCompleted and ProgressChanged
- Allow reporting progress and cancellation

Background Worker

- Implementujemy procedurę obsługi Click Event'a dla guzików Start i Cancel.

```
private void StartButton_Click(object sender, EventArgs e)
    {
     try
        {
        //Get the data from form
        int iter = (int)IterationsUD.Value;
        int sleep = (int)SleepUD.Value;
        //and make arguments object to pass to the worker
        ThreadsArguments param = new ThreadsArguments() { Iterations =
iter, SleepingTime = sleep };

        bw.RunWorkerAsync(param);
        }
    catch (Exception ex)
        {
        MessageBox.Show(ex.Message);
        }
    }

private void CancelButton_Click(object sender, EventArgs e)
    {
     bw.CancelAsync();
     }
```

Background Worker

- ThreadArguments jest prostą klasą służącą do przekazania parametrów do background workera

```
public class ThreadsArguments
{
    public int Iterations { get; set; }
    public int SleepingTime { get; set; }
}
```

Background Worker

- Zdarzenie DoWork jest obsłużone jak poniżej

```
private void Bw_DoWork(object sender, DoWorkEventArgs e)
    {
     ThreadsArguments arguments = (ThreadsArguments)e.Argument;
     for (int i = 0; i < arguments.Iterations; i++)
        {
        Console.WriteLine("I'm doing operation no: " + i);
        System.Threading.Thread.Sleep(arguments.SleepingTime);
        bw.ReportProgress((i + 1) * 100 / arguments.Iterations);
        if (bw.CancellationPending)
            {
            e.Cancel = true;
            return;
            }
        }
        string message = "In total " + (arguments.Iterations * 
arguments.SleepingTime) + " milisec";
             e.Result = message;
    }
```

Background Worker

- Obsługa zdarzeń *ProgressChanged* i *RunWorkerCompleted*

```
private void Bw_ProgressChanged(object sender, ProgressChangedEventArgs e)
    {
     ProgressBar.Value = e.ProgressPercentage;
     }


private void Bw_RunWorkerCompleted(object sender,
RunWorkerCompletedEventArgs e)
    {
     if (e.Cancelled)
        MessageBox.Show("Canceled");
     else if (e.Error != null)
        MessageBox.Show("Error: " + e.Error.ToString());
    else
        MessageBox.Show("Work completed\n" + e.Result);
    }
```

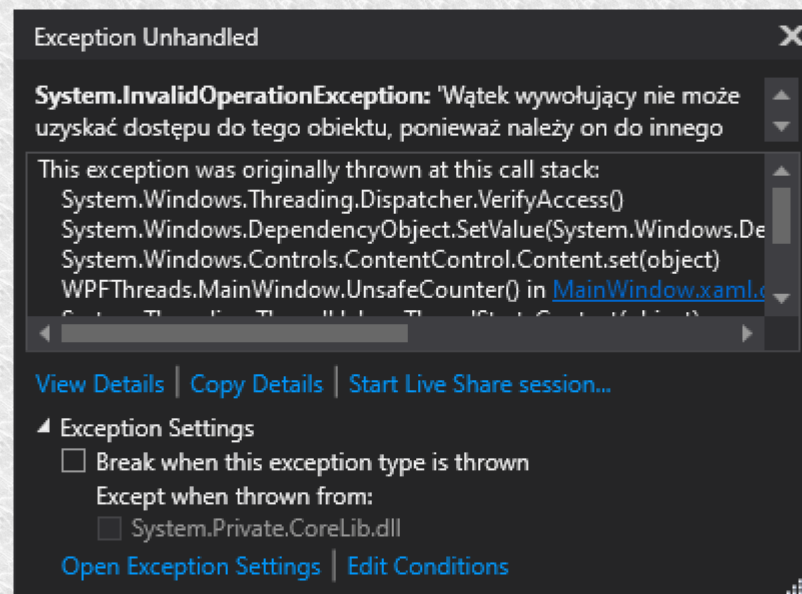
Wątki w WPF

- Jeżeli w aplikacji WPF chcemy zmienić stan kontrolki spoza wątku, który ją utworzył, także będziemy mieli problem podobnie jak z aplikacją WinForms:

```
private void UnsafeCounter()
        {
            for (int number = 0; number < 21; number++)
            {
                counterLabel.Content = number.ToString();

                Thread.Sleep(200);
            }
        }
```

```
private void Button_Click(object sender,
RoutedEventArgs e)
    {
     //version first
     Thread thcount = new
Thread(UnsafeCounter);
    thcount.Start();
    }
```



Example: WPFThreads

Wątki w WPF

- Druga wersja teraz poprawna:

```
delegate void WriteTextDel(string tekst);
private void WriteText(string tekst)
    {
    counterLabel.Content = tekst;
    }
private void ThreadSafeCounter()
    {
    WriteTextDel writeDel = new WriteTextDel(WriteText);
    for (int number = 0; number < 21; number++)
        {
        if (!counterLabel.Dispatcher.CheckAccess())
            {
            counterLabel.Dispatcher.Invoke(writeDel, new object[]
{ number.ToString() });
            }
        else
            {
            counterLabel.Content = number.ToString();
            }
        Thread.Sleep(200);
        }
    }
```


Example: WPFThreads

Wątki w WPF

- Komponenty wizualne kontrolowane są przez wątek okna aplikacji. Wszelkie zmiany zgłaszane są jako żądania z innych wątków do wątku kontrolującego za pośrednictwem obiektu *Dispatcher*, po wcześniejszym sprawdzeniu za pomocą *CheckAccess.*
- Metodę Invoke można przeciążyć podając priorytet wywołania

```
label.Dispatcher.Invoke(new Action(() =>
   { label.Content = content; }

),System.Windows.Threading.DispatcherPriority.ApplicationIdle);
```

- Jest też możliwość zgłaszania, żądań asynchronicznie z pomocą *BeginInvoke*.

Example: WPFThreads

Wątki w WPF

- Trzecia wersja poprawna, osobna klasa statyczna z bezpiecznymi metodami

```
class ThreadSafeCallsWPF
    {
    public static void setLabelContent(Label label, object content)
        {
        if (!label.Dispatcher.CheckAccess())
            {
            label.Dispatcher.Invoke(new Action(() =>
                { label.Content = content;}),
            System.Windows.Threading.DispatcherPriority.ApplicationIdle);
            }
        else
            {
            label.Content = content;
            }
         }
...
```

```
 private void ThreadSafeCounter2()
        {
            for (int number = 0; number < 21; number++)
            {
                ThreadSafeCallsWPF.setLabelContent(counterLabel,
number.ToString());
                Thread.Sleep(200);
            }
        }
```

Example: WPFThreads

Wątki w WPF

- w czwartej wersji użyjemy klasy Canvas, aby sobie porysować.

```
private void addFigureToCanvasUnsafe()
        {
            Random random = new Random();
            int w, h, x, y;

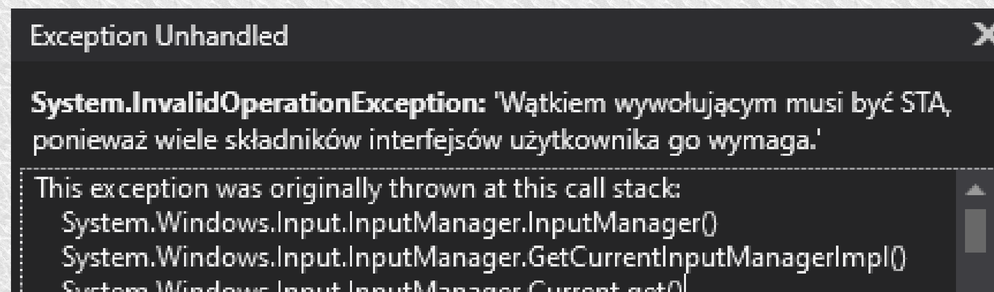
            switch (random.Next(0, 3))
            {
                case 0:
                    {
                        //   Ellipse
                        w = random.Next(5, 30);
                        h = random.Next(5, 30);
                        x = random.Next(0, (int)(canvas1.ActualWidth - w));
                        y = random.Next(0, (int)(canvas1.ActualHeight - h));
                        Ellipse ellipse = new Ellipse();
                        ellipse.Width = w;
                        ellipse.Height = h;
                        ellipse.Fill = Brushes.Red;
                        canvas1.Children.Add(ellipse);
                        Canvas.SetLeft(ellipse, x);
                        Canvas.SetTop(ellipse, y);

                        break;
                    }
                  .....
```


Example: WPFThreads

Wątki w WPF

- Przy próbie narysowania elipsy na kontrolce *canvas* bez synchronizacji dostajemy następujący błąd:



- *Singe Threaded Apartment* – model wywodzący się z architektury aplikacji wielowątkowych wykorzystujących obiekty COM (technologia tworzenia obiektów rejestrowanych globalnie w systemie operacyjnym dzięki czemu mogą być wykorzystywane przez inne aplikacje. COM jest podstawą Microsoft OLE, COM+, DCOM czy ActiveX.

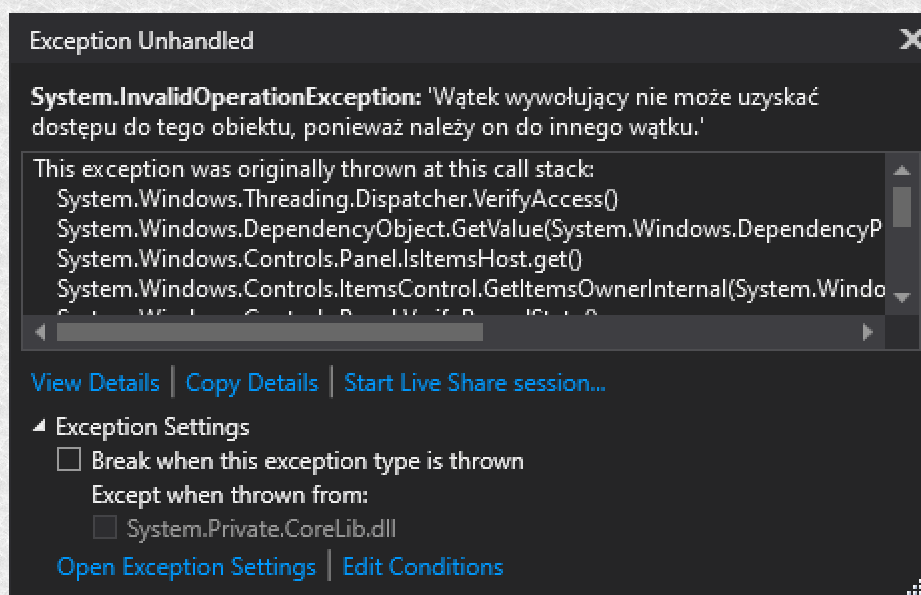
Example: WPFThreads

Wątki w WPF

- W piątek wersji ustawiamy STA

```
Thread thcount = new Thread(UnsafeCounter2); //but with STA
thcount.SetApartmentState(ApartmentState.STA);
thcount.Start();
```

- Dostaniemy błąd, który już znamy i wiemy jak sobie z nim poradzić.



Example: WPFThreads

Wątki w WPF

- Dodajemy bezpieczną metodę rysującą elipsy na canvasie:

```
private delegate void drawEllipseDelegate(Canvas canvas, double w, double h,
double x, double y, Brush brush);
public static void drawEllipse(Canvas canvas, double w, double h, double x,
double y, Brush brush)
    {
    if (!canvas.Dispatcher.CheckAccess())
        {
        canvas.Dispatcher.Invoke(new drawEllipseDelegate(drawEllipse), new
object[] { canvas, w, h, x, y, brush });
        }
    else
        {
        Ellipse ellipse = new Ellipse();
        ellipse.Width = w;
        ellipse.Height = h;
        ellipse.Fill = brush;
        canvas.Children.Add(ellipse);
        Canvas.SetLeft(ellipse, x);
        Canvas.SetTop(ellipse, y);
        }
}
```


Example: WPFThreads

Wątki w WPF

- Teraz powinno wszystko działać (wersja szósta).

```
ThreadSafeCallsWPF.drawEllipse(canvas1, w, h, x, y, Brushes.Red);
```

- Jednak gdy chcielibyśmy tworzyć różne kolory a nie używać statycznego:

```
SolidColorBrush scb = new SolidColorBrush(RandomColor());
ThreadSafeCallsWPF.drawEllipse(canvas1, w, h, x, y, scb);
```

- Pojawia się kolejny problem: „*The Application is in break mode*”, z konsoli można odczytać błąd: **„Nie można użyć obiektu DependencyObject należącego do innego wątku niż nadrzędny obiekt Freezable.”**

- Na szczęście rozwiązanie jest proste. Trzeba wykonać metodę *Freeze* na obiekcie *scb* przed jego użyciem.

```
scb.Freeze();
```


Example: WPFThreads

Wątki w WPF

- Siódma wersja - użycie kontekstu synchronizacji

```
private void ThreadSafeCounter3(object par)
    {
     DispatcherSynchronizationContext context = par as
DispatcherSynchronizationContext;
     for (int number = 0; number < 21; number++)
        {
        context.Send((object s) => {
            counterLabel.Content = s as string;
            }, number.ToString());
        Thread.Sleep(200);
        }
    }
```

```
Thread thcount = new Thread(ThreadSafeCounter3);
thcount.Start(SynchronizationContext.Current);
```


Example: WPFThreads

Wątki w WPF

- Uwaga na niebezpieczeństwo blokady. Nie powinno się tak robić (efekt taki jak byśmy wykonali funkcję sekwencyjnie), ale przy dodaniu oczekiwania na wątek
- ```
  Thread thcount = new Thread(ThreadSafeCounter3);
  thcount.Start(SynchronizationContext.Current);
  thcount.Join();
  ```
- W momencie wykonywania funkcji obsługi kliknięcia buttonStart_Click startujemy wątek potomny, który w pętli stara się zakolejkować zgłoszenie czynności do wykonania przez wątek macierzysty (Control.Invoke lub SynchronizeContext.Send).
- Zatrzymują one dalsze działania do momentu zakończenia czynności wątku interfejsu. Niestety wątek macierzysty jest zajęty bo czeka na zakończenie się buttonStart_Click, gdzie Joinem czeka na zakończenie wątku potomnego.

Example: WPFThreads
  ```

Wątki w WPF

- Zamiast *Send*, można użyć do kolejkowania zgłoszeń metody asynchronicznej *Post*
- Spowoduje to wykonanie całej pętli jednak obsługą zgłoszeń interfejs użytkownika zajmie się dopiero po zakończeniu wątku potomnego. Efekt będzie taki, że wszystkie żądania mogą wykonać się jednocześnie, czyli zobaczymy ostateczny wynik licznika. Nie jest to poprawne działanie ale przynajmniej nie mamy już blokady.

Example: WPFThreads