Intro to Async and Parallel Programming in .NET 4:

Coordinating, Canceling, and Exception Handling of Tasks

Necessary requirements in most applications…

pluralsight
see what you can learn

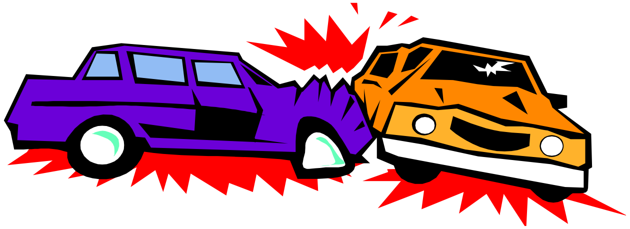
Overview

- **Your presenter: Joe Hummel, PhD**
  - PhD in field of high-performance computing
  - drjoe@pluralsight.com

- **Agenda for this module:**
  - *Exception handling*
  - *Task cancellation*
  - *Task priorities*
  - *Parent-child tasks*
  - *Parameter passing*

Exception Handling

- Task Parallel Library provides a good exception handling story:



If a task throws an exception E that goes unhandled:

- *task is terminated*
- *E is caught, saved as part of an* `AggregateException` *AE, and stored in task object's* `Exception` *property*
- *AE is re-thrown upon .Wait, .Result, or .WaitAll*

Example

No exception handling:

```
Task<int> T = Task.Factory.StartNew( code );
 .
 .
 .

int r = T.Result;
```

With exception handling:

```
Task<int> T = Task.Factory.StartNew( code );
 .
 .
 .

try
{
  int r = T.Result;
}
catch(AggregateException ae)
{
  Console.WriteLine(ae.InnerException.Message);
}
```

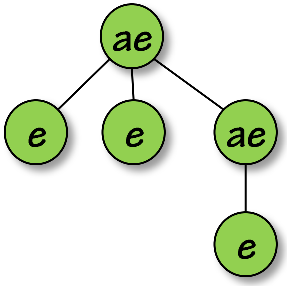
Example #2

- A more general approach…

```
Task T2 = Task.Factory.StartNew( code );
 .
 .
 .

try
{
  T2.Wait();
}
catch(AggregateException ae)
{
  ae = ae.Flatten();
  foreach(Exception ex in ae.InnerExceptions)
    Console.WriteLine(ex.Message);
}
```

Prefer this approach!

If T2 creates sub-tasks, the result could be a tree of exceptions…



==> flatten tree to process exceptions at the leaves…

Exception handling design

- **You should design to "observe" all unhandled exceptions**
  - *otherwise exception is re-thrown when task is garbage-collected*

- **How to observe? Explicitly or implicitly, you must:**

  1. call `.Wait` or touch `.Result` — exception re-thrown at this point, or
  2. call `Task.WaitAll` — exception(s) re-thrown when all have finished, or
  3. touch task's `.Exception` property *after* task has completed, or
  4. subscribe to `TaskScheduler.UnobservedTaskException`

Example — redesigning WaitAllOneByOne

```
List<Task> tasks = new List<Task>();

for (int i=0; i<N; i++)  // Start tasks:
{
  Task t = Task.Factory.StartNew( code );
  tasks.Add(t);
}

while (tasks.Count > 0)  // Wait all, 1-by-1:
{
  int i = Task.WaitAny( tasks.ToArray() );
   .
   .
   .
  tasks.RemoveAt(i);
}
```

no exception thrown here

```
try {
  tasks[i].Wait();
}
catch(AggregateException ae)
{ ... }
```

```
if (tasks[i].Exception != null)
{ ... }
```

```
try {
  var r = tasks[i].Result;
}
catch(AggregateException ae)
{ ... }
```

Last resort exception handling…

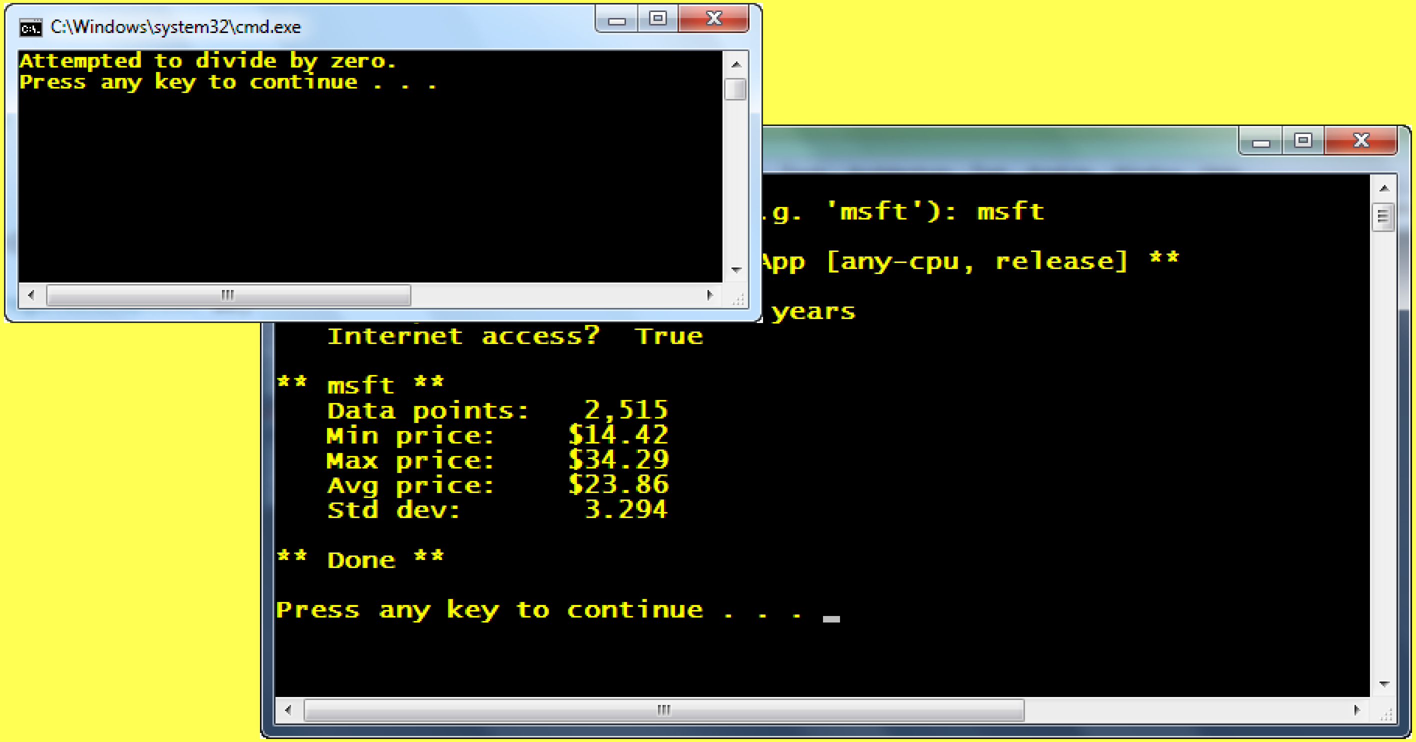
- **Why? Many reasons…**
  - speculative tasks that you don't cancel — what if they throw an exception?
  - 3rd-party libraries that you don't trust…
- **Subscribe to TaskScheduler.UnobservedTaskException**

```
static void Main(string[] args)
{
  TaskScheduler.UnobservedTaskException +=
     new EventHandler<UnobservedTaskExceptionEventArgs>(
     TaskUnobservedException_Handler);
```

```
static void TaskUnobservedException_Handler(
   object sender,
   UnobservedTaskExceptionEventArgs e)
{
   Console.WriteLine("**Unobserved: " + e.Exception.Message);
   e.SetObserved();
}
```

DEMO

- Exception handling
- Historical stock data revisited…



```
Attempted to divide by zero.
Press any key to continue . . .
```

```
.g. 'msft'): msft

App [any-cpu, release] **

 years
   Internet access?  True

** msft **
   Data points:   2,515
   Min price:    $14.42
   Max price:    $34.29
   Avg price:    $23.86
   Std dev:       3.294

** Done **

Press any key to continue . . .
```

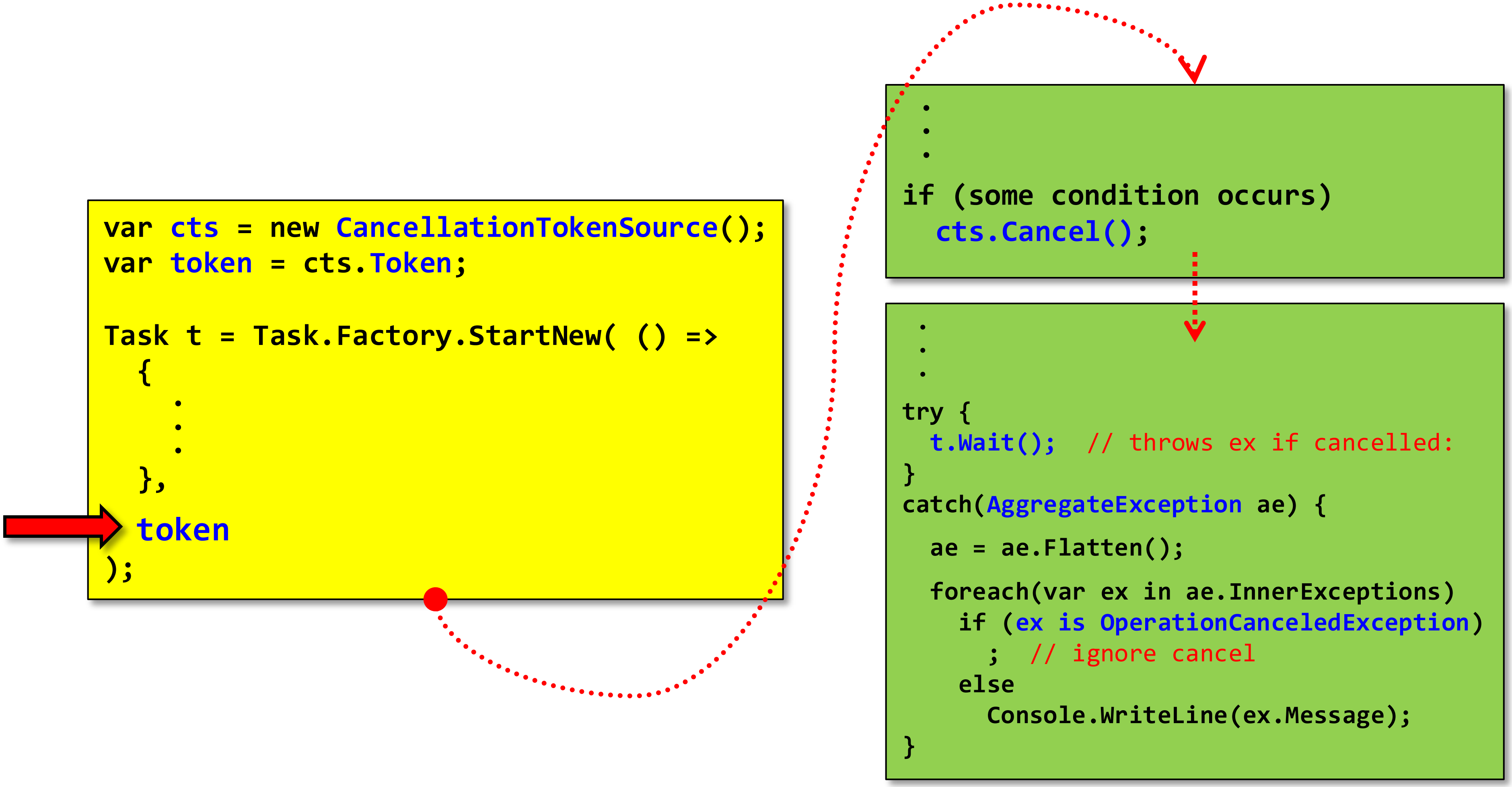
Task cancellation

- Cancellation is a cooperative model

- ***Creator*** *passes a cancellation token, starts task, later signals cancel…*
- ***Task*** *monitors token, if cancelled performs cleanup & throws exception*

Adopting this model guarantees task's Status is set to "Canceled"

Task cancellation — the creator…



```
var cts = new CancellationTokenSource();
var token = cts.Token;

Task t = Task.Factory.StartNew( () =>
  {
     .
     .
     .
  },
  token
);
```

```
 .
 .
 .
if (some condition occurs)
  cts.Cancel();
```

```
 .
 .
 .
try {
  t.Wait();  // throws ex if cancelled:
}
catch(AggregateException ae) {

  ae = ae.Flatten();

  foreach(var ex in ae.InnerExceptions)
    if (ex is OperationCanceledException)
      ;  // ignore cancel
    else
      Console.WriteLine(ex.Message);
}
```

Task cancellation — the task…

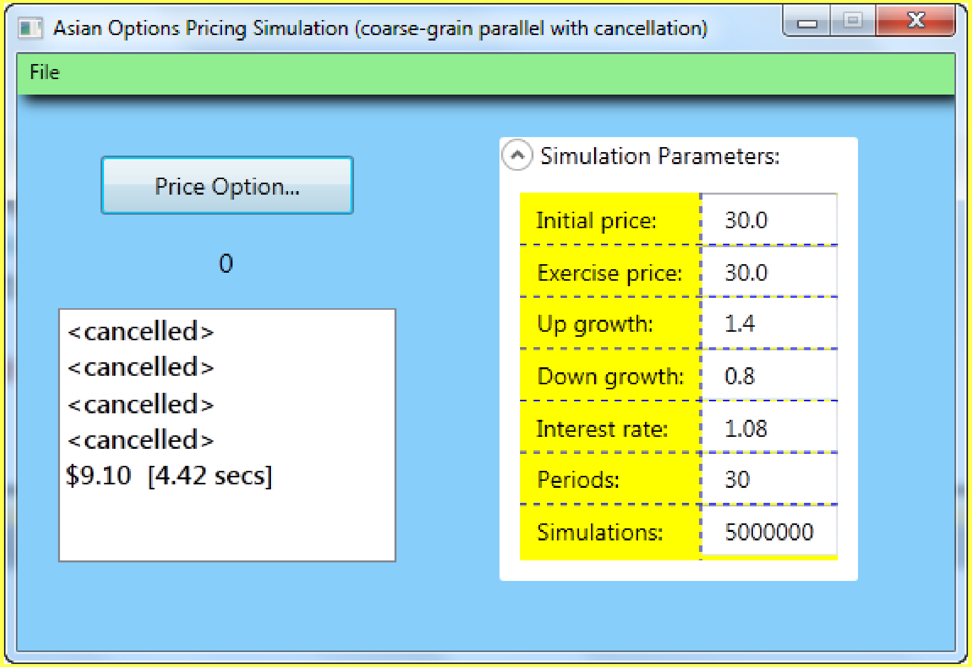
```
var cts = new CancellationTokenSource();
var token = cts.Token;

Task t = Task.Factory.StartNew( () =>
  {
    .
    .
    .
  },
  token
);
```

```
try {
  while(…)  // perform computation:
  {
    // check for cancellation:
    if (token.IsCancellationRequested)
    {
      ...  // cleanup:
      token.ThrowIfCancellationRequested();
    }
    .
    .
    .
  }
}
catch (OperationCancelledException)
{ throw; }
catch (Exception ex)
{ ... }
```

Task must let exception, *OperationCancelledException*, escape back to caller…

DEMO

- Task cancellation
- Asian options pricing


Asian Options Pricing Simulation (coarse-grain parallel with cancellation)
File
Price Option...
Simulation Parameters:

0

```
<cancelled>
<cancelled>
<cancelled>
<cancelled>
$9.10  [4.42 secs]
```

Initial price:	30.0
Exercise price:	30.0
Up growth:	1.4
Down growth:	0.8
Interest rate:	1.08
Periods:	30
Simulations:	5000000

Some final observations…

Task priorities?   Child tasks?

- **Task Parallel Library does not offer notion of task priority**
  - Priorities can be added via custom task scheduler

- **Tasks may form parent-child relationship**
  - Child tasks “attach” to parent
  - Parent task doesn’t complete until all children complete
  - Parent task represents single point of exception handling

```
Task parent = Task.Factory.StartNew( () =>
{
  Task child1 = Task.Factory.StartNew( () => {...},
                     TaskCreationOptions.AttachedToParent );
  Task child2 = Task.Factory.StartNew( () => {...},
                     TaskCreationOptions.AttachedToParent );
    :
    :
});
```

```
try {
  parent.Wait();  // wait for all
}
catch(AggregateException ae)
{ ... }
```

Beware using closures to pass data…

- Suppose you want to create 10 tasks, assigning each a unique id
  - 0, 1, 2, …, 9

```
for (int i=0; i<10; i++)
{
   Task.Factory.StartNew( () =>
   {
     int  taskid = i;
      .
      .
      .
     Console.WriteLine(taskid);
   });
}
```

This does NOT work!

```
10
10
10
10
.
.
.
```

Correct parameter passing

- If value may change, don't use closure --- pass as a task parameter…

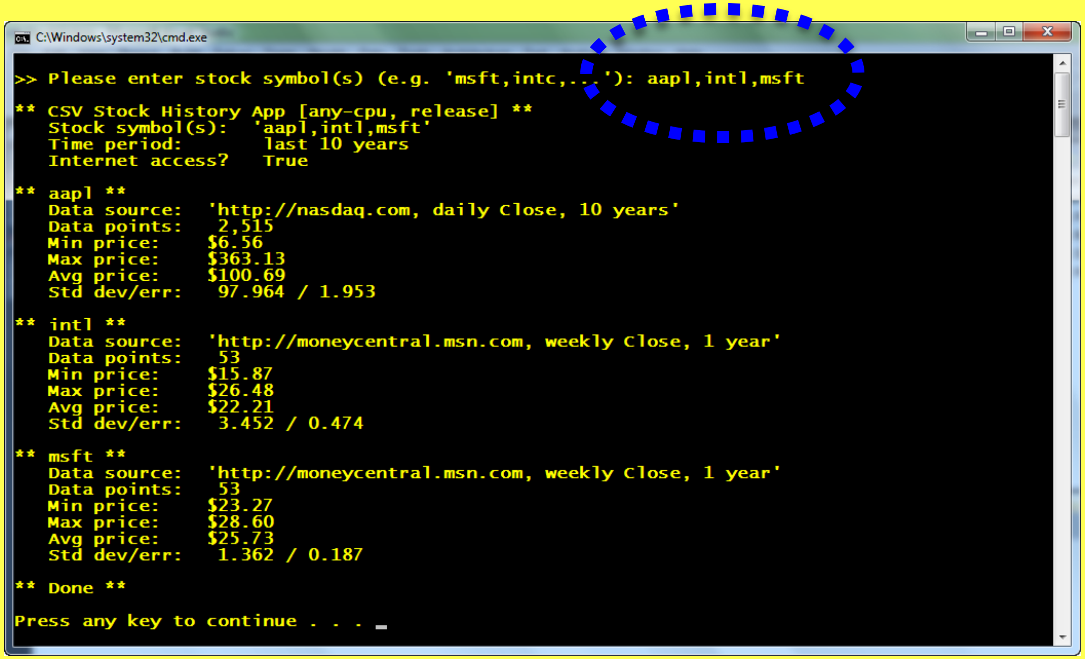
```
for (int i=0; i<10; i++)
{
    Task.Factory.StartNew( (arg) =>
      {
        int  taskid = (int)arg;
         .
         .
         .
        Console.WriteLine(taskid);
      },
      i
    );
}
```

```
0
1
2
3
.
.
.
```

Solution — pass as parameter

DEMO

- Passing parameters safely
- Historical stock data with comma-separated symbols…



```
>> Please enter stock symbol(s) (e.g. 'msft,intc,...'): aapl,intl,msft

** CSV Stock History App [any-cpu, release] **
   Stock symbol(s):  'aapl,intl,msft'
   Time period:       last 10 years
   Internet access?   True

** aapl **
   Data source:  'http://nasdaq.com, daily Close, 10 years'
   Data points:   2,515
   Min price:    $6.56
   Max price:    $363.13
   Avg price:    $100.69
   Std dev/err:   97.964 / 1.953

** intl **
   Data source:  'http://moneycentral.msn.com, weekly Close, 1 year'
   Data points:   53
   Min price:    $15.87
   Max price:    $26.48
   Avg price:    $22.21
   Std dev/err:   3.452 / 0.474

** msft **
   Data source:  'http://moneycentral.msn.com, weekly Close, 1 year'
   Data points:   53
   Min price:    $23.27
   Max price:    $28.60
   Avg price:    $25.73
   Std dev/err:   1.362 / 0.187

** Done **

Press any key to continue . . . _
```

Summary

- **Task Parallel Library provides the necessary support for robust, realistic apps:**
  - *Exception Handling --- be sure to observe all exceptions!*
  - *Task Cancellation --- cooperative model!*
  - *Parent-child tasks*

- **Beware of common mistakes when working with tasks:**
  - *Parameter passing*
  - *Jumbled output*

References

- **Microsoft's main site for all things parallel:**
  - http://msdn.microsoft.com/concurrency
- **MSDN technical documentation:**
  - http://tinyurl.com/pp-on-msdn
- **I highly recommend the following short, easy-to-read book:**
  - *Parallel Programming with Microsoft .NET: Design Patterns for Decomposition and Coordination on Multicore Architectures*, by C. Campbell, R. Johnson, A. Miller and S. Toub, Microsoft Press

    Online: http://tinyurl.com/tpl-book