/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* Part 8: Threading

\*

\* Topic: Using the ThreadStart delegate, which does not provide the

\* capability to send in arguments to the containing method.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* Be sure to provide an integer value on the command line. To do this:

\*

\* 1) Right-click on the project in Solution Explorer and click

\* Properties.

\*

\* 2) In the Properties window, click the Debug tab.

\*

\* 3) In the "Command line arguments" field, enter a whole number.

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

usingSystem**;**

usingSystem.Collections.Generic**;**

usingSystem.Text**;**

usingSystem.Threading**;**

namespaceThreadsDemo

**{**

internalclassComplicatedCalculator

**{**

// Member variable that represent the number of milliseconds

// to pause the thread.

privateint\_millisecondsToPause**;**

// Member variables that are used for the calculation.

privatedouble\_firstNumber**;**

privatedouble\_secondNumber**;**

privatedouble\_results**;**

// Provide the default constructor and set a default value for

// private member variables via a call to a different constructor.

// NEW: Needed to initialize fields.

publicComplicatedCalculator**() :** this **(**1000**)**

**{**

FirstNumber=0**;**

SecondNumber=0**;**

Results=0**;**

**}**

publicComplicatedCalculator**(**intmillisecondsToPause**)**

**{**

MillisecondsToPause=millisecondsToPause**;**

**}**

// Provide another method method that performs the calculation

// without any input or output.

// NEW: Needed to call method without passing in values directly

// from the client. Instead, this method is going to get the

// values stored in the properties. This is because the

// ThreadStart delegate does not allow for any arguments.

internalvoidCalculateValue**()**

**{**

Results=CalculateValue**(**FirstNumber**,** SecondNumber**);**

**}**

// This method represents a task that could potentially run for

// a long period of time.

internaldoubleCalculateValue

**(**doublefirstNumber**,** doublesecondNumber**)**

**{**

doubleanswer=0**;**

// Save the foreground color of the console window.

ConsoleColororiginalcolor=Console.ForegroundColor**;**

// Change the foreground color in the console.

Console.ForegroundColor=ConsoleColor.Red**;**

// Display a message that we're starting the task.

Console.WriteLine**(**"\n\tStarting the calculation task..."**);**

// Set the console color back to the original value.

Console.ForegroundColor=originalcolor**;**

// Pause for a moment.

System.Threading.Thread.Sleep**(**MillisecondsToPause**);**

// Perform the calculation.

answer=Math.Pow**(**firstNumber**,** secondNumber**);**

// Pause for another moment.

System.Threading.Thread.Sleep**(**MillisecondsToPause**);**

// Change the foreground color in the console.

Console.ForegroundColor=ConsoleColor.Red**;**

// Display a message that we're done with the task.

Console.WriteLine**(**"\n\tDone with the calculation task."**);**

// Set the console color back to the original value.

Console.ForegroundColor=originalcolor**;**

returnanswer**;**

**}**

// Make this available to code outside of this class.

internalintMillisecondsToPause

**{**

get **{** return\_millisecondsToPause**; }**

set

**{**

if **(**value<0**)**

**{**

thrownewArgumentException

**(**"Milliseconds must be greater than or equal to 0."**);**

**}**

\_millisecondsToPause=value**;**

**}**

**}**

// Provide access to the first number.

// NEW: Needed to store value for CalculateValue method to use.

internaldoubleFirstNumber

**{**

get **{** return\_firstNumber**; }**

set **{** \_firstNumber=value**; }**

**}**

// Provide access to the second number.

// NEW: Needed to store value for CalculateValue method to use.

internaldoubleSecondNumber

**{**

get **{** return\_secondNumber**; }**

set **{** \_secondNumber=value**; }**

**}**

// Provide a way to allow code outside this class to access

// the results. This is read-only to code outside this class

// (notice the private accessor on set).

// NEW: Needed for client to retrieve resulting value from method.

internaldoubleResults

**{**

get **{** return\_results**; }**

privateset **{** \_results=value**; }**

**}**

**}**

classProgram

**{**

privatestaticintGetMilliseconds**(**strings**)**

**{**

intmilliseconds=0**;**

// If this call fails, milliseconds will be set to zero.

if **(**int.TryParse**(**s**,** outmilliseconds**))**

**{**

// If the user types in a low number, let's assume

// that they entered in the number of seconds and

// convert the value to milliseconds.

if **(**milliseconds<250**)**

**{**

milliseconds\*=1000**;**

**}**

**}**

returnmilliseconds**;**

**}**

staticvoidMain**(**string**[]** args**)**

**{**

try

**{**

// Display a message to show we're in Main().

Console.WriteLine**(**"Starting the program."**);**

// Get the number of milliseconds from the arguments

// passed in from the command line.

intmilliseconds=GetMilliseconds**(**args**[**0**]);**

// Create the ComplicatedCalculator object.

ComplicatedCalculatorcc=

newComplicatedCalculator**(**milliseconds**);**

// Populate the data in the ComplicatedCalculator object.

// NEW: Store values into class so method can use them.

cc.FirstNumber=10.4**;**

cc.SecondNumber=7.451**;**

// Create the ThreadStart delegate. This delegate contains

// the method that will be called when the secondary thread

// starts.

ThreadStartthreadedMethod=

newThreadStart**(**cc.CalculateValue**);**

// Create the thread object and start the thread.

ThreadsecondaryThread=newThread**(**threadedMethod**);**

secondaryThread.Start**();**

// Display some messages to show that Main() is still

// responsive while the calculation is going on.

Console.WriteLine

**(**"\nNow I'm going to go do something else."**);**

System.Threading.Thread.Sleep**(**1500**);**

Console.WriteLine**(**"\nLike talk about the weather."**);**

System.Threading.Thread.Sleep**(**1500**);**

Console.WriteLine**(**"\nOr the latest news."**);**

System.Threading.Thread.Sleep**(**1500**);**

Console.WriteLine**(**"\nYou know, my foot hurts."**);**

System.Threading.Thread.Sleep**(**1500**);**

Console.WriteLine**(**"\nI love hotdogs!"**);**

System.Threading.Thread.Sleep**(**1500**);**

Console.WriteLine**(**"\nHow much is a shake at Burgermaster?"**);**

System.Threading.Thread.Sleep**(**1500**);**

Console.WriteLine**(**"\nOk, now I'm getting hungry!"**);**

System.Threading.Thread.Sleep**(**1500**);**

// How do we know that we have the answer? At this point

// we don't. We'll take a chance for now and hope that the

// answer is there. If the argument passed into Main() is

// set to 2500, then this will work. If it's set to 7500,

// then the result will be 0.

Console.WriteLine**(**"\n\tThe result is: {0}"**,** cc.Results**);**

**}**

catch **(**Exceptione**)**

**{**

Console.WriteLine**(**"\nEXCEPTION: {0}."**,** e.Message**);**

**}**

// Pause so we can look at the console window.

Console.Write**(**"\n\nPress <ENTER> to end: "**);**

Console.ReadLine**();**

**}**

**}**

**}**