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

\*

\* Part 10: Threading

\*

\* Topic: Thread properties: Name, ManagedThreadID, CurrentThread,

\* IsAlive.

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

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

\*

\* 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\_results**;**

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

**{**

MillisecondsToPause=millisecondsToPause**;**

**}**

// Provide yet another method that takes a single object

// argument. This will parse the object and get the input values

// from it.

internalvoidCalculateValue**(**objectinput**)**

**{**

// Attempt to convert the input object to an array of

// doubles.

double**[]** inputValues=inputasdouble**[];**

// If the conversion worked and there are at least two elements

// in the double array, run the calculation.

if **(**null!=inputValues&&inputValues.Length>=2**)**

**{**

Results=CalculateValue**(**inputValues**[**0**],** inputValues**[**1**]);**

**}**

**}**

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

// a long period of time.

internaldoubleCalculateValue

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

**{**

doubleanswer=0**;**

// Get the currently-running thread object.

ThreadthreadObject=Thread.CurrentThread**;**

// 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 on thread {0}: {1}."**,**

threadObject.ManagedThreadId**,** threadObject.Name**);**

// 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 on thread {0}: {1}."**,**

threadObject.ManagedThreadId**,** threadObject.Name**);**

// 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 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).

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**)**

**{**

// Get the currently-running thread object.

ThreadprimaryThreadObject=Thread.CurrentThread**;**

// Set the name of the thread. This will help with debugging

// when looking at the Threads window.

primaryThreadObject.Name="The Main Thread"**;**

// Get the thread ID so that we can use it in output statements.

intthreadId=primaryThreadObject.ManagedThreadId**;**

try

**{**

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

Console.WriteLine**(**"{0}: Starting the program."**,** threadId**);**

// 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**);**

// Create the ParameterizedThreadStart delegate. This

// delegate will be used to pass an array of doubles

// to the method on the secondary thread.

double**[]** numbers= **{** 10.4**,** 7.451 **};**

ParameterizedThreadStartthreadedMethod=

newParameterizedThreadStart**(**cc.CalculateValue**);**

// Create the thread object and start the thread. In this

// case, when we call Start(), we pass in the double array

// as an argument.

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

// Set the name of the secondary thread.

secondaryThread.Name="Calculation Thread"**;**

// Start the thread.

secondaryThread.Start**(**numbers**);**

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

// responsive while the calculation is going on.

Console.WriteLine

**(**"\n{0}: Now I'm going to go do something else."**,**

threadId**);**

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

Console.WriteLine**(**"\n{0}: Like talk about the weather."**,**

threadId**);**

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

Console.WriteLine**(**"\n{0}: Or the latest news."**,**

threadId**);**

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

Console.WriteLine**(**"\n{0}: You know, my foot hurts."**,**

threadId**);**

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

Console.WriteLine**(**"\n{0}: I love hotdogs!"**,**

threadId**);**

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

Console.WriteLine

**(**"\n{0}: How much is a shake at Burgermaster?"**,**

threadId**);**

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

Console.WriteLine**(**"\n{0}: Ok, now I'm getting hungry!"**,**

threadId**);**

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

// This time we'll poll for the IsAlive property. Once it's

// false, we can get the results. This is still dangerous

// because what if the thread never got started for some

// reason?

while **(**secondaryThread.IsAlive**)**

**{**

Thread.Sleep**(**750**);**

Console.WriteLine

**(**"\n{0}: Seeing if the thread is done."**,**

threadId**);**

**}**

Console.WriteLine**(**"\n\t{0}: The result is: {1}"**,**

threadId**,** cc.Results**);**

**}**

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

**{**

Console.WriteLine**(**"\n{0}: EXCEPTION: {1}."**,**

threadId**,** e.Message**);**

**}**

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

Console.Write**(**"\n\n{0}: Press <ENTER> to end: "**,**

threadId**);**

Console.ReadLine**();**

**}**

**}**

**}**