**GIVE IT A TRY #1:**

* Open the starter solution found in **Exercise Starters\GiveItATry\01\_CallbackMethods**.
* Locate each **// TODO:** comment and implement the code that the comment describes.
* The solution can be found in **⮨  
  Exercise Solutions\GiveItATry\01\_CallbackMethods**.

**GIVE IT A TRY #2:**

* Create a new Console application in Visual Studio.
* Be sure to include using **System.Threading** at the top of your file.
* Create a class called **IntervalCounter**.
  + It has two private member variables of type **int** called **interval** and **divisor**.
  + Supply a constructor that takes two **int** arguments and stores them in the member variables declared above. Make sure divisor is not 0! You might want to use properties to include validation code.
  + Implement a method called **DisplayIntervals()** that takes no parameters and returns **void**. This is the method all worker threads will start with.
  + Get the current thread by calling **Thread.CurrentThread**.
  + Display a message with the name of the thread.
  + Create a loop from 1 to interval. If the counter is evenly divisible by the divisor, display a status message with the current counter value.
* Test threading in your **Main()** method
  + Prompt for the **interval** and **divisor** values.
  + Create a new **IntervalCounter** object.
  + Create the **ThreadStart** delegate, providing **DisplayIntervals()** as the target method. You can use an anonymous method if you want.
  + Create the **Thread** object giving it the **ThreadStart** delegate object.
  + Set the **Name** property of the thread object.
  + Call **Start()** on the thread object and then call **Join()**.
* Test the program and walk through it in debug to see how the thread works.
* The solution can be found in **⮨  
  Exercise Solutions\GiveItATry\02\_IntervalCounter**.

**GIVE IT A TRY #3:**

* Open the starter solution found in **⮨ Exercise Starters\GiveItATry\03\_CriticalSections**.
* Locate each **// TODO:** comment and implement the code that the comment describes.
* The solution can be found in **⮨  
  Exercise Solutions\GiveItATry\03\_CriticalSections**.

EXERCISE #1:

|  |  |
| --- | --- |
| WHAT TO DO | HOW TO DO IT |
| Update a single-threaded application by implementing threads. | Open the ThreadSorting.sln file with Visual Studio .NET in the Exercise Starters\TakeHomeExercises\01\_NumberSearchAndSort.  Click the View menu, highlight Show Tasks and then click All. The Task List should appear, usually at the bottom of the screen.  Double-click each task and add or modify the code that the task describes.  Compile and run the program to make sure it works.  Rerun the program, but this time step through it in debug. If you need to, freeze one of the threads so you can see what the other one is doing. However, try to debug the program with both threads running and see if you can keep their paths straight in your mind. This will give you a good idea of how to debug a multithreaded application. |

EXERCISE #2:

|  |  |
| --- | --- |
| WHAT TO DO | HOW TO DO IT |
| Modify the multi-threaded application from exercise 1 by replacing the threading logic with asynchronous calls. | Open the AsyncSorting.sln file with Visual Studio .NET in Exercise Starters\TakeHomeExercises\02\_NumberSearchAndSort.  Click the View menu, highlight Show Tasks and then click All. The Task List should appear, usually at the bottom of the screen.  Double-click each task and add or modify the code that the task describes.  Compile and run the program to make sure it works.  Rerun the program, but this time step through it in debug. Try to follow the steps that the runtime takes to make the call and provide information in the callback method. |