Lab Assignment 6

Jerry Belmonte
Keira Wong

CODE

PART I: async & wait

```
// JERRY: redo lab 3 using async and await for synchronization
public class StockBroker
{
        // Reducing the semaphore's initialCount to 1 reduces the maximum
        // parallelism to 1, turning this into an asynchronous lock.
        private static SemaphoreSlim _semaphore = new SemaphoreSlim(1);

        // StockBroker class constructor and non async methods omitted ...

        /// <summary>
        ///     Adds stock objects to the stock list
        /// </summary>
        /// <param name="stock">Stock object</param>
        public void AddStock(Stock stock)
        {
                stocks.Add(stock);
                stock.StockValueChanged += async (sender, args) =>
                {
                   string statement = $"{BrokerName,-10}{args}";
                   await WriteStatementAsync(statement);
                };
        } // End of the AddStock method

        async Task WriteStatementAsync(string text)
        {
                await _semaphore.WaitAsync();
                try
                {
                   using (FileStream stream = new FileStream(docPath,
```

```
                                                FileMode.Append, FileAccess.Write,
                                                FileShare.None, 4096, true))
                    using (StreamWriter writer = new StreamWriter(stream))
                    {
                        await writer.WriteLineAsync(text);
                        Console.WriteLine(text);
                    }
                }
                finally
                {
                    _semaphore.Release();
                }
        } // End of the WriteStatementAsync method
} // End of the StockBroker class
```

Lab 3 was converted to an asynchronous program by replacing the ReaderWriterLockSlim with SemaphoreSlim to limit concurrency with asynchronous operations. By instantiating the semaphore with a capacity of 1, we were able to make it similar to a lock, except that any thread could call Release on a Semaphore because it is thread agnostic.In the AddStock method we implemented an asynchronous lambda expression to attach the event handler. The semaphore was necessary to limit concurrency and prevent multiple threads from executing the WriteStatementAsync method. The _semaphore.WaitAsync() and _semaphore.Release() methods were used to make asynchronous operations execute sequentially. Lastly, we modified the block of code that performed the duties of writing to an output text file by using a FileStream with the asynchronous parameter set to true and using the StreamWriter's WriteLineAsync(string) method.

PART II: Queries A, B, & C
JoiningTableData.cs

```
private void JoiningTableData_Load(object sender, EventArgs e)
      {
         // Entity Framework DBContext
         BooksEntities dbcontext =
           new BooksEntities();


         // KEIRA: A) Titles & Authors ----------------------------------------------

         // get titles and the authors who wrote them
```

```
        var titlesAndAuthors =
          from book in dbcontext.Titles
          from author in book.Authors
          orderby book.Title1
          select new
          {
             book.Title1,
             author.FirstName,
             author.LastName
          };

        outputTextBox.AppendText("\r\n\r\nTitles and Authors:");

        // display authors and titles in tabular format
        foreach (var element in titlesAndAuthors)
        {
            outputTextBox.AppendText(
              String.Format("\r\n\t{0,-10} {1} {2}",
                element.Title1, element.FirstName, element.LastName));
        } // end foreach


        // KEIRA: B) Authors & Titles ----------------------------------------------

        // get authors and titles w/ authors sorted for each title
        var authorsAndTitles =
          from book in dbcontext.Titles
          from author in book.Authors
          orderby book.Title1, author.LastName, author.FirstName
          select new
          {
             book.Title1,
             author.FirstName,
             author.LastName
          };

        outputTextBox.AppendText("\r\n\r\nAuthors and titles with authors sorted for
each title:");

        // display authors and titles in tabular format
        foreach (var element in authorsAndTitles)
        {
            outputTextBox.AppendText(
              String.Format("\r\n\t{0,-10} {1} {2}",
                element.Title1, element.FirstName, element.LastName));
        } // end foreach
```

```
        // KEIRA: C) Authors Grouped by Title ----------------------------------------

        // get titles and authors of each book
        // group by title
        var authorsByTitle =
          from title in dbcontext.Titles
          orderby title.Title1
          select new
          {
             Title = title.Title1,
             Authors =
               from author in title.Authors
               orderby author.LastName, author.FirstName
               select author.FirstName + " " + author.LastName
          };

        outputTextBox.AppendText("\r\n\r\nAuthors grouped by title:");

        // display titles written by each author, grouped by author
        foreach (var title in authorsByTitle)
        {
           // display title's name
           outputTextBox.AppendText("\r\n\t" + title.Title + ":");

           // display titles written by that author
           foreach (var author in title.Authors)
           {
              outputTextBox.AppendText("\r\n\t\t" + author);
           } // end inner foreach
        } // end outer foreach


    } // end method JoiningTableData_Load
```

"dbcontext" represents our database of Books. To get a list of titles and authors, for each Title that is currently in our database's collection of Titles (which we refer to as "book"), for each author in that Title's collection of Authors, we want to select the title of that book and the first name and last name of the author of that book. We save this selection as a collection of "titlesAndAuthors" and use this var to print the attributes we've selected in our query. To get a list of authors and titles, with the authors sorted for each title, for each Title that is currently in our database's collection of Titles (which we refer to as "book"), for each author in that Title's collection of Authors, we select the same attributes as in the previous query -- the title of that book and the first and last name of the author of that book but order it by title of the book, then by the first name of

that author, and then by the last name of that author. To get a list of Authors grouped by Title, for each Title in our database's collection of Titles, we select the title name of that Title (called "Title) and the first and last name of the authors from that Title's list of Authors (called "Authors") so that for each title in our selection ("authorsByTitle") we can display the title of that title and loop through that Title's list of authors ("Authors") and print the name of each author in that Title's list of authors.