**Advance C#, Dot Net Framework**

**Delegates**

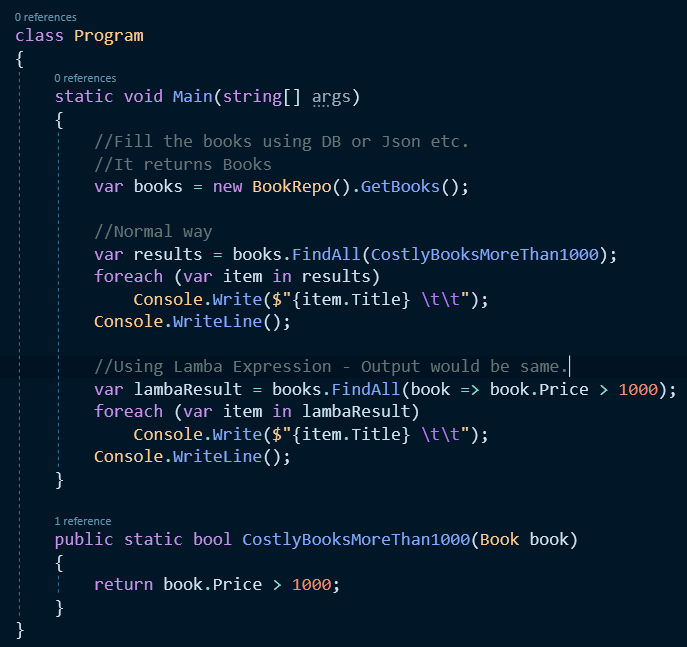
1. An object that knows how to call a method (or group of methods).
2. A reference to a function.
3. Generic Delegates – Action<T>, Func<T, out result> & Predicate<T>
4. Why do we need a delegates?
   1. For designing extensible and flexible application e.g. frameworks etc.
5. Interfaces or Delegates for extensibility
   1. Use a delegate when
      1. Event based mechanism is required.
      2. The caller doesn’t need to access other properties or methods on the object implementing the methods.
6. Short Example – Delegate vs Generic Delegates
   1. The actual implementation would not be able to consume “RemoveNumber” function. Hence we opted for delegate to extend the functionality.



1. For normal implementation check the github repo – “AdvanceGenericExample” example.

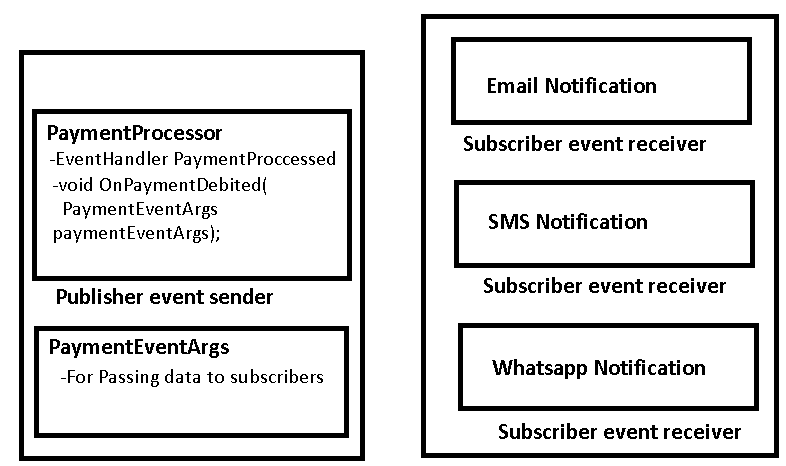
**Lamba Expressions**

1. Lamba Expressions are nothing but anonymous methods that doesn’t have name, access modifier.
2. Why do we use Lamba Expressions
   1. For convenience
3. Example –



**Events**

1. Events
   1. A mechanism for communication between objects. (notify other objects regarding an event happened(past tense) or happening.(present tense))
   2. Used in building Loosely coupled Applications
   3. Helps extending applications
2. Events helps us build publisher subscriber architecture that leads to loosely coupled apps & the apps can be easily extended for new features. For example Payment System
   1. Payment Processor class doesn’t need to know about Email, SMS & Whatsapp Notifications and vice-verse
   2. Payment Processor only need to provide event handler object that can be used by the subscriber’s classes to subscribe to this event.
   3. If event handler is not subscribed than it will not notify the notification class regarding the event.



1. Example



**Linq**

1. Linq gives us the capability to query objects, database tables, datatables, xmls & entities.
2. Examples of some linq extension methods



**Asynchronous Programming – How & Why & Syntax - Async and Await.**

1. Synchronous Vs Asynchronous
   1. Synchronous – Performing steps one after the another

E.g. Buying a Railway ticket offline

* + 1. User need to reach railway station
    2. Get in the line to buy a ticket
    3. Wait till his/her turns to buy the ticket comes
    4. Board the train

As you can see, we need to perform step 1, 2, 3 & 4 one after another. Basically we are performing the activity in a synchronously manner.

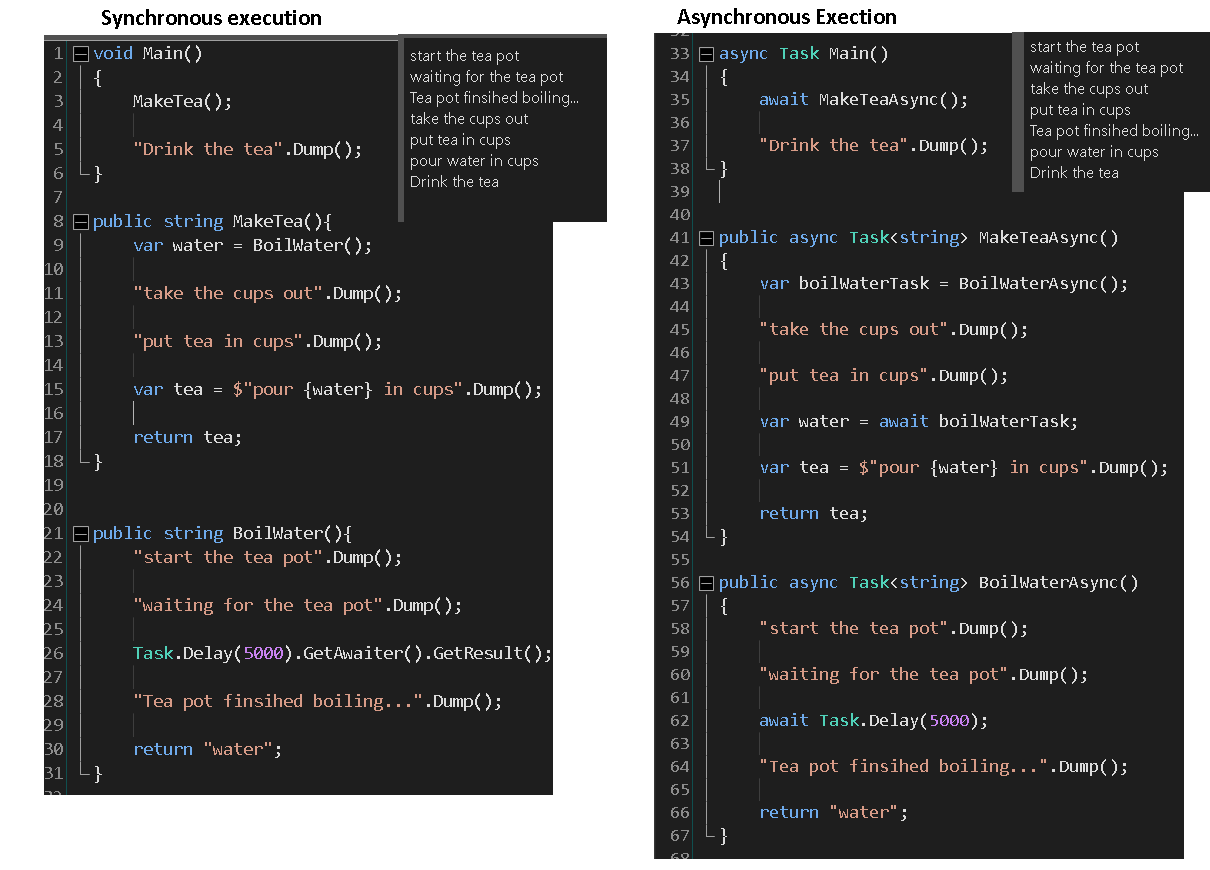
* 1. Asynchronous – Performing steps simultaneously.

E.g. Buying a Railway ticket online.

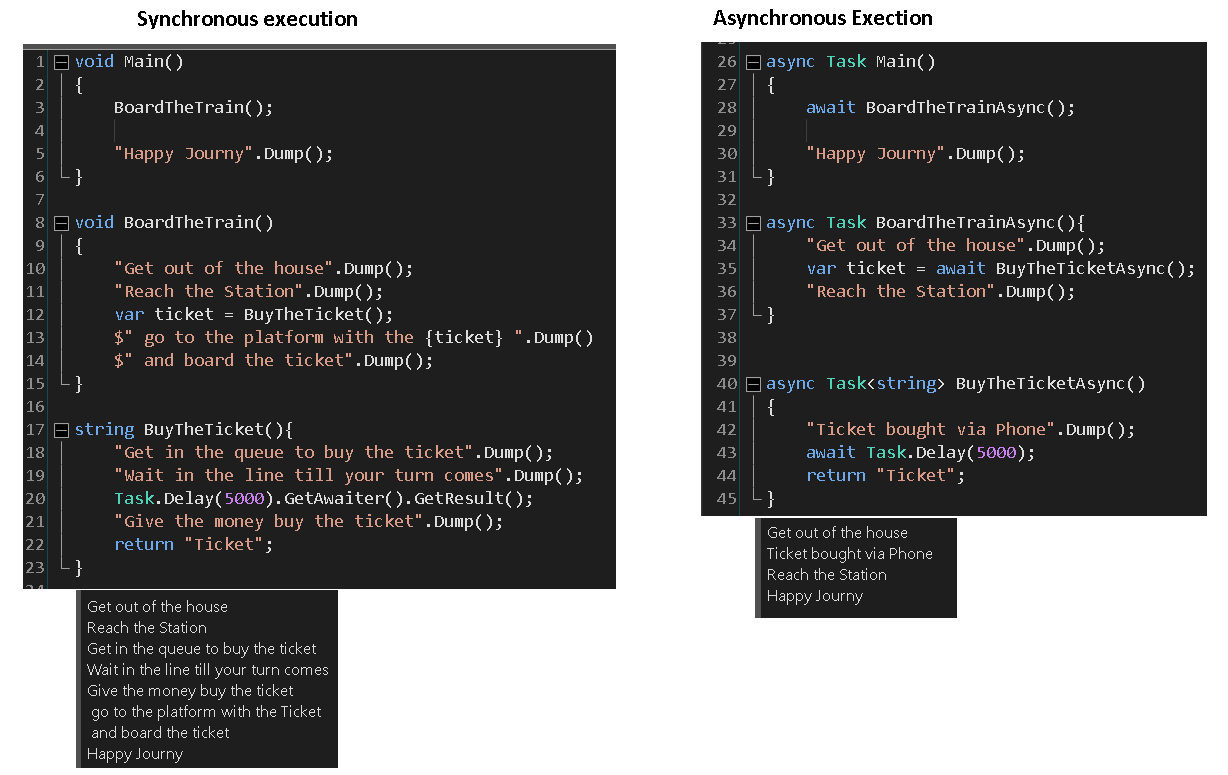
* + 1. User starts his journey towards the railway station.
    2. Meanwhile using his phones app – UTS or xyz App.
    3. Logs in & buy the ticket
    4. Boards the train.

As you can see, User minimized his waiting time by performing step 1 & 2 simultaneously.

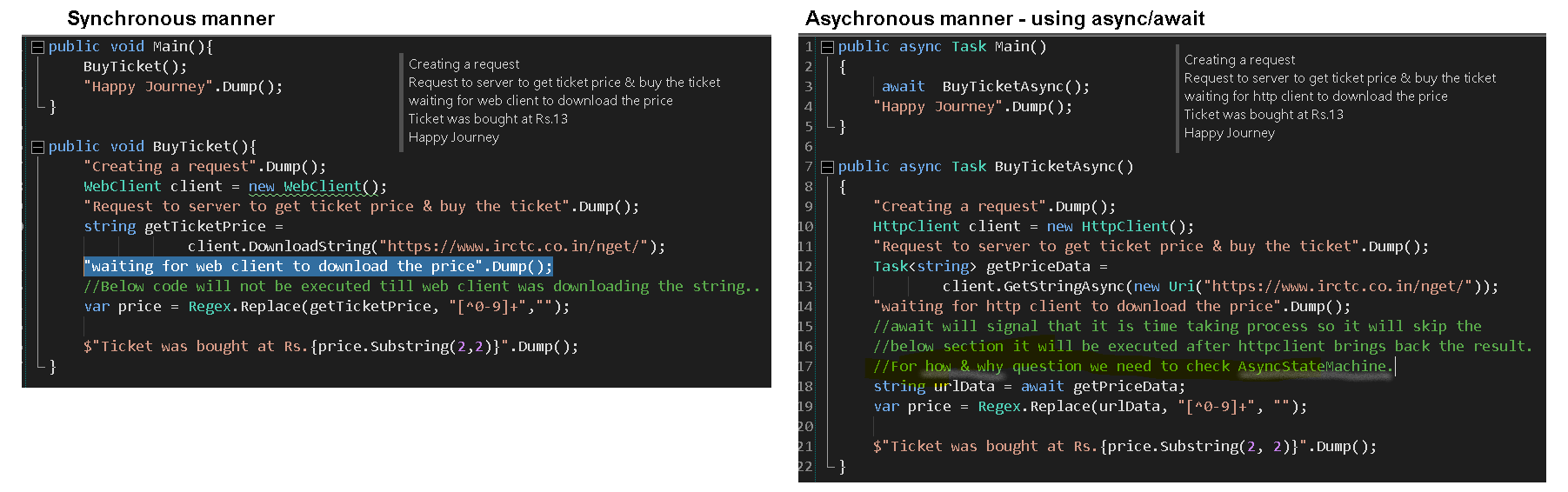
1. Benefits of Asynchronous mannerism
   1. Performance – We were able to do the task in less amount of time. i.e. we minimized the waiting time in ticket counter.
   2. Scalability – Railway was able to cater more people in less amount of time.
2. So Microsoft has provided us with two keywords **async** & **await**
3. A simple code example – Making a Tea & Drink a Tea in Sync & Async manner.



1. A simple code to demo – buy railway ticket offline (synchronous) vs online (asynchronous)…



1. Asynchronous programming is not same as creating multiple threads to perform the work in parallel.
2. **async/await** does not create any new threads.it just utilizes the current execution thread more efficiently so that we don’t need to worry about race conditions.
3. Two types of tasks that block the execution thread(Main Thread)
   1. IO Bound Tasks
      1. App waiting for database query to return call.
      2. App waiting for response of an http call.
      3. App waiting for azure storage SDK to return data.
   2. CPU bound Tasks
      1. App is waiting for some complex computation to complete.
4. A simple example of IO Bound Task- it looks the same but it is not…



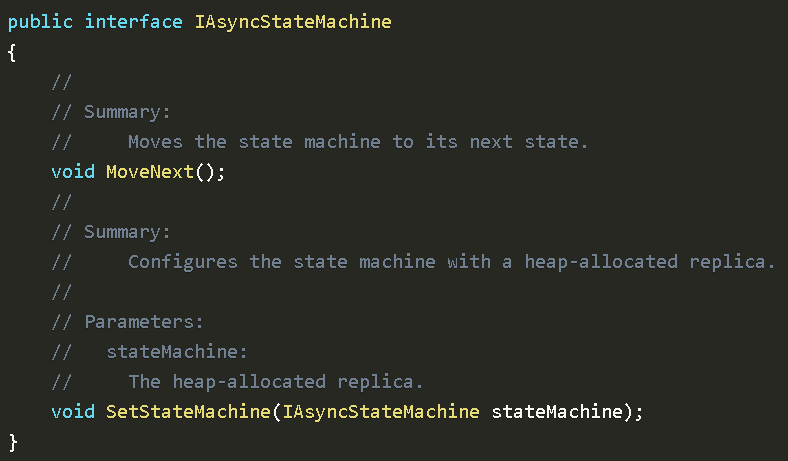
1. When we use async/await the code splits in to before await keyword and after await keyword.

**More await keyword the compiler detects it splits the code into more parts.**

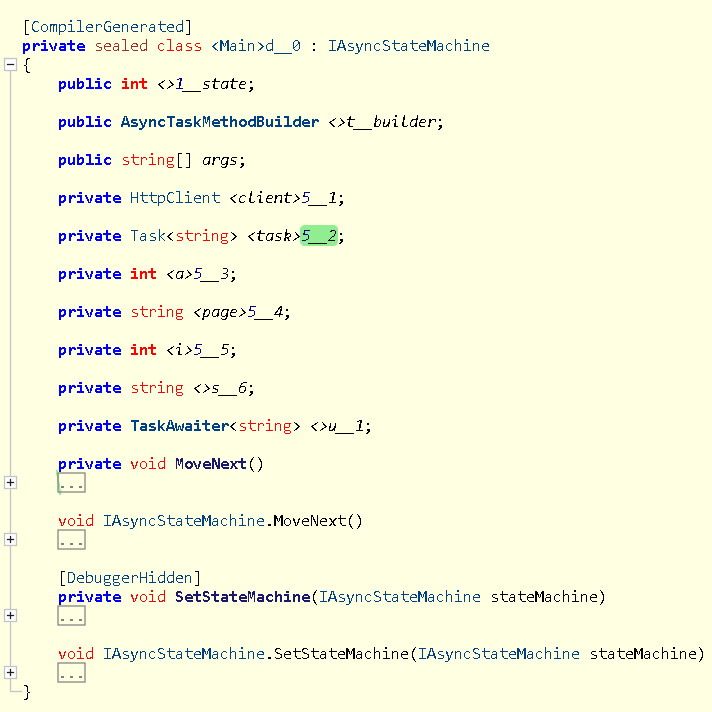
As you see in the below example, Thread 1 was used for completing all the steps till step 4 and then after Step 4 it detected await keyword and Thread 24 completed Step 5 & 6.



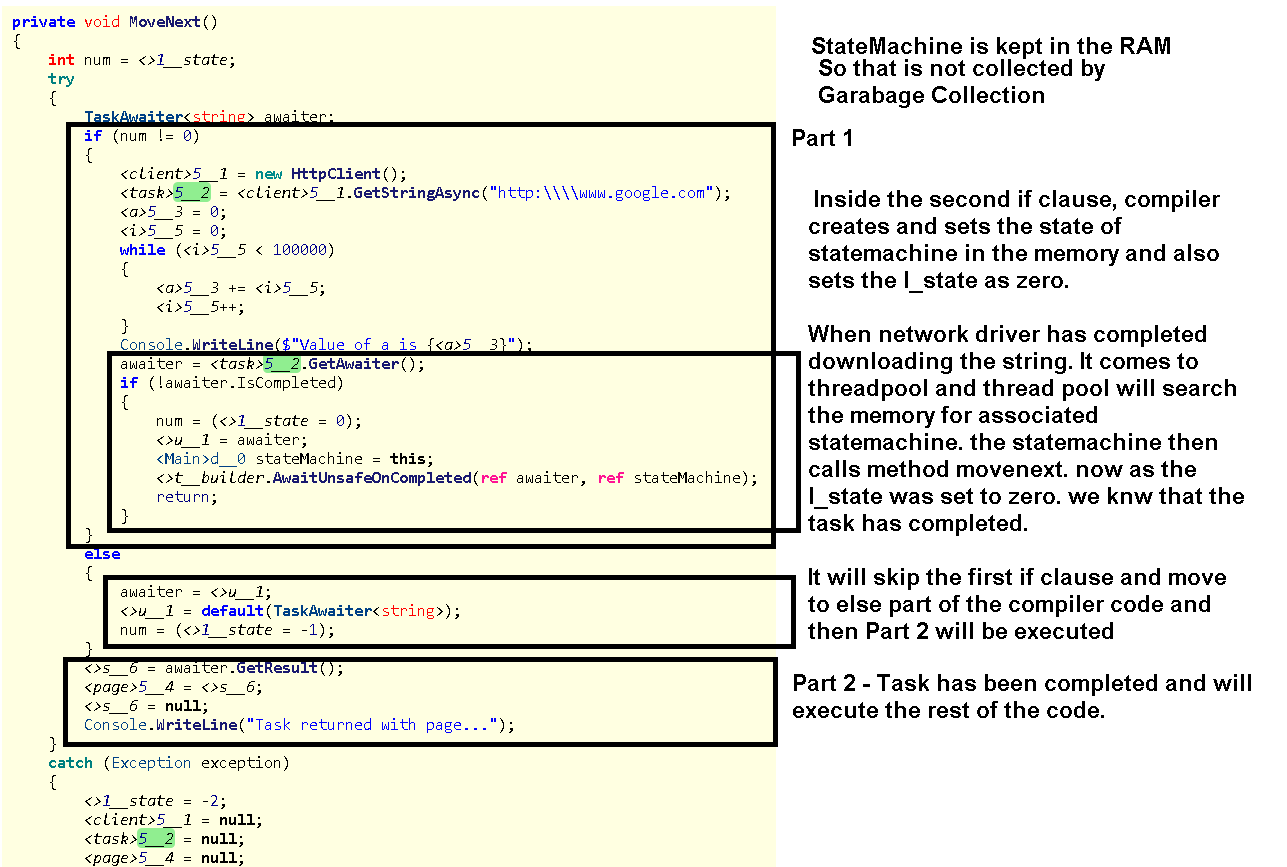
1. Now let’s have a sneak peek on Async State Machine which helped us to achieve the code split
   1. AsyncStateMachine stays in the RAM so that it is not collected by garbage collector.
   2. IAsyncStateMachine is interface which contains two methods MoveNext() & SetStateMachine(IAsyncStateMachine)



* 1. Compiler generated code for async/await



* 1. Now let see what is inside MoveNext() method where the magic happens

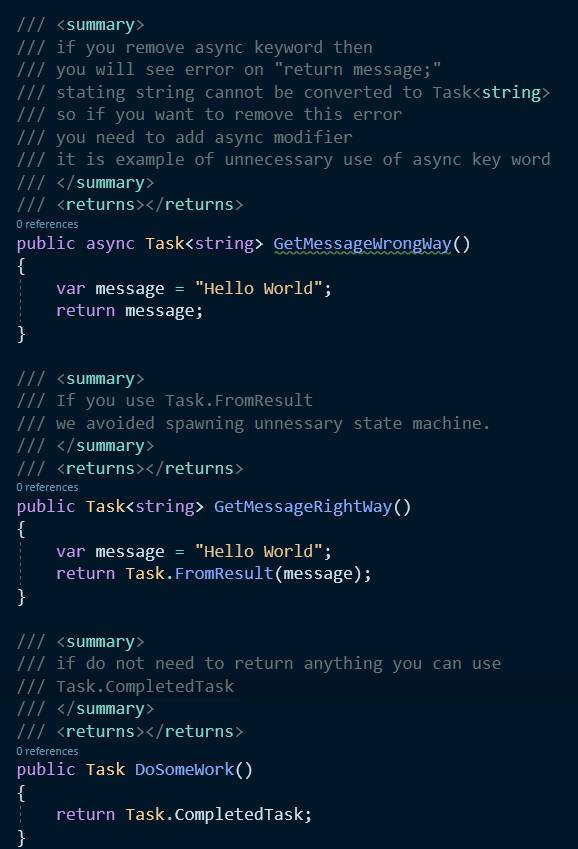


* 1. More the await keyword in the code, more async state machine will be created in the memory to preserve the state and more complex code will be generated by the compiler.

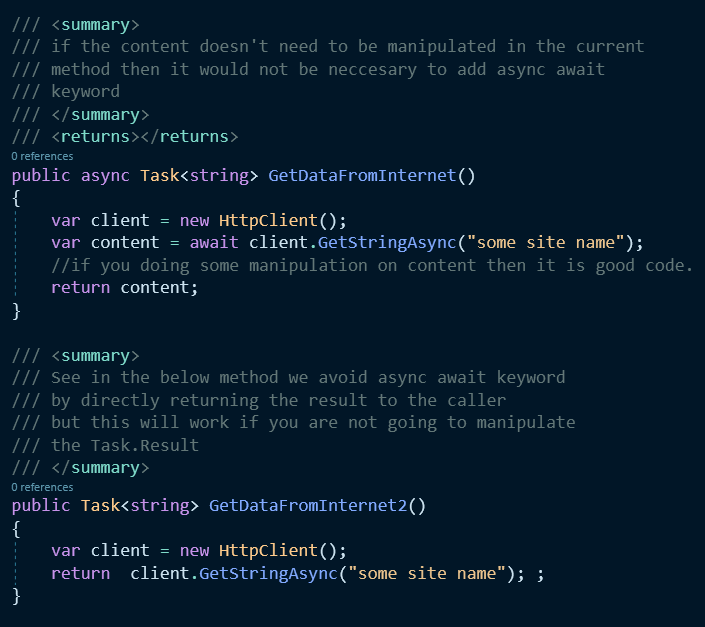
1. In other programming languages async and await is known as promise and future.

**Asynchronous Programming – Pitfalls - Async and Await.**

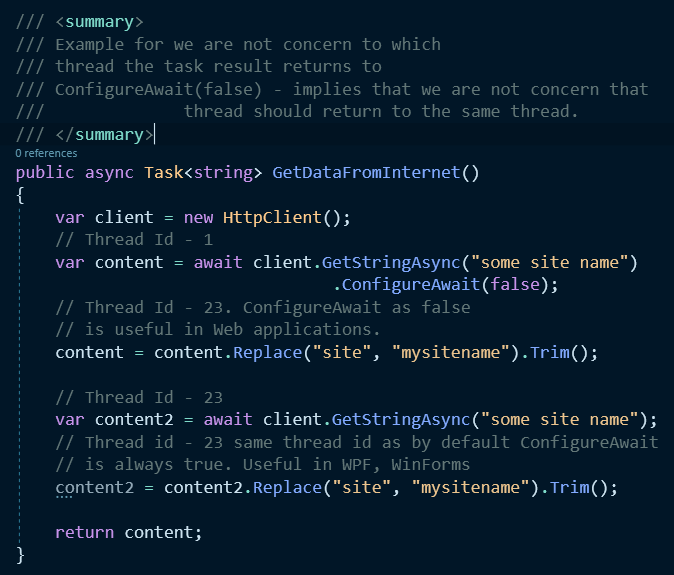
1. Try avoiding state machine by not adding unnecessary async await keywords in the code
2. Async\Await should only be used when there is input output scenarios involved in the code.
   1. When your code will communicates with some external source i.e. Database, Disk drive, Network drive etc.
3. Some example to avoid unnecessary async await & optimization of the code.
   1. Example 1



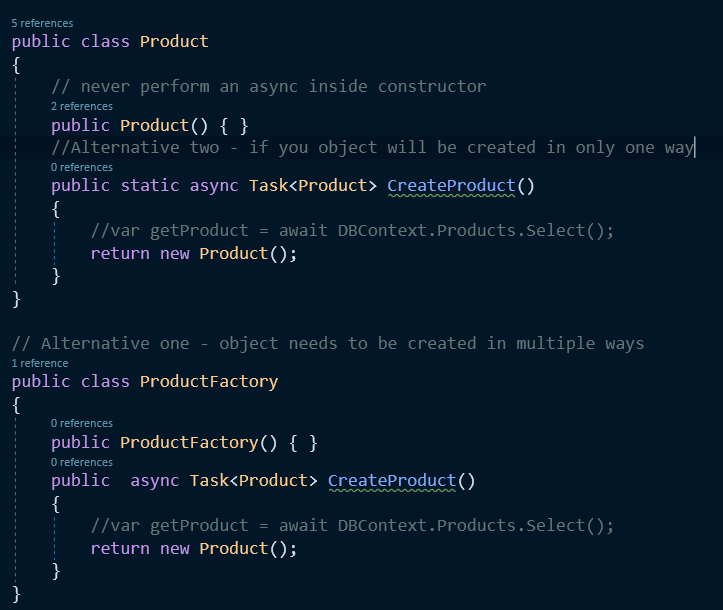
* 1. Example 2 – Return the async directly to the caller if result doesn’t need to be manipulated.



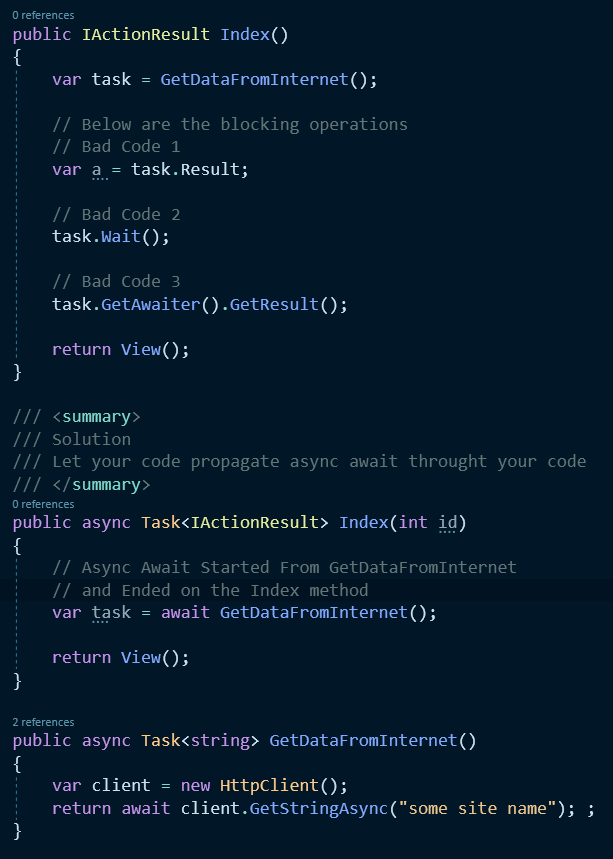
* 1. Example 3 – In web apps, we don’t need the task to return to the same thread but in wpf, winforms we require that the awaited task should return to the same thread.



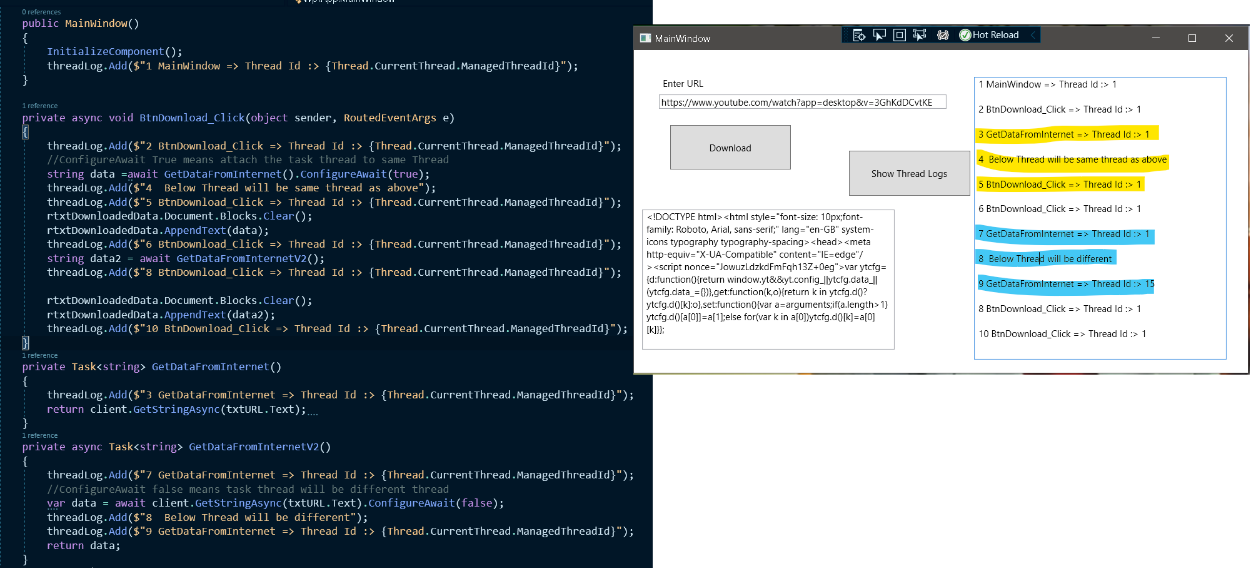
* 1. Example 4 – **Don’t async in constructor ever if you need to do then use static method or factory pattern.**



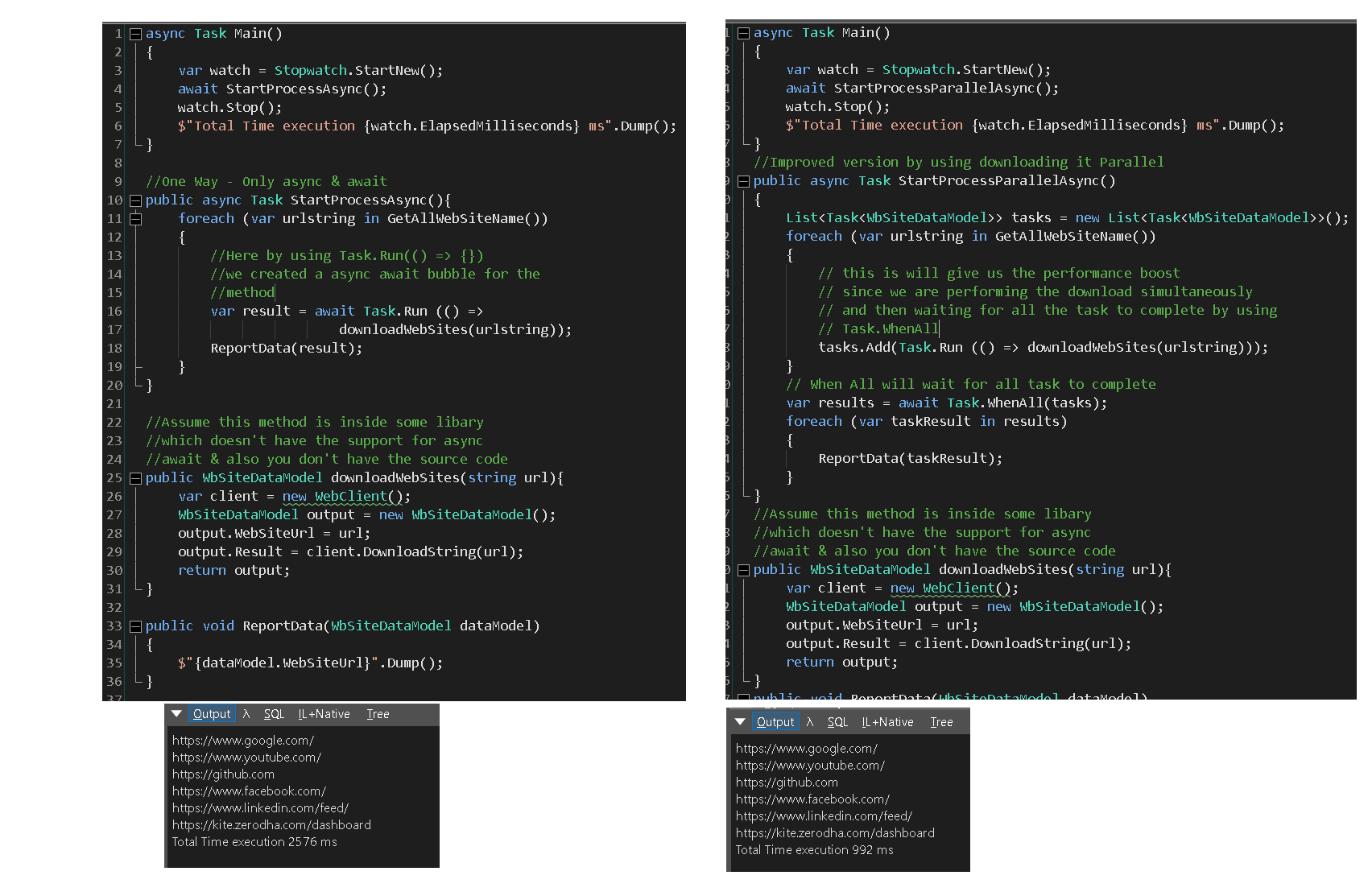
* 1. Example 5 – Blocking the Thread by using task.GetAwaiter(), task.Result etc.



* 1. Example 6 – Non Task Main method Example. Check “AdvanceAsyncAwaitWebApp\_Pitfalls” -> “WhereDoesTaskStart” example in the github repo link & also you can check “Raw Coding Part 2” YouTube link. Time Stamp : 15:20
  2. Example 7 – Using ConfigureAwait(true) & ConfigureAwait(false) - UI thread.



1. How to create async await bubble – it will be used if the method is not using async await keyword or we cannot change the source code
   1. Normal Async Await(Task.Run) & also Parallel Async Await(Task.WhenAll)



Reference:

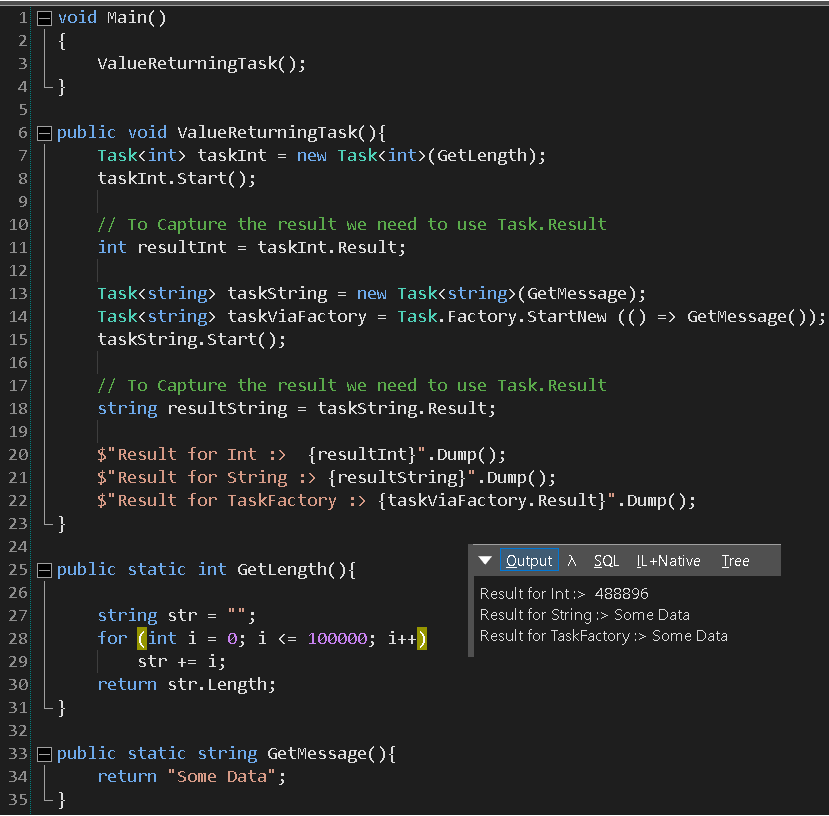
* <https://mykkon.work/async-state-machine/>
* <https://www.youtube.com/watch?v=il9gl8MH17s&t=330s> Raw Coding Part 1 – How & Why Part
* <https://www.youtube.com/watch?v=3GhKdDCvtKE> Raw Coding Part 2 – Pit Falls
* Tim Corey Async & Await video
* <https://ranjeet.dev/Getting-Started-With-Asynchronous-programming/>

**Task Parallel Library**

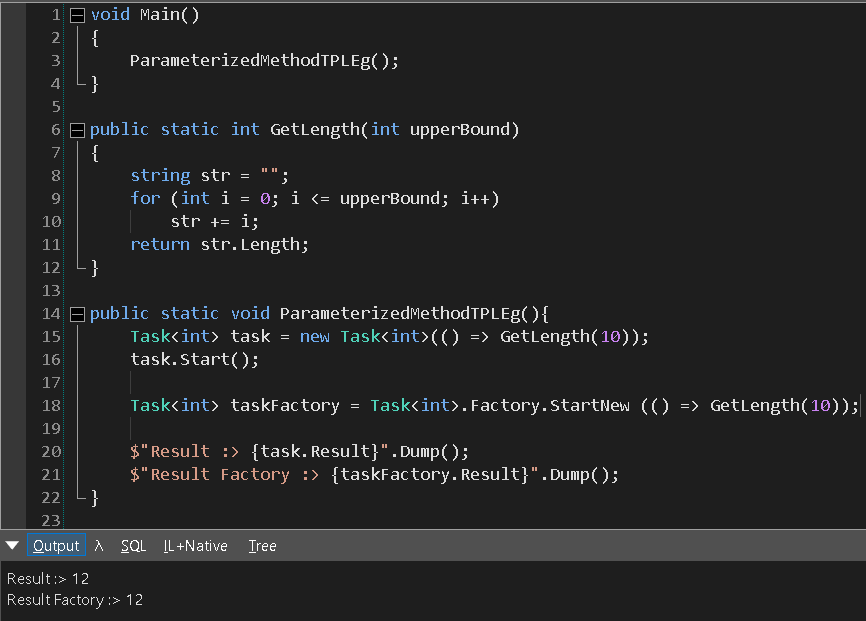
1. Tasks are built over the thread with better control. Task are superset of Thread.



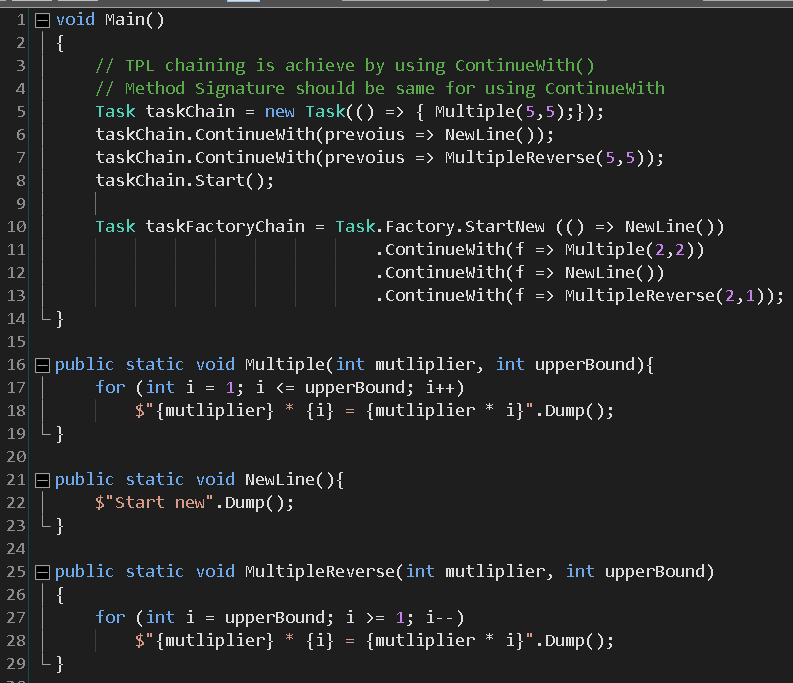
1. Task.Wait() and Thread.Join() have similar feature of waiting until all the child thread/task have been finished execution and then exists.
2. **Task** uses Generic delegates such as Func<T, out TResult> and Action<T> extensively.
3. Calling value returning method with Task.



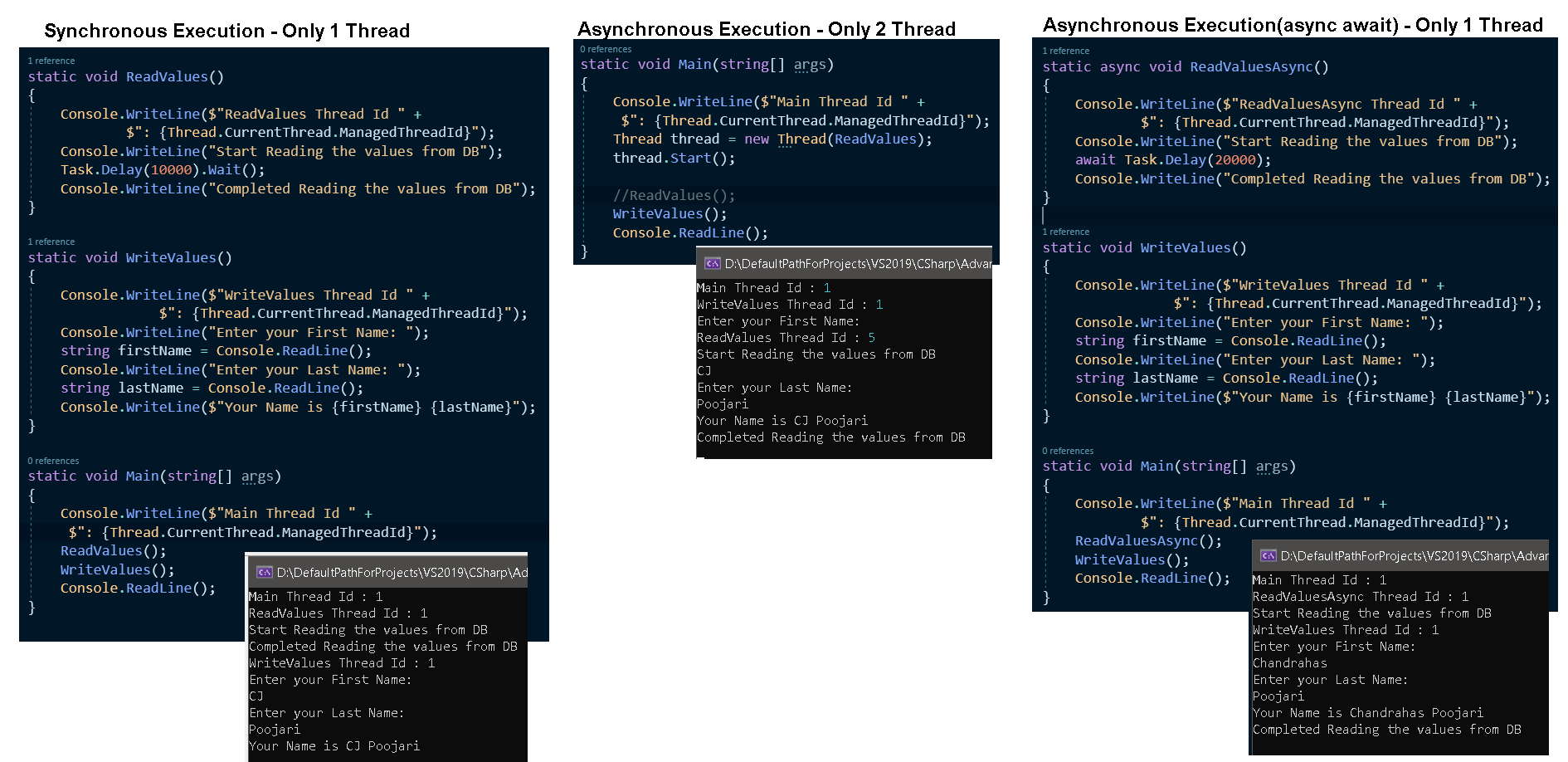
1. Calling Parameterized method with Task.



1. Method Chaining in TPL via ContinueWith()
   1. Method Signature should be same for using the method in ContinueWith



1. Parallelism – without creating new threads, we can achieve asynchronous execution (Parallelism) by using async await keywords.



C# Repo:

* [Advance C# Repo](https://github.com/chandrahasJ/Chandu816_2021_2022_V2/tree/main/Interview2.0/CSharpRepo/AdvanceCSharp)

Reference:

* [Free Workshop TPL by BangaRaju](https://www.youtube.com/watch?v=rkOuP1QlpqM&t=1688s)

**Dot Net Framework**

Coming Soon…