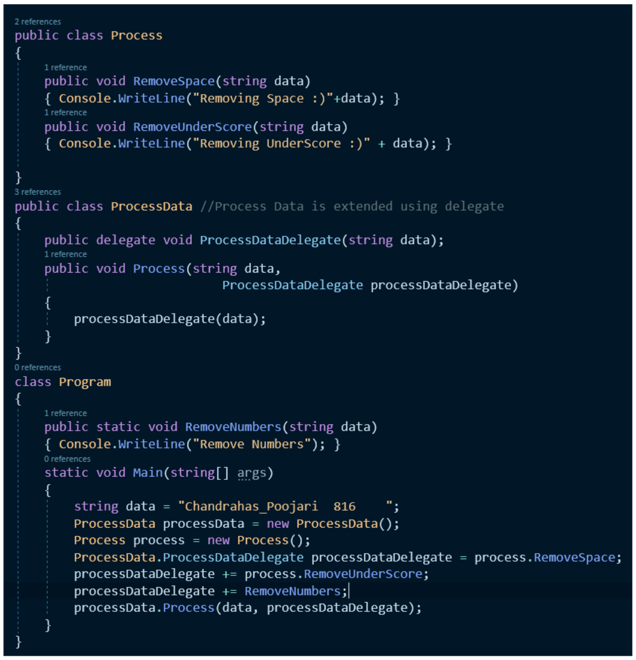
Advance C#, Dot Net Framework

Delegates

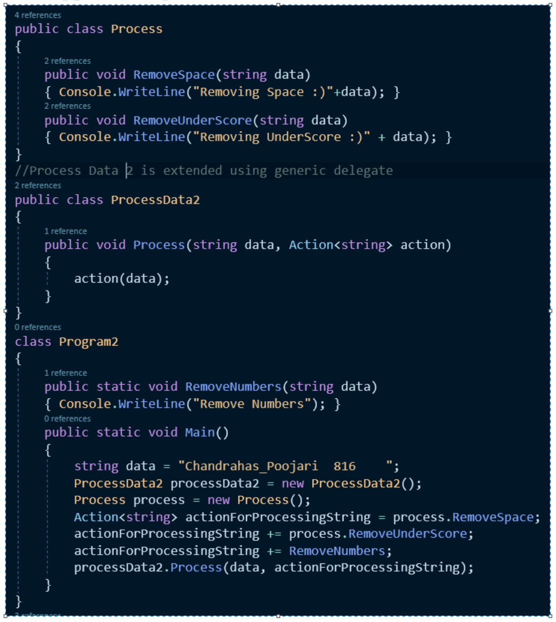
1) An object that knows how to call a method (or group of methods).
2) A reference to a function.
3) Generic Delegates – Action<T>, Func<T, out result> & Predicate<T>
4) Why do we need a delegates?
    a. For designing extensible and flexible application e.g. frameworks etc.
5) Interfaces or Delegates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

Using Delegate, we have extended ProcessData.



```
2 references
public class Process
{
    1 reference
    public void RemoveSpace(string data)
    { Console.WriteLine("Removing Space :)"+data); }
    1 reference
    public void RemoveUnderScore(string data)
    { Console.WriteLine("Removing UnderScore :)" + data); }

}
3 references
public class ProcessData //Process Data is extended using delegate
{
    public delegate void ProcessDataDelegate(string data);
    1 reference
    public void Process(string data,
                        ProcessDataDelegate processDataDelegate)
    {
        processDataDelegate(data);
    }
}
0 references
class Program
{
    1 reference
    public static void RemoveNumbers(string data)
    { Console.WriteLine("Remove Numbers"); }
    0 references
    static void Main(string[] args)
    {
        string data = "Chandrahas_Poojari  816    ";
        ProcessData processData = new ProcessData();
        Process process = new Process();
        ProcessData.ProcessDataDelegate processDataDelegate = process.RemoveSpace;
        processDataDelegate += process.RemoveUnderScore;
        processDataDelegate += RemoveNumbers;
        processData.Process(data, processDataDelegate);
    }
}
```

using generic delegate we have extended ProcessData2

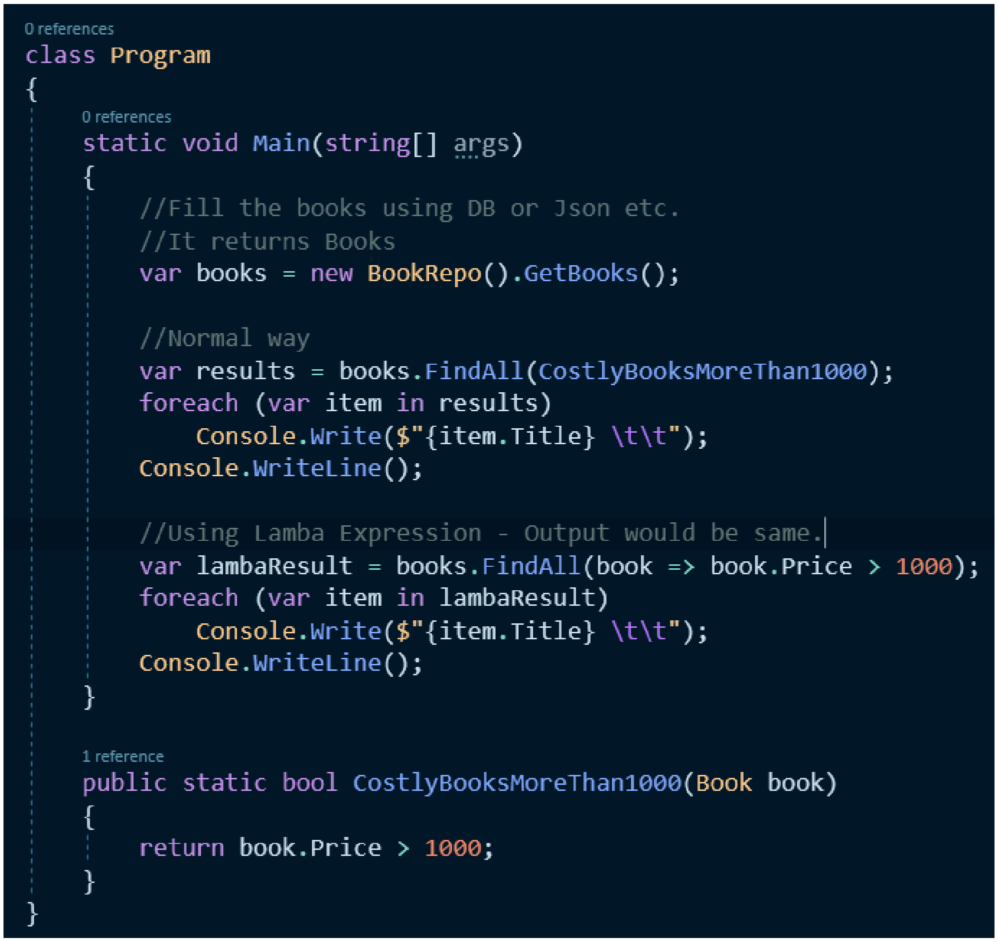


```
4 references
public class Process
{
    2 references
    public void RemoveSpace(string data)
    { Console.WriteLine("Removing Space :)"+data); }
    2 references
    public void RemoveUnderScore(string data)
    { Console.WriteLine("Removing UnderScore :)" + data); }
}
//Process Data 2 is extended using generic delegate
2 references
public class ProcessData2
{
    1 reference
    public void Process(string data, Action<string> action)
    {
        action(data);
    }
}
0 references
class Program2
{
    1 reference
    public static void RemoveNumbers(string data)
    { Console.WriteLine("Remove Numbers"); }
    0 references
    public static void Main()
    {
        string data = "Chandrahas_Poojari  816    ";
        ProcessData2 processData2 = new ProcessData2();
        Process process = new Process();
        Action<string> actionForProcessingString = process.RemoveSpace;
        actionForProcessingString += process.RemoveUnderScore;
        actionForProcessingString += RemoveNumbers;
        processData2.Process(data, actionForProcessingString);
    }
}
```

7) For normal implementation check the github repo – "AdvanceGenericExample" example.

Lamba Expressions

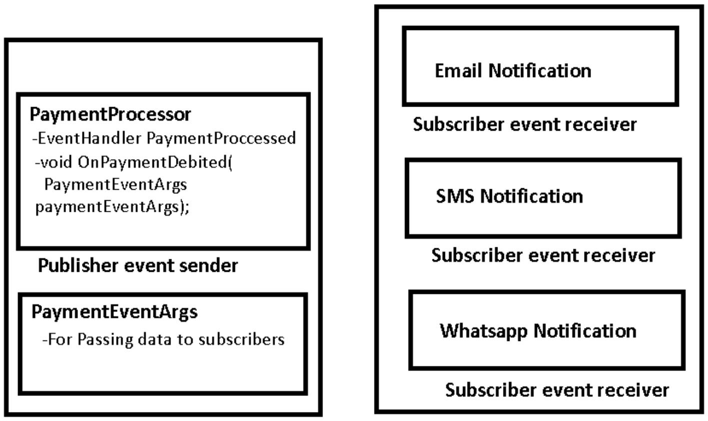
1) Lamba Expressions are nothing but anonymous methods that doesn't have name, access modifier.
2) Why do we use Lamba Expressions
    a. For convenience
3) Example –



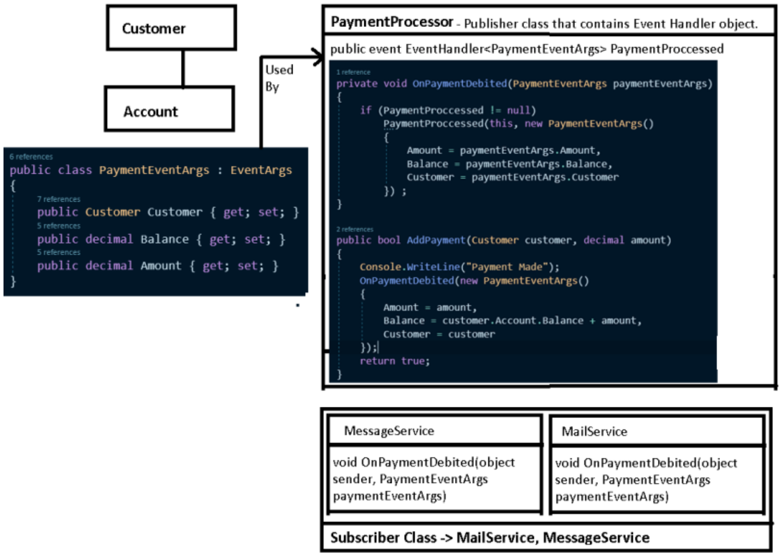
```
0 references
class Program
{
    0 references
    static void Main(string[] args)
    {
        //Fill the books using DB or Json etc.
        //It returns Books
        var books = new BookRepo().GetBooks();

        //Normal way
        var results = books.FindAll(CostlyBooksMoreThan1000);
        foreach (var item in results)
            Console.Write($"{item.Title} \t\t");
        Console.WriteLine();

        //Using Lamba Expression - Output would be same.
        var lambaResult = books.FindAll(book => book.Price > 1000);
        foreach (var item in lambaResult)
            Console.Write($"{item.Title} \t\t");
        Console.WriteLine();
    }

    1 reference
    public static bool CostlyBooksMoreThan1000(Book book)
    {
        return book.Price > 1000;
    }
}
```

Events

1) Events
    a. A mechanism for communication between objects. (notify other objects regarding an event happened(past tense) or happening.(present tense))
    b. Used in building Loosely coupled Applications
    c. 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
    a. Payment Processor class doesn't need to know about Email, SMS & Whatsapp Notifications and vice-verse
    b. Payment Processor only need to provide event handler object that can be used by the subscriber's classes to subscribe to this event.
    c. If event handler is not subscribed than it will not notify the notification class regarding the event.

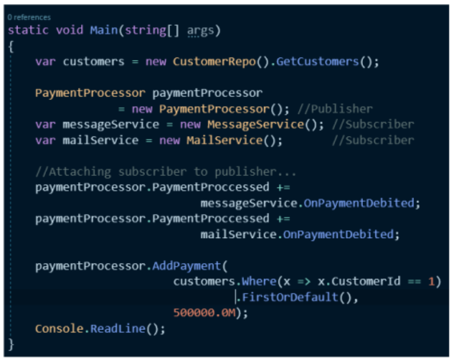


3) Example

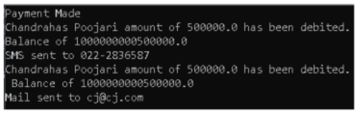

Customer
Account
Used By
PaymentProcessor - Publisher class that contains Event Handler object.
public event EventHandler<PaymentEventArgs> PaymentProccessed
MessageService
void OnPaymentDebited(object sender, PaymentEventArgs paymentEventArgs)
MailService
void OnPaymentDebited(object sender, PaymentEventArgs paymentEventArgs)
Subscriber Class -> MailService, MessageService

```
6 references
public class PaymentEventArgs : EventArgs
{
    7 references
    public Customer Customer { get; set; }
    5 references
    public decimal Balance { get; set; }
    5 references
    public decimal Amount { get; set; }
}
```

```
1 reference
private void OnPaymentDebited(PaymentEventArgs paymentEventArgs)
{
    if (PaymentProccessed != null)
        PaymentProccessed(this, new PaymentEventArgs()
        {
            Amount = paymentEventArgs.Amount,
            Balance = paymentEventArgs.Balance,
            Customer = paymentEventArgs.Customer
        }) ;
}

2 references
public bool AddPayment(Customer customer, decimal amount)
{
    Console.WriteLine("Payment Made");
    OnPaymentDebited(new PaymentEventArgs()
    {
        Amount = amount,
        Balance = customer.Account.Balance + amount,
        Customer = customer
    });
    return true;
}
```



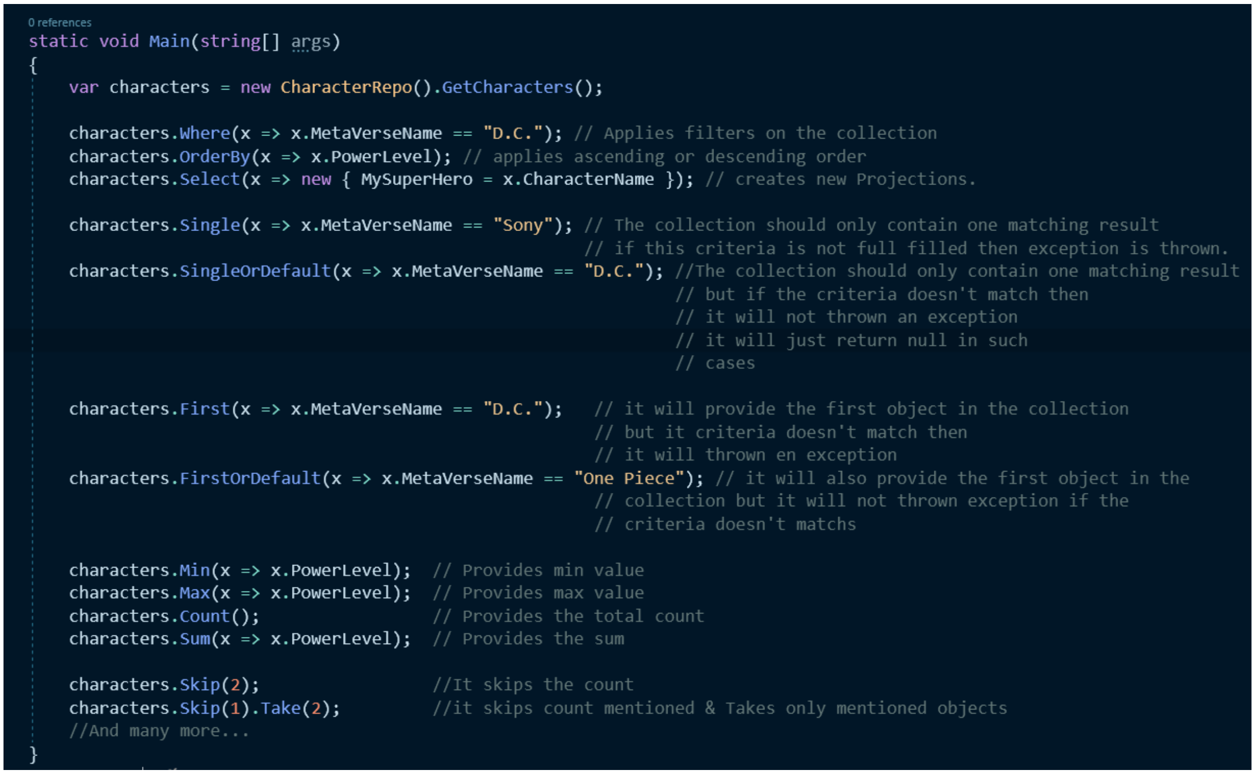
```
0 references
static void Main(string[] args)
{
    var customers = new CustomerRepo().GetCustomers();

    PaymentProcessor paymentProcessor
                = new PaymentProcessor(); //Publisher
    var messageService = new MessageService(); //Subscriber
    var mailService = new MailService();       //Subscriber

    //Attaching subscriber to publisher...
    paymentProcessor.PaymentProccessed +=
                        messageService.OnPaymentDebited;
    paymentProcessor.PaymentProccessed +=
                        mailService.OnPaymentDebited;

    paymentProcessor.AddPayment(
                    customers.Where(x => x.CustomerId == 1)
                            .FirstOrDefault(),
                    500000.0M);
    Console.ReadLine();
}
```



```
Payment Made
Chandrahas Poojari amount of 500000.0 has been debited.
Balance of 1000000000500000.0
SMS sent to 022-2836587
Chandrahas Poojari amount of 500000.0 has been debited.
 Balance of 1000000000500000.0
Mail sent to cj@cj.com
```

Linq

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



```
0 references
static void Main(string[] args)
{
    var characters = new CharacterRepo().GetCharacters();

    characters.Where(x => x.MetaVerseName == "D.C."); // Applies filters on the collection
    characters.OrderBy(x => x.PowerLevel); // applies ascending or descending order
    characters.Select(x => new { MySuperHero = x.CharacterName }); // creates new Projections.

    characters.Single(x => x.MetaVerseName == "Sony"); // The collection should only contain one matching result
                                                       // if this criteria is not full filled then exception is thrown.
    characters.SingleOrDefault(x => x.MetaVerseName == "D.C."); //The collection should only contain one matching result
                                                                // but if the criteria doesn't match then
                                                                // it will not thrown an exception
                                                                // it will just return null in such
                                                                // cases

    characters.First(x => x.MetaVerseName == "D.C.");   // it will provide the first object in the collection
                                                        // but it criteria doesn't match then
                                                        // it will thrown en exception
    characters.FirstOrDefault(x => x.MetaVerseName == "One Piece"); // it will also provide the first object in the
                                                        // collection but it will not thrown exception if the
                                                        // criteria doesn't matchs

    characters.Min(x => x.PowerLevel);  // Provides min value
    characters.Max(x => x.PowerLevel);  // Provides max value
    characters.Count();                 // Provides the total count
    characters.Sum(x => x.PowerLevel);  // Provides the sum

    characters.Skip(2);                 //It skips the count
    characters.Skip(1).Take(2);         //it skips count mentioned & Takes only mentioned objects
    //And many more...
}
```

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

1) Synchronous Vs Asynchronous

   a. 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.

   b. 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.

2) Benefits of Asynchronous mannerism

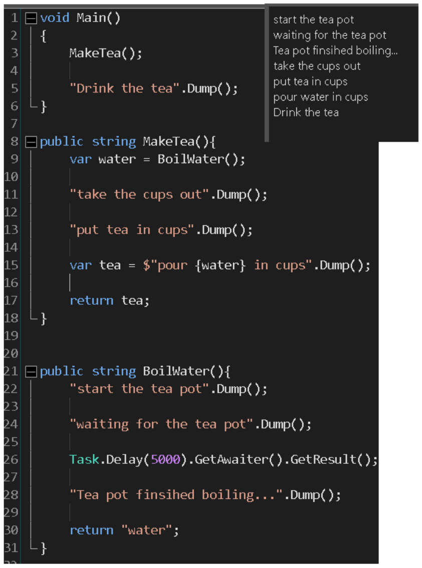
   a. Performance – We were able to do the task in less amount of time. i.e. we minimized the waiting time in ticket counter.

   b. Scalability – Railway was able to cater more people in less amount of time.

3) So Microsoft has provided us with two keywords **async** & **await**

4) A simple code example – Making a Tea & Drink a Tea in Sync & Async manner.

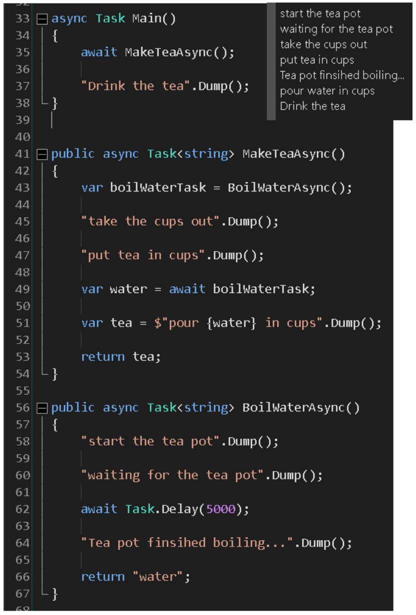
Synchronous execution



```
void Main()
{
    MakeTea();

    "Drink the tea".Dump();
}

public string MakeTea(){
    var water = BoilWater();

    "take the cups out".Dump();

    "put tea in cups".Dump();

    var tea = $"pour {water} in cups".Dump();

    return tea;
}


public string BoilWater(){
    "start the tea pot".Dump();

    "waiting for the tea pot".Dump();

    Task.Delay(5000).GetAwaiter().GetResult();

    "Tea pot finsihed boiling...".Dump();

    return "water";
}
```

```
start the tea pot
waiting for the tea pot
Tea pot finsihed boiling...
take the cups out
put tea in cups
pour water in cups
Drink the tea
```

Asynchronous Exection

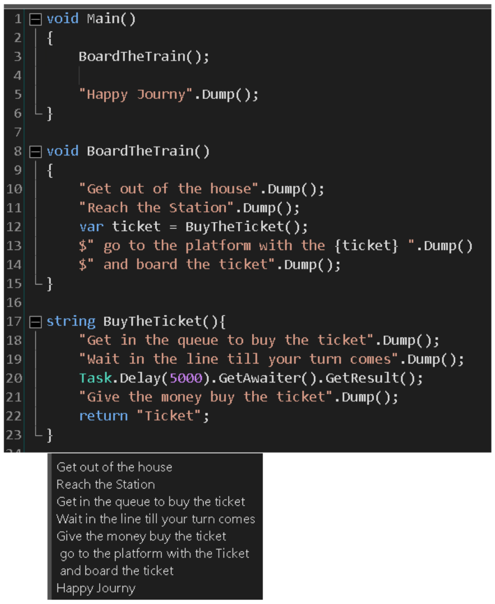


```
async Task Main()
{
    await MakeTeaAsync();

    "Drink the tea".Dump();
}


public async Task<string> MakeTeaAsync()
{
    var boilWaterTask = BoilWaterAsync();

    "take the cups out".Dump();

    "put tea in cups".Dump();

    var water = await boilWaterTask;

    var tea = $"pour {water} in cups".Dump();

    return tea;
}

public async Task<string> BoilWaterAsync()
{
    "start the tea pot".Dump();

    "waiting for the tea pot".Dump();

    await Task.Delay(5000);

    "Tea pot finsihed boiling...".Dump();

    return "water";
}
```

```
start the tea pot
waiting for the tea pot
take the cups out
put tea in cups
Tea pot finsihed boiling...
pour water in cups
Drink the tea
```

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

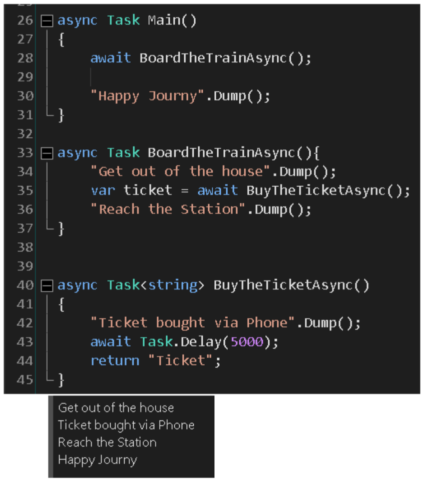
Synchronous execution



```
void Main()
{
    BoardTheTrain();

    "Happy Journy".Dump();
}

void BoardTheTrain()
{
    "Get out of the house".Dump();
    "Reach the Station".Dump();
    var ticket = BuyTheTicket();
    $" go to the platform with the {ticket} ".Dump()
    $" and board the ticket".Dump();
}

string BuyTheTicket(){
    "Get in the queue to buy the ticket".Dump();
    "Wait in the line till your turn comes".Dump();
    Task.Delay(5000).GetAwaiter().GetResult();
    "Give the money buy the ticket".Dump();
    return "Ticket";
}
```

```
Get out of the house
Reach the Station
Get in the queue to buy the ticket
Wait in the line till your turn comes
Give the money buy the ticket
 go to the platform with the Ticket
 and board the ticket
Happy Journy
```

Asynchronous Exection



```
async Task Main()
{
    await BoardTheTrainAsync();

    "Happy Journy".Dump();
}

async Task BoardTheTrainAsync(){
    "Get out of the house".Dump();
    var ticket = await BuyTheTicketAsync();
    "Reach the Station".Dump();
}


async Task<string> BuyTheTicketAsync()
{
    "Ticket bought via Phone".Dump();
    await Task.Delay(5000);
    return "Ticket";
}
```

```
Get out of the house
Ticket bought via Phone
Reach the Station
Happy Journy
```

6) Asynchronous programming is not same as creating multiple threads to perform the work in parallel.

7) 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.

8) Two types of tasks that block the execution thread(Main Thread)

   a. IO Bound Tasks

      1) App waiting for database query to return call.

      2) App waiting for response of an http call.

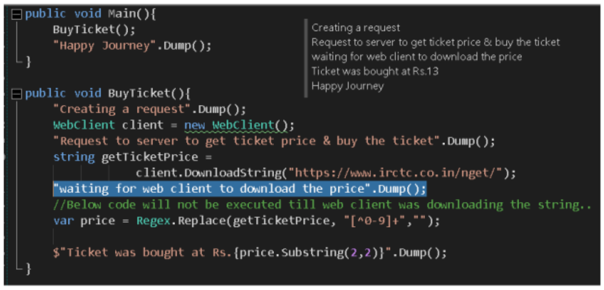
      3) App waiting for azure storage SDK to return data.

   b. CPU bound Tasks

      1) App is waiting for some complex computation to complete.

9) A simple example of IO Bound Task- it looks the same but it is not…

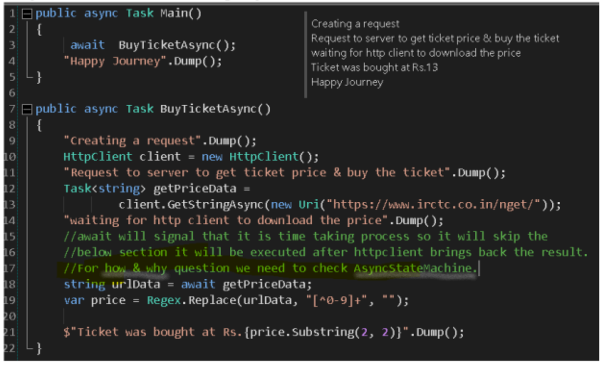
Synchronous manner



```
public void Main(){
    BuyTicket();
    "Happy Journey".Dump();
}

public void BuyTicket(){
    "Creating a request".Dump();
    WebClient client = new WebClient();
    "Request to server to get ticket price & buy the ticket".Dump();
    string getTicketPrice =
                client.DownloadString("https://www.irctc.co.in/nget/");
    "waiting for web client to download the price".Dump();
    //Below code will not be executed till web client was downloading the string..
    var price = Regex.Replace(getTicketPrice, "[^0-9]+","");

    $"Ticket was bought at Rs.{price.Substring(2,2)}".Dump();
}
```

```
Creating a request
Request to server to get ticket price & buy the ticket
waiting for web client to download the price
Ticket was bought at Rs.13
Happy Journey
```

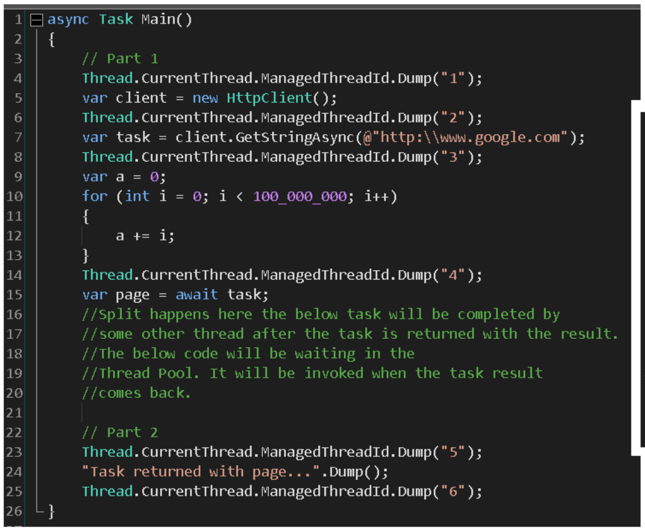
Asychronous manner - using async/await



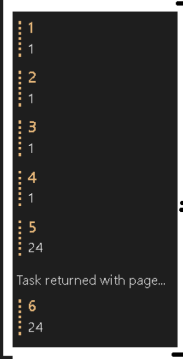
```
public async Task Main()
{
     await  BuyTicketAsync();
    "Happy Journey".Dump();
}

public async Task BuyTicketAsync()
{
    "Creating a request".Dump();
    HttpClient client = new HttpClient();
    "Request to server to get ticket price & buy the ticket".Dump();
    Task<string> getPriceData =
            client.GetStringAsync(new Uri("https://www.irctc.co.in/nget/"));
    "waiting for http client to download the price".Dump();
    //await will signal that it is time taking process so it will skip the
    //below section it will be executed after httpclient brings back the result.
    //For how & why question we need to check AsyncStateMachine.
    string urlData = await getPriceData;
    var price = Regex.Replace(urlData, "[^0-9]+", "");

    $"Ticket was bought at Rs.{price.Substring(2, 2)}".Dump();
}
```

```
Creating a request
Request to server to get ticket price & buy the ticket
waiting for http client to download the price
Ticket was bought at Rs.13
Happy Journey
```

10) 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.



```
async Task Main()
{
    // Part 1
    Thread.CurrentThread.ManagedThreadId.Dump("1");
    var client = new HttpClient();
    Thread.CurrentThread.ManagedThreadId.Dump("2");
    var task = client.GetStringAsync(@"http:\\www.google.com");
    Thread.CurrentThread.ManagedThreadId.Dump("3");
    var a = 0;
    for (int i = 0; i < 100_000_000; i++)
    {
        a += i;
    }
    Thread.CurrentThread.ManagedThreadId.Dump("4");
    var page = await task;
    //Split happens here the below task will be completed by
    //some other thread after the task is returned with the result.
    //The below code will be waiting in the
    //Thread Pool. It will be invoked when the task result
    //comes back.

    // Part 2
    Thread.CurrentThread.ManagedThreadId.Dump("5");
    "Task returned with page...".Dump();
    Thread.CurrentThread.ManagedThreadId.Dump("6");
}
```



```
1
1
2
1
3
1
4
1
5
24
Task returned with page...
6
24
```

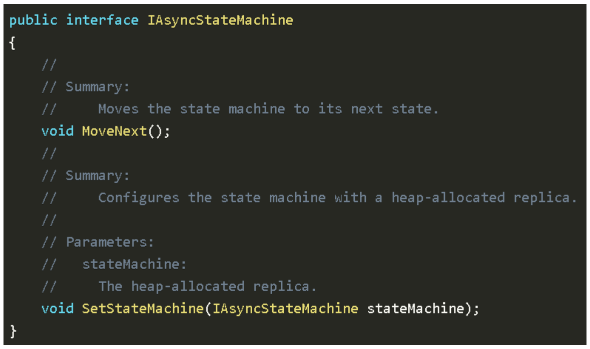
Current Thread Id is 1 for Step 1, 2, 3 & 4.
In Step 2, a async network call is done which will take some time to complete
Step 3, The thread "1" is performing in the some work.
Step4, Compiler see await keyword. now below tasks is pushed in thread pool waiting for the task to be completed.

Step 5, Task returns with page i.e. it is completed it checks the thread pool for the task. it invokes the task and execution is completed.
Note : Since Current thread id "1" is now performing some other task the task will be assigned to some other thread.
In our case thread id is "24"

11) Now let's have a sneak peek on Async State Machine which helped us to achieve the code split

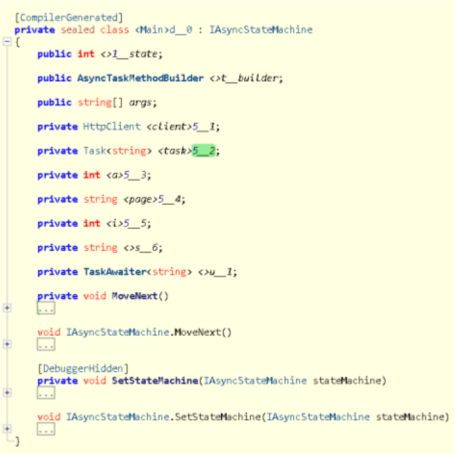
a. AsyncStateMachine stays in the RAM so that it is not collected by garbage collector.

b. IAsyncStateMachine is interface which contains two methods MoveNext() & SetStateMachine(IAsyncStateMachine)



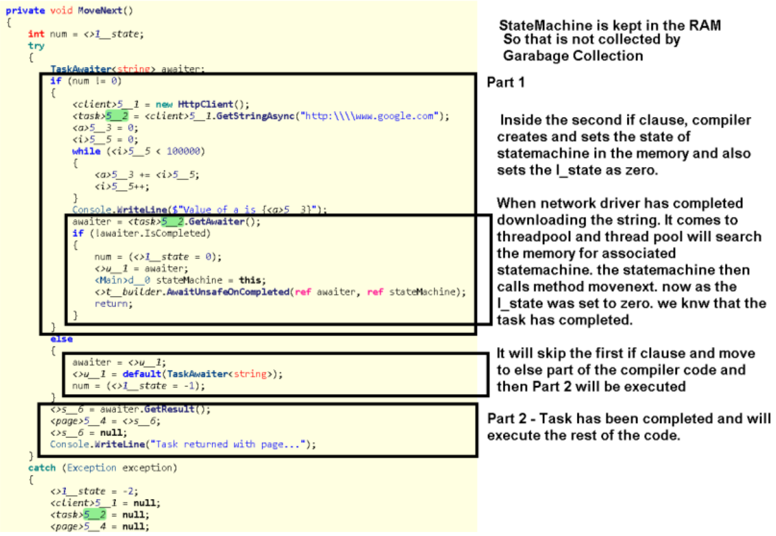
```
public interface IAsyncStateMachine
{
    //
    // Summary:
    //     Moves the state machine to its next state.
    void MoveNext();
    //
    // Summary:
    //     Configures the state machine with a heap-allocated replica.
    //
    // Parameters:
    //   stateMachine:
    //     The heap-allocated replica.
    void SetStateMachine(IAsyncStateMachine stateMachine);
}
```

c. Compiler generated code for async/await



```
[CompilerGenerated]
private sealed class <Main>d__0 : IAsyncStateMachine
{
    public int <>1__state;

    public AsyncTaskMethodBuilder <>t__builder;

    public string[] args;

    private HttpClient <client>5__1;

    private Task<string> <task>5__2;

    private int <a>5__3;

    private string <page>5__4;

    private int <i>5__5;

    private string <>s__6;

    private TaskAwaiter<string> <>u__1;

    private void MoveNext()
    ...

    void IAsyncStateMachine.MoveNext()
    ...

    [DebuggerHidden]
    private void SetStateMachine(IAsyncStateMachine stateMachine)
    ...

    void IAsyncStateMachine.SetStateMachine(IAsyncStateMachine stateMachine)
    ...
}
```

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



```
private void MoveNext()
{
    int num = <>1__state;
    try
    {
        TaskAwaiter<string> awaiter;
        if (num != 0)
        {
            <client>5__1 = new HttpClient();
            <task>5__2 = <client>5__1.GetStringAsync("http:\\\\www.google.com");
            <a>5__3 = 0;
            <i>5__5 = 0;
            while (<i>5__5 < 100000)
            {
                <a>5__3 += <i>5__5;
                <i>5__5++;
            }
            Console.WriteLine($"Value of a is {<a>5__3}");
            awaiter = <task>5__2.GetAwaiter();
            if (!awaiter.IsCompleted)
            {
                num = (<>1__state = 0);
                <>u__1 = awaiter;
                <Main>d__0 stateMachine = this;
                <>t__builder.AwaitUnsafeOnCompleted(ref awaiter, ref stateMachine);
                return;
            }
        }
        else
        {
            awaiter = <>u__1;
            <>u__1 = default(TaskAwaiter<string>);
            num = (<>1__state = -1);
        }
        <>s__6 = awaiter.GetResult();
        <page>5__4 = <>s__6;
        <>s__6 = null;
        Console.WriteLine("Task returned with page...");
    }
    catch (Exception exception)
    {
        <>1__state = -2;
        <client>5__1 = null;
        <task>5__2 = null;
        <page>5__4 = null;
```

StateMachine is kept in the RAM
So that is not collected by
Garabage Collection

Part 1

Inside the second if clause, compiler creates and sets the state of statemachine in the memory and also sets the l_state as zero.

When network driver has completed downloading the string. It comes to threadpool and thread pool will search the memory for associated statemachine. the statemachine then calls method movenext. now as the l_state was set to zero. we knw that the task has completed.

It will skip the first if clause and move to else part of the compiler code and then Part 2 will be executed

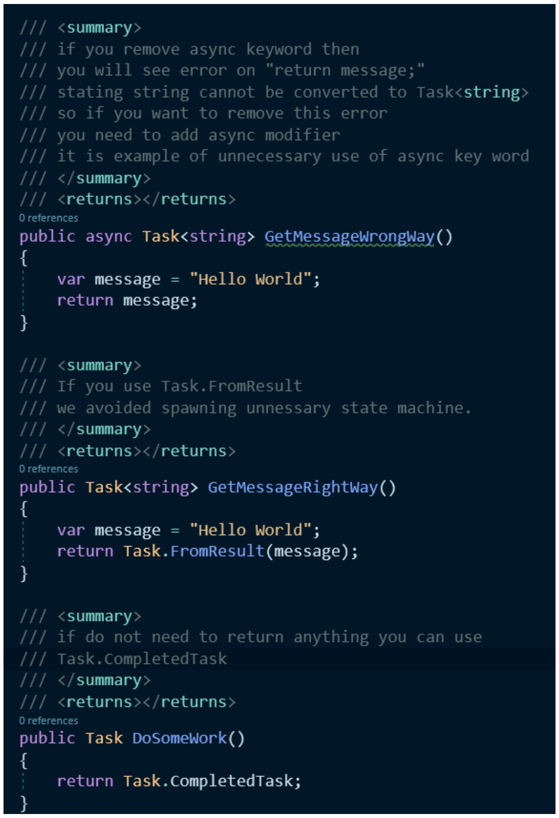
Part 2 - Task has been completed and will execute the rest of the code.

e. 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.

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

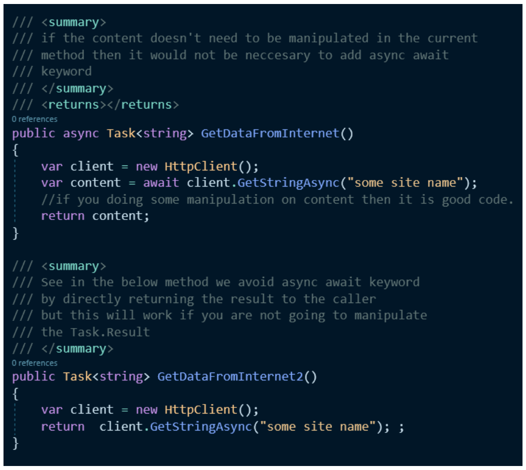
Asynchronous Programming – Pitfalls - Async and Await.

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



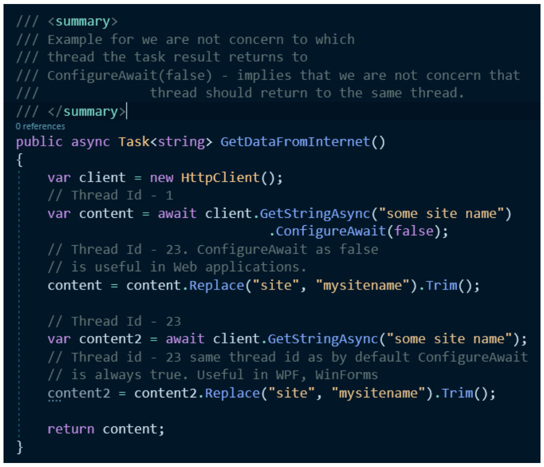
```
/// <summary>
/// if you remove async keyword then
/// you will see error on "return message;"
/// stating string cannot be converted to Task<string>
/// so if you want to remove this error
/// you need to add async modifier
/// it is example of unnecessary use of async key word
/// </summary>
/// <returns></returns>
0 references
public async Task<string> GetMessageWrongWay()
{
    var message = "Hello World";
    return message;
}

/// <summary>
/// If you use Task.FromResult
/// we avoided spawning unnessary state machine.
/// </summary>
/// <returns></returns>
0 references
public Task<string> GetMessageRightWay()
{
    var message = "Hello World";
    return Task.FromResult(message);
}

/// <summary>
/// if do not need to return anything you can use
/// Task.CompletedTask
/// </summary>
/// <returns></returns>
0 references
public Task DoSomeWork()
{
    return Task.CompletedTask;
}
```

    b. Example 2 – Return the async directly to the caller if result doesn't need to be manipulated.



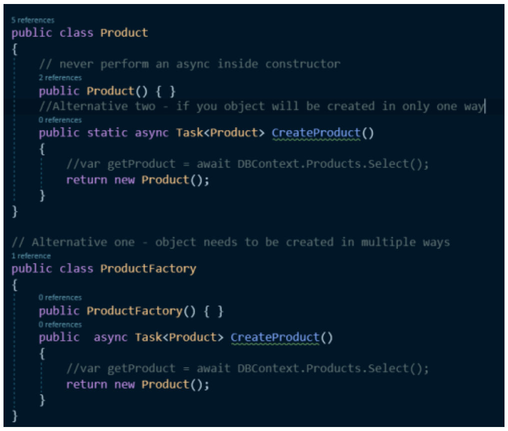
```
/// <summary>
/// if the content doesn't need to be manipulated in the current
/// method then it would not be neccesary to add async await
/// keyword
/// </summary>
/// <returns></returns>
0 references
public async Task<string> GetDataFromInternet()
{
    var client = new HttpClient();
    var content = await client.GetStringAsync("some site name");
    //if you doing some manipulation on content then it is good code.
    return content;
}

/// <summary>
/// See in the below method we avoid async await keyword
/// by directly returning the result to the caller
/// but this will work if you are not going to manipulate
/// the Task.Result
/// </summary>
0 references
public Task<string> GetDataFromInternet2()
{
    var client = new HttpClient();
    return  client.GetStringAsync("some site name"); ;
}
```

c. 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.



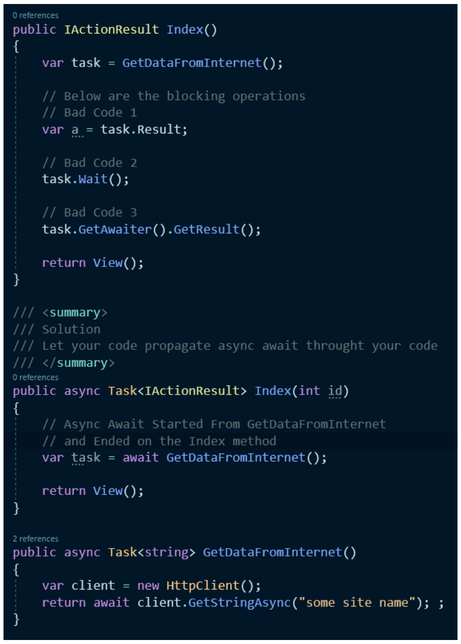
```
/// <summary>
/// Example for we are not concern to which
/// thread the task result returns to
/// ConfigureAwait(false) - implies that we are not concern that
///              thread should return to the same thread.
/// </summary>
0 references
public async Task<string> GetDataFromInternet()
{
    var client = new HttpClient();
    // Thread Id - 1
    var content = await client.GetStringAsync("some site name")
                            .ConfigureAwait(false);
    // Thread Id - 23. ConfigureAwait as false
    // is useful in Web applications.
    content = content.Replace("site", "mysitename").Trim();

    // Thread Id - 23
    var content2 = await client.GetStringAsync("some site name");
    // Thread id - 23 same thread id as by default ConfigureAwait
    // is always true. Useful in WPF, WinForms
    content2 = content2.Replace("site", "mysitename").Trim();

    return content;
}
```

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



```
5 references
public class Product
{
    // never perform an async inside constructor
    2 references
    public Product() { }
    //Alternative two - if you object will be created in only one way
    0 references
    public static async Task<Product> CreateProduct()
    {
        //var getProduct = await DBContext.Products.Select();
        return new Product();
    }
}

// Alternative one - object needs to be created in multiple ways
1 reference
public class ProductFactory
{
    0 references
    public ProductFactory() { }
    0 references
    public  async Task<Product> CreateProduct()
    {
        //var getProduct = await DBContext.Products.Select();
        return new Product();
    }
}
```

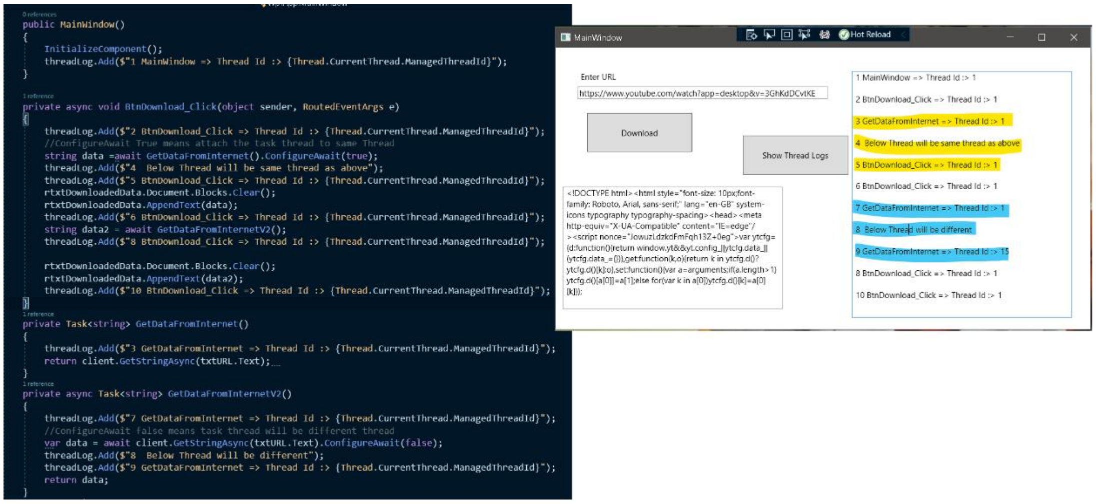
e. Example 5 – Blocking the Thread by using task.GetAwaiter(), task.Result etc.



```
0 references
public IActionResult Index()
{
    var task = GetDataFromInternet();

    // Below are the blocking operations
    // Bad Code 1
    var a = task.Result;

    // Bad Code 2
    task.Wait();

    // Bad Code 3
    task.GetAwaiter().GetResult();

    return View();
}

/// <summary>
/// Solution
/// Let your code propagate async await throught your code
/// </summary>
0 references
public async Task<IActionResult> Index(int id)
{
    // Async Await Started From GetDataFromInternet
    // and Ended on the Index method
    var task = await GetDataFromInternet();

    return View();
}

2 references
public async Task<string> GetDataFromInternet()
{
    var client = new HttpClient();
    return await client.GetStringAsync("some site name"); ;
}
```

f. 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

g. Example 7 – Using ConfigureAwait(true) & ConfigureAwait(false) - UI thread.



```
0 references
public MainWindow()
{
    InitializeComponent();
    threadLog.Add($"1 MainWindow => Thread Id :> {Thread.CurrentThread.ManagedThreadId}");
}

1 reference
private async void BtnDownload_Click(object sender, RoutedEventArgs e)
{
    threadLog.Add($"2 BtnDownload_Click => Thread Id :> {Thread.CurrentThread.ManagedThreadId}");
    //ConfigureAwait True means attach the task thread to same Thread
    string data =await GetDataFromInternet().ConfigureAwait(true);
    threadLog.Add($"4  Below Thread will be same thread as above");
    threadLog.Add($"5 BtnDownload_Click => Thread Id :> {Thread.CurrentThread.ManagedThreadId}");
    rtxtDownloadedData.Document.Blocks.Clear();
    rtxtDownloadedData.AppendText(data);
    threadLog.Add($"6 BtnDownload_Click => Thread Id :> {Thread.CurrentThread.ManagedThreadId}");
    string data2 = await GetDataFromInternetV2();
    threadLog.Add($"8 BtnDownload_Click => Thread Id :> {Thread.CurrentThread.ManagedThreadId}");

    rtxtDownloadedData.Document.Blocks.Clear();
    rtxtDownloadedData.AppendText(data2);
    threadLog.Add($"10 BtnDownload_Click -> Thread Id :> {Thread.CurrentThread.ManagedThreadId}");
}
1 reference
private Task<string> GetDataFromInternet()
{
    threadLog.Add($"3 GetDataFromInternet => Thread Id :> {Thread.CurrentThread.ManagedThreadId}");
    return client.GetStringAsync(txtURL.Text);
}
1 reference
private async Task<string> GetDataFromInternetV2()
{
    threadLog.Add($"7 GetDataFromInternet => Thread Id :> {Thread.CurrentThread.ManagedThreadId}");
    //ConfigureAwait false means task thread will be different thread
    var data = await client.GetStringAsync(txtURL.Text).ConfigureAwait(false);
    threadLog.Add($"8  Below Thread will be different");
    threadLog.Add($"9 GetDataFromInternet => Thread Id :> {Thread.CurrentThread.ManagedThreadId}");
    return data;
}
```

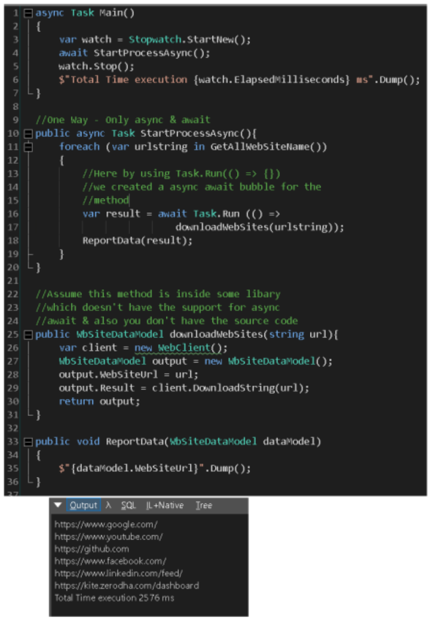
https://www.youtube.com/watch?app=desktop&v=3GhKdDCvtKE

```
<!DOCTYPE html><html style="font-size: 10px;font-family: Roboto, Arial, sans-serif;" lang="en-GB" system-icons typography typography-spacing><head><meta http-equiv="X-UA-Compatible" content="IE=edge"/><script nonce="JowuzLdzkdFmFqh13Z+0eg">var ytcfg={d:function(){return window.yt&&yt.config_||ytcfg.data_||(ytcfg.data_={})},get:function(k,o){return k in ytcfg.d()?ytcfg.d()[k]:o},set:function(){var a=arguments;if(a.length>1)ytcfg.d()[a[0]]=a[1];else for(var k in a[0])ytcfg.d()[k]=a[0][k]}};
```

```
1 MainWindow => Thread Id :> 1
2 BtnDownload_Click => Thread Id :> 1
3 GetDataFromInternet => Thread Id :> 1
4  Below Thread will be same thread as above
5 BtnDownload_Click => Thread Id :> 1
6 BtnDownload_Click => Thread Id :> 1
7 GetDataFromInternet => Thread Id :> 1
8  Below Thread will be different
9 GetDataFromInternet => Thread Id :> 15
8 BtnDownload_Click => Thread Id :> 1
10 BtnDownload_Click => Thread Id :> 1
```

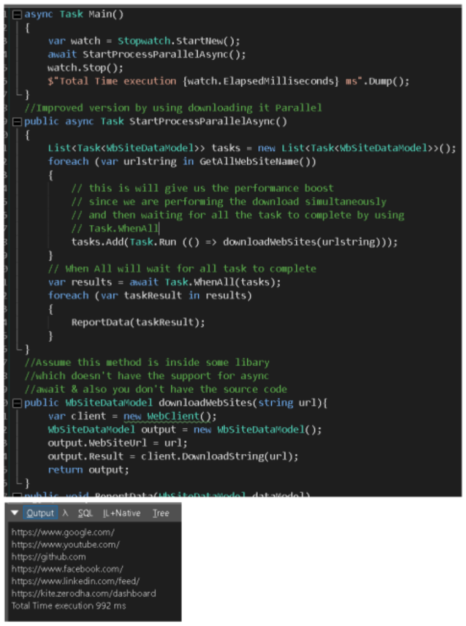
16) How to create async await bubble – it will be used if the method is not using async await keyword or we cannot change the source code

a. Normal Async Await(Task.Run) & also Parallel Async Await(Task.WhenAll)



```
async Task Main()
{
    var watch = Stopwatch.StartNew();
    await StartProcessAsync();
    watch.Stop();
    $"Total Time execution {watch.ElapsedMilliseconds} ms".Dump();
}

//One Way - Only async & await
public async Task StartProcessAsync(){
    foreach (var urlstring in GetAllWebSiteName())
    {
        //Here by using Task.Run(() => {})
        //we created a async await bubble for the
        //method
        var result = await Task.Run (() =>
                        downloadWebSites(urlstring));
        ReportData(result);
    }
}

//Assume this method is inside some libary
//which doesn't have the support for async
//await & also you don't have the source code
public WbSiteDataModel downloadWebSites(string url){
    var client = new WebClient();
    WbSiteDataModel output = new WbSiteDataModel();
    output.WebSiteUrl = url;
    output.Result = client.DownloadString(url);
    return output;
}

public void ReportData(WbSiteDataModel dataModel)
{
    $"{dataModel.WebSiteUrl}".Dump();
}
```

```
https://www.google.com/
https://www.youtube.com/
https://github.com
https://www.facebook.com/
https://www.linkedin.com/feed/
https://kite.zerodha.com/dashboard
Total Time execution 2576 ms
```



```
async Task Main()
{
    var watch = Stopwatch.StartNew();
    await StartProcessParallelAsync();
    watch.Stop();
    $"Total Time execution {watch.ElapsedMilliseconds} ms".Dump();
}
//Improved version by using downloading it Parallel
public async Task StartProcessParallelAsync()
{
    List<Task<WbSiteDataModel>> tasks = new List<Task<WbSiteDataModel>>();
    foreach (var urlstring in GetAllWebSiteName())
    {
        // this is will give us the performance boost
        // since we are performing the download simultaneously
        // and then waiting for all the task to complete by using
        // Task.WhenAll
        tasks.Add(Task.Run (() => downloadWebSites(urlstring)));
    }
    // When All will wait for all task to complete
    var results = await Task.WhenAll(tasks);
    foreach (var taskResult in results)
    {
        ReportData(taskResult);
    }
}
//Assume this method is inside some libary
//which doesn't have the support for async
//await & also you don't have the source code
public WbSiteDataModel downloadWebSites(string url){
    var client = new WebClient();
    WbSiteDataModel output = new WbSiteDataModel();
    output.WebSiteUrl = url;
    output.Result = client.DownloadString(url);
    return output;
}
public void ReportData(WbSiteDataModel dataModel)
```

```
https://www.google.com/
https://www.youtube.com/
https://github.com
https://www.facebook.com/
https://www.linkedin.com/feed/
https://kite.zerodha.com/dashboard
Total Time execution 992 ms
```

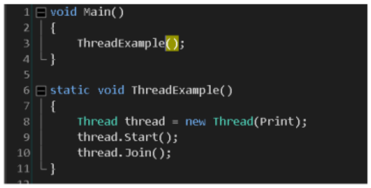
Reference:

- 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

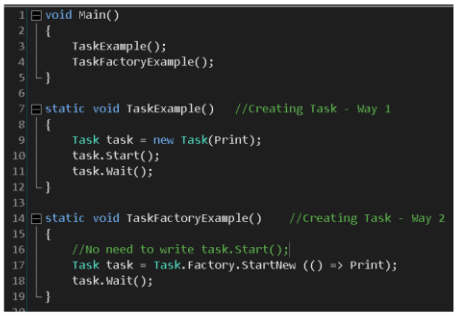
1. Tasks are built over the thread with better control. Task are superset of Thread.

Thread



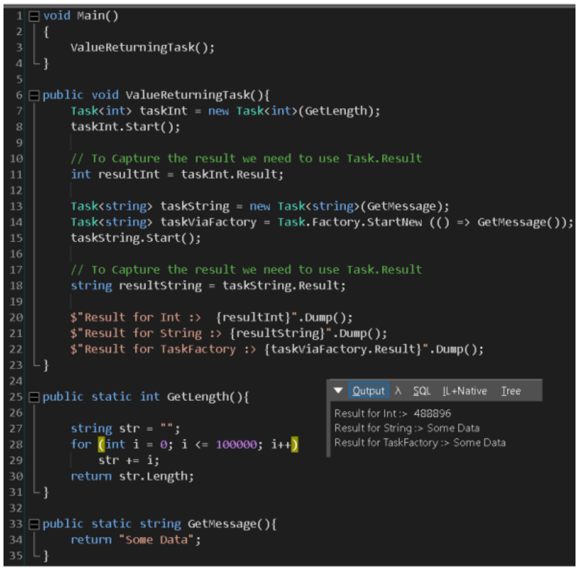
```
void Main()
{
    ThreadExample();
}

static void ThreadExample()
{
    Thread thread = new Thread(Print);
    thread.Start();
    thread.Join();
}
```

Task



```
void Main()
{
    TaskExample();
    TaskFactoryExample();
}

static void TaskExample()   //Creating Task - Way 1
{
    Task task = new Task(Print);
    task.Start();
    task.Wait();
}

static void TaskFactoryExample()    //Creating Task - Way 2
{
    //No need to write task.Start();
    Task task = Task.Factory.StartNew (() => Print);
    task.Wait();
}
```

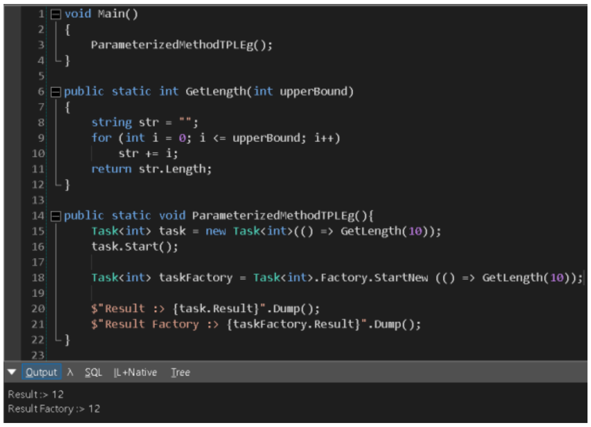
2. Task.Wait() and Thread.Join() have similar feature of waiting until all the child thread/task have been finished execution and then exists.
3. **Task** uses Generic delegates such as Func<T, out TResult> and Action<T> extensively.
4. Calling value returning method with Task.



```
void Main()
{
    ValueReturningTask();
}

public void ValueReturningTask(){
    Task<int> taskInt = new Task<int>(GetLength);
    taskInt.Start();

    // To Capture the result we need to use Task.Result
    int resultInt = taskInt.Result;

    Task<string> taskString = new Task<string>(GetMessage);
    Task<string> taskViaFactory = Task.Factory.StartNew (() => GetMessage());
    taskString.Start();

    // To Capture the result we need to use Task.Result
    string resultString = taskString.Result;

    $"Result for Int :>  {resultInt}".Dump();
    $"Result for String :> {resultString}".Dump();
    $"Result for TaskFactory :> {taskViaFactory.Result}".Dump();
}

public static int GetLength(){

    string str = "";
    for (int i = 0; i <= 100000; i++)
        str += i;
    return str.Length;
}

public static string GetMessage(){
    return "Some Data";
}
```

```
Result for Int :>  488896
Result for String :> Some Data
Result for TaskFactory :> Some Data
```

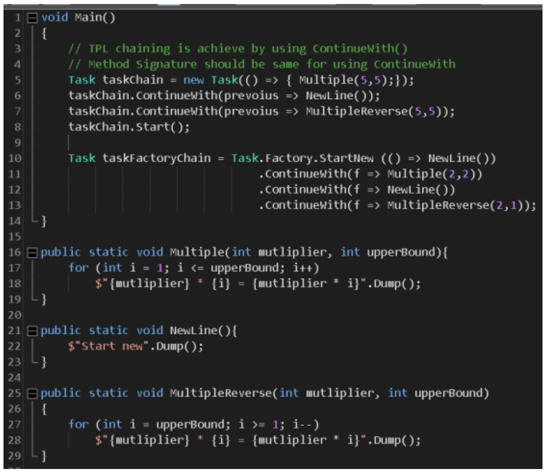
5. Calling Parameterized method with Task.



```
void Main()
{
    ParameterizedMethodTPLEg();
}

public static int GetLength(int upperBound)
{
    string str = "";
    for (int i = 0; i <= upperBound; i++)
        str += i;
    return str.Length;
}

public static void ParameterizedMethodTPLEg(){
    Task<int> task = new Task<int>(() => GetLength(10));
    task.Start();

    Task<int> taskFactory = Task<int>.Factory.StartNew (() => GetLength(10));

    $"Result :> {task.Result}".Dump();
    $"Result Factory :> {taskFactory.Result}".Dump();
}
```

```
Result :> 12
Result Factory :> 12
```

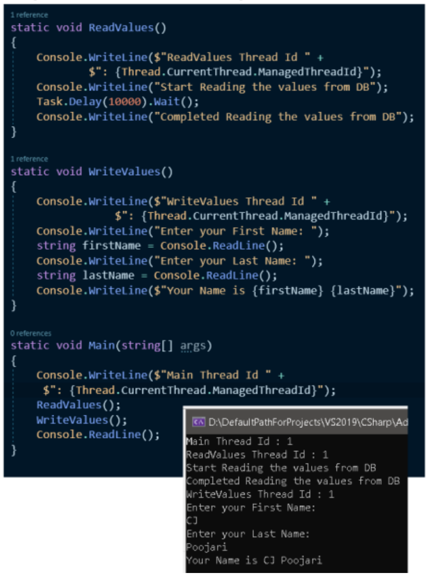
6. Method Chaining in TPL via ContinueWith()
    a. Method Signature should be same for using the method in ContinueWith



```
void Main()
{
    // TPL chaining is achieve by using ContinueWith()
    // Method Signature should be same for using ContinueWith
    Task taskChain = new Task(() => { Multiple(5,5);});
    taskChain.ContinueWith(prevoius => NewLine());
    taskChain.ContinueWith(prevoius => MultipleReverse(5,5));
    taskChain.Start();

    Task taskFactoryChain = Task.Factory.StartNew (() => NewLine())
                                       .ContinueWith(f => Multiple(2,2))
                                       .ContinueWith(f => NewLine())
                                       .ContinueWith(f => MultipleReverse(2,1));
}

public static void Multiple(int mutliplier, int upperBound){
    for (int i = 1; i <= upperBound; i++)
        $"{mutliplier} * {i} = {mutliplier * i}".Dump();
}

public static void NewLine(){
    $"Start new".Dump();
}

public static void MultipleReverse(int mutliplier, int upperBound)
{
    for (int i = upperBound; i >= 1; i--)
        $"{mutliplier} * {i} = {mutliplier * i}".Dump();
}
```

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

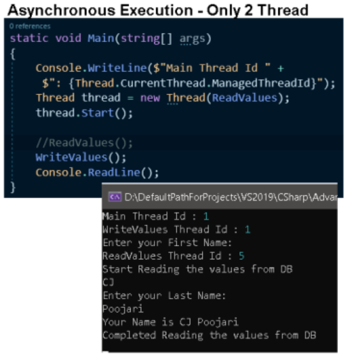
Synchronous Execution - Only 1 Thread



```
1 reference
static void ReadValues()
{
    Console.WriteLine($"ReadValues Thread Id " +
            $": {Thread.CurrentThread.ManagedThreadId}");
    Console.WriteLine("Start Reading the values from DB");
    Task.Delay(10000).Wait();
    Console.WriteLine("Completed Reading the values from DB");
}

1 reference
static void WriteValues()
{
    Console.WriteLine($"WriteValues Thread Id " +
                $": {Thread.CurrentThread.ManagedThreadId}");
    Console.WriteLine("Enter your First Name: ");
    string firstName = Console.ReadLine();
    Console.WriteLine("Enter your Last Name: ");
    string lastName = Console.ReadLine();
    Console.WriteLine($"Your Name is {firstName} {lastName}");
}

0 references
static void Main(string[] args)
{
    Console.WriteLine($"Main Thread Id " +
     $": {Thread.CurrentThread.ManagedThreadId}");
    ReadValues();
    WriteValues();
    Console.ReadLine();
}
```

```
Main Thread Id : 1
ReadValues Thread Id : 1
Start Reading the values from DB
Completed Reading the values from DB
WriteValues Thread Id : 1
Enter your First Name:
CJ
Enter your Last Name:
Poojari
Your Name is CJ Poojari
```

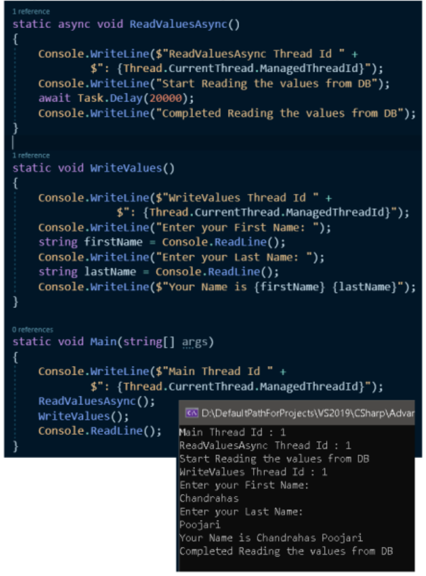
Asynchronous Execution - Only 2 Thread



```
0 references
static void Main(string[] args)
{
    Console.WriteLine($"Main Thread Id " +
     $": {Thread.CurrentThread.ManagedThreadId}");
    Thread thread = new Thread(ReadValues);
    thread.Start();

    //ReadValues();
    WriteValues();
    Console.ReadLine();
}
```

```
Main Thread Id : 1
WriteValues Thread Id : 1
Enter your First Name:
ReadValues Thread Id : 5
Start Reading the values from DB
CJ
Enter your Last Name:
Poojari
Your Name is CJ Poojari
Completed Reading the values from DB
```

Asynchronous Execution(async await) - Only 1 Thread



```
1 reference
static async void ReadValuesAsync()
{
    Console.WriteLine($"ReadValuesAsync Thread Id " +
            $": {Thread.CurrentThread.ManagedThreadId}");
    Console.WriteLine("Start Reading the values from DB");
    await Task.Delay(20000);
    Console.WriteLine("Completed Reading the values from DB");
}

1 reference
static void WriteValues()
{
    Console.WriteLine($"WriteValues Thread Id " +
                $": {Thread.CurrentThread.ManagedThreadId}");
    Console.WriteLine("Enter your First Name: ");
    string firstName = Console.ReadLine();
    Console.WriteLine("Enter your Last Name: ");
    string lastName = Console.ReadLine();
    Console.WriteLine($"Your Name is {firstName} {lastName}");
}

0 references
static void Main(string[] args)
{
    Console.WriteLine($"Main Thread Id " +
            $": {Thread.CurrentThread.ManagedThreadId}");
    ReadValuesAsync();
    WriteValues();
    Console.ReadLine();
}
```

```
Main Thread Id : 1
ReadValuesAsync Thread Id : 1
Start Reading the values from DB
WriteValues Thread Id : 1
Enter your First Name:
Chandrahas
Enter your Last Name:
Poojari
Your Name is Chandrahas Poojari
Completed Reading the values from DB
```

C# Repo:

- Advance C# Repo

Reference:

- Free Workshop TPL by BangaRaju

Dot Net Framework

Coming Soon…