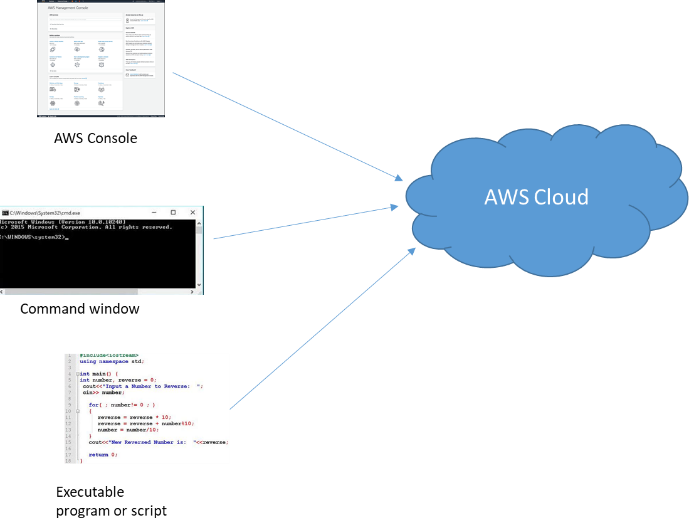
Module 6  
AWS SDK  
  
  
NOTE: I use C# (with the AWS SDK for C#) for all examples code I will give throughout the quarter. Therefore, I recommend you use C# with Visual Studio or VS Code.   
  
If you want to use a language other than C# (e.g., Java or Python) you are welcome to do that. But you have to translate the examples I give in the language of your choice. It is not hard (AWS SDKs exist for all popular languages – you can get them from [here](https://aws.amazon.com/tools/)), but it is additional work that you need to do on your own.  
  
  
Objectives

We said in a previous module that there are three ways to interact with AWS (Figure 1): (1) The AWS Console (i.e., the AWS web site). (2) The AWS CLI (typing commands in a terminal window). (3) Programmatically using a programming language API (Application Programming Interface).   
  
In this module we will look at (3) – that is, how to interact with AWS from code.  
  


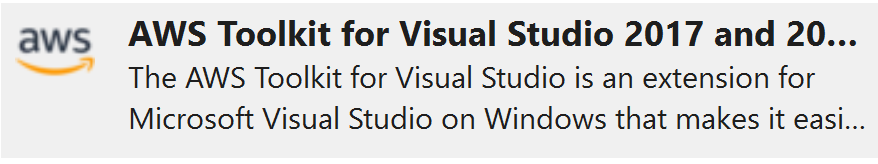
**Figure 1**: Three ways to interact with AWS: the AWS Console (that is, the AWS website), CLI, and APIs.

AWS SDK for .NET

First install the AWS Toolkit for the IDE you are using:

For Visual Studio:

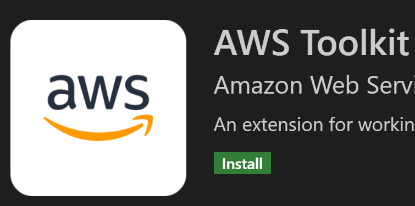
1. Open the Visual Studio IDE.
2. Click menu **Extensions** 🡪 **Manage Extensions**.
3. In the Search textbox in the upper right-hand corner of the dialog, type: AWS Toolkit.

This should bring the AWS Toolkit into view. Select it (select the one that supports your version of Visual Studio. For example, if you have Visual Studio 2022, then you should pick the AWS Toolkit for Visual Studio 2022).  
  


1. Click the Download button to install it.

### For VS Code

1. Open the VS Code IDE.
2. Click menu **File** 🡪 **Preferences** 🡪 **Extensions**.
3. In the Search textbox type: AWS Toolkit.   
   This should bring the AWS Toolkit into view.

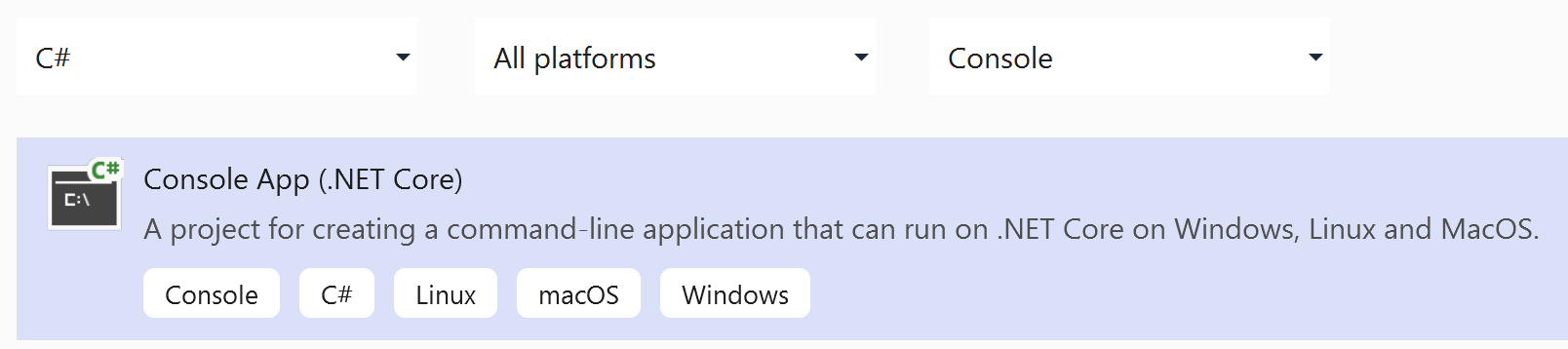
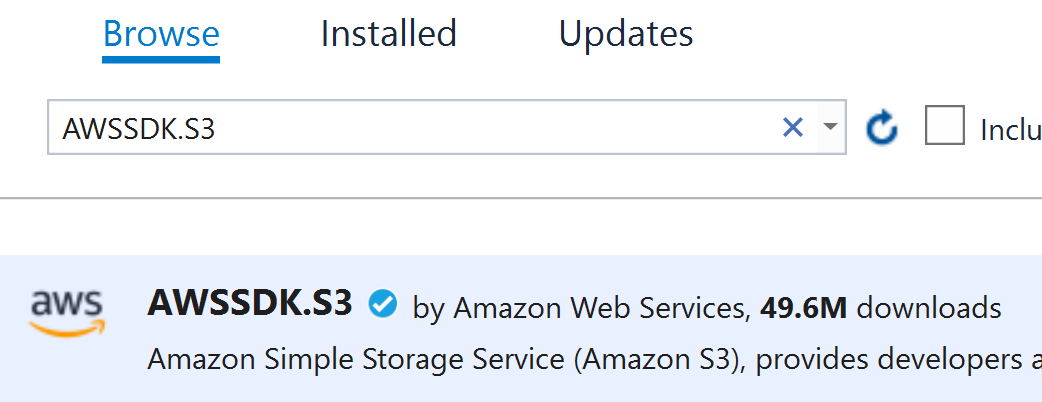
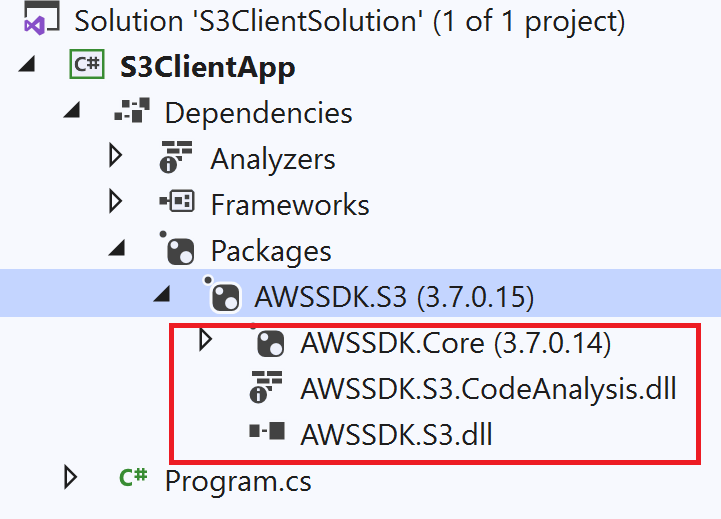
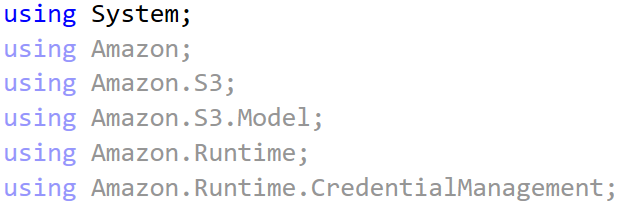


1. Click the green Install button.

## Testing the AWS .NET SDK

1. Open Visual Studio and create a Visual Studio solution (name it “S3ClientSolution”).

NOTE: if you are starting a new project, always create a solution first. Then you can add as many Visual Studio sub-projects to the solution. When we have the all-class meeting, I will explain the proper way to structure a solution in Visual Studio. Many people (especially those not used to Visual Studio end up having many unrelated projects under one solution – and this is incorrect).

1. Right-click on the S3ClientSolution and click menu **Add** 🡪 **New Project…** Choose C#, All platforms, Console. Select project template “Console App (.NET Core)”. Then click the **Next** button.  
     
   
2. Give the project a name: S3ClientApp.  
   Click the **Create** button.
3. Step 3 creates a project. However, we can’t interact with AWS yet. To do that we first need to add the needed AWS libraries. What libraries you add to a project depends on what you plan to do. In this project we plan to interact with the AWS S3 service. So at a minimum, we need the corresponding libraries that allows us to interact with S3.   
     
   I will show detailed steps on how to add libraries to a project. In future modules, you need to do the same when I ask you to add libraries:  
   1. In Solution Explorer, right-click on Dependencies and choose menu **Manage NuGet Packages…**
   2. While **Browse** is selected, type in the search box: AWSSDK.S3. This should bring the AWSSDK.S3 library into view.   
        
      
   3. Click the **Install** button. Then press **OK** on the Preview Changes dialog. This adds the libraries to the project.
   4. In Solution Explorer, expand Dependencies – Packages – AWSSDK.S3 to see what was added.  
        
      
4. Now you will write some code to use these AWS libraries:  
     
   Double-click on file Program.cs to open it. Add the following using statements at the top of the file:  
     
   

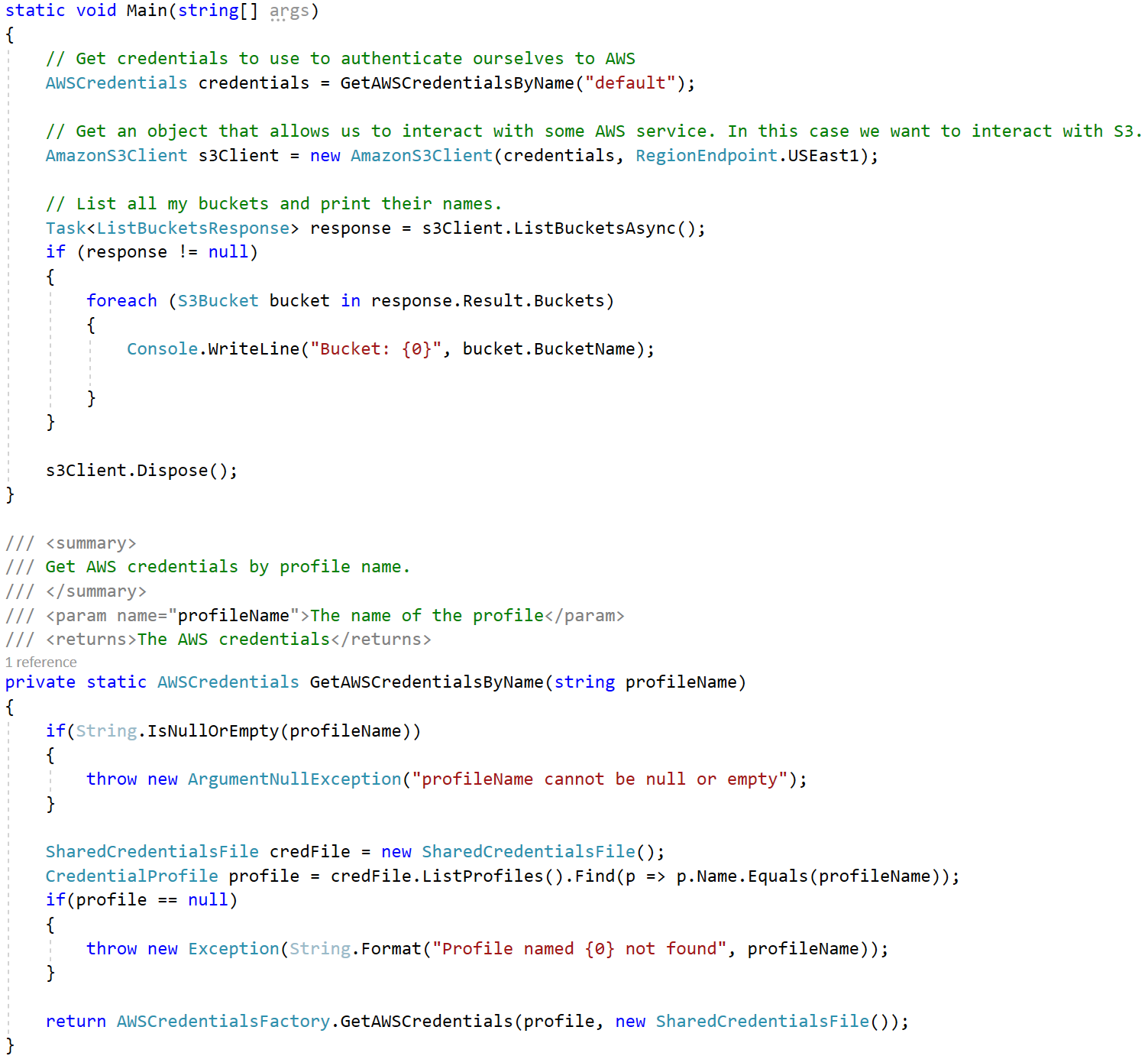
And remove the Hello world print statement.

You probably noticed that Visual Studio gave you intelli-sense code-completion as you typed Amazon… This is because the project is now aware of these Amazon libraries. If you ever try to use a namespace and Visual Studio doesn’t give you intelli-sense capability, most of the time it is due to some library reference issue. In this case, you go back and make sure that you have added said library (as we have done above in step 4).

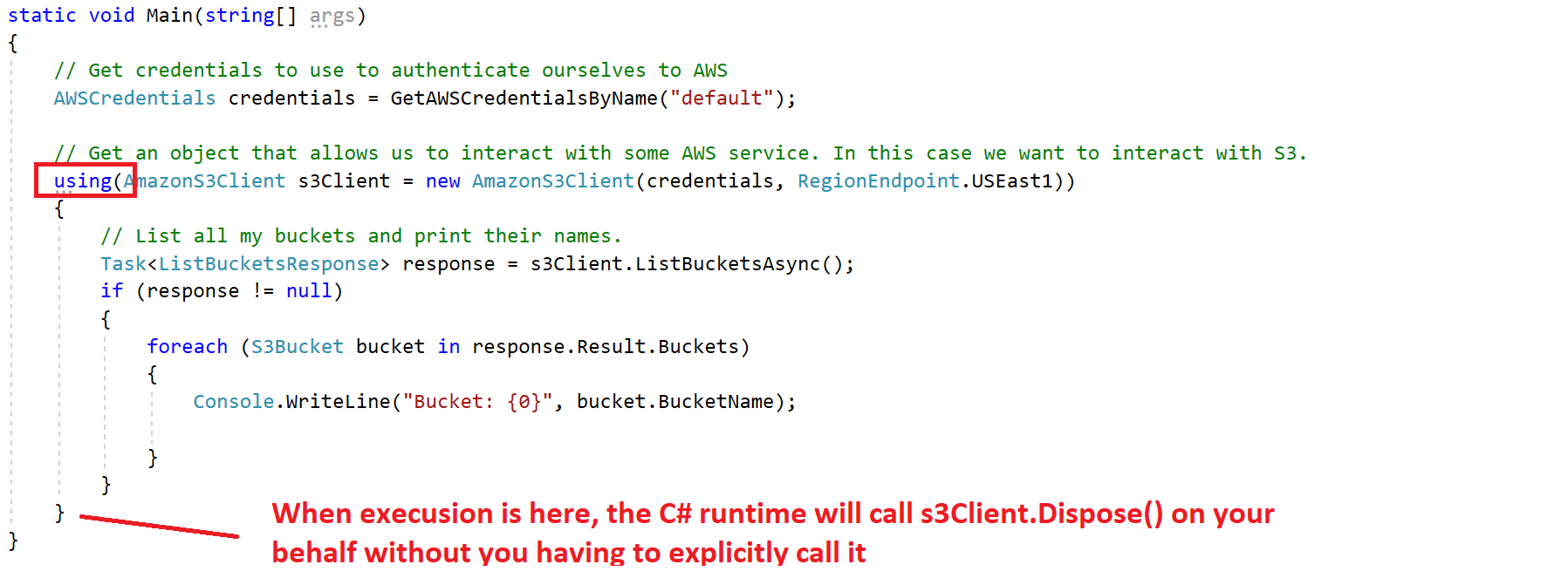
**Is there a way I can see from inside Visual Studio what is inside the libraries I added to my project (that is, namespaces, classes, methods, etc.)?**  
  
Yes. Try it:

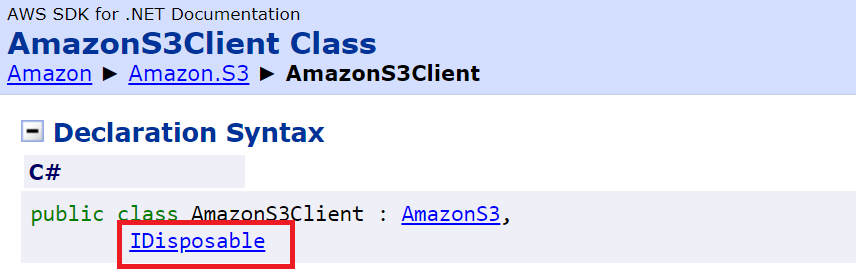
1. In Visual Studio, click menu **View** 🡪 **Object Browser**.
2. Expand AWSSDK.S3 – Amazon.S3 and look at all the classes that are defined in the Amazon.S3 namespace.
3. Click on any class (e.g., AmazonS3Client). In the right pane you will see methods defined in the class.
4. Click a method (e.g., ListObjectsAsync). The lower pane gives you description of the method, the method parameters and return type, etc.

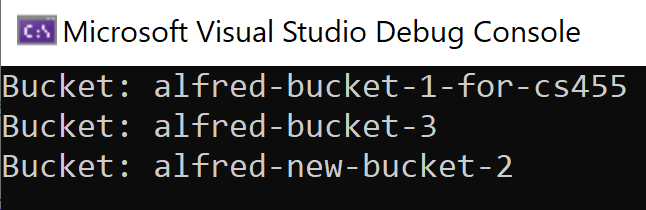
**The Object Browser is a great way to gain understanding and get help on libraries you are using. Get in the habit of using it whenever you need clarification.**

1. We plan to interact with AWS via code. Just like when you used the CLI, this too requires that you authenticate yourself to AWS. And we will use the credentials file again. Once we do that we can make calls to the AWS service we are interested in (in this example S3). Update Programs.cs with a code similar to Figure 2:  
     
     
     
   **Figure 2**: Using a profile from the credentials file to authenticate to AWS and interact with the S3 service.  
     
   Your credentials data are probably stale. Before you run the code above, close Visual Studio, update your credentials file, save it, then restart Visual Studio.  
     
   In method GetAWSCredentialsByName, we are using a few classes to return an AWSCredentials object. Because the credentials file supports multiple profiles, we are using argument profileName to make our function versatile. You can use this function as-is in future modules.

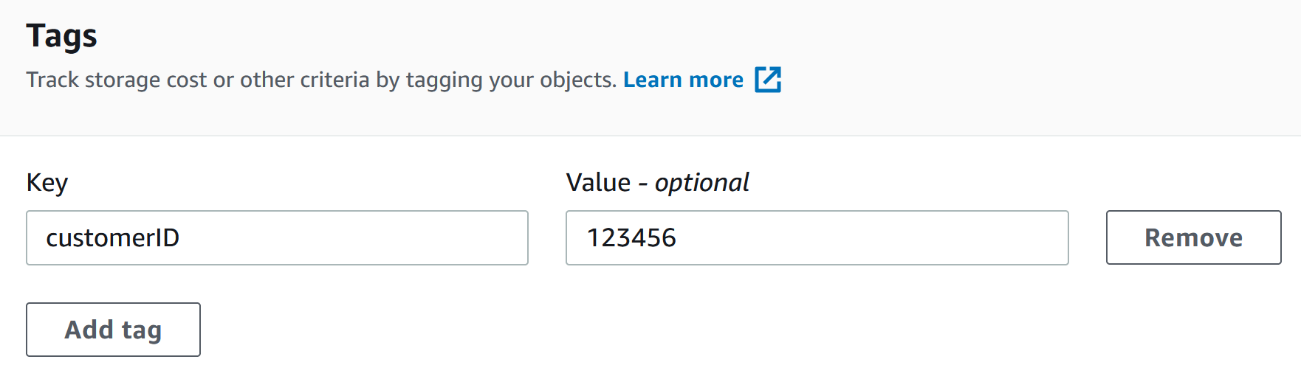
In the main function, we create object credentials (of type AWSCredentials). We then pass it to the constructor of class AmazonS3Client, which will allow us to interact with S3.  
This is a common pattern we can follow with other services; other services have client objects (similar to AmazonS3Client) to interact with said services. After that, we call the ListBucketsAsync method to get the buckets, loop through them, and print their names.  
  
In the last line we call s3Client.Dispose(). The reason we do this is because class AmazonS3Client implements interface [IDisposable](https://docs.microsoft.com/en-us/dotnet/api/system.idisposable?view=net-5.0). The IDisposable interface defines a Dispose method. When a class implement the IDisposable interface, it is in a way telling you this: “*I might be holding resources that needs to be cleaned or released. When done using me, call Dispose so that I do all needed cleanup*”.

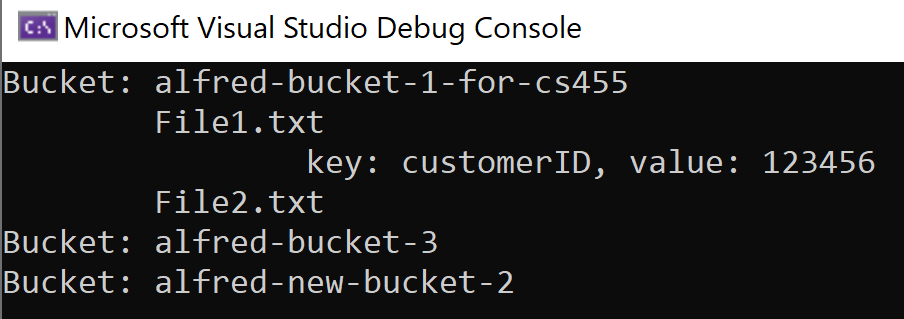
C# provides a syntax (using() { }) that calls Dispose for you implicitly (Figure 3). The main code in Figure 3 is equivalent to the main code in Figure 2.  
  
  
**Figure 3**: Using the using() {} construct, the runtime calls Dispose for you implicitly.

Good Programming Practice:  
  
When you use any object in .NET, always check if it implements the IDisposable interface (you can either look at the documentation or check if it has a Dispose method). If it does, call the Dispose method when done with it. Alternatively, you can wrap the creation of the object in a using () { } construct. The using () { } construct implicitly calls Dispose when it reaches the closing bracket.  
  
See using keyword documentation [here](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/using).  
  
The AmazonS3Client class [documentation](https://docs.aws.amazon.com/sdkfornet1/latest/apidocs/html/T_Amazon_S3_AmazonS3Client.htm) also shows you that it implements IDisposable.  
  


1. Run the program. The output should show the S3 buckets you have (if you don’t have any buckets, manually create one or two from the AWS Console so that you can test and verify that your code works). My output looks like this:  
     
   

## Exercise to do and submit

1. Modify the program you started in this module to create a bucket (that is, you need to create a bucket via code – not manually using the AWS console) .  
     
   Then list your buckets and print the bucket names.
2. Now login to the AWS Console, choose any of the buckets you have, and add 2 files to it (e.g., File1.txt and File2.txt – you can create those with some dummy content). When adding one of the files (e.g., File2.txt) specify some key/value metadata tag (for example, you can have a key of “customerID” and its value as “123456”).  
     
   

Custom metadata (or tags) are useful since they allow you to attach additional information to an S3 file. That information may be important to your cloud application. One can imagine many use cases where this can be useful.  
  
Now modify your program to do the following: For each bucket, print the names of files it has. And for each file, print the metadata tags associated with the file. In my case, my output looks like this:  
  
  
  
HINTS: To achieve the above I used a few classes I am listing below:  
  
ListObjectsV2Request  
ListObjectsV2Response  
GetObjectTaggingRequest  
GetObjectTaggingResponse  
  
And the following methods:  
  
s3Client.ListObjectsV2Async  
s3Client.GetObjectTaggingAsync

1. You might have noticed a common pattern how methods arguments and results are organized. Something like:

SomeTypeRequest request = new SomeTypeRequest();  
// Set request properties  
…  
…  
  
// Call method and receive results in a response object  
Task<SomeTypeResponse> response = client.DoSomething(request, …);  
  
Can you think of a reason why this is a good practice (encapsulating many arguments in a request object instead of say passing many individual arguments)? Consider things like the service might evolve with additional features, etc. How does that affect existing client code?

**What to submit:**

Nothing to submit for this module. No quiz for this module.