Distributed Computing

Multi-Tiered Architectures

Async-Await-Task

This Week

- Multi Tiered Architectures
  - What are they
  - Why we use them
  - How they work
  - Which ones suit which situations
- Asynchronous Communications
  - Why these are important
  - How to do them in C#

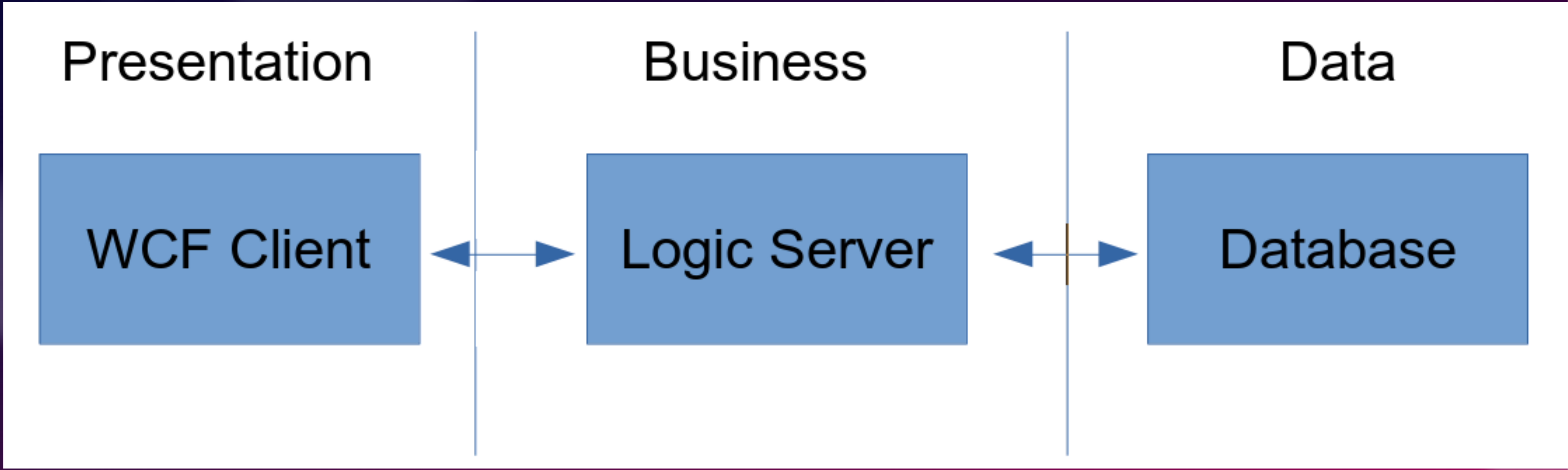
Distributing an app over a network

- We have so far covered an intro to Distribution and the ways of doing it in the past.
- We still have *no* idea how to tell which parts of a program you can distribute.
- Well, let’s investigate that today

The Three Tiered Model

- Fairly old standard of distribution
- A reasonable starting place, although outdated
- Consists of three parts:
  - The Data Tier
  - The Business Tier
  - The Presentation Tier

Three Tiers


Presentation
Business
Data
WCF Client
Logic Server
Database

Why distribute this way?

- This helps with modularisation:
  - Each tier exposes one interface and uses one interface
    - Which means very low coupling
    - You can even have multiple tier implementations!
  - Clear delineation of concerns between tiers
- Better security too:
  - This design limits connection between tiers
  - You can hide functionality you don’t want users to have access to!

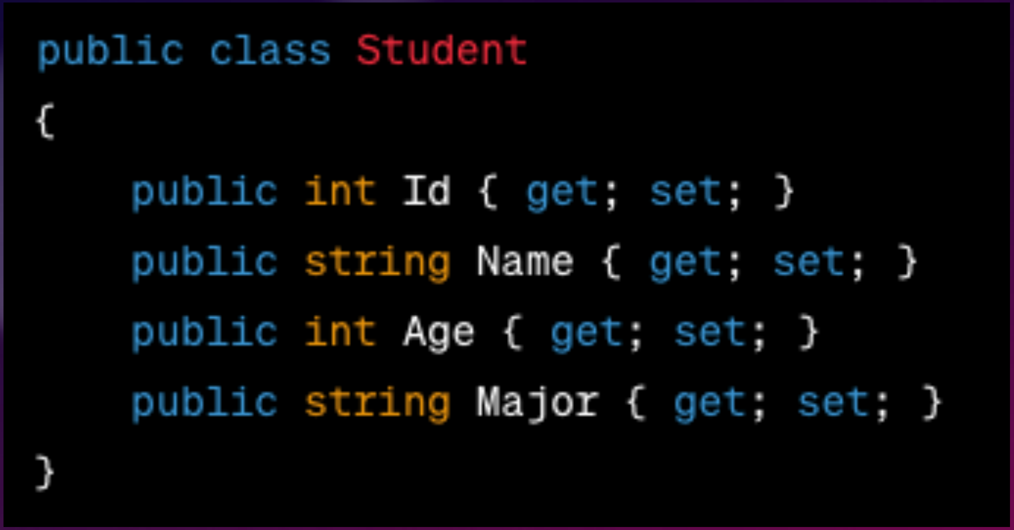
Data Tier

- Mostly just a database
  - *An interface *to* a database. Sometimes just a DB but not often.
- Purpose is to serve data to the business tier
- Tends to be a singleton
  - Databases generally don’t like race conditions
- Doesn’t need to consider *how* data is used, just provides interface for business tier.

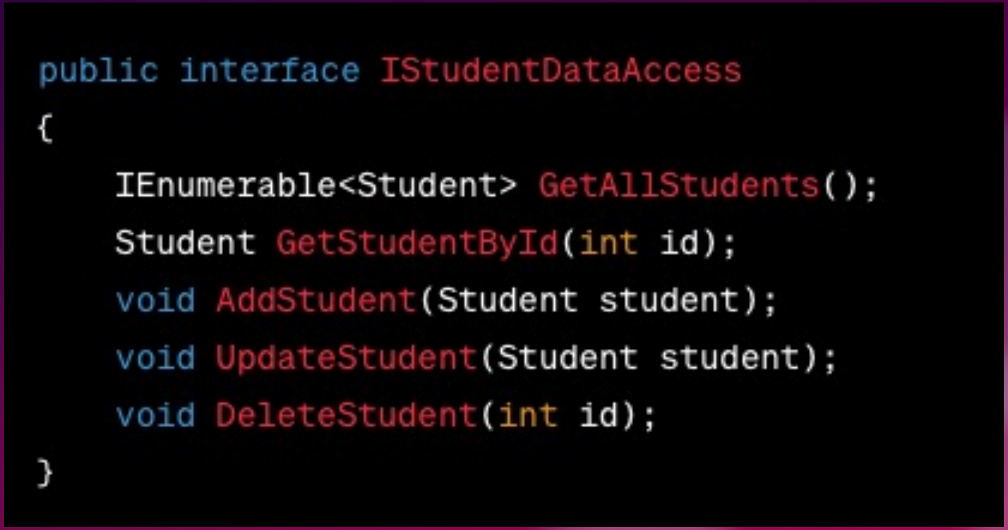
Advanced Data Tier

- Technically a Data Tier doesn’t need to be a database…. It could be a program, it could be *multiple* db’s.
  - Data tier can obfuscate *how* data is retrieved, and from what.
  - Can build complex data gathering systems with single unified component interface.

Data tier Interface example



```
public class Student
{
    public int Id { get; set; }
    public string Name { get; set; }
    public int Age { get; set; }
    public string Major { get; set; }
}
```



```
public interface IStudentDataAccess
{
    IEnumerable<Student> GetAllStudents();
    Student GetStudentById(int id);
    void AddStudent(Student student);
    void UpdateStudent(Student student);
    void DeleteStudent(int id);
}
```

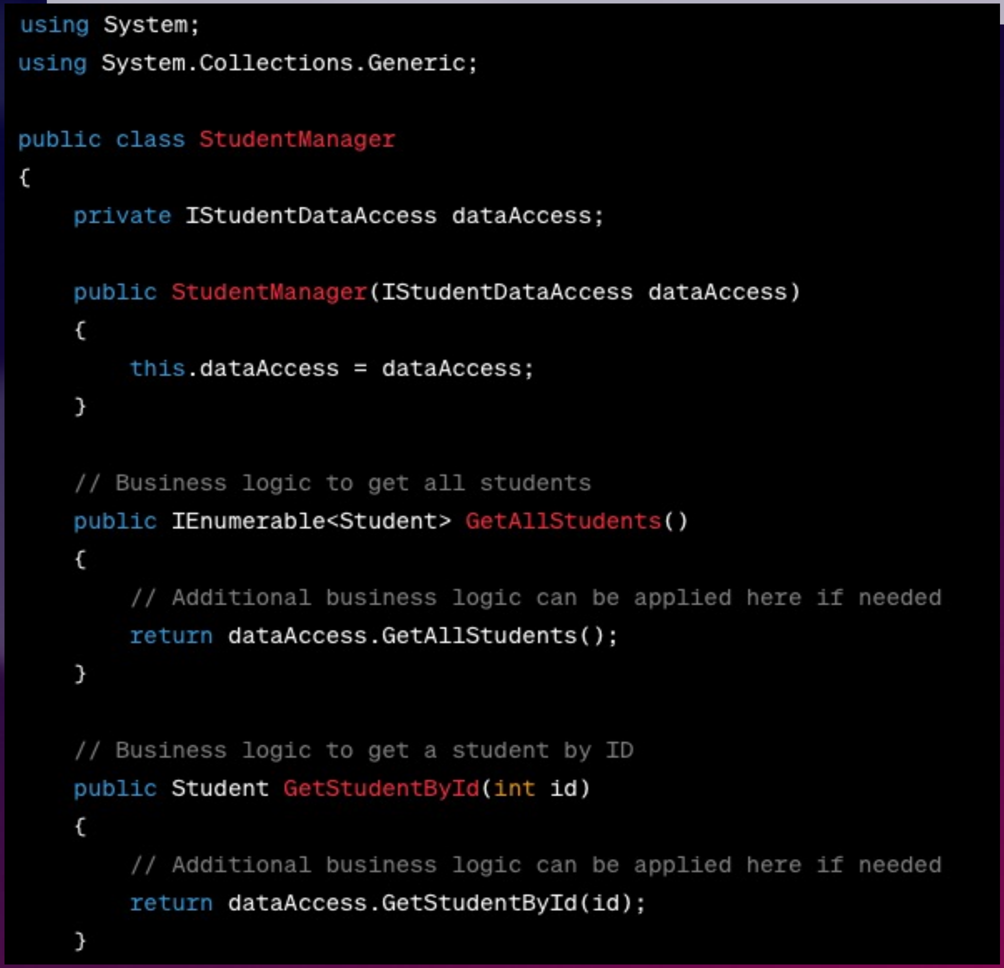
Business Tier

- A tier that does the “business logic”
- If it’s not directly related to the data, and it’s not related to presentation, it belongs here.
  - Logins
  - Game world management
  - Data processing

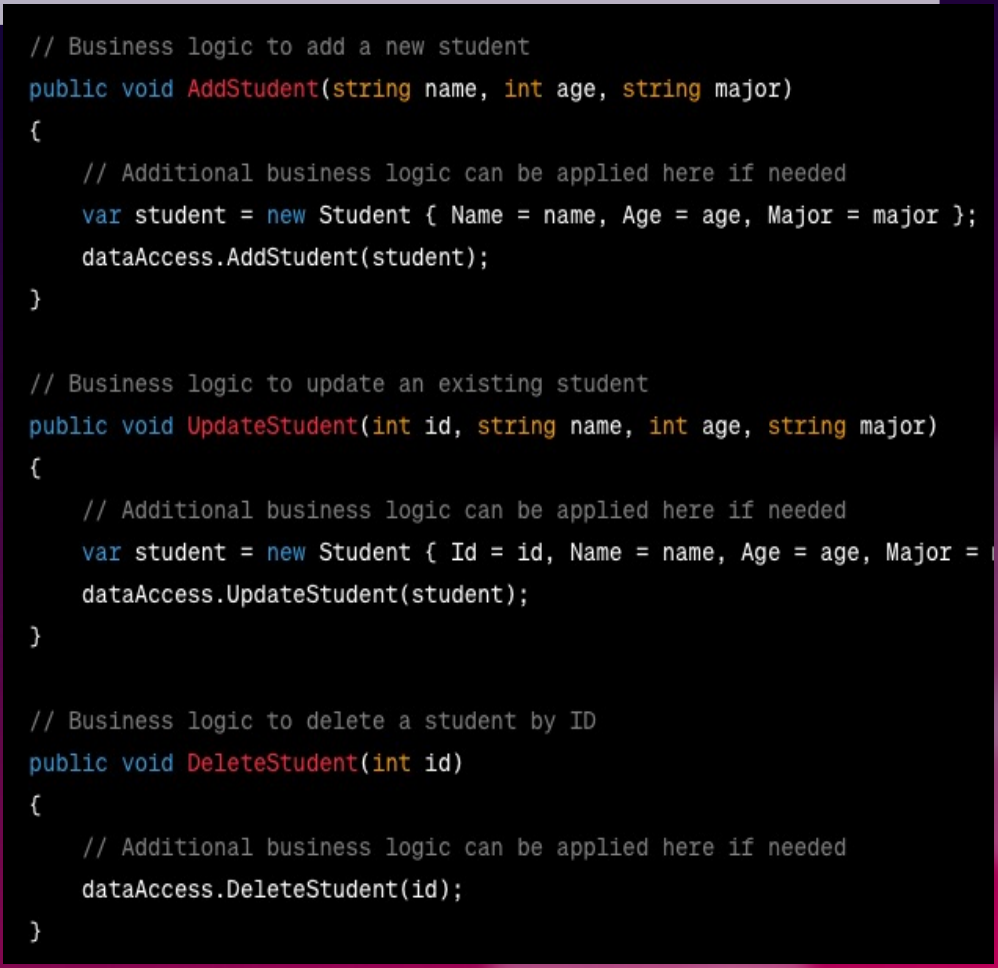
Business Tier

- Business tier can be distributed
  - Allows for load balancing of applications
  - Can allow for specialisation of application back ends
    - Have one business tier service that handles one specific service
    - Can even use a multi-level business tier to route clients to the requisite machines!
  - Can be used to act as a uniform platform access to multiple underlying systems

Business Tier Example



```
using System;
using System.Collections.Generic;

public class StudentManager
{
    private IStudentDataAccess dataAccess;

    public StudentManager(IStudentDataAccess dataAccess)
    {
        this.dataAccess = dataAccess;
    }

    // Business logic to get all students
    public IEnumerable<Student> GetAllStudents()
    {
        // Additional business logic can be applied here if needed
        return dataAccess.GetAllStudents();
    }

    // Business logic to get a student by ID
    public Student GetStudentById(int id)
    {
        // Additional business logic can be applied here if needed
        return dataAccess.GetStudentById(id);
    }
```



```
// Business logic to add a new student
public void AddStudent(string name, int age, string major)
{
    // Additional business logic can be applied here if needed
    var student = new Student { Name = name, Age = age, Major = major };
    dataAccess.AddStudent(student);
}

// Business logic to update an existing student
public void UpdateStudent(int id, string name, int age, string major)
{
    // Additional business logic can be applied here if needed
    var student = new Student { Id = id, Name = name, Age = age, Major =
    dataAccess.UpdateStudent(student);
}

// Business logic to delete a student by ID
public void DeleteStudent(int id)
{
    // Additional business logic can be applied here if needed
    dataAccess.DeleteStudent(id);
}
```

Business Tier Example (Cont..)

- We've defined a class StudentManager, which acts as the business tier.

- It uses IStudentDataAccess (the data tier interface) as a constructor parameter. This allows us to easily switch between different data access implementations (e.g., using a different database or data storage solution) without modifying the business tier code.

- The StudentManager class exposes methods that encapsulate the business logic for interacting with student data.

- Each method interacts with the data tier through the IStudentDataAccess interface to perform the desired operations (e.g., retrieving, adding, updating, or deleting students).

Presentation Tier

- The bit you see.

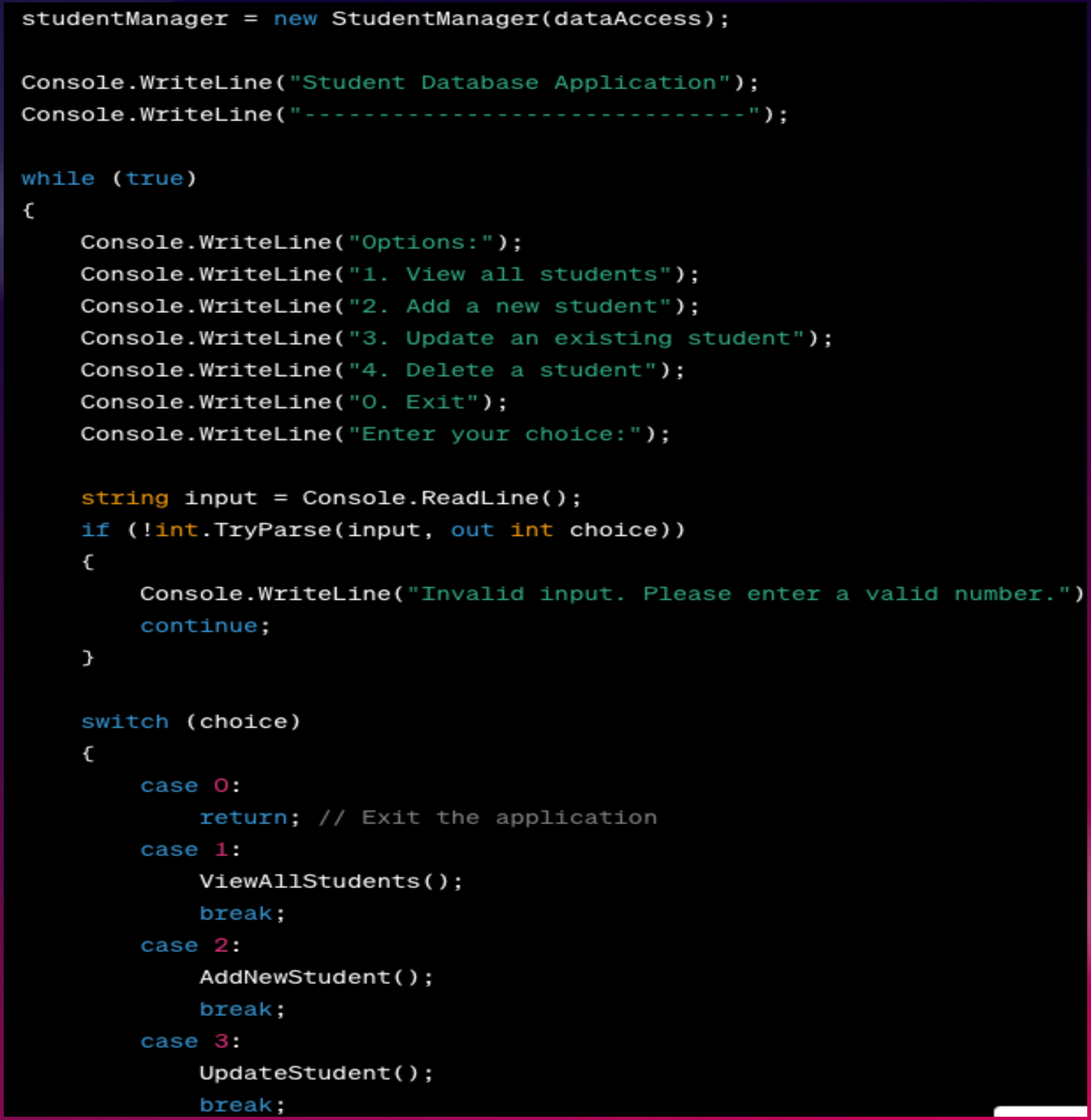
Presentation Tier

- Presents the information filtered through the Business Tier.
- Can be:
  - Window
  - Console program
  - Phone app?
- Mostly data to display processing
- Also input validation
  - Basically anything from HCI.

Presentation Tier

- Inherently distributed
  - Are you really only going to have one user?
- Can be multiple programs
  - Use a standard API on your Biz tier and it’s easy!
- Could theoretically have same user on multiple devices
  - Just hide everything by running one object per user at the business tier.

Presentation tier – Console Example

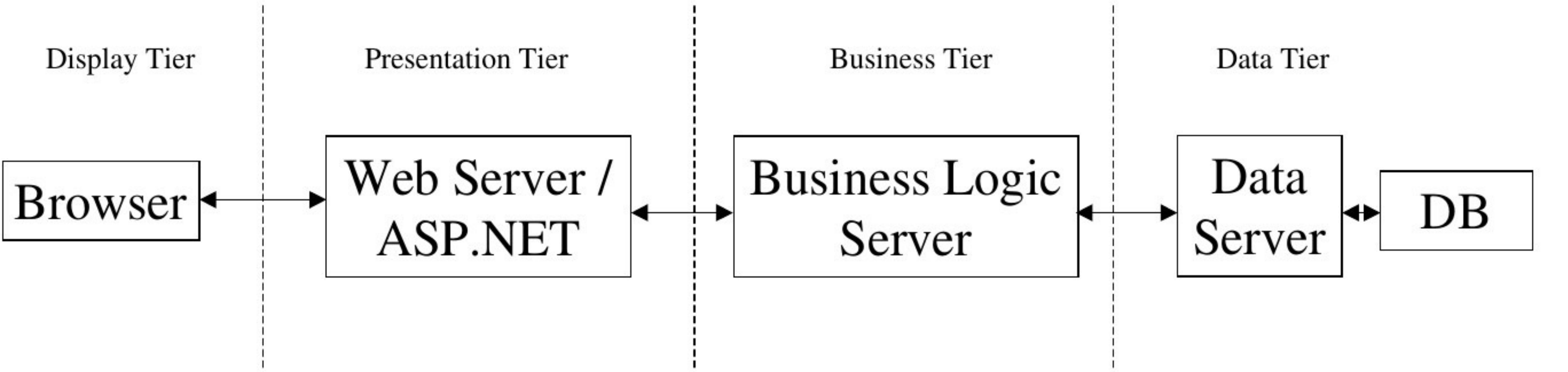


```
studentManager = new StudentManager(dataAccess);

Console.WriteLine("Student Database Application");
Console.WriteLine("------------------------------");

while (true)
{
    Console.WriteLine("Options:");
    Console.WriteLine("1. View all students");
    Console.WriteLine("2. Add a new student");
    Console.WriteLine("3. Update an existing student");
    Console.WriteLine("4. Delete a student");
    Console.WriteLine("0. Exit");
    Console.WriteLine("Enter your choice:");

    string input = Console.ReadLine();
    if (!int.TryParse(input, out int choice))
    {
        Console.WriteLine("Invalid input. Please enter a valid number.")
        continue;
    }

    switch (choice)
    {
        case 0:
            return; // Exit the application
        case 1:
            ViewAllStudents();
            break;
        case 2:
            AddNewStudent();
            break;
        case 3:
            UpdateStudent();
            break;
```

Going further

- You may have noticed a distinct lack of browser in what the Presentation tier can be.
- This is because the Browser doesn't do presentation, the web server does.
  - The Browser just displays the presentation….
- But hang on… isn't converting from HTTP to something nice work?
  - Definitely with Javascript it is!

The Four Tier System



Presentation Tier

- So if a Presentation tier isn't for the user….
  - Who on earth is it for?
- Answer: The "user"
  - Could be the display tier
  - Could be other presentation tiers
  - Could be other business tiers
  - Could be *anything*
- Presentation Tier is just a general access interface!

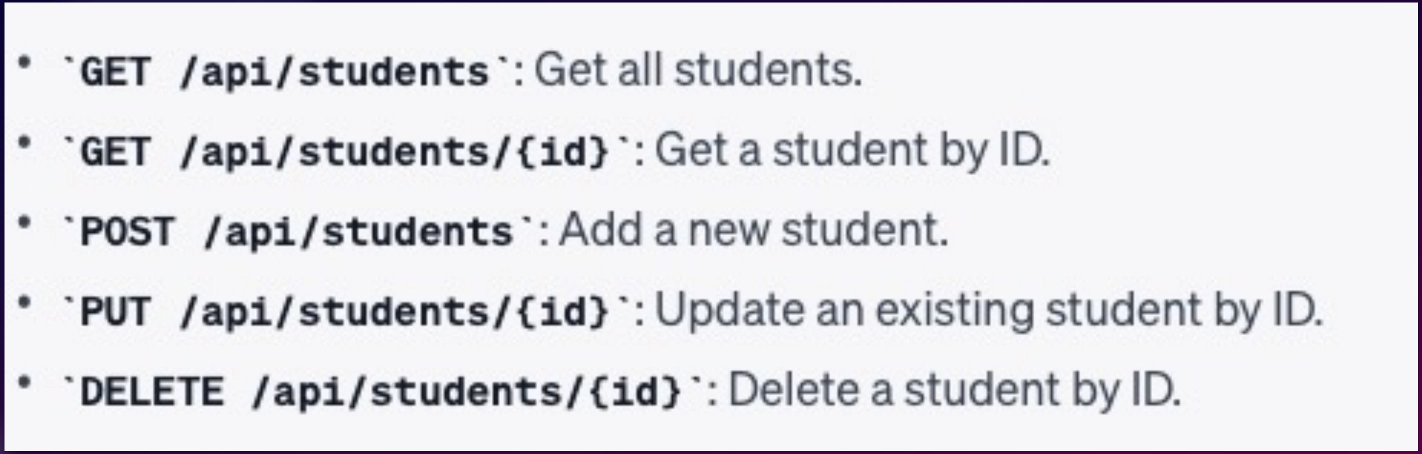
Presentation Tier (Cont..)

- Usually appears as an API for access to a service
  - Most common are web services that can be rendered by the browser
    - Usually REST or SOAP based.
- Presents an interface to the *entire program backend*.
  - Like some kind of SuperService

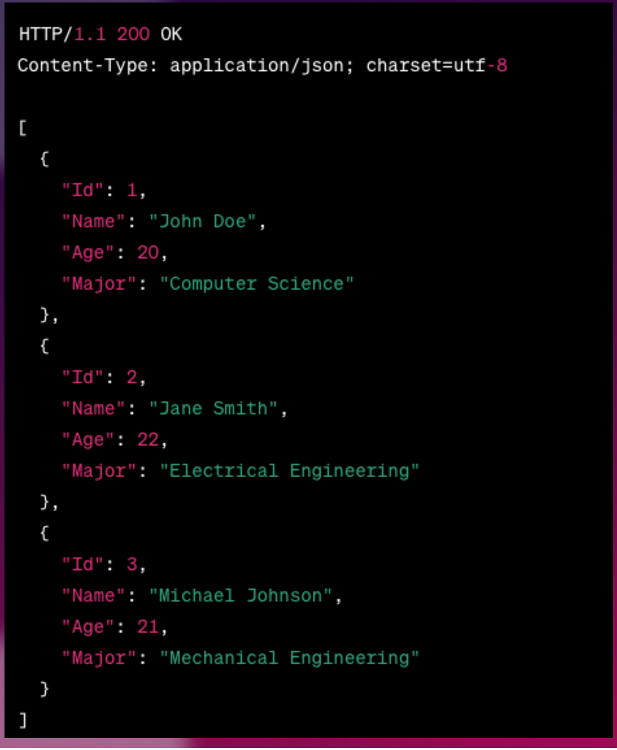
Presentation Tier (Cont..)

- Can you distribute?
- Of course you can!
  - Load balancing for massive user load
  - APIs for humans and bots
  - Specialised API's for specific services in the application
    - Distribute *entire* chunks of the application into multi-tier services? Why not?

Presentation Tier API Example



- `GET /api/students`: Get all students.
- `GET /api/students/{id}`: Get a student by ID.
- `POST /api/students`: Add a new student.
- `PUT /api/students/{id}`: Update an existing student by ID.
- `DELETE /api/students/{id}`: Delete a student by ID.



```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8

[
  {
    "Id": 1,
    "Name": "John Doe",
    "Age": 20,
    "Major": "Computer Science"
  },
  {
    "Id": 2,
    "Name": "Jane Smith",
    "Age": 22,
    "Major": "Electrical Engineering"
  },
  {
    "Id": 3,
    "Name": "Michael Johnson",
    "Age": 21,
    "Major": "Mechanical Engineering"
  }
]
```

You can easily change the output to different formats – json, xml, html, js etc.

Display Tier

- Displays information for human consumption
  - This is different to the Presentation tier… Even though I’m sure you can read XML.
- Does do actual work!
  - If window or client application, is still doing user input validation, and converting API data to things that make visual sense.
  - If browser, it has Javascript and CSS to convert the otherwise “unreadable” HTML output.

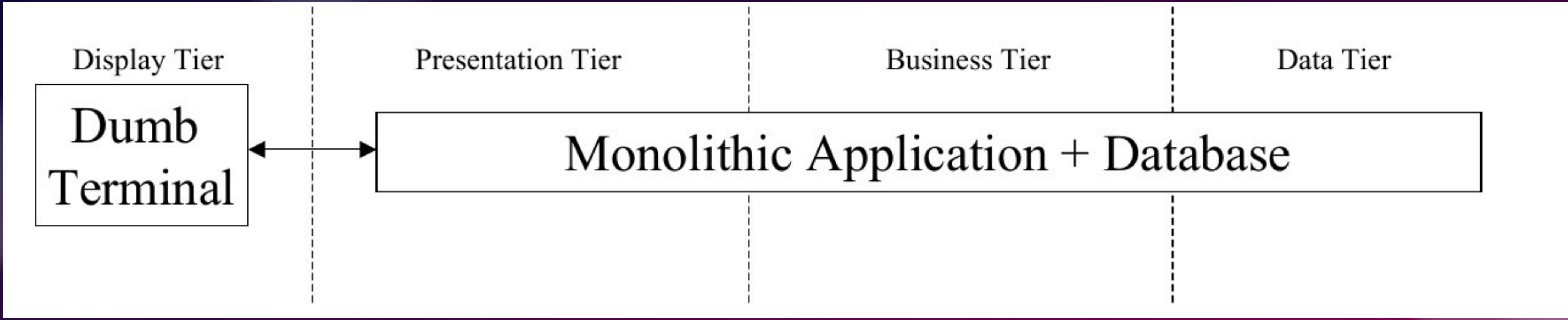
Display Tier

- Inherently distributed due to multiple users
- Can be multiple systems
  - Usually browsers, or other systems (local or distributed)
- Can have one user across multiple systems, although it starts to become hard to maintain state over three tiers.

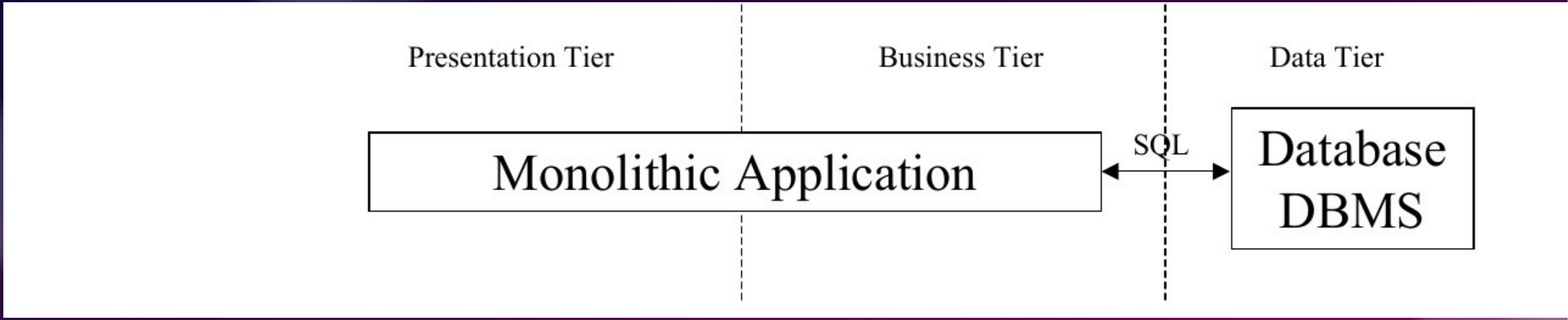
Advanced Techniques

- You can also hybridise systems across tiers.
- Easiest one is to simple integrate all four tiers
  - However this is simply a monolithic app…
- Lets have a look at some variants of this approach

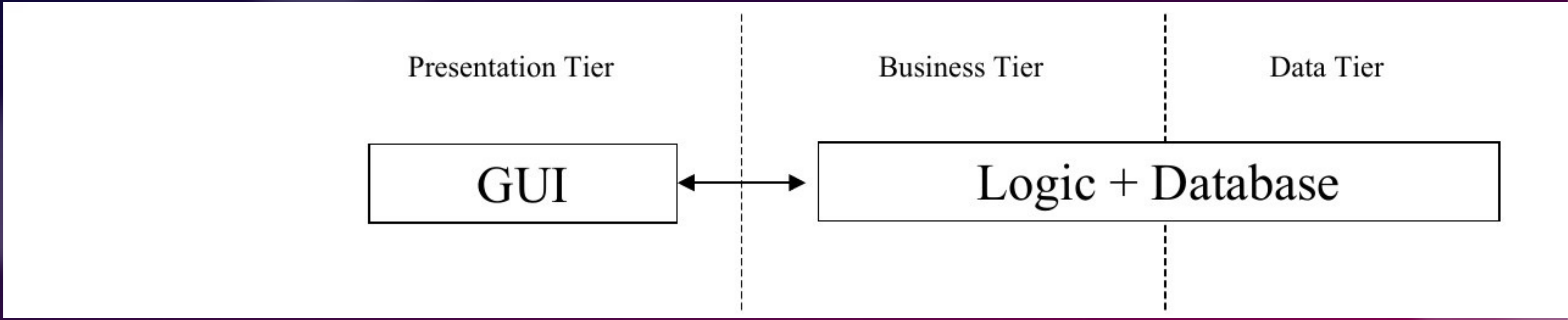
The Dumb Terminal



The SQL Backend



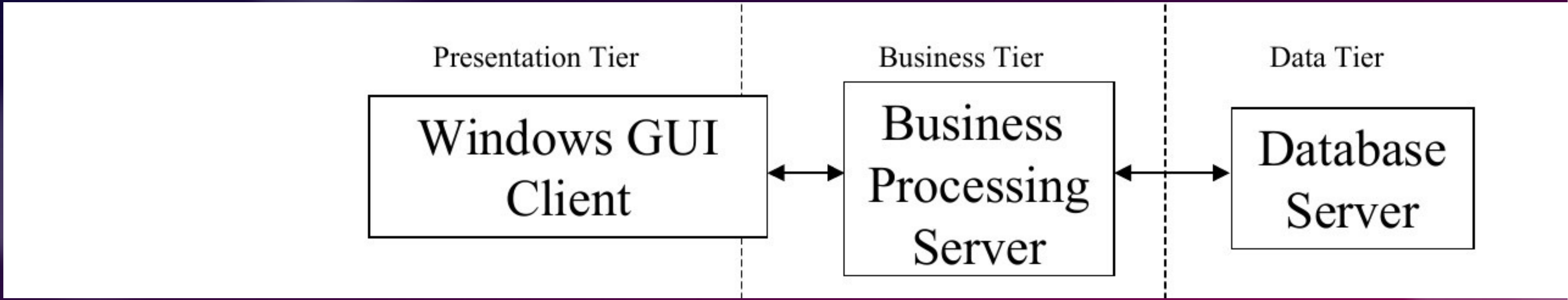
The Network Application



Partial Tiers

- Sometimes it makes sense to have multiple specialised business tiers
- Sometimes you might want to merge two tiers
- So sometimes you may have applications that partially leak over into another tier
  - Maybe your business tier has some kind of data cache

Example: The Destiny 2



Distributing your Tiers

- Parallel Computing:
  - Distribute Business Tier, can do heavy workloads on many pcs
- Load Balancing:
  - Any tier can be involved in this, use tier parallelism to decrease load on any one machine
- Fault Tolerance:
  - If one machine breaks, route to one that isn’t broken.

How many tiers is too many tiers?

- Realistically you won’t usually need all four tiers.
  - Most applications are reasonably simple
  - Web Services regularly combine business and presentation tiers
- Part of your job as a software architect is to figure out which tiers you need.

4 Tiers is a really large queue….

- Say you want to run something on the data tier (defrag or something)
- Display calls Presentation tier, blocks
  - Presentation calls Business, blocks
    - Business calls Data, blocks
      - Data takes about a trillion years to defrag…
- What does the user do? Wait?
  - No they throw their computer out the window.

People Hate Waiting

- More accurately, people hate feeling like their application has stopped responding.
- If you let your app just hang while computing, people will just close it.
- So, what you need to do is not let things hang
  - Progress bars
  - Countdown timers (even if they’re wrong)
  - A spinning icon will do

Asynchronicity

- Hey you guys like threads?
  - Did you enjoy OS?
- There are many ways of handling blocking with distributed applications:
  - Don’t block (aka OneWay connections)
  - Block (might be useful under some conditions)
  - Block in another thread

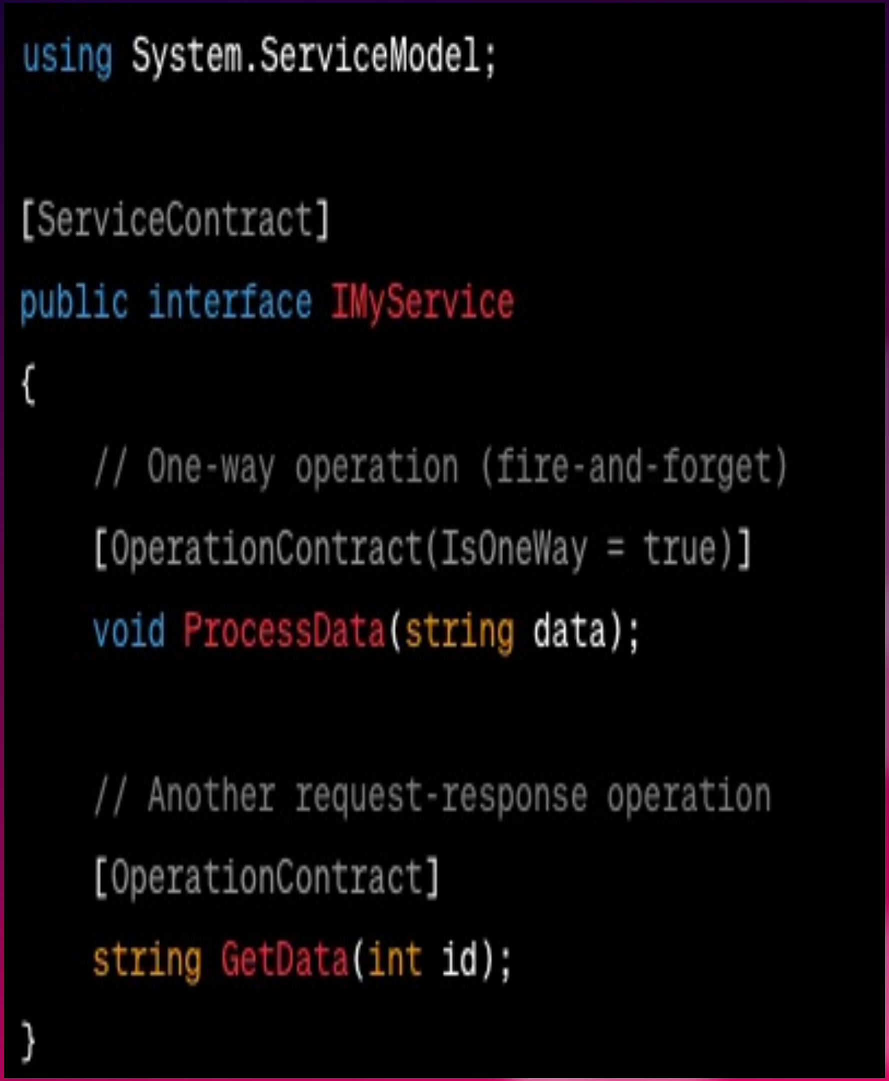
One Way Calls

- Simplest way of doing non blocking remote calls
- Problem: Does not return anything
  - Must be void type
  - Must have no out mode parameters.
- Defined with [OperationContract(IsOneWay=true)]

One-way calls (Example)

- The ProcessData method is a one-way operation, indicated by the IsOneWay=true attribute.
- It takes a string parameter data, and when the client calls this method, it sends the data to the server but doesn't wait for any response.

- On the other hand, the GetData method is a standard request-response operation that takes an int parameter id and returns a string result back to the client.



```
using System.ServiceModel;

[ServiceContract]
public interface IMyService
{
    // One-way operation (fire-and-forget)
    [OperationContract(IsOneWay = true)]
    void ProcessData(string data);

    // Another request-response operation
    [OperationContract]
    string GetData(int id);
}
```

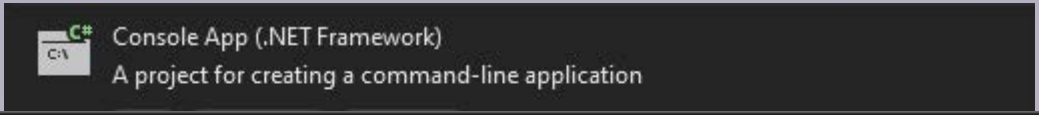
One Way Calls

- One Way still needs to connect to work
  - This means it can still time out, and still block
  - In some cases, you may want to delegate One Way calls.
- One Way *does not return*.
  - This means you will not know anything after the function is called.
  - You’ll need to keep this in mind.

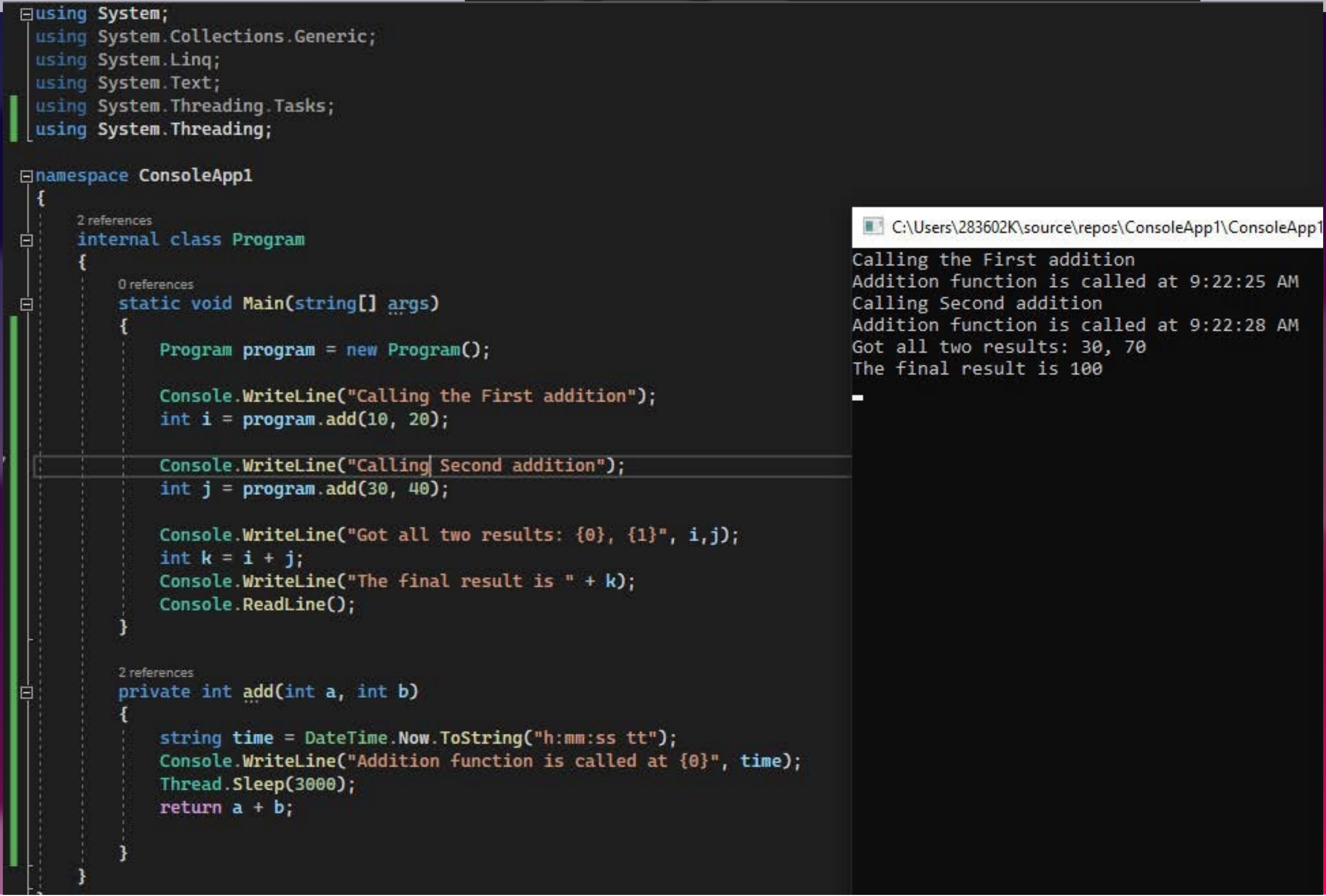
Threads, a recap

- Doing things at the same time as other things is possible on computers.
  - At least with anything that supports concurrency.
- Threads is a convenient way of managing multiple processes in a single application
  - Shared memory, independent execution.

Synchronous Method execution



Console App (.NET Framework)
A project for creating a command-line application



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Threading;

namespace ConsoleApp1
{
    2 references
    internal class Program
    {
        0 references
        static void Main(string[] args)
        {
            Program program = new Program();

            Console.WriteLine("Calling the First addition");
            int i = program.add(10, 20);

            Console.WriteLine("Calling Second addition");
            int j = program.add(30, 40);

            Console.WriteLine("Got all two results: {0}, {1}", i,j);
            int k = i + j;
            Console.WriteLine("The final result is " + k);
            Console.ReadLine();
        }

        2 references
        private int add(int a, int b)
        {
            string time = DateTime.Now.ToString("h:mm:ss tt");
            Console.WriteLine("Addition function is called at {0}", time);
            Thread.Sleep(3000);
            return a + b;

        }
    }
```

C:\Users\283602K\source\repos\ConsoleApp1\ConsoleApp1

```
Calling the First addition
Addition function is called at 9:22:25 AM
Calling Second addition
Addition function is called at 9:22:28 AM
Got all two results: 30, 70
The final result is 100
```

C# Async Overview

- The async keyword is used to mark a method as an asynchronous method. An asynchronous method can contain one or more await expressions.
- When a method is marked as async, it can use the await keyword to asynchronously wait for the completion of other asynchronous operations without blocking the current thread.

C# Task overview

- The Task class represents an asynchronous operation that may or may not return a result.
- It can be used to represent a single asynchronous operation that runs in the background, freeing up the calling thread to perform other tasks.
- When an asynchronous method is called, it returns a Task or Task<TResult>, allowing the caller to wait for or continue with other tasks using await.
- The Task class provides methods to handle the completion, cancellation, and exception handling of asynchronous operations.

C# Await overview

•The await keyword is used inside an asynchronous method to asynchronously wait for the completion of a Task or Task<TResult>.

•It allows the current method to suspend its execution and return control to its caller without blocking the thread.

•When the awaited Task completes, the method resumes execution from the point after the await statement.

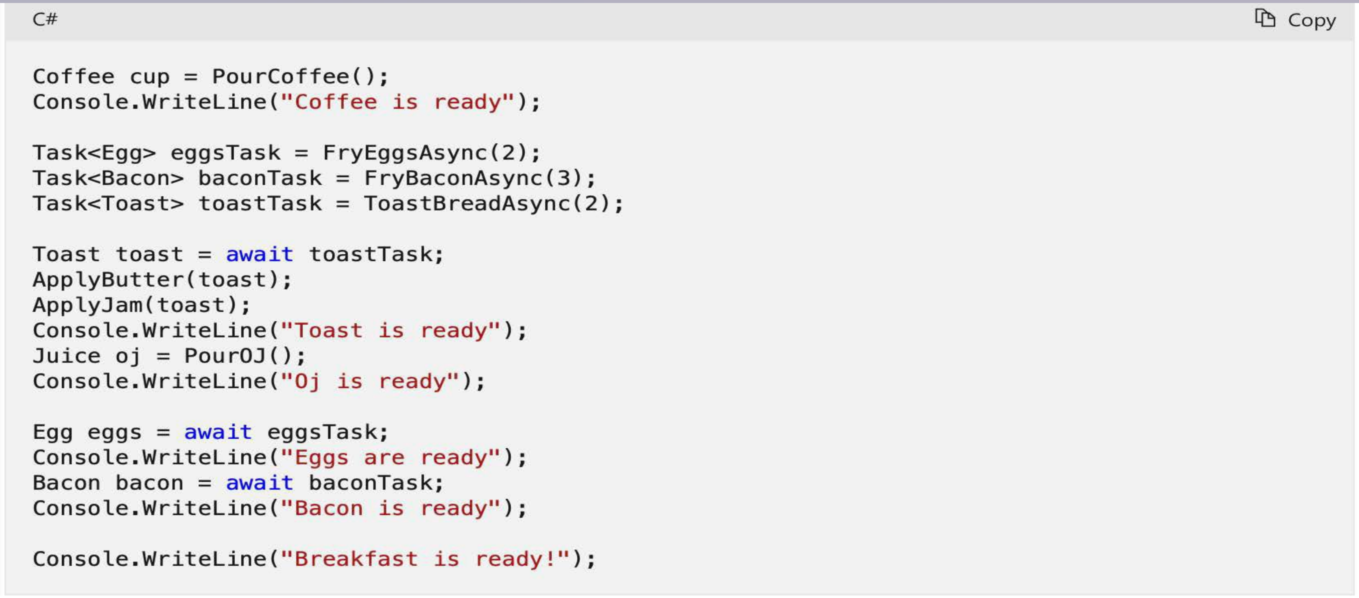
•If the awaited Task throws an exception, the await expression will rethrow the exception.

Reading file asynchronously

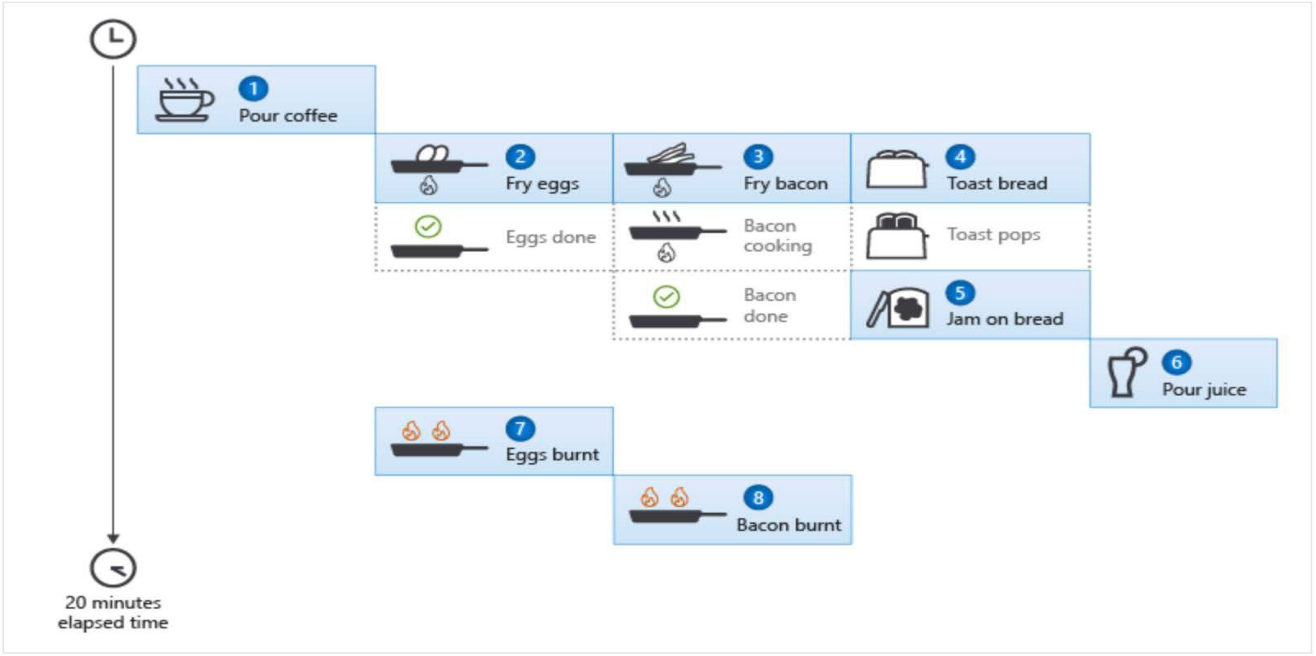
```csharp
static async Task Main(string[] args)
{
    string filePath = "example.txt";

    try
    {
        string fileContent = await ReadFileAsync(filePath);
        Console.WriteLine("File content:");
        Console.WriteLine(fileContent);
    }
    catch (FileNotFoundException)
    {
        Console.WriteLine($"File not found: {filePath}");
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Error reading file: {ex.Message}");
    }
}

static async Task<string> ReadFileAsync(string filePath)
{
    // Use StreamReader to read the file asynchronously.
    using (StreamReader reader = new StreamReader(filePath))
    {
        // Read the whole file asynchronously.
        string fileContent = await reader.ReadToEndAsync();
        return fileContent;
    }
}
```

The ReadFileAsync method uses the StreamReader class to open the file and read its content asynchronously. The await keyword ensures that the reading operation doesn't block the main thread, allowing the program to perform other tasks while the file is being read.

C# Task and Await (Example)

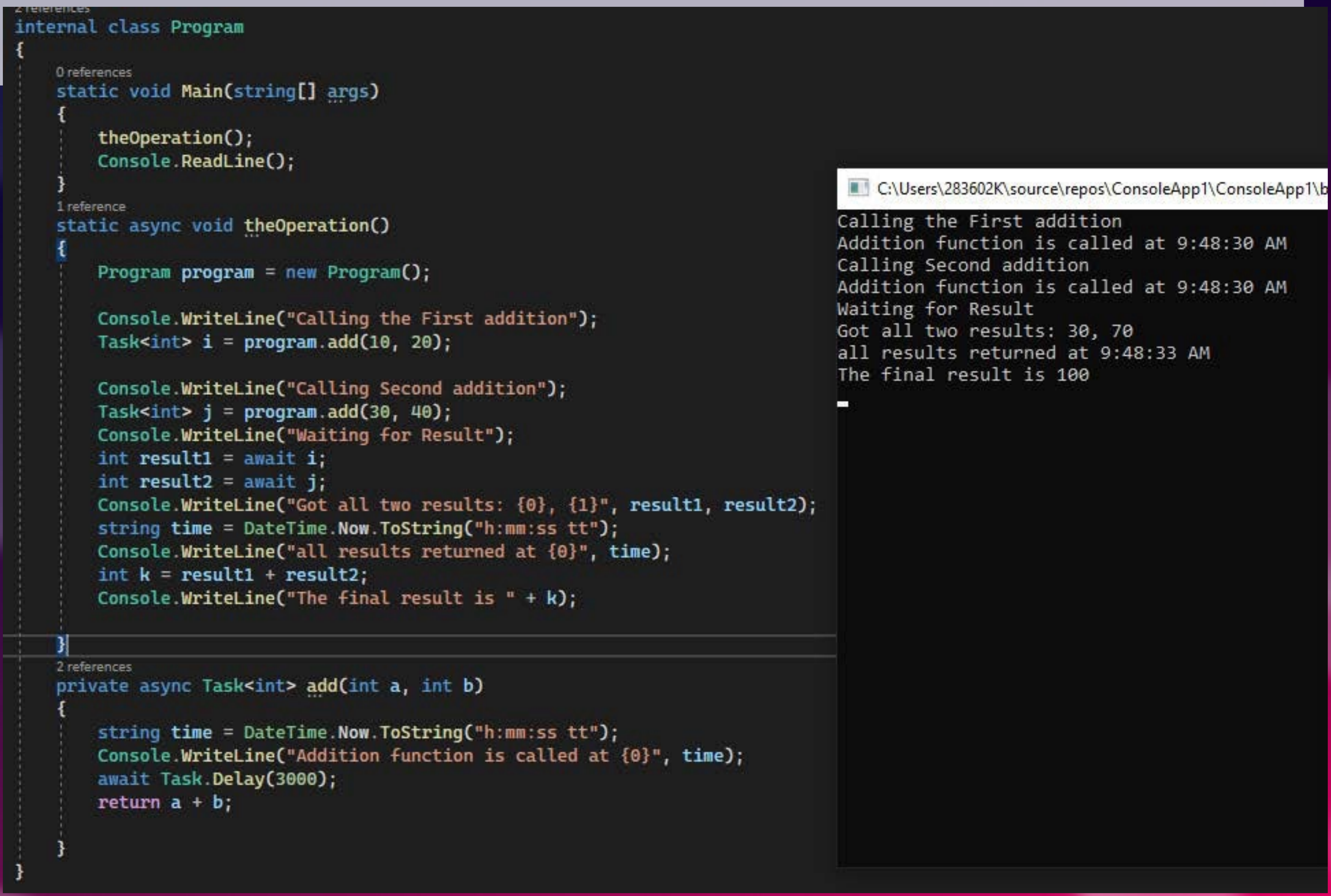

C# Copy

```csharp
Coffee cup = PourCoffee();
Console.WriteLine("Coffee is ready");

Task<Egg> eggsTask = FryEggsAsync(2);
Task<Bacon> baconTask = FryBaconAsync(3);
Task<Toast> toastTask = ToastBreadAsync(2);

Toast toast = await toastTask;
ApplyButter(toast);
ApplyJam(toast);
Console.WriteLine("Toast is ready");
Juice oj = PourOJ();
Console.WriteLine("Oj is ready");

Egg eggs = await eggsTask;
Console.WriteLine("Eggs are ready");
Bacon bacon = await baconTask;
Console.WriteLine("Bacon is ready");

Console.WriteLine("Breakfast is ready!");
```



ref: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/async/

Parallel execution – Async Addition



```
2 references
internal class Program
{
    0 references
    static void Main(string[] args)
    {
        theOperation();
        Console.ReadLine();
    }
    1 reference
    static async void theOperation()
    {
        Program program = new Program();

        Console.WriteLine("Calling the First addition");
        Task<int> i = program.add(10, 20);

        Console.WriteLine("Calling Second addition");
        Task<int> j = program.add(30, 40);
        Console.WriteLine("Waiting for Result");
        int result1 = await i;
        int result2 = await j;
        Console.WriteLine("Got all two results: {0}, {1}", result1, result2);
        string time = DateTime.Now.ToString("h:mm:ss tt");
        Console.WriteLine("all results returned at {0}", time);
        int k = result1 + result2;
        Console.WriteLine("The final result is " + k);

    }
    2 references
    private async Task<int> add(int a, int b)
    {
        string time = DateTime.Now.ToString("h:mm:ss tt");
        Console.WriteLine("Addition function is called at {0}", time);
        await Task.Delay(3000);
        return a + b;

    }
}
```

C:\Users\283602K\source\repos\ConsoleApp1\ConsoleApp1\b

```
Calling the First addition
Addition function is called at 9:48:30 AM
Calling Second addition
Addition function is called at 9:48:30 AM
Waiting for Result
Got all two results: 30, 70
all results returned at 9:48:33 AM
The final result is 100
```

Next class

- Thread Synchronisation
- Delegate
- Applying delegate and Async-await task in WPF